Introducción a MQL4

"Estoy profundamente en deuda co el Sr. Ranat Fatkhullin, el director general de MetaQuotes Software Corp.. por su confianza, apoyo profesional y toda la ayuda posible. Tambien estoy agradecido a los empleados de la empresa, Stanislav Starikov y Rashid Umarov, por su valioso asesoramento y en la ayuda en la redacción de este libro."

Sergey Kovalyov
http://autograf.dp.ua

S. Kovalyov

Programación en lenguaje algorítmico MQL4

Curso Introductorio

Hoy en día, el ordenador personal se ha convertido en algo indispensable para todo el mundo. El rápido desarrollo de Internet y el rendimiento de los ordenadores modernos han abierto nuevas perspectivas en muchos ámbitos de las actividades humanas. En solo un tiempo tan cercano como hace diez años el mercado financiero de trading solo estaba disponible para los bancos y para un numero limitado de una comunidad de especialistas. Hoy en día, cualquiera puede empezar en cualquier momento y unirse al mundo de los comerciantes y profesionales independientes de trading.

Cientos de miles de traders en todo el mundo ya han juzgado el terminal MetaTrader 4. La utilización de su lenguaje de programación incrutado MQL4, ha ascendiendo a los traders a un nuevo nivel de comercio, el comercio automatizado. Ahora, un comerciante puede poner en práctica sus ideas con un programa de aplicación - escribir un indicador personal, un script para realizar una operaciones simples, o crear un Asesor Experto ó un sistema de comercio automatizado (robot de comercio). Un Asesor Experto (AE) puede trabajar en 24/7 (24 horas al dia/ 7 dias a la semana), sin intervención, hacer un seguimiento de precios del valor, enviar mensajes electrónicos, SMS´s a su teléfono móvil, así como hacer muchas otras cosas útiles.

La principal ventaja de las aplicaciones es la posibilidad de hacer operaciones de acuerdo con el algoritmo establecido por el comerciante. Cualquier idea que se puede describir en un lenguaje algorítmico (intersección de dos medias móviles o el procesamiento digital de señales, triples pantallas de Elder o el análisis fractal de Pedros, una red neuronal o construcciones geométricas) puede ser codificada en una aplicación y luego utilizados en la práctica de comercio.

Desarrollo de aplicaciones para el Terminal de Usuario MetaTrader 4 requiere el conocimiento de MQL4. El presente manual le ayudará a crear sus propios Asesores Expertos, scripts e indicadores encarnando en él sus ideas - algoritmos de su rentabilidad comercial. El libro de texto está destinado a un gran número de lectores sin experiencia en programación que deseen aprender a desarrollar aplicaciones de comercio automatizado para Terminal de Usuario MetaTrader 4. El libro de texto está concebido de tal modo que para hacer del aprendizaje MQL4 tan conveniente y consecuente como sea posible.

Prefacio

Hay una especial dificultad en escribir un libro de texto sobre programación para los principiantes, ya que el área de conocimiento objeto de examen implica algunos nuevos conceptos que no se basan en nada previamente conocido o habituales.

En términos generales, problemas de este tipo puede ocurrir en cualquier otra esfera del conocimiento. Por ejemplo, el punto se conoce en matemáticas como círculo infinitesimal, mientras que el círculo en sí se define como un conjunto de puntos ordenados de una cierta manera. Como es fácil ver, estos términos se definen unos a través de otros. Al mismo tiempo, esta "inadvertencia" no se convierte en un escollo para las matemáticas. Los dos círculos, puntos, así como otras condiciones aprobadas en matemáticas van bien juntos. Por otra parte, todo el mundo entiende qué es un punto que es un círculo.

Introducción a MQL4

Es fácil saber que la inmensa mayoría de los términos ordinarios tienen límites indeterminados. Algunos de esos límites son tan difusos que volcaron dudas sobre la existencia del propio objeto o fenómeno definido por el término. Sin embargo, la naturaleza del hombre es que esta extraña (en términos de lógica normal) situación no viene entre un hombre y su existencia y fructíferas actividades. Después de un período de tiempo que estos términos han sido utilizados el concepto toma un sentido completo para nosotros. Es difícil responder a la pregunta de cómo y por qué ocurre de esta manera. Pero lo hace. Nosotros sólo sabemos que múltiples referencias a un término desempeña un papel importante en el notable proceso de aprendizaje de los términos.

Las siguientes tareas fueron establecidas en el presente trabajo:

- Desarrollar el sentido de utilizar nuevos términos con bien conocidos analogías;
- hacer que el significado de cada término intuitivamente evidente cuando se aparece por primera vez;
- Proveer a los lectores con la necesaria cantidad de información que caracterizan a los programas y la programación.

Con este fin, el libro contiene muchos ejemplos y cifras. El texto incluye referencias cruzadas que permiten al lector obtener información sobre temas afines.

Unas pocas palabras sobre la presentación de materiales. Algunos libros de texto sobre la programación invitan a sus lectores sobre la primera página a imprimir "Hola, mundo!" utilizando un programa simple. Sus autores piensan que tan pronto como sus lectores comienzan a aprender programación deben referirse a los textos del programa y poco a poco acostumbrarse a la forma en que el programa pude aparecer de tal modo que posteriormente facilite su aprendizaje. Sin embargo, este enfoque da lugar a que el lector tiene que tratar con varios términos desconocidos, al mismo tiempo, y justo adivinar el contenido y las propiedades de algunas líneas en el programa. Esto puede dar lugar a un malentendido y, consecutivamente, a vacíos en el conocimiento del lector.

A mi modo de ver, sería más eficaz utilizar un método donde el lector va a la siguiente sección en el libro de texto sólo después de que él o ella han tenido una profunda comprensión de los anteriores materiales. En el marco de este método, el primer programa se ofrece al lector sólo después de que él o ella han dominado todas las condiciones necesarias y ha obtenido alguna información sobre los principios básicos de codificación. Este es el método en el que se basa presente libro de texto.

El dominio del conocimiento dado en el libro, el lector tiene que ser un usuario de PC y tener cierta experiencia en trabajar con MetaTrader 4 producidas por MetaQuotes Software Corp.

Índice de contenidos

Introducción a la programación MQL4

Antes de comenzar a estudiar programación MQL4, vamos a definir el alcance de nuestro estudio. En primer lugar hay que señalar que los programas examinados en este libro sólo se pueden utilizar como aplicaciones para trabajar en MetaTrader Terminal de Usuario 4. Fig. 1 a continuación muestra el papel de estos programas en la gestión del comercio. Para una mejor comprensión de la importancia de estos programas en gestión del comercio, echemos un vistazo a la Fig. 1.

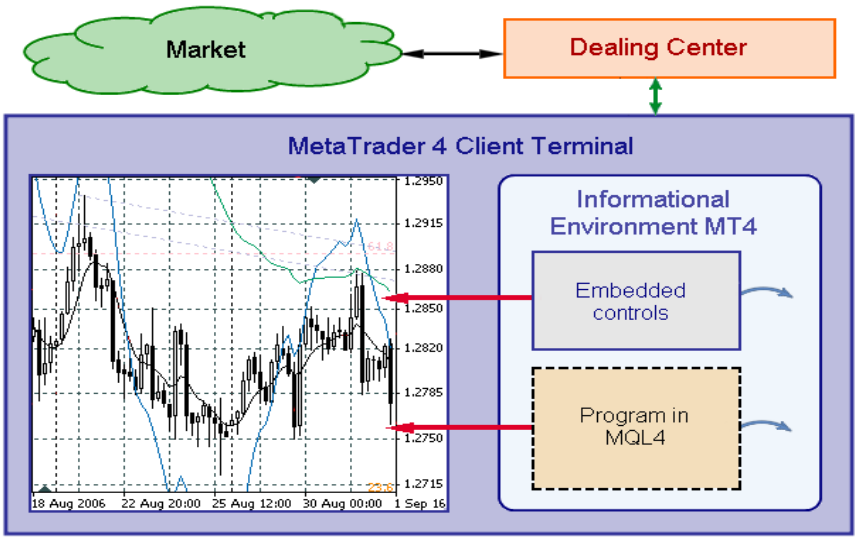


Fig. 1. Un programa en MQL4 como parte de MetaTrader Terminal de Usuario 4.

Si usted está interesado en la programación de MQL4, usted debe estar familiarizado con el Terminal de Usuario. El Terminal de Usuario forma parte del sistema de comercio en línea. Este sistema también incluye un servidor instalado en un dealing center. En el dealing center a su vez están conectados con otros participantes del mercado como bancos e instituciones financieras.

El Terminal de Usuario incluye informaciones del entorno: un conjunto de parámetros que informan sobre el estado del mercado y las relaciones entre el comerciante y el centro de tratamiento de datos. Contiene información sobre los precios actuales, las limitaciones en tamaño máximo y mínimo de las órdenes, distancia mínima de las órdenes de stop, permiso / prohibición del trading automático y muchos otros parámetros útiles que caracterizan el estado actual del mercado. El entorno informeativo se actualiza cuando un nuevo ticks es recibido en el terminal (línea verde en la Fig. 1).

Conjunto de herramientas

El Terminal de Usuario contiene herramientas que permiten la realización de análisis técnico del mercado y gestión de la ejecución manual de comercio. Para el análisis de mercado se pueden utilizar diferentes indicadores tecnicos y estudios: lineas de soporte y resistencia, canales de tendencia, niveles de Fibonacci, etc.

Para la gestión de trading manual se usan las órdenes desde barra de herramientas. Utilizando esta barra de herramientas un trader puede abrir, cerrar y modificar órdenes. Además, el terminal tiene la opción de administración automatizada de stops de la posición. Unas acciones de comercio que incorporen herramientas de gestión de trading dan como resultado la formación de órdenes comerciales las cuales son enviados al servidor.

Para obtener más información sobre el Terminal de Usuario, por favor consulte la sección "Userguide" ***(ClientTerminal_folder \ Terminal.chm).***

Herramientas de programación

El análisis de mercado y la gestión de comercio en Terminal de Usuario MetaTrader 4 se aplican con la ayuda de herramientas de programación. El lenguaje MQL4 permite la creación de tales programas. Hay tres tipos de aplicaciones creadas en MQL4 y creadas para trabajar en el Terminal de Usuario:

- **Costum Indicator**: (Indicador personal) Es un programa para mostrar gráficos de mercado escrito de acuerdo al algoritmo de su autor.
- **Expert Advisor**: (Asesor Experto) Es un programa que permite automatizar gran parte de operaciones comerciales y completo tranding automatico.
- **Script** – Es un programa para ejecutarlo una vez, las acciones incluyen la ejecución de operaciones de trading.

Fig.1 muestra que la aplicación tiene los mismos medios de acceso al Terminal de Usuario y entorno informativo como las herramientas de comercio manual (flechas azules). También puede formar la gestión de medios (flechas rojas), pasó al Terminal de Usuario. Intercambio de datos y programas de diferentes tipos pueden ser utilizados simultáneamente. El uso de estas aplicaciones permite que un programador pueda automatizar una gran parte de las operaciones de comercio o crear un robot que ralice comercio sin la intervencion del comerciante.

Las aplicaciones y herramientas de gestión manual pueden ser utilizadas en el Terminal de Usuario al mismo tiempo y complementarse entre sí.

Las características técnicas fundamentales de la negociación en línea utilizando el sistema de comercio MetaTrader es que la gestión de todas las acciones se producen en el Terminal de Usuario y luego es enviada a un servidor. Los programas de aplicación (Expert Advisor, script o Custom Indicator) sólo pueden trabajar como parte del Terminal de Usuario, siempre y cuando se conecte a un servidor (dealing center). Ninguno de los programas de aplicación se instala en el servidor.

El servidor sólo permite procesar señales procedentes del Terminal de Usuario. Si un Terminal de Usuario está desconectado de Internet o un programa de aplicación (Expert Advisor o script) que haya en ella, sus acciones no generan ninguna gestión, nada va a suceder en el servidor.

El alcance de nuestro estudio incluye los programas (Asesor Experto, scripts personalizados e indicador personal) que permitan llevar a cabo el trading de forma parcial o de forma totalmente automatizada y de esta forma, ampliar significativamente el mantenimiento de información comercial (ver Fig. 1). En este libro usted encontrará la descripción de los componentes del programa y las principales reglas de creación y uso de programas. También vamos a examinar en detalle ejemplos de programas y parámetros de información del entorno en el Terminal de Usuario que están disponibles al programa durante su ejecución.

Los programas para el tratamiento automatizado del comercio poseen mucho más potencial y posibilidades que las herramientas manuales de gestión de comercio.

En la mayoría de los casos, un programa le permite a un comerciante un trabajo más fácil, eliminando la necesidad de un constante seguimiento de la situación del mercado y tener que estar sentado ante un ordenador durante un largo periodo de tiempo. También puede ayudar a aliviar la tensión nerviosa y reducir el número de errores que aparecen en períodos de extrema tensión emocional. Pero lo principal es, que el uso del programa como método de gestión del comercio, permite desarrollar las propias ideas y probarlas con datos históricos, seleccionando los parámetros óptimos para la aplicación de estas ideas y, por último, permite aplicar las ideas sobre las estrategias comerciales.

Fundamentos de MQL4

Esta sección se representa la terminologia básica del lenguaje de programación MQL4:

- Algunos conceptos básicos

  Son descritos términos tales como "tick" (un cambio de precio), "control" en los algoritmos, "comentarios" que describen los programas. El principal acontecimiento de las cotizaciónes en los mercados financieros es el cambio de precio. Esta es la razón por la que el tick es un acontecimiento importante que hace que los mecanismos básicos de los programas de MQL4 se ejecutan ¿Qué hacer cuando ocurre un nuevo tick? ¿Qué medidas tomar? Este es el control que mueve la vanguardia aquí. Pero no se olvide de comentar su código.

- Constantes y Variables

  Los términos de constantes y variables serán explicadas; la diferencia entre estos términos sera explicada. Como el término sugiere, una constante es algo continuo, un valor fijo. A diferencia de la constante, una variable es un objeto del código de programación que puede modificar su contenido. Es imposible escribir un programa sin usar objetos inalterables (constantes) y/o objetos que puedan ser modificados durante la ejecución del programa (variables).

- Tipos de datos

  Ciertos tipos de datos se utilizan en cualquier lenguaje de programación. El tipo de una variable se elige de acuerdo a su finalidad. ¿Cómo podemos declarar una variable, ¿cómo podemos inicializarla (preset su valor inicial)? Una elección errónea del tipo de una variable puede frenar el programa o incluso dar lugar a un mal funcionamiento.

- Operaciones y expresiones

  Las operaciones se hacen sobre operandos ¿Qué tipo de operaciones hay? ¿Cuáles son las características especiales de las operaciones sobre enteros? ¿Por qué es importante recordar los diferentes tipos de datos precedentes? Sin conocer las características de algunas operaciones, pueden aparecer sutiles errores.

- Operadores

  Los operadores pueden ser simples y complejos. Una acción necesaria no siempre pueden ser ejecutadas por un operador simple. Si es necesario que un grupo de operadores se ejecute como un gran operador, este grupo debe incluirse como un operador compuesto. Serán dados los requerimientos necesarios y ejemplos especificos de utilización de los operadores.

- Funciones

  La necesidad de conseguir un código simple nos lleva al término de función. Para poder utilizar la función en distintos lugares del programa, es necesario establecer parámetros a la función. Tendremos en cuenta el proceso de creación de función definida por el usuario. Se ofrecerán ejemplos de uso de funciones estándar.

- Tipos de programa

  Scripts, indicadores personales y Expert Advisor son los tipos de programas de MQL4 que le permiten cubrir prácticamente toda la clase de problemas relacionados con el comercio en los mercados fincancieros. Es necesario comprender los efectos de cada tipo de programas con el fin de utilizar el Terminal de Usuario de MetaTrader 4 de la mejor manera.

Algunos conceptos básicos

Por lo tanto, el objeto de nuestro interés es un programa escrito en MQL4. Antes de empezar una presentación detallada de las reglas para escribir programas, es necesario describir los conceptos básicos que caracterizan a un programa y sus interrelaciones con el entorno de información. El Terminal de Usuario MetaTrader 4 es conocido para trabajar on-line a través de internet. La situación en los mercados financieros cambia continuamente, esto afecta al grafico del símbolo (instrumento) en el Terminal de Usuario. Los ticks proveen al Terminal de información acerca de los cambios de precios en el mercado.

La noción del tick

El **tick** es un evento que se caracteriza por establecer un nuevo precio del instrumento financiero (símbolo) en algún instante.

Los ticks son enviados a cada Terminal de Usuario desde un servidor instalado en un Centro de transaciones bursátiles. Según corresponda a la situación actual del mercado, los ticks pueden recibirse con más o menos frecuencia, pero cada uno de ellos trae una nueva cotización (en el comercio de divisas, es el coste de una moneda expresado en términos de otra moneda).

Una aplicación operando con el Terminal de Usuario puede funcionar durante de un largo período de tiempo, por ejemplo, varios días o semanas. Cada aplicación se ejecuta con arreglo a las normas establecidas para cada determinado tipo de programa. Por ejemplo, un Asesor Experto (AE) no funciona continuamente todo el tiempo. Un Asesor Experto es por lo general puesto en marcha en el momento en que se marca un nuevo tick. Por esta razón, no se caracteriza el tick como si se acabara de marcar un nuevo precio, sino como una orden de ejecución del programa para ser procesada por el Terminal de Usuario.

La duración de la operación del Asesor Experto depende de lo que esté incluido en el código del programa. Normalmente los AEs completan un ciclo de información / procesamiento en algunas décimas o centésimas de segundo. Dentro de este tiempo, el AE puede haber procesado algunos parámetros, tomar una decisión comercial, dar al trader alguna información útil, etc Después de haber terminado esta parte de su labor, el AE se pone en modo de espera hasta que se marque un nuevo tick. Este nuevo tick inicia de nuevo al Asesor Experto (Expert Advisors), el programa hace su trabajo y de nuevo vuelve al modo de espera. A continuación se describe detalladamente cómo la aparición de un nuevo tick actua sobre la operación del programa.

El concepto de control

Usaremos el término «control» cuando hablemos de cómo es el flujo de ejecución de código de un programa así como su interacción con el Terminal de Usuario.

El control es un proceso de realización de acciones preestablecidas por el algoritmo del programa y las caracteristicas del Terminal de Usuario. El control puede ser transferido dentro un programa desde una línea de código a otro, así como desde el programa hacia el Terminal de Usuario.

El control se transfiere de una manera similar a la manera como se da la palabra a alguien en una reunión. Al igual que el orador una reunion toma la palabra y luego se la da a los demás, el Terminal de Usuario y el programa de control de transferencias se pasan el control uno a otro. En este caso, es el Terminal de Usuario quien domina. Su estado es superior a la del programa, al igual que la autoridad del presidente de una reunión es más grande que las de un simple portavoz.

Antes de que el programa se ponga en marcha, el control está bajo la supervisión del Terminal de Usuario. Cuando un nuevo tick es recibido, el Terminal de Usuario transfiere el control al programa. El código del programa comienza a ser ejecutado en este momento.

El Terminal de Usuario, después de que ha transferido el control al programa, no detiene su funcionamiento. Sigue trabajando con el máximo rendimiento durante todo el período de tiempo que ha sido ejecutado desde el ordenador. El programa sólo puede comenzar a funcionar en el momento en que el Terminal de Usuario le ha transferido el control (al igual que el presidente de una reunión controla la reunión todo el tiempo, mientras que el actual orador tome la palabra sólo por un período de tiempo limitado).

El programa devuelve el control al Terminal de Usuario después de que ha completado su operación, y no puede ponerse en marcha por su propia cuenta. Sin embargo, cuando el control ha sido transferido al programa, este devuelve el control al Terminal de Usuario por sí mismo. En otras palabras, el Terminal de Usuario no puede tomar el control del programa por sí solo. La dinámica de las acciones del usuario (por ejemplo, obligar la finalizacion del programa) son una excepción.

Al discutir las cuestiones del rendimiento y las estructuras internas de los programas, estamos interesados la mayoría de la veces en la parte del control que se transfiere dentro de un programa. Vamos a hacer referencia a la Fig. 2 que muestra la naturaleza general de la transferencia del control hacia, desde y dentro de un programa. Los círculos que se muestra en la figura caracterizar algunos de los pequeños fragmentos logicos de un programa, mientras que las flechas entre los círculos muestran cómo se transfiere el control de un fragmento a otro.

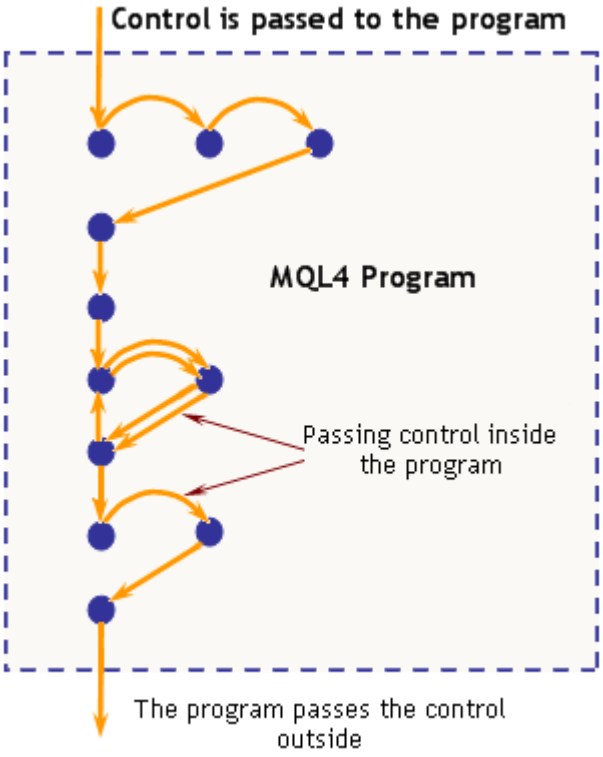


Fig. 2. La transferencia de control en un programa

Un programa que ha aceptado el control del Terminal de Usuario (el programa en ejecución) comienza a ejecutar algunas acciones en función de su algoritmo inherente. El programa contiene las líneas de programa, órdenes generales de ejecución de un programa secuencial que consiste en la transferencia del control de una línea a otra, de la parte superior hacia abajo. Qué y con arreglo a qué reglas se puede escribir en estas líneas se discute más adelante en todos sus detalles.

Aquí, sólo es importante destacar que, lógicamente, se ejecuta cada fragmento terminado (por ejemplo, algunos cálculos matemáticos, un mensaje en la pantalla, una orden comercial etc.) y mantiene el control hasta el fragmento actual del programa ejecutado. Después de que se haya completado todo, el control se transfiere a otro fragmento. Por lo tanto, el control dentro de un programa es transferido desde un fragmento lógico completado a otro. Tan pronto como el último fragmento es ejecutado, el programa de transferencia (ida y vuelta) pasa el control al Terminal de Usuario.

La noción del Comentario

Un programa consiste en dos tipos de registros: los del propio programa y los textos explicativos que están al código de programa.

El **Comentario** es opcional y no es una parte ejecutable del programa.

Por lo tanto, un comentario es opcional dentro de un programa. Esto significa que un programa hace el trabajo de acuerdo con su código, independientemente de si hay comentarios en ella o no. Sin embargo, los comentarios facilitan la comprensión del código del programa en gran medida. Hay comentarios de una línea y de múltiples líneas. El comentario de una linea es cualquier secuencia de caracteres que van a continuacin de una dobe barra (//). El signo de una línea de comentario queda terminado con el salto de línea. El comentrio multi-línea comienza con / * y termina con * / (ver Fig. 3).

Los comentarios son utilizados para explicar el código de programa. Un buen programa siempre contiene comentarios.

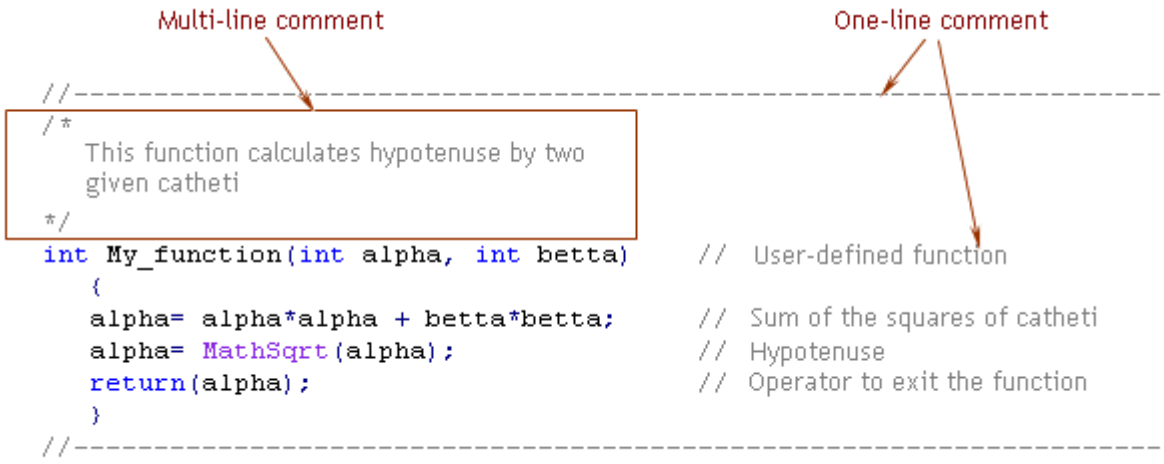


Fig. 3. Ejemplo de comentarios en un programa.

Los comentarios son ampliamente utilizados en la codificación. Por lo general se muestran en gris en los códigos. Vamos a utilizar los comentarios también con el fin de explicar nuestros códigos y hacerlos más inteligibles.

Constantes y Variables

Los términos de «constante» y «variable» se considerarán juntos dentro de esta sección, ya que estos sus conceptos son muy proximos entre sí.

El concepto de constante

Una **Constante** es una parte de un programa, un objeto que tiene un valor.

El término de constante en un programa es similar al que se usa en las ecuaciones matemáticas. Se trata de un valor invariable. Para describir la naturaleza de una constante utilizada en un lenguaje algorítmico en tantos detalles como sea posible, vamos a hacer referencia a su concepto físico y matemático.

La raza humana ha descubierto las constantes, los valores de los que no dependen de nosotros de ninguna manera. Estos son, por ejemplo, en la física: la aceleración de caída libre que es siempre igual a 9,8 m / s / s ó en matemáticas: Pi = 3,14. Las constantes de este tipo no pueden considerarse similares a las constantes de un lenguaje algorítmico.

El término constante también se utiliza en ecuaciones matemáticas. Por ejemplo, en la ecuación de Y = 3 * X + 7, los números 3 y 7 son constantes. Los valores de esas constantes son totalmente dependientes de la voluntad de la persona que ha hecho la ecuación. Esta es la analogía más cercana de constantes utilizadas en los programas de MQL4.

Una constante (como un valor) la coloca un programador en el código en la fase de creación del programa. La constante se caracteriza sólo por su valor, por lo que los términos de «constante» y «el valor de una constante" son sinónimos.

Ejemplo de constates

```
37, 3,14, true, "Kazan"
```

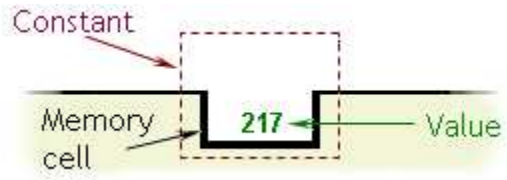


Fig. 4. Una constante en la memoria de un ordenador.

Propiedades de las constantes

La propiedad de una constante es su poder para conservar durante el tiempo de funcionamiento del programa el valor fijado por el programador y entregar este valor al programa cuando el programa se lo pide (Fig. 5). Para cada constante del programa, el ordenador asigna una parte de su memoria del tamaño necesario. El valor de una constante no se puede cambiar durante la ejecución del programa ni tampoco por parte del programador. (Fig. 6).

El valor de una constante es siempre el mismo.

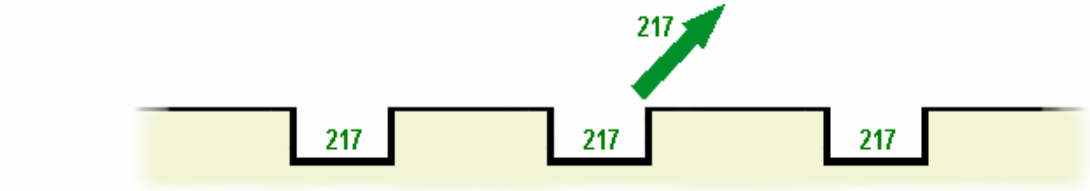


Fig. 5. El estado de una célula de memoria de una constante la hora de fijar el valor al programa.

El valor de una constante no puede cambiarse durante la operación del programa.

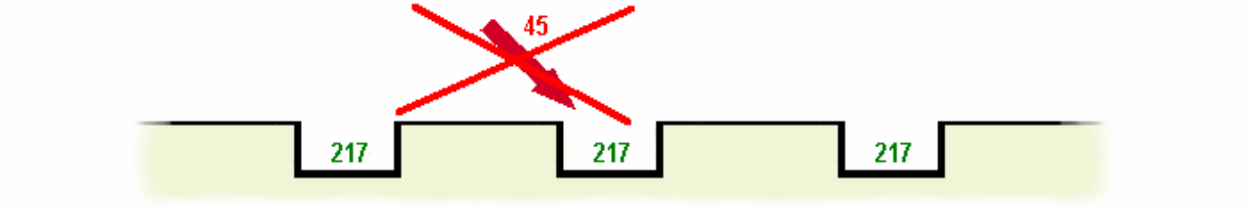


Fig. 6. Es imposible cambiar el valor de una constante durante la operación del programa.

El concepto de variable

Variable es una parte de un programa que tiene un nombre y un valor.

El término de variable en MQL4 es similar al aceptado por las matemáticas. La diferencia entre ellos consiste sólo en que el valor de una variable en matemáticas está siempre implícito, mientras que el valor de la variable en un programa de ejecución se almacena en una celda especial de la memoria del ordenador.

El término "identificador de variable" son plenamente sinonimas al de "nombre de variable". La variable la pone su autor en el código de texto en la fase de codificación como un nombre de variable. El nombre (identificador) de una variable puede constar de letras, dígitos y subrayado. Sin embargo, se debe comenzar con una letra. MQL4 distingue entre mayúsculas y minúsculas, es decir, **S** y **s** no son las mismas.

Ejemplo de nombres de variables: Alfa, alFa, beta, el_número, Num, A_37, A37, qwerty_123

Los siguientes identificadores de las variables representan nombres de variables diferentes: a_22 y A_22; Massa y MASSA.

Ejemplo de valores de variables: 37, 3,14, true, "Kazan".

Propiedades de las variables

La caracteristica de una variable es su capacidad para obtener un valor determinado del programa, conservalo durante todo el período de funcionamiento del programa y entregar este valor para uso del programa cuando este lo solicite. Para cada una de las variables del programa, el ordenador destina el tamaño necesario de una parte de su memoria.

Vamos a hacer referencia a la Fig. 7 y estudiar la construcción de una variable.

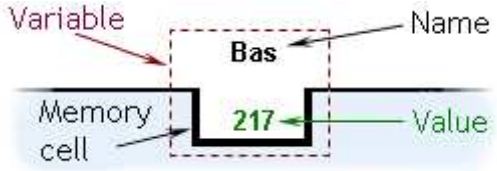


Fig. 7. Una variable en la memoria de un ordenador.

Tenemos el valor de una variable en la celda de memoria de la computadora. Este valor se puede leer y modificar por el programa. El nombre de una variable no cambia nunca. Al escribir un código, el programador puede poner cualquier nombre a la variable. Sin embargo, tan pronto como el programa se pone en marcha, ni el programador, ni el programa pueden cambiar el nombre de la variable.

Si el programa que se está ejecutando encuentra el nombre de una variable, el programa toma esta variable con el fin de obtener su valor para el proceso. Si un programa ha hecho referencia a una variable, esta fija su valor para el programa. Cuando el programa hace una copia del valor de la variable, este valor de la sigue siendo el mismo que figura en la celda de memoria asignada a esta variable (Fig. 8).

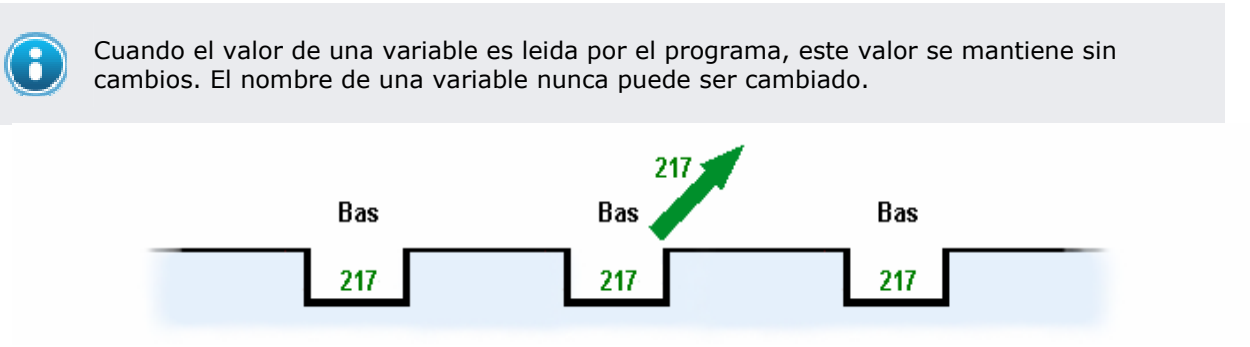


Fig. 8. El estado de la memoria celular de una variable la hora de fijar el valor al programa.

Una variable no está relacionada con la ejecución del programa durante un cierto período de tiempo. Durante este período, el programa puede referirse a otras variables o hacer otros cálculos necesarios. Entre las "sesiones" de comunicación con el programa, la variable conserva su valor, es decir, se mantiene sin cambios.

De acuerdo con el algoritmo del programa, puede ser necesario cambiar el valor de una variable. En este caso, el programa establece la variable a su nuevo valor, y la variable recibe el nuevo valor del programa. Todas las necesarias modificaciones se realizan en la memoria. Esto da lugar a la eliminación del anterior valor de la variable mientras el nuevo valor fijado por el programa toma su lugar. (Fig. 9).

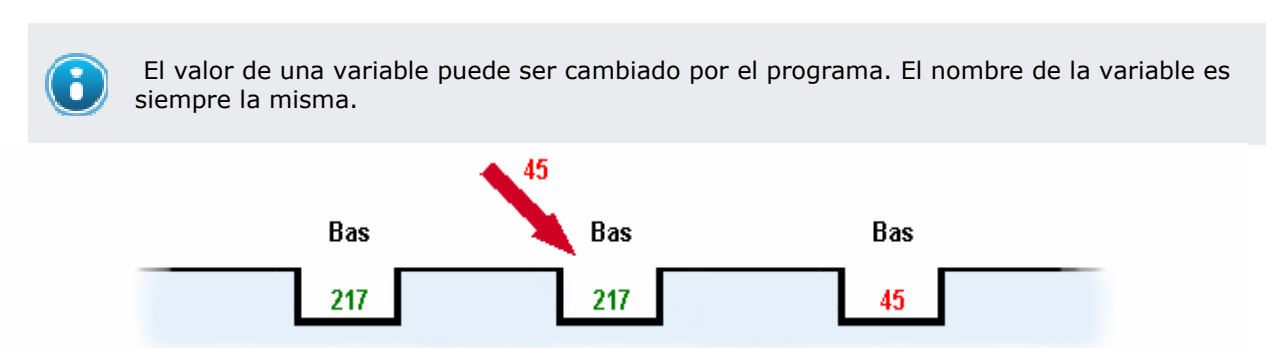


Fig. 9. El estado de la memoria celular de una variable al obtener el valor del programa.

Ejemplo de Constantes y Variables en un programa

Constantes y variables se pueden encontrar en los operadores de un programa. En el código que aparece a continuación, A y B son variables, 7 y 3 son constantes:

```
A = 7; // Línea 1
B = A + 3; // Línea 2
```

Vamos a estudiar la forma de un programa trabaja con constantes y variables. Al ejecutar estas líneas, el programa hará los siguientes pasos:

Línea 1:

1. La constante 7 establece su valor al programa.
2. Una variable recibe el valor 7 del programa.

Línea 2:

1. El programa ha encontrado una expresión a la derecha del signo igual y está tratando de calcularlo.
2. Constante 3 establece su valor al programa.
3. El programa hace referencia a la variable A por su nombre.
4. Una variable fija su valor (7) al programa.
5. El programa hace los cálculos (7 + 3).
6. Variable B obtiene el valor 10 del programa.

El valor de una variable puede ser cambiado durante la operación del programa. Por ejemplo, puede haber una línea en el programa que contenga lo siguiente:

```
B = 33 // Línea 3
```

En este caso, los siguientes pasos se llevarán a cabo en ejecución del programa:

1. Constante 33 establece su valor al programa.
2. Variable B establece el valor 33 (nuevo) del programa.

Es fácil notar que la variable B recibe el valor 10 en una cierta fase de la ejecución del programa y, a continuación, recibe el valor de 33. El nombre de la variable B no ha cambiado durante todos estos acontecimientos mientras que el valor de la variable va a cambiar.

Fig. 10 muestra las constantes y variables en el código de programa:

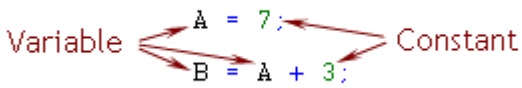


Fig. 10. Un constante y una variable en un programa.

Tipos de datos

Se trata de un conocimiento común que sólo tipos iguales de valores pueden ser sumados o restados. Por ejemplo, las manzanas se pueden añadir a las manzanas, pero las manzanas no pueden sumarse a metros cuadrados o grados centigrados. Similares limitaciones se encuentran en la mayoría de los modernos lenguajes algorítmicos.

Al igual que los objetos normales de la vida tienen determinados tipos que caracterizan su color (rojo, azul, amarillo, verde), su sabor (amargo, ácido, dulce), cantidad (una y media, dos, siete), MQL4 utiliza datos de diferentes tipos. Al hablar de tipo de datos, se entenderá el tipo del valor de una constante, de una variable y el valor devuelto por una función (la noción de función es considerado en la sección de Funciones).

En MQL4, se distinguen los siguientes tipos de datos (para los valores de las constantes, variables, y los valores devueltos por funciones):

- **int** – números enteros;
- **double** - números reales;
- **bool** - Bolean de valores logicos;
- **string**- valores de tipo cadena de caracteres;
- **color** - valores de tipo color ;
- **datetime** - valores de fecha y hora.

Tipo int

Los valores de tipo **int** son enteros. Este tipo incluye los valores que por su naturaleza son de tipo entero. Los siguientes valores son enteros, por ejemplo: cantidad de barras en la ventana de símbolo o instrumento (16000 barras), distancia entre puntos en el símbolo actual de precios y el precio de apertura de la orden (15 puntos). Las cantidades que representan esos objetos como eventos también pueden ser sólo números enteros. Por ejemplo, la cantidad de intentos para abrir una orden no puede ser igual a uno y medio, sólo a uno, dos, tres, etc.

Hay 2 tipos de valores:

- **Decimal** estos valores se presentan en forma de dígitos del 0 al 9 y pueden ser positivos o negativos: 10, 11, 12, 1, 5, -379, 25, -12345, -1, 2.
- Los valores **hexadecimales** estan formados por las letras de la A a la F y los dígitos del 0 al 9. Deben comenzar con 0x o 0x y tomar valores positivos o negativos: 0x1a7b, 0xff340, 0xAC3 0X2DF23, 0X13AAB, 0x1.

Los valores de tipo **int** deben estar dentro del rango de -2 147 483 648 a 2 147 483 647. Si el valor de una constante o una variable está fuera de este rango, el resultado de la operación del programa será nulo. Los valores de constantes y variables de tipo **int** ocupan 4 bytes en la memoria del ordenador.

Un ejemplo que utiliza variable de tipo **int** en un programa:

```
int Art = 10; // Ejemplo variable integer
int B_27 = - 1; // Ejemplo variable integer
int num = 21; // Ejemplo variable integer
int Max = 2147483647 // Ejemplo variable integer
int min = - 2147483648; // Ejemplo variable integer
```

Tipo double

Los valorer de tipo **double** son números reales que contienen una parte decimal.

Los valores de ejemplo de este tipo puede ser cualquier valor que tengan una parte decimal: inclinación de la línea de apoyo, símbolo de precios con una media de cantidad de órdenes abiertas dentro de un día.

A veces se pueden afrontar los problemas designando las variables al escribir su código, es decir, no siempre es evidente para un programador a qué tipo **(int** o **double)** pertenece la variable. Veamos un pequeño ejemplo:

Un programa ha abierto 12 órdenes en una semana. ¿Cuál es el tipo de una variable que considera la cantidad media diaria de órdenes abiertas por este programa? La respuesta es obvia: A = 12 órdenes / 5 días. Esto significa que la variable A = 2,4 deben ser consideradas en el programa como **double,** ya que este valor tiene una parte fraccional. ¿Qué tipo debe ser la misma variable A, si el importe total de órdenes que abrió en el plazo de una semana es de 10? Se puede pensar que si 2 (10 órdenes / 5 días = 2) no tiene parte decimal, una variable puede ser considerada como **int.** Sin embargo, este razonamiento es erróneo. El valor actual de una variable puede tener una pequeña parte que consta sólo de ceros. Es importante saber que el valor de esa variable es real, por su propia naturaleza. En este caso, una variable también ha de ser de tipo **double**. La separación del punto decimal también debe ser mostrada en el registro del programa: A = 2,0

Los valores reales de constantes y variables consistirá en una parte entera, un punto decimal, y una parte decimal. Los valores pueden ser positivos o negativos. La parte entera y la parte decimal se forman con los dígitos del 0 al 9. La cantidad de cifras significativas después del punto decimal puede alcanzar el valor de 15.

Ejemplo:
27,12 -1,0 2,5001 -765456,0 198732,07 0,123456789012345

Los valores de tipo **double** puede oscilar entre -1,7 * e-308 a 1,7 * e308. En la memoria de ordenador, los valores de constantes y variables de tipo **double** toman 8 bytes.

Un ejemplo de utilizar una variable de tipo **double** en un programa:

```
double Arte = 10,123; // Ejemplo de variable real
double B_27 = - 1,0; // Ejemplo de variable real
double Num = 0,5; // Ejemplo de variable real
doble MMM = - 12,07 // Ejemplo de variable real
doble Price_1 = 1.2756; // Ejemplo de variable real
```

Tipo bool

Los valores de tipo **bool** son valores de Boléanos (lógicos) que contienen valores del tipo **true** (verdadero) o **false** (falso).

Con el fin de aprender la noción de los tipos boléanos, vamos a examinar un pequeño ejemplo de nuestra vida cotidiana. Digamos, un profesor necesita para tener en cuenta la presencia de los libros de texto de los alumnos. En este caso, el profesor, confeccionará una lista de todos los alumnos en una hoja de papel y, a continuación se marca en línea si un alumno tiene libro de texto o no. Por ejemplo, el profesor podrá utilizará marcas y guiones en la gráfica:

Lista de alumnos		Libro de texto de Física	Libros de Texto en Biología	Libro de texto en Química
1	Smith	V	--	--
2	Jones	V	--	V
3	Marrón	--	V	V
...	...	...	...	...
25	Thompson	V	V	V

Los valores en las columnas pueden ser sólo de 2 tipos: verdadero o falso. Estos valores no se pueden atribuir a cualquiera de los tipos considerados anteriormente, ya que no son números en absoluto. No son los valores de color, sabor, cantidad, etc. En MQL4, esos valores se denominan boléanos, o valores lógicos. Constantes y variables de tipo **bool** se caracterizan por que sólo pueden tomar 2 valores posibles: **true** (Es cierto, TRUE, 1) o **false** (false, false, 0). Los valores de constantes y variables de tipo **bool** ocupan 4 bytes en la memoria de ordenador.

Ejemplo que utiliza una variable de tipo **bool** en un programa:

```
bool aa = True; // la variable Boolean aa tiene el valor de verdadero
bool B17= TRUE // la variable Boolean B17 tiene el valor de verdadero
bool Hamma = 1; // la variable Boolean Hamma tiene el valor de verdadero

bool TEA = False; // la variable Boolean TEA tiene el valor de falso
bool Nol = false; // la variable Boolean Nol tiene el valor de falso
bool Prim = 0; // la variable Boolean Prim tiene el valor de falso
```

Tipo string (cadena de caracteres)

El valor de tipo **string** es un valor representado como un conjunto de caracteres ASCII.

En la vida cotidiana, un contenido similar pertenecen aquellos tipos que, por ejemplo, almacenar nombres, coches, etc Un un valor tipo **string** se registra como un conjunto de caracteres colocados entre comillas dobles (no para ser mezclado con dobles comillas simples). Las comillas se utilizan sólo para marcar el comienzo y el final de una constante de tipo **string**. El valor en sí es el conjunto de caracteres enmarcados por las comillas.

Si hay necesidad de introducir dobles comillas ("), se debe poner una barra diagonal inversa (\) antes. Cualquier carácter especial puede ser introducido en una constante de tipo **string** tras la barra inversa (\). La longitud de una constante de tipo **string** va de 0 a 255 caracteres. Si la longitud de una constante de tipo **string** es superior a su máximo, el exceso de caracteres en el lado derecho se trunca y el compilador dará el correspondiente aviso. Una combinación de dos caracteres, el primero de los cuales es la barra inversa (\), es comúnmente aceptado y percibido por la mayoría de los programas como una instrucción para ejecutar un determinado formato de texto. Esta combinación no se muestra en el texto. Por ejemplo, la combinación de \ n indica la necesidad de un salto de línea; \ t demanda de tabulación, etc.

El valor de tipo **string,** se registra como un conjunto de caracteres enmarcados por comillas dobles: "MetaTrader 4", "Detener la Pérdida", "Ssssstop_Loss", "stoploss", "10 pips". El valor de la cadena como tal, es el conjunto de caracteres. Las comillas se utilizan sólo para enmarcar las fronteras de la cadena de caracteres. La representación interna es una estructura de 8 bytes.

Ejemplo de utilización de una variable de tipo **string** en un programa:

```
string prefix = "MetaTrader 4"; // Ejemplo variable string
string Postfix = "_of_my_progr. OK"; // Ejemplo variable string
string Name_Mass = "Historial"; // Ejemplo variable string
string texto = "Línea de Alta \n Bajo la línea" //el texto contiene caracteres  de salto de línea
```

Tipo color

El valor del tipo **color** es un valor cromático.

El significado de 'color' (azul, rojo, blanco, amarillo, verde, etc) es de conocimiento común. Es fácil imaginar lo que una variable o una constante de tipo **color** puede significar. Es una constante o una variable, cuyo valor es un color. En términos generales puede parecer ser un poco inusual, pero es muy simple. Al igual que el valor de una constante de tipo entero es un número, el valor de una constante color es un color.

Los valores constantes y variables de tipo color pueden ser representados con una de tres formas distintas:

- **Literales**

  El valor de tipo **color** es representado como un literal y consta de tres partes que representan los valores numéricos de intensidad de tres colores básicos: rojo, verde y azul (RGB). Un valor de este tipo empieza con "C" y el valor numerico esta encerrado entre comillas simples.

  Los valores numéricos RGB de intensidad de 0 a 255 y se pueden grabar tanto en decimal como en hexadecimal.

  Ejemplos: C'128128128'(gris), C'0x00, 0x00, 0xFF' (azul), C'0xFF, 0x33, 0x00’(rojo).

- **Representación Integer** (Representación por enteros)

  La representación integer se registra como número hexadecimal o un número decimal. Un número hexadecimal se muestra como 0xRRGGBB, donde RR es el valor de intensidad de color rojo; GG, verde; y BB, azul. Las constantes decimales no se reflejan directamente en RGB. Representan el valor decimal de un número entero en representación hexadecimal.

  La representación de los valores de tipo color enteros y literales como numeros hexadecimales es muy fácil de usar. La mayoría de los textos modernos y editores gráficos proporcionar información sobre la intensidad de rojo, verde y azul en el valor seleccionado de color. Sólo se tiene que seleccionar un color en su editor y copiar los valores encontrados en su descripción correspondiente a la representación del valor color en su código.

  Ejemplos: 0xFFFFFF (blanco), 0x008000 (verde), 16777215 (blanco), 32.768 (verde).

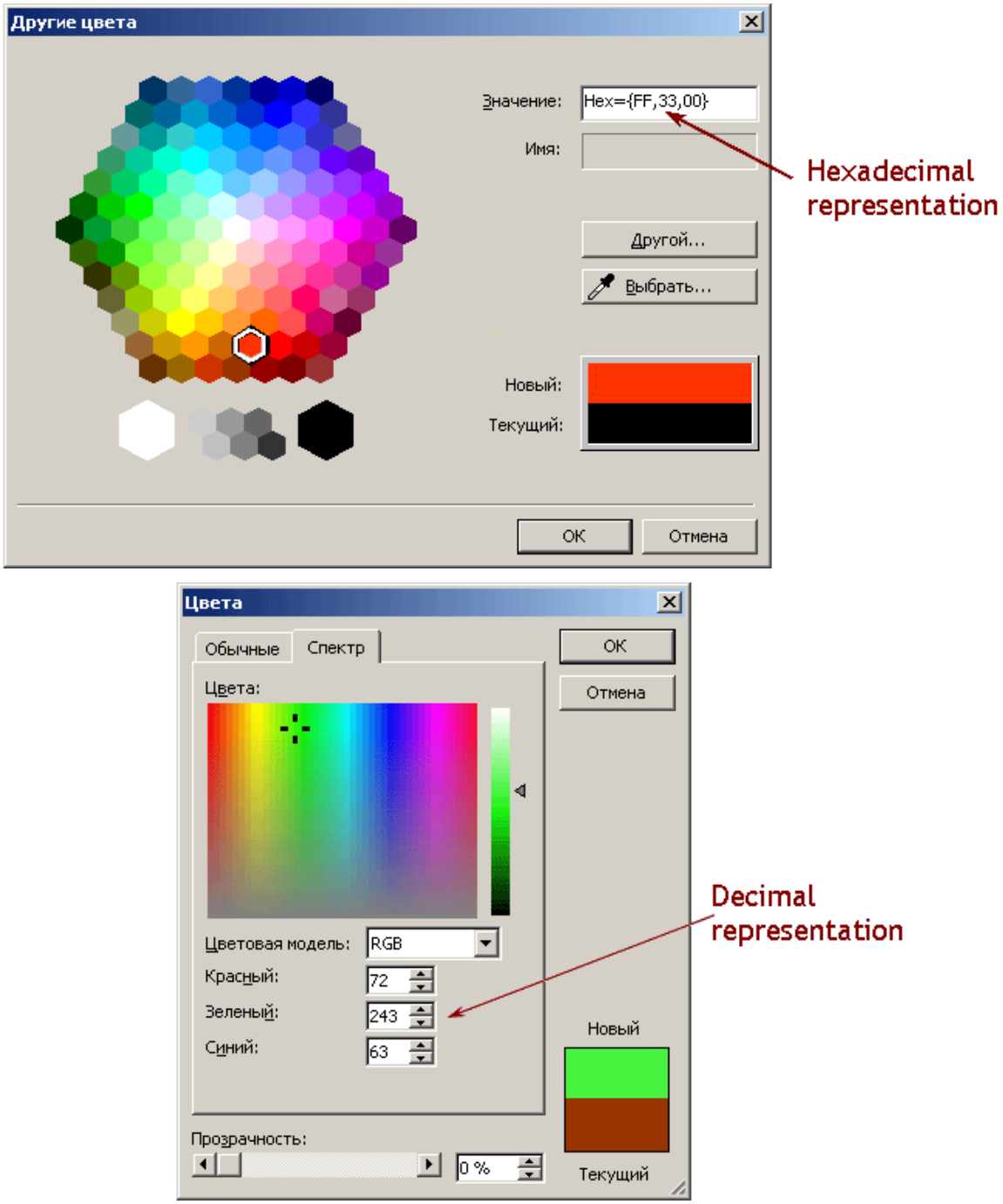


Fig. 11. Parámetros Color para literal entero y representación del valor constante de color que se pueden tomar en los modernos editores.

- **Nombres de colores**

La forma más fácil de definir colores es especificar su nombre de acuerdo con la gráfica de colores web. En este caso, el valor de un color se representa como una palabra que corresponda con el color, por ejemplo, Red - el color rojo.

Black	DarkGreen	DarkSlateGray	Olive	Green	Teal	Navy	Purp
Maroon	Indigo	MidnightBlue	DarkBlue	DarkOliveGreen	SaddleBrown	ForestGreen	OliveD
eaGreen	DarkGoldenrod	DarkSlateBlue	Sienna	MediumBlue	Brown	DarkTurquoise	DimG
tSeaGreen	DarkViolet	FireBrick	MediumVioletRed	MediumSeaGreen	Chocolate	Crimson	SteelB
oldenrod	MediumSpringGreen	LawnGreen	CadetBlue	DarkOrchid	YellowGreen	LimeGreen	Orang
rkOrange	Orange	Gold	Yellow	Chartreuse	Lime	SpringGreen	Aqu
epSkyBlue	Blue	Magenta	Red	Gray	SlateGray	Peru	BlueV
tSlateGray	DeepPink	MediumTurquoise	DodgerBlue	Turquoise	RoyalBlue	SlateBlue	DarkK
ndianRed	MediumOrchid	GreenYellow	MediumAquamarine	DarkSeaGreen	Tomato	RosyBrown	Orch
diumPurple	PaleVioletRed	Coral	CornflowerBlue	DarkGray	SandyBrown	MediumSlateBlue	Ta
rkSalmon	BurlyWood	HotPink	Salmon	Violet	LightCoral	SkyBlue	LightSa
Plum	Khaki	LightGreen	Aquamarine	Silver	LightSkyBlue	LightSteelBlue	LightB
aleGreen	Thistle	PowderBlue	PaleGoldenrod	PaleTurquoise	LightGray	Wheat	Navajo
Moccasin	LightPink	Gainsboro	PeachPuff	Pink	Bisque	LightGoldenrod	Blanched
nonChiffon	Beige	AntiqueWhite	PapayaWhip	Cornsilk	LightYellow	LightCyan	Line
Lavender	MistyRose	OldLace	WhiteSmoke	Seashell	Ivory	Honeydew	AliceB
enderBlush	MintCream	Snow	White				

Las constantes y variables de tipo **color** toman 4 bytes en la memoria de ordenador.

Ejemplo de la utilización de esa variable en un programa:

```
color Paint_1 = C '128, 128, 128'; // El valor gris se asignó a la variable
color Colo= C '0 x00, 0 x00, 0 xff "// El valor azul fue asignado a la variable
color BMP_4  = C '0 xff, 0 x33, 0 x00 "// El valor rojo fue asignado a la variable

color K_12= 0 xFF3300; // El valor rojo fue asignado a la variable
color N_3 = 0 x008000; // El valor verde fue asignado a la variable
color Color = 16777215; // El valor blanco se asignó a la variable
color Alfa = 32768; // El valor verde fue asignado a la variable


color Un = Rojo; // El valor rojo fue asignado a la variable
color B = amarillo; // El valor amarillo fue asignado a la variable
color Colorit = Negro // El valor negro fue asignado a la variable
color B_21 = Blanco // El valor blanco se asignó a la variable
```

Tipo datetime

El valor de tipo **datetime** es un valor de fecha y hora.

Los valores de este tipo puede ser utilizado en los programas para analizar el momento de inicio o finalización de algunos eventos, entre ellos las emisiones de noticias importantes, de trabajo de inicio / finalización, etc. Las constantes de fecha y hora se pueden representar como una línea de un literal constituido de 6 partes que representan el valor numérico del año, mes y día (o día, mes, año), hora, minuto y segundo.

La constant se enmarca entre comillas simples y comienza con 'D'. Está permitido el uso truncado de valores: o bien sin fecha o sin tiempo, o simplemente un valor vacío. El rango de valores va desde el 1 de enero de 1970 al 31 de diciembre de 2037. Los valores de constantes y variables de tipo **datetime** ocupan 4 bytes en la memoria de ordenador. Un valor representa la cantidad de segundos transcurridos desde las 00:00 del 1 de enero de 1970.

Un ejemplo de utilización una variable de tipo **datetime** en un programa:

```
datetime Alfa = D '2004.01.01 00: 00' // Año Nuevo
datetime Tim = D "01.01.2004"; // Año Nuevo
datetime Tims = D '2005.05.12 16: 30: 45'; // 12 de Mayo de 2005 a las 4:30:45 PM
datetime N_3 = D '12/05/2005 16: 30: 45'; // 12 de Mayo de 2005 a las 4:30:45 PM
datetime Compilar = D''; //equivalente de D '[compilación fecha] // 00:00:00
```

Declaración de variables e inicialización

Con el fin de evitar posibles preguntas por el programa acerca de qué tipo de datos pertenece de tal o cual a variable, MQL4 acepta que se especifiquen explícitamente los tipos de variables al inicio del programa. Antes de que una variable empiece a utilizarse en cualquier cálculo deber ser declarada.

La Declaración de Variables es lo primero que se debe hacer con cualquier variable dentro de un programa. En la declaración de una variable siempre ha de especificarse su tipo.

La inicialización de Variables significa la asignación de un valor acorde con su tipo y que se efectua en su declaración. Todas las variables pueden ser inicializadas. Si no hay valor inicial que se establezca explícitamente, la variable se inicializa a cero (0), o si la variable es de tipo string, esta se inicializa como una cadena de caracteres vacía.

En MQL4 se acepta que se especifiquen los tipos de variables explícitamente en su declaración. El tipo de una variable solo se declara en la primera mención del nombre de esta variable. Cuando se menciona el resto de las veces su tipo ya no se vuelve especificar más. En el curso de la ejecución del programa, el valor de la variable puede cambiar, pero su tipo y nombre siguen siendo los mismos. El tipo de una variable puede ser declarada en líneas simples o en los operadores.

Una variable puede declararse en una sola línea:

```
int Var_1; // declaración de variable en una sola línea
```

Este registro significa que existirá la variable **Var_1** (declaración de variable como tal) y el tipo de esta variable en el programa será **int**.

Pueden ser declaradas varias variables en una sola línea.

```
int Var_1, Box, Com // Declaración de varias variables en una línea
```

Este registro significa que en el programa tendremos las variables Var_1, Box y Com, todas de tipo **int**. Esto significa que las variables enumeradas anteriormente serán consideradas por el programa como variables de tipo entero.

Las variables pueden también ser inicializado dentro de los operadores:

```
double Var_5 = 3,7; // Variable inicializada en un operador de asignación
```

Este registro significa que existirá una variable Var_5 de tipo **double** utilizada en el programa y cuyo valor inicial es de 3,7.

El tipo de variable ya no se especifica más en ninguna parte de las líneas siguientes del programa. Sin embargo, cada vez que el programa llama a una variable, "recuerda" que esa variable es del tipo que se ha especificado en su declaración. A medida que avanza el programa los valores de las variables pueden cambiar con los cálculos pero no así su tipo.

El nombre de una variable no tiene relación con su tipo, es decir, no se puede juzgar sobre el tipo de una variable por su nombre. Un nombre dado a una variable también puede utilizarse en variables de todo tipo en diferentes programas. Sin embargo, el tipo de variable solo puede ser declarada una vez dentro de un programa. El tipo de variables declaradas no se modificará durante la ejecución del programa.

Ejemplos de declaración e inicialización de variables

Las variables pueden ser declaradas en varias líneas o en una sola línea.

Se pueden declarar varias variables de un mismo tipo simultáneamente. En este caso, se enumeran las variables separadas por comas, al final de línea se colocará un punto y coma.

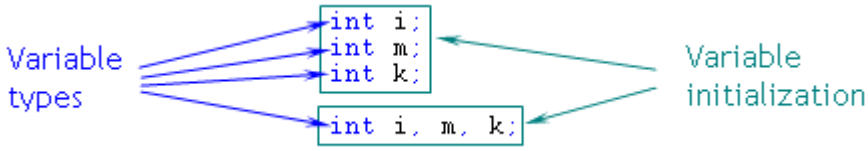


Fig. 12. Ejemplo de declaración de variables en una sola línea.

El tipo de variable se declara una sola vez, en la primera mención de la variable. El tipo no se vuelve a especificar más ni cuando el resto de las veces se menciona la variable.

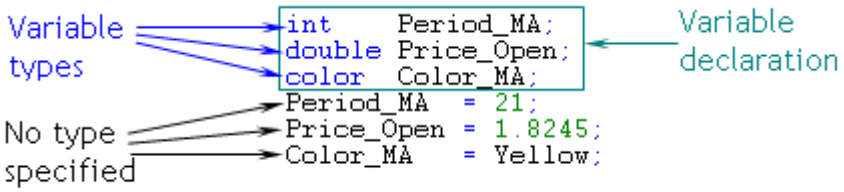


Fig. 13. Ejemplo de declaración de variables en una sola línea.

Se permite declarar e inicializar las variables en los operadores.

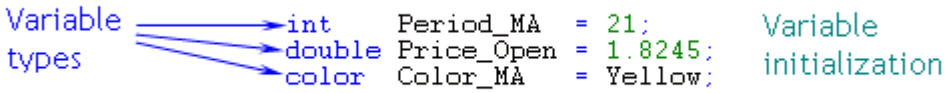


Fig. 14. Ejemplo de inicialización de variables.

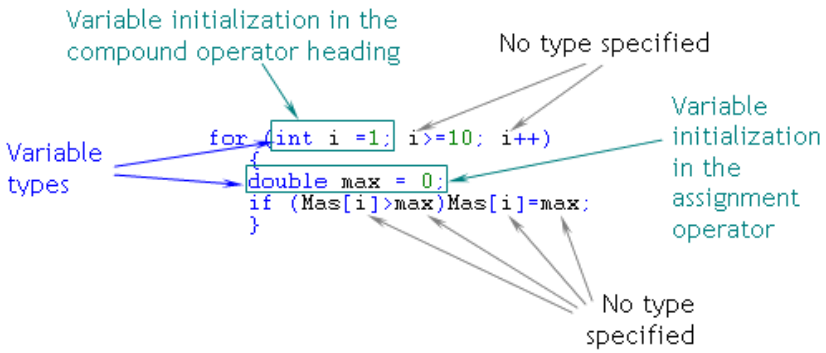


Fig. 15. Inicialización de variable en la cabecera de un operador compuesto.

Operaciones y expresiones

No son necesarias analogías especiales con el fin de comprender la importancia que las operaciones y las expresiones tienen en MQL4. Prácticamente, son las mismas que las operaciones y las expresiones de la simple aritmética. Todo el mundo entiende que en en el registro **f = n + m,** los miembros **f**, **n** y **m** son variables y los signos **+** y **=** son los signos de la operación, mientras que **n + m** es una expresión.

En la sección anterior del libro, aprendimos la necesidad de presentar los datos de diferentes tipos. Aquí vamos a entrar en las posibles relaciones entre estos datos (metros cuadrados no se pueden sumar a manzanas). En MQL4, hay algunas reglas y naturales limitaciones de uso de las operaciones de las expresiones.

Nociones de 'operando', 'Operación', ' Símbolo de la Operación ' y 'Expresión'

Operando es una constante, una variable, un conjunto de componentes o un valor devuelto por una función (el término de función se considera en la sección de Funciones, el de array, en la sección de Arrays; en la presente etapa de aprendizaje, es suficiente comprender los operandos como constantes y variables los cuales ya hemos estudiado antes).

Operación es una acción hecha con los operandos.

Símbolo de la Operación es un carácter preseleccionado o un grupo de caracteres que ordenan ejecutar una operación.

Expresión es una secuencia de operandos y operaciónes de símbolo, es un registro de programa, el valor calculado del cual se caracteriza por un tipo de datos.

Tipos de Operaciones

Existen los siguientes tipos de operaciones en MQL4:

- operaciones aritméticas;
- operaciones de asignación;
- operaciones relacionales;
- operaciones Boolean (lógicas);
- operaciones bitwise;
- operación coma;
- llamada a función.

Las operaciones se utilizan en los operadores (ver Operadores). Sólo en los operadores su utilización tiene sentido y se realiza en un programa. La posibilidad de utilizar una operación está determinada por las propiedades de los operadores (si las propiedades del operador le permiten utilizar esta operación específica, puede usarlo, de lo contrario, usted no debe utilizar esta operación). No está permitido el empleo de las operaciones fuera de los operadores.

Operaciones aritméticas

Los siguientes símbolos pertenecen a símbolos de operaciones aritméticas:

Símbolo	Operación	Ejemplo	Analógica
+	Adición de valores	x + 2	
-	La resta de valores o de cambio de signo	x - 3, y = - y	
*	Multiplicación de valores	3 * x	
/	Cociente de la división	x / 5	
%	Resto de la división	minutos = tiempo 60%	
+ +	Incrementar 1 el valor de la variable	y + +	y = y + 1
--	Decrementar 1 el valor de la variable	y --	y = y - 1

Operaciones de Asignación

Los siguientes símbolos pertenecen a símbolos de operaciones de asignación:

Símbolo	Operación	Ejemplo	Analógica
=	Asignación del valor x a la variable y	y = x	
+ =	Aumento de la variable y en el valor x	y + x =	y = y + x
-=	Reducción de la variable y en el valor x	y -= x	y = y - x
*=	Multiplicación de la variable y por x	y *= x	y = x * y
/ =	División de la variable y entre x	y / x =	y = y / x
% =	Residuo de la división de la variable y por x	y = x%	y = x% y

Operaciones Relacionales

Los siguientes símbolos pertenecen a los símbolos de operaciones relacionales:

Símbolo	Operación	Ejemplo
==	Es cierto si x es igual a y	x == y
! =	Es cierto si x no es igual a y	x! = y
<	Es cierto si x es menor que y	x <y
>	Es cierto si x es más y más	x> y
<=	Es cierto si x es igual o inferior a y	x <= y
>=	Es cierto si x es igual o superior y	x> = y

Operaciones Boolean (lógicas)

Los siguientes símbolos pertenecen a los símbolos de operaciones booleanas:

Símbolo	Operación	Ejemplo	Explicaciones
!	NO (negación lógica)	! x	TRUE (1), si el valor del operando es FALSO (0); FALSO (0), si el valor del operando no es FALSO (0).
||	O (disyunción lógica)	x <5 || x>7	TRUE (1), si algúna de las dos expresiones es cierta
&&	Y (conjunción lógica)	x==3 && y<5	TRUE (1), si las dos expresiones son ciertas

Operaciones Bitwise

Las operaciones Bitwise sólo pueden realizarse con números enteros. Las siguientes operaciones pertenecen a operaciones bitwise:

Complemento a uno del valor de la variable. El valor de la expresión contiene 1 en todos los lugares, en los cuales los valores de la variable contienen 0, y contienen 0 en todos los lugares, en los cuales los valores de la variable contienen 1.

```
b= ~ n;
```

La representación binaria de x que es desplazada y lugares a la derecha. Este desplazamiento lógico a la derecha, significa que en todos los lugares que se han vaciado a la izquierda será rellenado con ceros.

```
x = x>>y;
```

La representación binaria de x que es desplazada y lugares a la izquierda. Este desplazamiento lógico a la izquierda será rellenado con ceros a la derecha.

```
x = x<<y;
```

La operación bitwise AND de las representaciones binarias de x e y. El valor de la expresión contiene 1 (TRUE) en todos los lugares, en tanto que x e y contienen uno, y el valor de la expresión contiene 0 (FALSO) en todos los demás casos.

```
b = ((x + y)! = 0);
```

La operación bitwise OR de las representaciones binarias de x e y. El valor de la expresión contiene 1 en todos los lugares, en la que x o y contienen 1. Contiene 0 en todos los demás casos.

```
b = x | y;
```

La operación bitwise exclusiva o de las representaciones binarias de x e y. El valor de la expresión contiene 1 en los lugares, en los que x e y tienen diferentes valores binarios. Contiene 0 en todos los demás casos.

```
b = x ^ y;
```

Operación de la coma

Las expresiones separadas por comas se calculan de izquierda a derecha. Los efectos de los cálculos a la izquierda de la expresión ocurren siempre antes de que se calcule el lado derecho de la expresión. El tipo y el valor del resultado son coincidentes con el tipo y el valor del lado derecho de expresión.

```
for (i = 0, j = 99; i <100; i++, j--) Imprimir (array [i] [j]); // Loop declaración
```

La lista de parámetros transferidos (véase más adelante) puede considerarse como un ejemplo.

```
My_function (Alf, Bet, Gam, Del)  // La llamada a una función con argumentos
```

Los operadores y las funciones se consideran en las secciones de Operadores y Funciones y en el capítulo de Operadores

Función Call

llamada a la función Call se describe en todos sus detalles en la sección de Función Call.

Operaciones sobre operandos similares

Si en una escuela primaria a los alumnos se les dice que, al resolver el problema sobre el número de lápices, él o ella tendría que basar su presentación en los términos tales como operandos, operadores y expresiones, a los pobres alumnos, sin duda, les resulta imposible. En cuanto a los símbolos de las operaciones, uno puede pensar que la codificación es un misterioso y complicado proceso, accesibles sólo para una especie de elite. Sin embargo, la codificación no es realmente difícil en absoluto, sólo tiene que hacer la cola o la cabeza de algunas intenciones. Para estar seguro de que esto es realmente así, vamos a examinar algunos ejemplos.

Problema 1. Juan tiene 2 lápices, Pedro tiene 3 lápices. ¿Cuántos lápices tienen estos muchachos?:)

Solución. Vamos a indicar el número de lápices de Juan como una variable A y el número de lápices de Pedro como variable B, mientras que el resultado será denominado C.

La respuesta será: A + B=C

En la sección de nombre Tipos de datos, hemos considerado los métodos de declaración de variables. Los lápices son las cosas, es decir, es algo que, básicamente, puede existir como una parte (por ejemplo, puede haber una mitad de un lápiz). Por lo tanto, vamos a considerar como lápices, variables reales, es decir, las variables de tipo *double.*

Por tanto, podemos poner el código de la solución como sigue, por ejemplo:

```
double A = 2,0; // El número de lápices de Juan
double B = 3,0; // El número de lápices de Pedro
double C = A + B // Número total
```

En este caso, la operación "+" se aplica a la suma de los valores de un tipo de variables.

El tipo de valor de la expresión:

```
A + B
```

Será de mismo tipo que el de las variables que componen de la expresión. En nuestro caso, este será de tipo *double*.

Vamos a obtener la respuesta similar para la diferencia entre los valores (¿Cuántos lápices tiene Pedro más que Juan?):

```
double A = 2,0; // El número de lápices de Juan
double B = 3,0; // El número de lápices de Pedro
double C = B - A; // La diferencia entre dos números reales
```

Otras operaciones aritméticas se utilizan de manera similar:

```
double C = B * A; // Multiplicación de dos números reales
double C = B / A // División de dos números reales
```

Cálculos similares pueden realizarse con números enteros también.

Problema 2. Los alumnos van a la pizarra y responden en clase. Juan fue 2 veces, Pedro fue 3 veces. ¿Cuántas veces fueron en total los muchachos a la pizarra?

Solución. Vamos a denotar los viajes de Juan como variable X y viajes de Pedro como variable Y, el resultado como Z.

En este ejemplo, tenemos que utilizar las variables de tipo *int,* ya que consideramos que son eventos entero por su propia naturaleza (no se puede ir a la pizarra 0,5 veces o 1,5 veces; la respuesta a la pizarra puede ser buena o mala, pero sólo interesa el número de esas respuestas o viajes).

La solución de este problema puede escribirse como:

```
int X = 2;                  // Número de viajes de Juan
int Y = 3;                  // Número de viajes de Pedro
int Z = X + Y;              // Número total
```

En el caso de cálculo de la diferencia entre los productos o de cociente de enteros, la operación "-" se utiliza en la forma simple de modo similar:

```
int X = 2;                  // Entero
int Y = 3;                  // Entero
int Z = Y - X;              // Diferencia entre dos enteros
int Z = Y * X;              // Productpo de dos enteros
int Z = Y / X;              // Cociente de dos enteros
```

La situación es algo diferente con las variables de tipo string (*cadena de caracteres):*

Problema 3. En una esquina de la casa, hay una carnicería denominada "Ártico". En otra esquina de la misma casa, hay un establecimiento llamado "Salón de peluquería". ¿Qué está escrito en la casa?

Solución. En MQL4, se permite agregar, junto a la cadena de valores constantes y variables. Si se suman las variables de tipo *cadena,* las cadenas simplemente se añadirán una a otra en la secuencia que se mencionan en la expresión.

Es fácil codificar un programa que nos diera la respuesta:

```
string W1  = "Artico";                 // String 1
string W2  = "Salon de peluqueria";    // String 2
string Ans = W1 + W2;                  // Suma de strings
```

El valor de la variable Ans será la cadena que aparece como sigue:

```
ArticoSalon de peluqueria
```

Se obtuvo una cadena no de muy buen ver, pero absolutamente correcta. Por supuesto, hay que tener en cuenta las lagunas y otros puntuacion en nuestra práctica de codificación de este tipo de problemas.

El resto de operaciones aritméticas con variables de tipo cadena no están permitidas. Por ejemplo:

```
string Ans = W1 - W2 // No permitido
string Ans = W1 * W2; // No permitido
string Ans = W1 / W2 // No permitido
```

Typecasting

El **Typecasting** modifica los tipos de los valores de un operando o de una expresión. Antes de la ejecución de las operaciones (todas las operaciones de asiganción), los valores son modificados al del tipo de más alta prioridad, mientras que antes de la ejecución del signo de asignación los valores se modifican para el tipo objetivo.

Vamos a considerar algunos problemas que se ocupan de typecasting.

Problema 4. Juan tiene 2 lápices, mientras que Pedro se fue 3 veces a la pizarra. ¿Cuántos en total?

Por lo que respecta a la lógica formal se refiere, la incongruencia del problema es evidente. Es evidente que es un error pues no se pueden sumar acontecimientos y cosas.

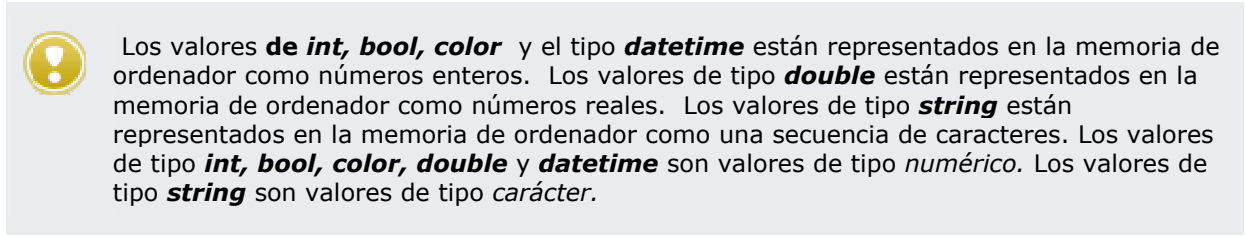
Problema 5. En una esquina de la casa, hay una carniceria denominada "Ártico", mientras que Juan tiene 2 lápices. :)

Con el mismo grado de desesperanza (en la medida que se trate de un normal razonamiento) podemos preguntar:

1. ¿Cuántos en total?, O
2. ¿Qué está escrito en la casa?

Si desea resolver ambos problemas anteriormente en MQL4 correctamente, usted debe hacer referencia a las normas typecasting. En primer lugar, tenemos que hablar de cómo variables de diferentes tipos están representadas en la memoria de ordenador.

Los tipos de datos, tales como ***int, bool, color, double* y *datetime***, pertenecen al tipo de datos *numéricos.* La representación interna de constantes y variables de *int, double, bool, color* y el tipo *datetime* es un número. Las variables de tipo ***int, bool, color* y *datetime*** están representadas en la memoria de ordenador como enteros, mientras que las variables de tipo *double* están representadas como valores de doble precisión, como números en coma flotante, es decir, números reales. El valor de constantes y variables de tipo ***string*** son un conjunto de caracteres (Fig. 16).



Los valores **de *int, bool, color*** y el tipo ***datetime*** están representados en la memoria de ordenador como números enteros. Los valores de tipo ***double*** están representados en la memoria de ordenador como números reales. Los valores de tipo ***string*** están representados en la memoria de ordenador como una secuencia de caracteres. Los valores de tipo ***int, bool, color, double*** y ***datetime*** son valores de tipo *numérico.* Los valores de tipo ***string*** son valores de tipo *carácter.*

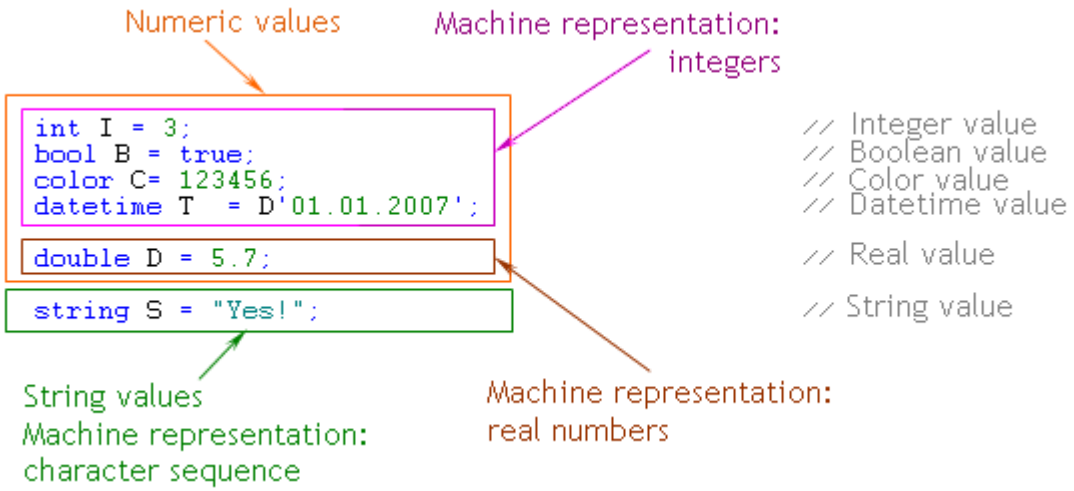
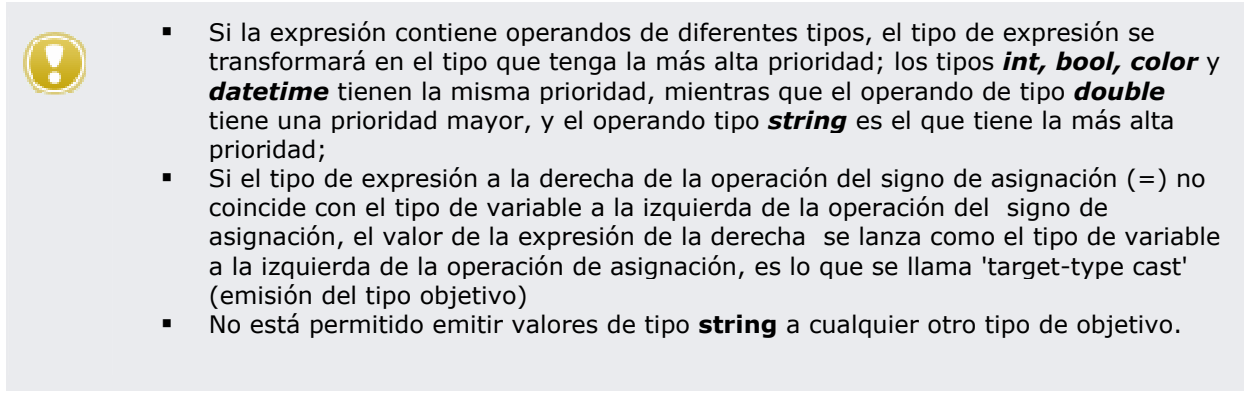


Fig. 16. Representación de diferentes tipos de datos en la memoria de ordenador.

Hemos mencionado anteriormente que los valores de las variables de tipo ***int, bool, color*** y ***datetime*** están representados en la memoria de ordenador como enteros, mientras que los de tipo ***double*** como números reales.Por lo tanto, si queremos encontrar información de una expresión que consta de variables de diferentes tipos, sólo podemos elegir entre tres tipos de datos: ***int, double* y *string***. Los valores de ***bool, color*** y el tipo ***datetime*** resultarán en una expresión iguales que los valores de tipo ***int***.

Entonces, ¿de qué tipo será el valor de una expresión compuesta de operandos con diferentes tipos? En MQL4, la regla implícita de typecasting es la siguiente:



- Si la expresión contiene operandos de diferentes tipos, el tipo de expresión se transformará en el tipo que tenga la más alta prioridad; los tipos ***int, bool, color*** y ***datetime*** tienen la misma prioridad, mientras que el operando de tipo ***double*** tiene una prioridad mayor, y el operando tipo ***string*** es el que tiene la más alta prioridad;
- Si el tipo de expresión a la derecha de la operación del signo de asignación (=) no coincide con el tipo de variable a la izquierda de la operación del signo de asignación, el valor de la expresión de la derecha se lanza como el tipo de variable a la izquierda de la operación de asignación, es lo que se llama 'target-type cast' (emisión del tipo objetivo)
- No está permitido emitir valores de tipo **string** a cualquier otro tipo de objetivo.

Bueno, vamos a volver al Problema 4. No puede haber dos soluciones para ello.

Solución 4,1. Cálculo del resultado de tipo *int:*

```
double A = 2.0;                 // El número de lápices de Juan
 int   Y = 3;                   // El número de viajes Pedro
 int   F = A + Y;               // Número total
```

En primer lugar, necesitamos saber el valor de la expresión siempre que sus operandos sean de distinto tipo. En la expresión:

```
A + Y,
```

Los operandos que se utilizan son de dos tipos de datos: el operando A es de tipo ***double***, y el operando Y es de tipo ***int***.

De conformidad con la regla implícita de typecasting, el valor de esta expresión será un número de tipo ***double***. Tenga en cuenta que: estamos hablando sólo sobre el tipo de la expresión A + Y, pero no sobre el tipo de variable F que está a la izquierda del signo de asignación (=). El valor de esta expresión es el número real de 5,0. Para emitir el tipo de expresión A + Y, se aplicó la primera parte de la regla implícita typecasting.

Después del cálculo de la expresión A + Y se ejecuta la operación de asignación (=). En este caso, los tipos no coinciden, el tipo de la expresión A + Y es el ***double***, mientras que el tipo de variable F es ***int***. Durante el periodo de ejecución de la operación de asignación el tipo de expresión A + Y se emite como tipo ***int*** (según la regla del cálculo) y se convierte en número entero 5, a continuación, este resultado se convierte en el valor del entero variable F. Los cálculos se han realizado de acuerdo con la segunda parte de la regla implícita typecasting - «emisión de tipo objetivo». El resultado final de los cálculos y manipulaciones es la siguiente: El valor del entero de la variable F es un entero de valor 5.

Solución 4,2. Una situación similar se produce, si tratamos de tener un resultado como un valor de tipo ***double***:

```
double A = 2.0;                 // El número de lápices de Juan
int    Y = 3;                   // El número de viajes Pedro
double   F = A + Y;             // Número total
```

Esta situación difiere de la anterior en que el objetivo de la variable tipo F (a la izquierda de la operación del signo de asiganción), en este caso es de tipo ***double*** , coincide con el tipo (***double***) de la expresión A + Y, por lo que no tenemos ningúna emisión de tipo objetivo aquí. El resultado de los cálculos (el valor de tipo ***double*** de variable F es el número real de 5,0.

Vamos a encontrar ahora una solución para el Problema 5. No hay preguntas hasta llegar a la inicialización de variables:

```
string W1 = "Ártico";           // String 1
double A  = 2;                  // El número de lápices de Juan
```

Solución 5,1. Una posible solución para este problema:

```
string W1  = "Ártico";          // String 1
double A   = 2;                 // Número de lápices Juan
string Sum = W1 + A;            // Transformación implicita a la derecha
```

Aquí, en la parte derecha de la expresión existen dos tipos de variables: uno de tipo **string** y otro de tipo ***double***. De acuerdo con la regla implícita de typecasting, el valor de la variable será emitida en primer lugar como tipo string ya que este tipo tiene una prioridad más alta que el tipo double. A continuación este tipo de valores se suman (concatenación). El tipo del valor resultante a la derecha de la operación de asignación sera de tipo **string**. En la siguiente etapa, este valor se le asignará a la variable ***string*** Suma Como resultado, el valor de la variable Suma resultará con el siguiente texto:

```
Arcaico 2.00000000
```

Solución 5,2. Esta solución es errónea:

```
string W1  = "Ártico";        // String 1
double A   = 2;               // Número de lápices Juan
double Sum = W1 + A;          // Transformación implicita a la derecha
```

En este caso, se pretende romper la prohibición de la emision de tipos objetivos de los los valores de tipo ***string***. El tipo del valor de la expresión W1 + A, de tipo **string**, como en la anterior solución. Cuando la operación de asignación se ejecuta, debe realizarse la emisión de tipo objetivo. Sin embargo, de acuerdo a la norma, la emisión del tipo objetivo ***string*** desde tipos de menor prioridad no está permitida. Se trata de un error que detecta el MetaEditor durante la creación del programa (en la compilación).

En general, las reglas que figuran en esta sección son claras y sencillas: Para cualquier cálculo de valores, usted debe emitir todos los diferentes tipos desde un tipo de prioridad más alta. El typecasting de prioridad mas baja solo está permitido para valores numéricos pero no con cadena de caracteres porque estas no pueden transformarse en números.

Características de los cálculos Integer

Se sabe que los números enteros son valores sin parte fraccional o decimal. Estos valores se pueden suman o restar juntos y el resultado que se obtiene es muy intuitivo. Por ejemplo, si:

```
int X = 2;                    // Primera variable int
int Y = 3;                    // Segunda variable int
```

y:

```
int Z = X + Y;                // Operación de adición
```

no hay ningún problema para calcular el valor de la variable Z: 2 + 3 = 5

Del mismo modo, si se ejecuta una operación de multiplicación:

```
int Z = X * Y;                // Operación de Multiplicación,
```

el resultado es muy previsible: 2 * 3 = 6

Pero, ¿qué resultado obtenemos si nuestro programa tiene que ejecutar una operación de división?

```
int Z = X / Y;                // División de las variables
```

Es fácil escribir 2 / 3. Sin embargo, no es un entero. Así que, ¿cuál será el valor de la expresión X/Y y Z variable?

La regla de cálculo de valores de tipo entero es que siempre se descarta la parte decimal.

En el ejemplo anterior, la expresión a la derecha del signo de igual sólo contiene números enteros, es decir, en este caso no se lleva a cabo typecasting. Y esto significa que el tipo de expresión X / Y es *int.* Así que el resultado de hallar el valor de la expresión X/Y (= 2 / 3) es 0 (cero). Este valor (cero) se le asignará a la variable Z al final.

De la misma manera, otros valores de las X e Y produciran otros resultados. Por ejemplo, si:

```
int X = 7;                    // El valor de una variable int
int Y = 3;                    // El valor de una variable int
int Z = X / Y;                // División de las variables
```

En este caso el valor de 7/3 para la expresión X/Y y Z variable es igual a 2 (dos).

Orden de cálculo de las operaciones

La regla para el cálculo de las operciones es la siguiente:

El valor de una expresión se calcula de acuerdo a las prioridades de las operaciones aritméticas y de izquierda a derecha, cada resultado intermedio se calculará de acuerdo a las normas typecasting.

Vamos a considerar el fin para el cálculo de una expresión en el siguiente ejemplo:

```
Y = 2.0*(3*X/Z - N) + D;                    // Expresión ejemplo
```

La expresión a la derecha del signo igual se compone de dos sumandos: 2,0 * (3 * X / Z - N) y D. El sumando 2,0 * (3 * X / Z - N) consta de dos factores, a saber: 2 y (3 * X / Z - N). La expresión entre paréntesis, 3 * X / Z - N, a su vez, consta de dos summands, sumando los 3 * X / Z que consta de tres factores, a saber: 3, X y Z.

Con el fin de calcular la expresión a la derecha de la igualdad de signo, entonces, en primer lugar, calcular el valor de expresión 3 * X / Z. Esta expresión contiene dos operaciones (multiplicación y división) del mismo rango, por lo que el cálculo de esta expresión de izquierda a derecha. En primer lugar, vamos a calcular el valor de expresión 3 * X, el tipo de esta expresión es el mismo que el tipo de la variable X. A continuación, vamos a calcular el valor de expresión 3 * X / Z, su tipo se calculará de acuerdo con la regla typecasting. Después de eso, el programa va a calcular el valor y el tipo de la expresión 3 * X / Z - N, entonces, de la expresión 2,0 * (3 * X / Z - N), y el último de toda la expresión * 2,0 (3 * X / Z - N) + D.

Como es fácil de ver, el orden de las operaciones en un programa es similar a la de matemáticas. Sin embargo, el anterior se diferencia en el cálculo de los tipos de valores en los resultados intermedios, que ejerce una influencia significativa en el resultado final de los cálculos. En particular, (a diferencia de las normas aceptadas en las matemáticas), el orden de los operandos en una expresión es muy importante. Para demostrar esto, vamos a examinar un pequeño ejemplo.

Problema 6.Calcular los valores de las expresiones A/B*C y A*C/B para los enteros A,B y C.

El resultado del cálculo es intuitivamente esperado que sea el mismo, en ambos casos. Sin embargo, esta afirmación es cierta sólo para los números reales. Si queremos calcular los valores de las expresiones compuestas de los operandos de tipo ***int***, debemos siempre considerar el resultado intermedio. En tal caso, la secuencia de operandos es de fundamental importancia:

```
int A = 3;                  // Valor de tipo int
int B = 5;                  // Valor de tipo int
int C = 6;                  // Valor de tipo int
int Res_1 = A/B*C;          // Result 0 (cero)
int Res_2 = A*C/B;          // Resultado 3 (tres)
```

Vamos a seguir el proceso de cálculo de la expresión A / B * C:

1. En primer lugar (de izquierda a derecha) se calculará el valor de la expresión A / B. De acuerdo con las reglas anteriores, el valor de la expresión (3/5) es integer 0 (cero).
2. Cálculo de la expresión 0*C (cero multiplicado por C). El resultado es integer 0 (cero).
3. Otros resultados (el valor de la variable Res_1) es integer **0 (cero).**

Vamos a seguir ahora la evolución del cálculo de la expresión A * C / B.

1. Cálculo de A * C. El valor de esta expresión es entero 18 (3 * 6 = 18).

2. Cálculo de la expresión 18 / B. La respuesta es evidente: después de la parte decimal se ha descartado, (18 / 5) es entero 3 (tres).

3. Otros resultados (el valor de la variable Res_2) es entero **3 (tres).**

En el ejemplo anterior, consideramos que sólo un pequeño fragmento de código, en la que se calculan los valores de las variables de tipo ***int***. Si queremos sutituir estas variables con constantes con los mismos valores, el resultado final sera el mismo. Al calcular las expresiones que contengan enteros, se debe prestar mayor atención a los contenidos de sus líneas de programa. De lo contrario, puede ocurrir un error en su código, que sería muy difícil de encontrar y arreglar más tarde (sobre todo en programas grandes). Tales problemas no se producen en los cálculos utilizando números reales. Sin embargo, si utiliza operandos de diferentes tipos en una expresión compleja, el resultado final puede depender completamente de un fragmento formado fortuitamente que contiene la división de números enteros.

En la sección de operadores, son considerados el término y las propiedades generales de los operadores y también las propiedades especiales de cada operador se describen en el capítulo llamado operadores.

Operadores

El termino operador

Uno de los principales conceptos de cualquier lenguaje de programación es el término de «operador». La codificación parece ser imposible para la persona que no ha aprendido por completo este término. Cuanto antes y mejor, aprende un programador lo que son los operadores, y cómo se aplican en un programa, antes se inicia éste en la escritura de sus propios programas.

El operador es parte de un programa, una frase de un lenguaje algorítmico que prescribe un determinado método de conversión de datos.

Cualquier programa contiene operadores. La analogía más cercana a «operador» es una frase. Así como una sentencia compone el texto normal de una novela, así los operadores componen un programa.

Propiedades de los operadores

Hay dos tipos de propiedades de los operadores: una propiedad común y las propiedades específicas de los distintos operadores.

La Propiedad común de los operadores

Todos los operadores tienen una propiedad común: todos ellos se ejecutan.

Podemos decir que el operador es una instrucción que contiene la guía de operaciones, la descripción de una orden. Para un ordenador ejecute un programa significa que (consecutivamente, pasando de un operador a otro) se cumplan las órdenes, (las recetas, instrucciones) que figura en los operadores.

Un Operador, como tal, es sólo un registro, una cierta secuencia de caracteres. Un Operador no tiene palanca, cables o células físicas de memoria. Es por ello que, cuando un ordenador está ejecutando un programa, no pasa nada en los operadores como tales, ellos siguen permaneciendo en el programa como los compuso el programador. Sin embargo, el equipo es el que tiene todas esas células de memoria y los vínculos entre ellas y todas las experiencias dentro de las transformaciones. Pero si su PC ha ejecutado algunas transformaciones de datos con arreglo a las instrucciones contenidas en un operador, usted dice: el operador se ha ejecutado.

Propiedades específicas de los operadores

Existen varios tipos de operadores. Cada tipo de operadores tiene sus propiedades específicas. Por ejemplo, la propiedad de un operador de asignación es su capacidad para asignar un valor a una variable dada, la propiedad de un operador de bucle es sus múltiples ejecuciones, etc. Las propiedades específicas de los operadores se consideran en todos los detalles en las secciones correspondientes del capítulo Los operadores de este libro. Vamos a decir aquí que cada tipo de operador tiene sus propias propiedades, que son típicas sólo por su tipo y no se repiten en ningún otro tipo.

Tipos de Operadores

Hay dos tipos de operadores: los operadores simples y los compuestos.

Operadores simples

Los operadores simples de MQL4 terminar con el carácter ";" (punto y coma). El uso de este separador, es para que el PC pueda detectar cuando un operador termina y otro comienza. El carácter ";" (punto y coma) es tan necesario en un programa como carácter "." (punto) lo es en un texto normal para separar las frases. Un operador puede tener varias líneas. Se pueden colocar varios operadores en una línea.

Cada operdor simple finaliza con el carácter ";" (punto y coma).

Ejemplos de operadores simples:

```
Day_Next= TimeDayOfWeek(Mas_Big[n][0]+60);   // Operador simple

Go_My_Function_ind();                        // Operador simple

a=3;  b=a*x+n;  i++;                         // Varios operadores colocados en linea

   Print("   Day= ",TimeDay(Mas_Big[s][0]),    // Un operador...        "                    "
     "   Hour=",TimeHour(Mas_Big[s][0]),   // colocado..
     "   Minute=",TimeMinute(Mas_Big[s][0]),  // en varias..
     "   Mas_Big[s][0]= ",Mas_Big[s][0],       // lineas
     "   Mas_Big[s][1]= ",Mas_Big[s][1]);
```

Operadores compuestos

Un operador compuesto consta de varios operadores simples separados por el carácter ";" y se vinculan entre llaves. Con el fin de poder utilizar varios operadores donde se espera que haya solo uno, los programadores utilizar un operador compuesto (también lo llaman "bloque" o "bloque de código"). El conjunto de operadores simples estan ubicados en un recinto separado por llaves. La presencia de una llave de cierre marca el final de un operador compuesto.

Un ejemplo de utilización de un operador compuesto es un operador condicional. Comienza con el operador condicional if (expresión), seguido por un bloque compuesto de operadores simples llamado cuerpo. Este cuerpo contiene una lista de operadores ejecutables.

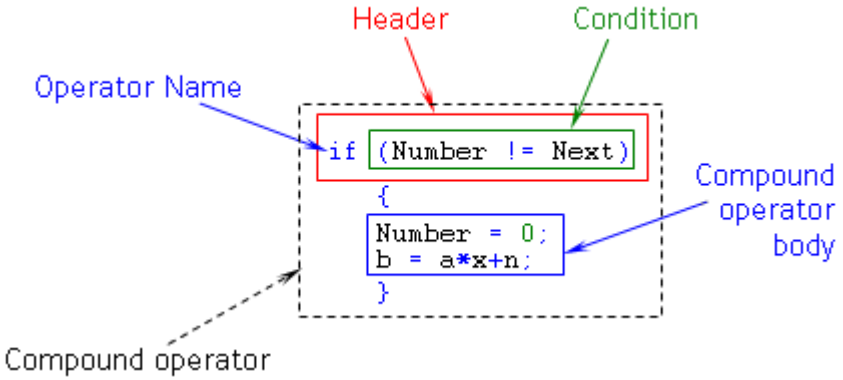


Fig. 17. Operador compuesto.

El cuerpo de un operador compuesto se situa entre llaves. Todos los operadores compuestos finalizan con una llave de cierre.

Ejemplos de operadores compuestos:

```
                                 // Ejemplo del operador switch
switch(ii)                         // Operador switch(expresion)
  {                                // Apertura de llave
  case 1: Buf_1[Pok-f+i]= Prognoz;  break; // Nested operators (cuerpo del operador)
  case 2: Buf_2[Pok-f+i]= Prognoz;  break; // Nested operators (cuerpo del operador)
  case 3: Buf_3[Pok-f+i]= Prognoz;  break; // Nested operators (cuerpo del operador)
  }                                // Cierre de llave que..
                                   // .. muestra donde acaba el operador compuesto
//-------------------------------------------------------------------------------------
                                 // Ejemplo de uso en un bucle o lazo.
for (tt=1; tt<=Kol_Point[7]; tt++)         // Operador for(expresiones)
  {                                // Apertura de llave
  Numb = Numb + Y_raz[tt]*X_raz[ii][tt];   // Nested operators (cuerpo del operador)
  }                                // Cierre de llave que..
                                   // .. muestra donde acaba el operador compuesto
//-------------------------------------------------------------------------------------
                                 // Ejemplo de operador condicional if
if (TimeDay(Mas_Big[f][0])!= 6)            // if (expresion)
  {                                // Apertura de llave
  Sred =(Nabor_Koef[ii][vv][2]+ NBh)*Point// Nested operators (cuerpo del operador)
  Ind = Nabor_Koef[ii][vv][0] + f;         // Nested operators (cuerpo del operador)
  Print(" Ind= ",Ind);                  // Nested operators (cuerpo del operador)
  }                                // Cierre de llave que..
                                   // .. muestra donde acaba el operador compuesto
```

El cuerpo de un operador compuesto esta siempre encerrado entre llaves y puede estar formado de cero, uno, o varios operadores.

Ejemplos de operadores simples:

```
//------------------------------------------------------------------------------
                           // Ejemplo del operador for
  for (n=1; n<=Numb; n++)         // for(expresiones)
    Mas[n]= Const_1+ n*Pi;       // Nested operators (cuerpo del operador)
//------------------------------------------------------------------------------
                           // Ejemplo del operador condicional if
  if (Table > Chair)             // if (expresion)
    Norma = true;                // primer operador (suboperador 1)
  else                          // Else-condición
    Norma = false;               // segundo operador (suboperador 2)
//------------------------------------------------------------------------------
```

Varios operadores simples se pueden combinar en un operador compuesto sin tener estricta necesidad.

Esta es una rara enfermedad, pero una construcción absolutamente admisible. En este caso, los operadores se encierran ente llaves, y se denominan "bloque de operadores". Este uso es completamente aceptable. El programador es el que decide si incluir o no a los operadores entre llaves, simplemente en aras de una representación conveniente del código.

Ejemplo de bloque de operadores:

```
{                                  // Apertura de llave
  Day_Next= TimeDayOfWeek(Mas_Big[n][0]+60);   // Operador simple
  b=a*x+n;                              // Operador simple
  }                                 // Cierre de llave..
```

Requisitos de los Operadores

Los operadores deben estar escritos en el texto de un programa de acuerdo a las normas de formato (la forma en que deben estar representados en un código). Ningún operador puede estar construido más allá de estas reglas. Si el programa contiene un operador compuesto con un formato fuera de las reglas, el MetaEditor producirá un mensaje de error en la compilación. Esto significa que el programa que contiene el operador erróneo no puede utilizarse.

Se debe entender la frase "formato del operador" como un conjunto de normas de formato, típico para todos los operadores del mismo tipo. Cada tipo de operador tiene su propio formato. Los formatos del Operador son considerados en todos sus detalles en las secciones de este libro correspondientes al capítulo Operadores.

Orden de ejecución de los operadores

Una característica muy importante de cualquier programa es el orden de ejecución de los operadores dentro de él. Los operadores no pueden ser ejecutados sin razón o por excepción. En MQL4, el orden de ejecución de los operadores es el siguiente:

Los operadores se ejecutan en el orden, en el que se aparecen en el programa. La dirección de los operadores de ejecución va de izquierda a derecha y de arriba a abajo.

Esto significa que tanto los los operadores simples como los compuestos y se ejecutan uno a uno (como en las líneas de los poemas: en primer lugar se lee la línea superior, después la siguiente inferior, después la siguiente y así sucesivamente). Si hay varios operadores en una línea, deben ser ejecutadas consecutivamente, uno después de otro, de izquierda a derecha, a continuación, los agentes se ejecutan en la línea inferior más cercana en el mismo orden.

Los operadores que integran un operador compuesto se ejecutan de la misma manera: todo operador del bloque de código comienza a ser ejecutados sólo después de que lo ha hecho el anterior.

Redacción y Ejecución de Operadores: Ejemplos

El texto de un programa que contiene los operadores no es diferente en aspecto a un texto normal o una notación matemática. Sin embargo, esta similitud es sólo formal. Un texto normal permite que las anotaciones puedan ser colocadas en cualquier secuencia, mientras que en un programa se debe mantener un orden bien definido.

A modo de ejemplo, veamos como trabaja un operador de asignación. Vamos a resolver un simple sistema de ecuaciones lineales y comparar la representación de algunos cálculos matemáticos en un texto normal con los operadores de un código de programa..

Problema 7. Tenemos un sistema de ecuaciones:
Y = 5
Y - X = 2
Hayar el valor numérico de la variable.

Solución 1. En un texto normal en una hoja de papel:

1. 5 - X = 2
2. X = 5 - 2
3. X = 3

Solución 2. Un texto en un programa:

```
Y = 5;              // Linea 1
X = Y - 2;          // Linea 2
```

Tanto en la primera y la segunda en las soluciones, las anotaciones (líneas) han completado un contenido. Sin embargo, las líneas en Solución 1 no se pueden utilizar en un programa tal como son, porque su apariencia no cumple con el formato del operador de asignación.

Las anotaciones que figuran en la Solución 1 representan algunas funciones matemáticas en papel. Sólo pueden utilizarse para informar a los programadores sobre las relaciones entre las variables. Los operadores en un programa se asignan para otros fines, que informará a la máquina de las operaciones y en qué orden se debe ejecutar. Todos los operadores, sin excepción alguna, representan instrucciones precisas que no permiten ambigüedades.

Los operadores de la Solución 2 son los operadores de asignación. Cualquier operador de asignación da literalmente a la máquina la siguiente orden:

Calcula el valor de la expresión que se encuentra a la derecha de la igualdad y asigna el valor obtenido a la variable situada a la izquierda del signo de igualdad. Es decir, en el lado izquierdo de la igualdad solo puede haber una variable y en el lado derecho una expresión con cualquier grado de complejidad

Por esta razón, solo una variable puede estar situada a la izquierda del signo de igualdad de un operador de asignación. Por ejemplo, un registro de 5 - X = 2 utilizada en la primera solución contendría un error, porque el conjunto de caracteres 5 - X no es una variable.

Vamos a seguir al ordenador durante la ejecución de los operadores de la segunda solución.

1. Paso por el operador (línea 1).

```
Y =5; // Línea 1
```

2. Referencia a la parte derecha del operador (la parte derecha se encuentra entre el signo de igualdad y el punto y coma).
3. El ordenador ha detectado que la parte derecha del operador contiene un valor numérico.
4. Registro en la memoria fisica del ordenador del valor numérico (5) de la variable Y.
5. Paso al siguiente operador (línea 2).

```
X = Y-2; // Línea 2
```

6. Referencia a la parte derecha del operador.
7. El ordenador ha detectado que la parte derecha del operador contiene una expresión.
8. Cálculo del valor numérico de la parte derecha del operador (5 - 2).
9. Registro en la memoria física del valor numérico (3) de la variable X.

La realización de los pasos 1-4 en la computadora es la ejecución del primer operador (línea 1). La realización de los pasos 5-9 en la computadora es la ejecución del segundo operador (línea 2).

Con el fin de código de un programa viable, el programador debe también darse cuenta de qué, y en qué orden se ejecutará este programa. En particular, no todos los cálculos matemáticos se pondrán en un programa, a veces es necesario antes preparar a los operadores.

Por ejemplo, muchos cálculos intermedios se realizan cuando se ha hayado la solución de otros problemas matemáticos. Estos pasos intermedios pueden ayudar a un matematico a encontrar una solución adecuada, pero resultan inútiles desde el punto de vista de la programación. Sólo soluciones significativas pueden ser incluidos en un programa, por ejemplo: valores originales de las variables o las fórmulas para calcular los valores de otras variables. En el ejemplo anterior, el primer operador tiene información sobre el valor numérico de la variable Y, y el segundo operador establece la fórmula para calcular el valor de la variable X que nos interesa.

Cualquier programa viable contiene expresiones de significado familiar, pero también se puede encontrar expresiones que usted será capaz de comprender sólo si se comentan tanto como sea posible, en el programa. Por ejemplo, el registro de abajo…

```
X = X + 1; // Ejemplo de un contador
```

parece ser erróneo desde el punto de vista logico matemático y el propio sentido común. Sin embargo, es bastante aceptable si se considera como un operador (por cierto, este operador se utiliza ampliamente en la codificación).

Con este operador, hemos calculado un nuevo valor de la variable X: cuando se ejecuta el operador de asignación (es decir, el cálculo del valor de la parte derecha del operador), el ordenador toma el valor de la memoria física que contiene el valor numérico de la variable X (que en el ejemplo resulta ser igual a 3 en el momento de referirse a ella), calcula la expresión en la parte derecha del operador de asignación (3 + 1), y escribe el valor obtenido (4) en la memoria celular (física) de la variable X. La máquina almacenará este valor de la variable X hasta que la variable X se produce en la parte izquierda del signo de igualdad en otro operador de asignación. En este caso, el nuevo valor de esta variable se calculará y se almacenan hasta el próximo posible cambio.

Funciones

Término de una función

El más importante avance tecnológico en ingeniería informática es la posibilidad de creación y almacenamiento de fragmentos de código separado que describan normas de procesamiento de datos para resolver un problema o una pequeña tarea. Esta posibilidad tambien existe en MQL4.

Función es el nombre que recibe una parte específica de un programa que describe un método de conversión de datos.

Hablando de funciones, vamos a considerar dos aspectos: descripción o definición de la función y funcion de llamada

Función descripción o definición es la parte del programa destinada a su ejecución.

Función de llamada (función de referencia) es un registro, es el acto que conduce a la ejecución de la función.

En nuestra vida cotidiana, podemos encontrar muchas análogias de la función. Tomemos, por ejemplo, el sistema de frenado de un coche. El mecanismo de accionamiento que lleva a cabo el frenado en un vehiculo. La idea implentada por el ingeniero es análoga a la definición/descripción de función, mientras que el pedal de freno es el análogo de la llamada a la función. El conductor presiona el pedal, y los mecanismos de accionamiento realizar ciertas acciones y detenienen el coche.

Del mismo modo, si la llamada a una función se produce en un programa, entonces la función del mismo nombre será llamada y ejecutada, es decir, se llevarán a cabo una cierta secuencia de cálculos u otras acciones (por ejemplo, se muestra un mensaje o una orden de apertura, etc) El sentido general de una función es la adopción de una lógica que se completa fuera del texto base del programa, mientras que sólo se mantiene dentro del texto base del programa la parte del código que se ocupa de la llamada de esta. Este programa de construcción tiene algunas ventajas incontestables:

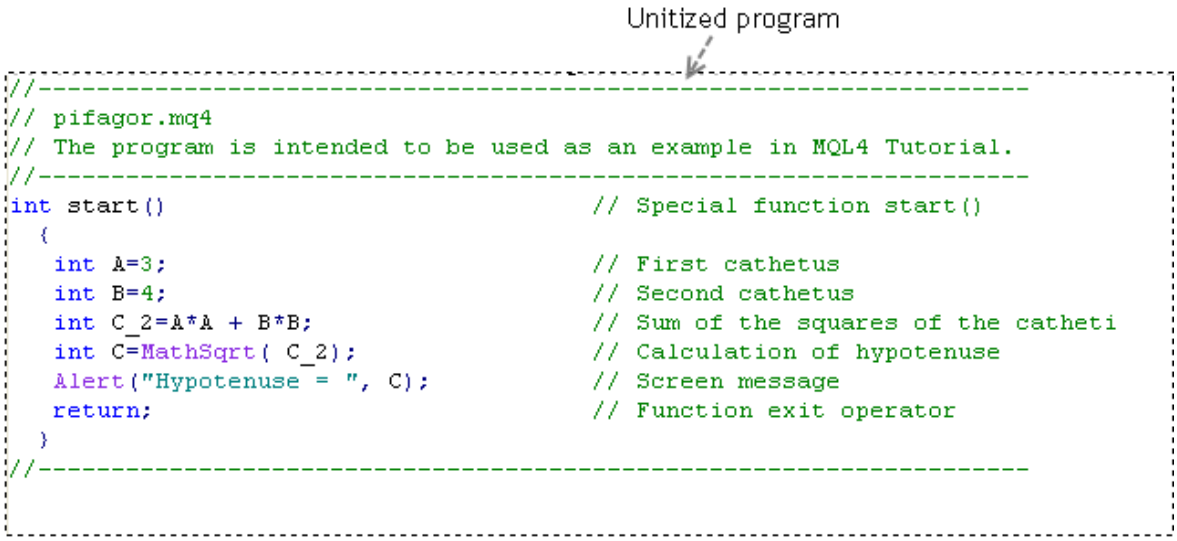
- Primera, el texto del programa está integrado de tal manera que se lee mucho más fácil.
- Segunda, se puede ver con facilidad y, si es necesario, modificar el texto de una función sin realizar ningún cambio en el código básico ó programa principal.
- Y tercera, una función puede estar compuesta como un solo archivo y usarse en otros programas, el cual liberará al programador de la necesidad de insertar el mismo fragmento de código en cada programa de nueva creación.

Podemos decir que la mayor parte del código de los programas que usan MQL4 está escrito en forma de funciones. Este enfoque se extendió y actualmente es un estándar.

Composición de una función

Por lo tanto, una función está compuesta de **la descripcion** y **la llamada**. Vamos a considerar un ejemplo. Supongamos que tenemos un pequeño programa (Fig. 18) que considera la longitud de la hipotenusa utilizando los otros dos lados del triángulo rectangulo y el teorema de Pitágoras.

En este programa, todos los cálculos se encuentran juntos, los operadores son ejecutados uno por uno en el orden en el que se producen en el programa (de arriba hacia abajo).



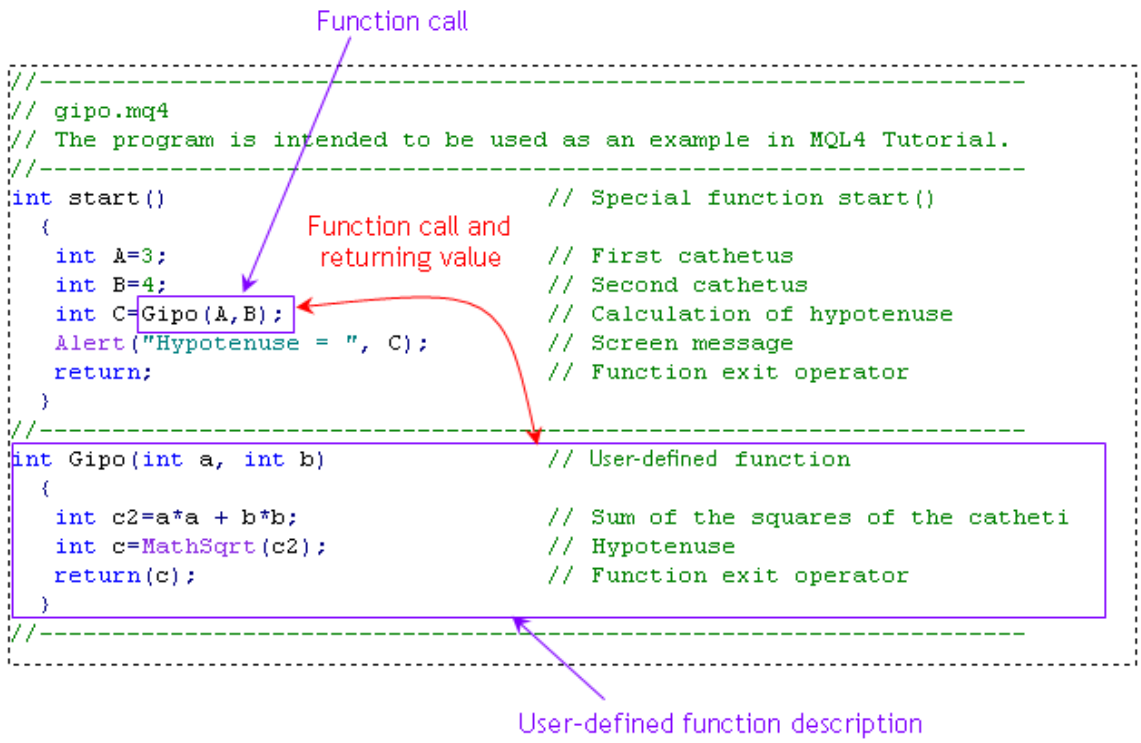

```
//+------------------------------------------------------------------+
// pifagor.mq4
// The program is intended to be used as an example in MQL4 Tutorial.
//+------------------------------------------------------------------+
int start()                                 // Special function start()
  {
   int A=3;                                 // First cathetus
   int B=4;                                 // Second cathetus
   int C_2=A*A + B*B;                       // Sum of the squares of the catheti
   int C=MathSqrt( C_2);                    // Calculation of hypotenuse
   Alert("Hypotenuse = ", C);               // Screen message
   return;                                  // Function exit operator
  }
//+------------------------------------------------------------------+
```

Fig. 18 años. El código de programa único pifagor.mq4.

Problema 8. Redactar una parte del código del programa como una función.

Sería razonable hacer una función utilizando las dos líneas que encuentran el valor buscado. El mismo programa pero usando una función se muestra en la Fig. 19.

En este programa, una parte de los cálculos se integra como una función. El código básico contiene una llamada a la función definida por el usuario. La descripción de la función definida por el usuario se encuentra fuera (después de) del código básico.




```
//+------------------------------------------------------------------+
// gipo.mq4
// The program is intended to be used as an example in MQL4 Tutorial.
//+------------------------------------------------------------------+
int start()                                 // Special function start()
  {
   int A=3;                                 // First cathetus
   int B=4;                                 // Second cathetus
   int C=Gipo(A,B);                         // Calculation of hypotenuse
   Alert("Hypotenuse = ", C);               // Screen message
   return;                                  // Function exit operator
  }
//+------------------------------------------------------------------+
int Gipo(int a, int b)                      // User-defined function
  {
   int c2=a*a + b*b;                        // Sum of the squares of the catheti
   int c=MathSqrt(c2);                      // Hypotenuse
   return(c);                               // Function exit operator
  }
//+------------------------------------------------------------------+
```

Fig. 19. El código de un programa que contiene la descripción y la llamada a función definida por el usuario gipo.mq4.

Ambas versiones del programa dará el mismo resultado. Sin embargo, el código se compone de un único módulo en la primera versión (Fig. 18), mientras que en la segunda versión (Fig. 19) una parte de los cálculos se ejecuta en la llamada a una función desde el texto de base. Una vez terminados los cálculos separados en la función, continuarán los cálculos en el código del programa principal.

Para conseguir la ejecución de la función, tenemos que llamarla. Esta es la razón por que la función esta representada en dos partes: el propio código que compone la función (función descripción) y la llamada a la función para poner en marcha la función (refiriéndose a ella). Si no se llama a la función, esta no será ejecutada. Al mismo tiempo, si se llama a una función inexistente, esto se traducirá en nada (El MetaEditior dará un mensaje de error si se intenta compilar un programa así en MQL4).

Descripción de la Función

La descripción de una función consta de dos partes básicas: cabecera de la función y cuerpo de la función.

La **cabecera** de una función está formada por: el tipo del valor de return, el nombre de función y la lista de parámetros formales. La lista de parámetros formales están encerrados entre paréntesis y se colocan después del nombre de la función. El tipo del valor de return puede ser uno de los tipos que ya conocemos: ***int, double, bool, color, datetime,* o *string***. Si la función no devuelve ningún valor, su tipo puede ser denominado ***void*** ("sin contenido, vacío") como se denominaríacualquier otro tipo.

El **cuerpo** de una función se encierra entre llaves. El cuerpo de la función puede contener operadores simples y/o complejos, así como la llamada a otras funciones. El valor devuelto por la función se da en el paréntesis de operador return (). El tipo del valor devuelto es el utilizando el operador return () y debe coincidir con el tipo de la función especificada en la cabecera de la función. La descripción de la función se termina con el cierre de la llave.

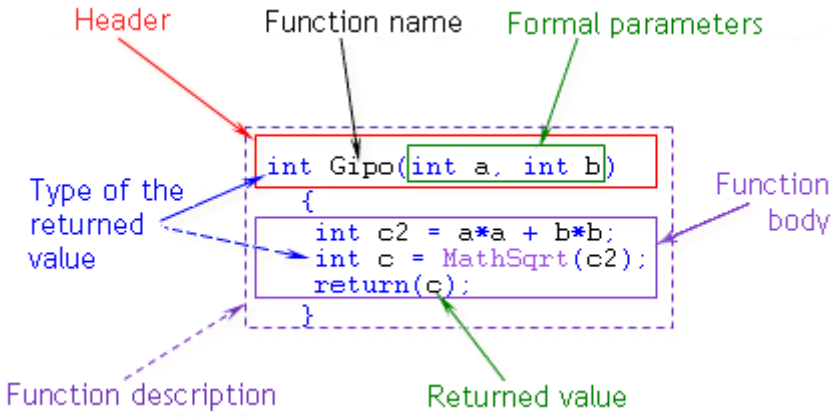


Fig. 20. Descripción de la función.

La Descripción de la función debe estar ubicada separada del programa, al margen de cualesquiera otras funciones (es decir, no dentro de otra función, si no fuera de ella).

Llamada a la Función

La **llamada a la Función** se representa con el nombre de la función y la lista de parámetros transferidos. La lista de parámetros se transfieren entre paréntesis. La llamada a la función puede ser representada como un operador independiente o como parte de un operador.

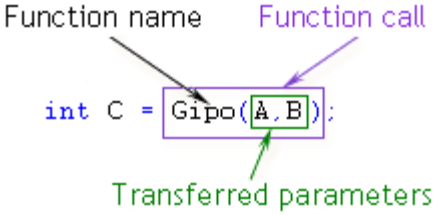


Fig. 21. Llamada a la Función (referencia a una función).

Cualquier llamada a la función es siempre dentro de otra función (es decir, no fuera de todas las demás funciones, pero dentro de una de ellas).

Tipos de funciones

Hay tres tipos de funciones: funciones especiales, funciones estándar (built-in o predefinidas), y funciones definidas por el usuario.

Funciones especiales

En MQL4, hay 3 funciones especiales en total. Ellas tienen nombres predefinidos: **init (), start (),** y **deinit ()** que no pueden utilizarse como nombres de cualquiera otras funciones. El examen detallado de las funciones especiales se da en el capítulo Funciones especiales. Sólo decir aquí que el código básico de un programa se encuentra dentro de estas funciones (Fig. 18, 19).

La característica especial de las funciones especiales es el hecho de que son llamadas para su ejecución desde el Terminal de Usuario. Aunque las funciones especiales tienen todas las propiedades de las funciones en general, no se les suele llamar desde el programa si éste está codificado correctamente.

Funciones estándar

MQL4 tiene una serie de útiles funciones en las cuales, cuando se escribe la codificación del programa no es necesario hacer su descripción. Por ejemplo, el cálculo de raíces cuadradas, la impresión de mensajes en el sistema o en la pantalla. Todas estas y muchas otras funciones estándar se realizan de acuerdo con un algoritmo predefinido. El usuario no necesita saber el contenido de estas funciones. Él o ella sólo puede estar seguro de que todas las funciones estándar son desarrolladas debidamente por profesionales y de acuerdo a la mejor algoritmo posible.

La característica singular de las funciones estándar es que no están descritas en el texto del programa. Las funciones estándar son llamadas en el programa de la misma manera a como lo hace cualquier otra función (es una práctica común).

En nuestro ejemplo (Fig. 18, 19), se utilizan dos funciones estándar: MathSqrt () y Alerta (). La primera está destinada al cálculo de raíces cuadradas, mientras que la segunda está diseñada para mostrar un determinado mensaje de texto, puesto entre paréntesis, en la pantalla. Las propiedades de las funciones estándar son consideradas con más detalles en la sección Funciones estándar. Aquí vamos a observar que estos registros representan llamadas a funciones estándar, mientras que descripciones de estas funciones no se las puede encontrar en el programa. Estas funciones, también pueden ser denominadas funciones built-in, o funciones predefinidas. Usted puede usar cualquiera de estos términos.

Funciones definidas por el usuario

En algunos casos, los programadores crean y utilizan sus propias funciones y hacen la llamada estas funciones (definidas por el usuario). Las Funciones definidas por el usuario se utilizan en los programas con la descripción de la función y las llamadas a la funcion.

Cuadro 1. La descripción de la Función y la llamada a la función son utilizadas en los programas dependiendo de los diferentes tipos de funciones.

Tipo de función	Descripción de la Función	Llamada a la Función
Especial	Implementable	No procede (*)
Estandar	No se implementa	Aplicable
Definida por el usuario	Implementable	Aplicable

(*) A pesar de que las funciones especiales puede ser técnicamente llamadas desde un programa, no se recomienda hacerlo.

Propiedades de las Funciones

Ejecución de la Función

La principal propiedad de todas las funciones es que la llamada a la función hacen que estás se ejecuten. Las funciones se ejecutan de acuerdo a sus códigos.

El paso de parámetros y el valor de return

Una función se comporta como una calculadora estándar, en cuanto a que la información se recibe y se devuelve transformada. Se puede escribir (usando el teclado) una cierta expresión que consta de varios valores van entrando uno a uno, y se obtendrá un valor como respuesta. Una función puede recibir y procesar uno o varios parámetros del programa que le ha llamado para su ejecución, y la función terminará su operación de retornando (transmitiendo, dando, entregando) un parámetro como respuesta a este programa.

El **paso de los parámetros** se especifican encerrandos entre paréntesis después del nombre de la función que se llama. El paso de parámetros se hace usualmente separandolos medianter comas. El número de parámetros transferidos a la función no debe superar 64. La función también puede omitir el uso de paso de parámetros. En este caso se especifica una lista vacía de parámetros, es decir, simplemente hay que poner un paréntesis de apertura y uno de cierre directamente después del nombre de función.

El número, tipos y orden de los parámetros transferidos en la llamada a la función debe coincidir con los parámetros de formación que se especifica en la descripción de la función (la llamada de una función que tiene parámetros por defecto es una excepción - véase la Llamada a la función y Descripción de funciones y el operador " return "). Si no coinciden, el MetaEditor le dará un mensaje de error. Constantes, variables, expresiones y arrays pueden ser utilizadas como parámetros de paso.

El **valor de return** se especifica en los paréntesis del operador return () (ver Descripción de funciones y el operador "return"). El tipo del valor devuelto utilizando en el operador return () debe coincidir con el tipo de la función dada en la cabecera de la función. También es posible que una función no devuelva ningún valor. En este caso, no se especifica nada en el paréntesis del operador return ().

En nuestro ejemplo, el paso de parámetros son variables **A** y **B** (Fig. 21), mientras que el valor de return es la variable **c** (Fig. 20). El requisito de que concuerden los tipos de paso y los parámetros formales pueden verse en la Fig. 22.

El número, tipo y orden de los parámetros transferidos en la llamada a la función debe coincidir con los parámetros que se especifican en la descripción de la función. Solamente se pueden utilizar variables en los parámetros formales de la cabecera de la descripción de la función.

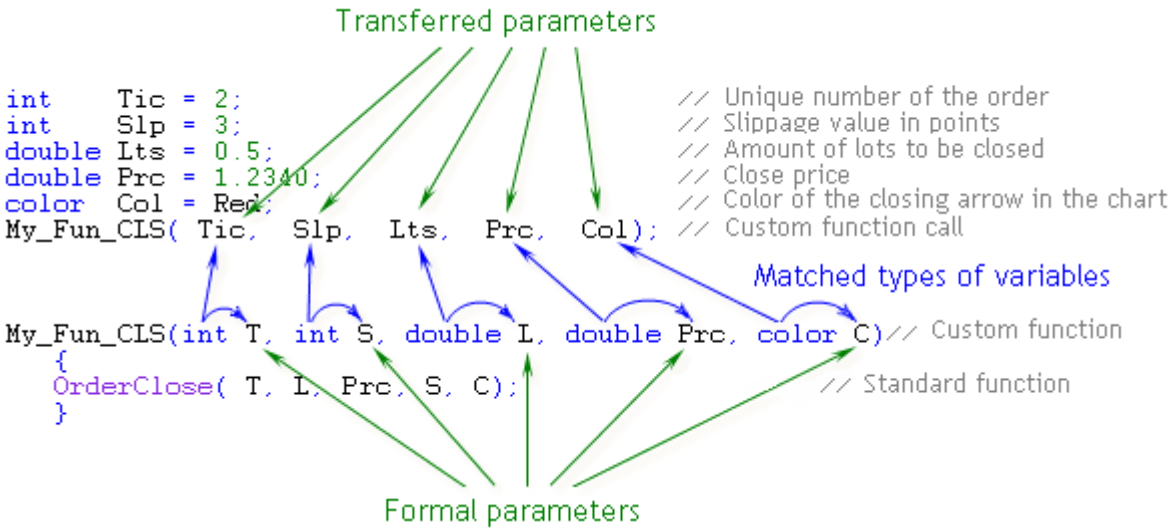


Fig. 22. Match en el número, tipo, parámetros transferidos y parámetros formales. En este caso solamente se utilizan variables como parámetros transferidos.

Como parámetros transferidos, solo se pueden utilizar variables, constantes y expresiones.

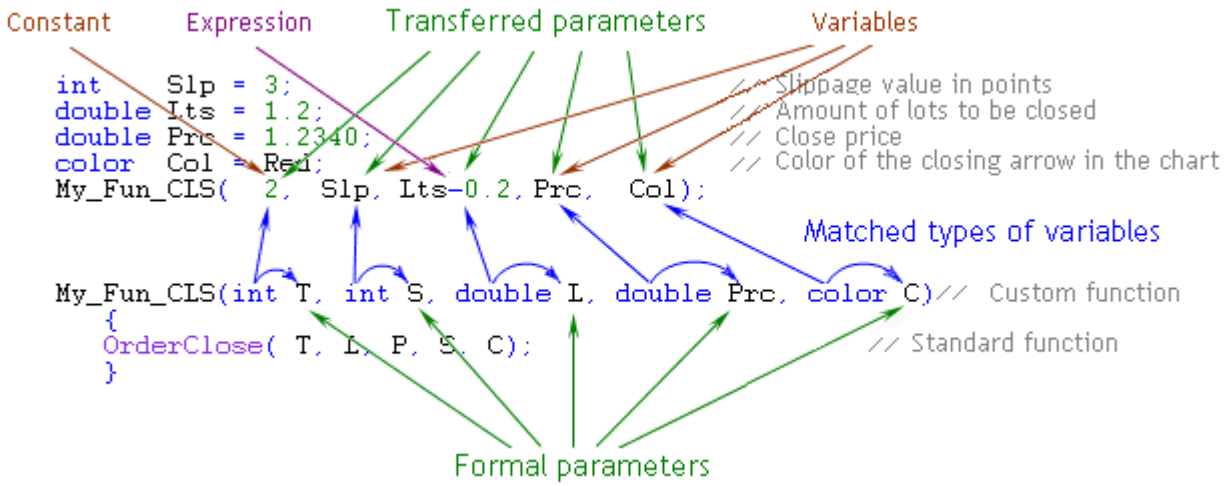


Fig. 23. Match en el número, tipo, parámetros transferidos y parámetros formales. En este caso, en los parámetros transferidos se utilizan una constante, una expresión, y las variables del tipo correspondiente.

Parámetros formales

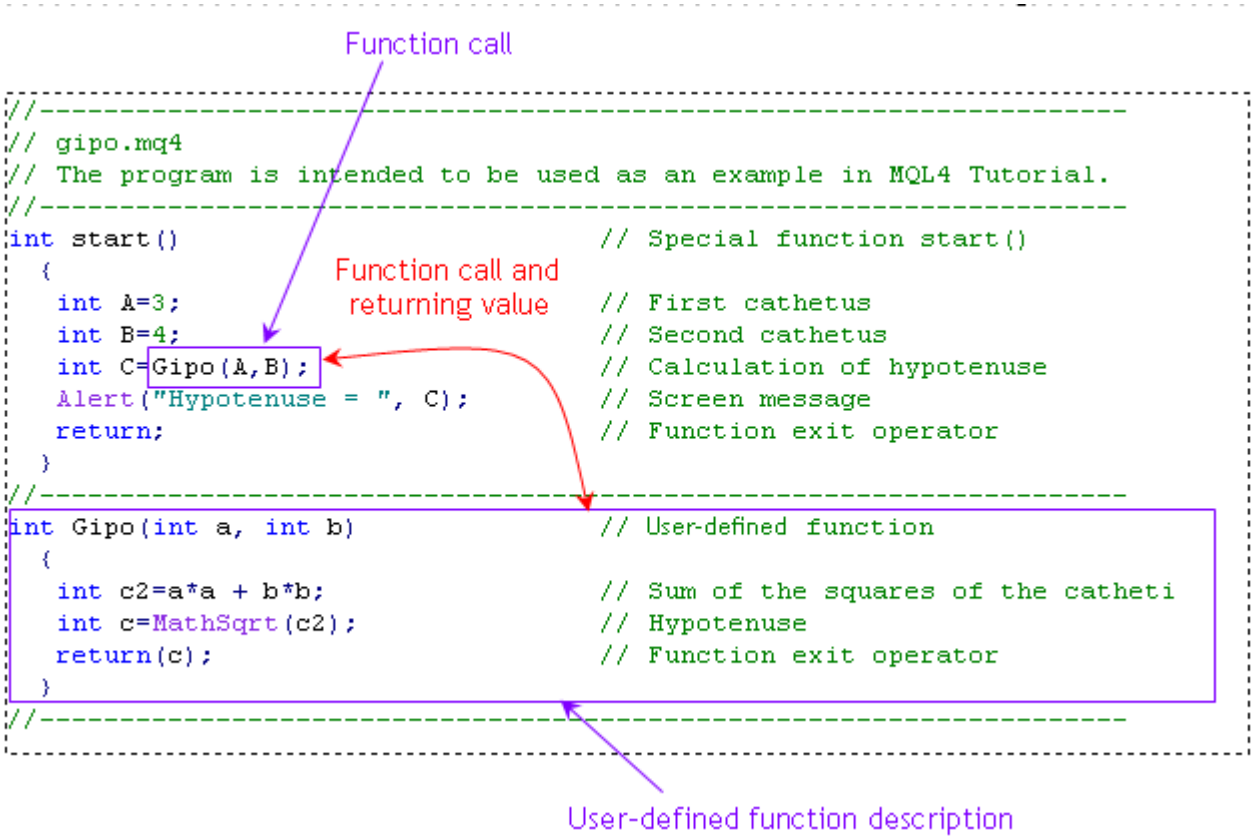
El punto culminante de las funciones es el uso de los parámetros formales.

Los **Parámetros formales** son una lista de variables especificadas en la cabecera de la descripción de la función.

Ya mencionamos antes que una misma función podría ser utilizada en varios programas. Sin embargo, los diferentes programas utilizan diferentes nombres para las variables. Si las funciones requirieran de forma estricta que se pusieran determinados nombres en las variables de los parámetros a transferir (y, correlativamente su valor), no sería conveniente para los programadores. De hecho, usted tendría que poner en cada programa recientemente desarrollado los nombres de variables que ya se han utilizado en sus funciones anteriores. Sin embargo, afortunadamente, las variables utilizadas dentro de las funciones no tienen relación con las variables utilizadas en el programa que lo llama.

Vamos a hacer referencia a la Fig. 19 una vez más. Cabe señalar que los nombres de los parámetros transferidos **(A** y **B** que se dan en el paréntesis de la llamada a la función) no coinciden con los nombres de los parámetros **(a** y **b)** que se especifica en la descripción de la función. Hemos observado en la sección Constantes y Variables que MQL4 distingue entre mayúsculas y minúsculas. De este modo, los nombres **a** y **A, B** y **b** deben considerarse aquí como diferentes nombres de variables. Sin embargo, no hay error en este código.

Las variables utilizadas en los **parámetros formales** de la descripción de la función no están relacionados con las variables utilizadas en el código básico del programa. Estas son variables diferentes. Solamente pueden especificarse variables (pero no constantes) como parámetros formales en cabecera de la función.




```
//+------------------------------------------------------------------+
// gipo.mq4
// The program is intended to be used as an example in MQL4 Tutorial.
//+------------------------------------------------------------------+
int start()                          // Special function start()
  {
   int A=3;                          // First cathetus
   int B=4;                          // Second cathetus
   int C=Gipo(A,B);                  // Calculation of hypotenuse
   Alert("Hypotenuse = ", C);        // Screen message
   return;                           // Function exit operator
  }
//+------------------------------------------------------------------+
int Gipo(int a, int b)               // User-defined function
  {
   int c2=a*a + b*b;                 // Sum of the squares of the catheti
   int c=MathSqrt(c2);               // Hypotenuse
   return(c);                        // Function exit operator
  }
//+------------------------------------------------------------------+
```

Fig. 19. El código de un programa que contiene la descripción y la llamada a función definida por el usuario gipo.mq4.

Cómo funciona

- En el programa, se produce una llamada a la función, las variables **A** y **B** están especificadas en su paréntesis.
- El programa llama a la función con ese nombre y que tiene los parámetros formales **a** y **b** que se especifican en su cabecera.
- El valor de variable **A** se le asigna a la varible **a**.
- El valor de la variable **B** se le asigna a la variable **b.**
- La función ejecutable realiza los cálculos utilizando los valores de las variables **a** y **b.**

Cualquier nombres se pueden usar en los parámetros formales (mientras no coincidan con otros nombres de variables ya utilizados en el programa). En este ejemplo, hemos utilizado los identificadores de los parámetros formales **a** y **b.** Sin embargo, podríamos utilizar cualquier otro, por ejemplo, **m** y **n,** o **Kat_1** y **Kat_2.** Por supuesto, en la redacción de un programa, usted debe especificar en el cuerpo de la función para los cálculos los nombres de variables que están en la cabecera. Puesto que hemos dado **a** y **b** en la cabecera, obviamente tenemos que utilizar **a** y **b,** dentro de la función y no **m** y **n.**

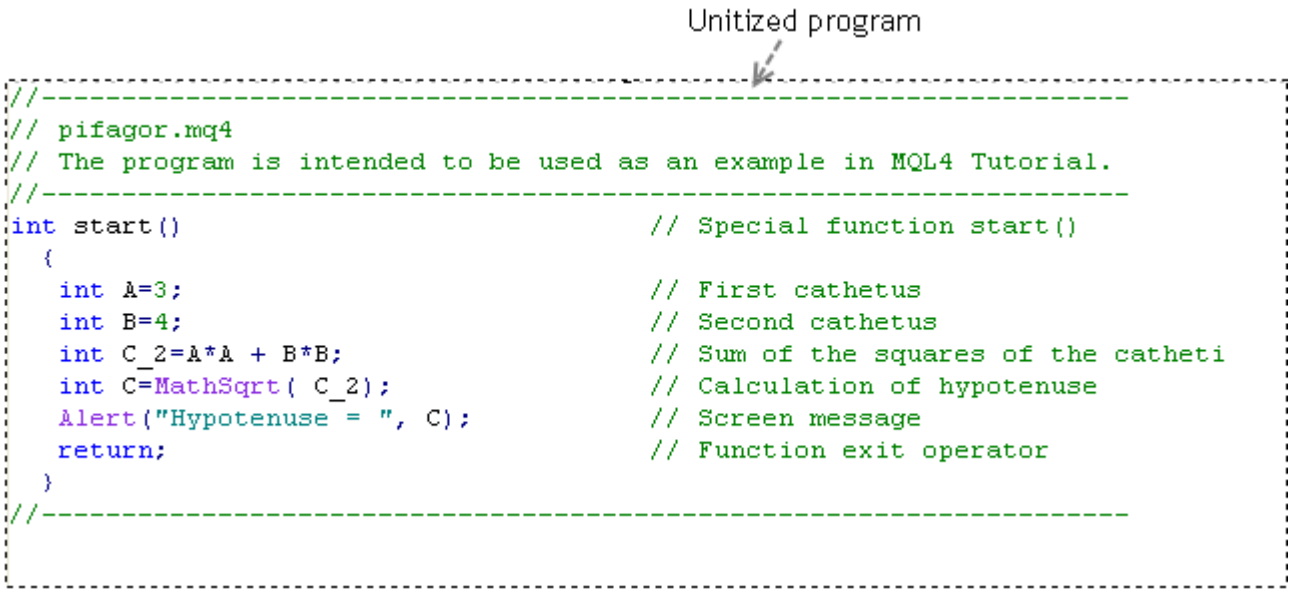
En la función, los cálculos se hacen de forma que se involucren parámetros **a** y **b y** no las variables **A** y **B** como ya hemos dicho**.** Una función se permite que cualquier acción se pueda realizar con los parámetros formales de **a** y **b** (incluidos los cambios en los valores de estas mismas variables) y esto no tendría ningún efecto sobre las variables **A** y **B** correspondientes a los parámetros de transferencia del operador de llamada del programa principal**.**

El valor de return calculado en la función se da en el paréntesis del operador return (). Como valor de return puede utilizarse el valor de una variable, el resultado de una expresión o una constante. En nuestro caso, el valor de retur es el valor de la variable local **c** (una variable local es una variable declarada dentro de una función, a la salida de la función, los valores de todas las variables locales se pierden, y por tanto su ámbito de trabajo es solo válido dentro de la función. Examinaremos con más detalles las variables locales en la sección Tipos de variables. La función devuelve al programa que lo llamó el valor de la variable local **c** (Fig. 19). Esto significa que este valor será asignado a la variable **C.**

Otros cálculos dentro del programa, si fuera el caso, se podrían realizar con las variables declaradas dentro de la función que la llama. En nuestro caso, la función que la llama es la función especial start () (que contiene la línea para llamar a la función definida por el usuario), mientras que las variables declaradas dentro de la función que la llama son **A, B** y **C,** en la función que realiza los cálculos se usan los **parámetros formales**, lo que nos permite crear funciones usando nombres de variables arbitrarios e independientes de los nombres de valores utilizados en el programa principal o función que le llama.

Ejemplo de función estándar en un programa

En primer lugar, vamos a examinar el comportamiento del programa se muestra en la Fig. 18. Debemos tener en cuenta que todo el código del programa se encuentra dentro de la función especial start (). En esta etapa de aprendizaje, no vamos a prestar especial atención a esta función (las funciones especiales y sus propiedades seran consideradas en más detalle en la sección Funciones especiales).




```
//+------------------------------------------------------------------+
// pifagor.mq4
// The program is intended to be used as an example in MQL4 Tutorial.
//+------------------------------------------------------------------+
int start()                                 // Special function start()
  {
   int A=3;                                 // First cathetus
   int B=4;                                 // Second cathetus
   int C_2=A*A + B*B;                       // Sum of the squares of the catheti
   int C=MathSqrt( C_2);                    // Calculation of hypotenuse
   Alert("Hypotenuse = ", C);               // Screen message
   return;                                  // Function exit operator
  }
//+------------------------------------------------------------------+
```

Vamos a seguir la ejecución del programa, empezando con el operador de asignación:

```
int A = 3;                    // Primer cateto
```

1. La parte derecha del operador de asignación contiene una constante, su valor es de 3.
2. El valor de 3 (el valor de la parte derecha) se le asigna a la variable **A** (que está situada a la izquierda del signo de igualdad).

Se le da el control a la línea siguiente:

```
int B = 4;                    // Segundo cateto
```

3. La parte derecha del operador de asignación contiene especificada una constante, su valor es de 4.

4. El valor de 4 se asigna a la variable **B.**

El programa se destina a la ejecución de la línea siguiente:

```
int C_2 = A*A + B*B;                    // Suma de los cuadrados de los catetos
```

5. Cálculo de la parte derecha del operador de asignación. El resultado de los cálculos es el valor de 25 (los detalles de cómo el programa se refiere a las variables para obtener sus valores son considerados en la sección Constantes y variables, por lo que no se particularizah este punto aquí y ahora).

6. Asignación del valor 25 a la variable **C_2.**

La siguiente línea representa un operador de asignación, de las cuales la parte derecha contiene una llamada a una función estándar:

```
int C = MathSqrt( C_2);                 // Cálculo de la hipotenusa
```

El programa tiene como objetivo ejecutar el operador de asignación. Con este próposito se ejecuta primero los cálculos situados a la derecha de la igualdad.

7. El programa requiere la ejecución de la función estándar MathSqrt () (que calcula raíces cuadradas). El valor de la variable C_2 (en nuestro caso igual a 25) se utilizará como parámetro el paso. Hay que tener en cuenta que no existe una descripción de esta función estándar en ninguna parte del programa. Las descripciones de funciones estándar no están situadas en los programas. En el texto de un programa, usted puede distinguir fácilmente la regla de llamada a la función por su aspecto: que se destacan en MetaEditor con color morado (programador puede elegir a voluntad los colores).

8. Los cálculos se realizan en el nivel de la función MathSqrt ().

9. La función estándar MathSqrt () completa sus cálculos y devuelve el valor obtenido. En nuestro caso, es el valor de **5** (la raíz cuadrada de 25).

El valor devuelto por la función es ahora el contenido del registro:

```
MathSqrt( C_2)
```

Este registro puede ser considerado como una especie de variable compleja especial, dentro de los cálculos que se realizan. Después de que estos cálculos se han completado, esta especie de variable toma un valor. Lo importante de esto aquí es el hecho de que el valor devuelto por la función puede ser asignado a otra variable o considerado de alguna manera en cualquier otro cálculo.

10. En este caso, nuestro valor es el valor de la parte derecha del operador de asignación. En la continuación de la ejecución, el operador de asignación el programa asigna el valor de 5 a la variable C.

11. La siguiente línea es el operador que contiene referencias a la función estándar de **Alert ()** (llamada a función).

```
Alert("Hipotenusa = ", C);              // Mensaje para la pantalla
```

La función estándar **Alert ()** abre un cuadro de diálogo en donde se muestran los valores de los parámetros transferidos. En este caso, la función ha tomado dos valores como parámetros de paso:

-- La cadena de valor constante: “Hipotenusa = ”

-- El valor integer de la variable **C:** 5

Se ha señalado anteriormente que no todas las funciones deberán devolver el valor (que es el resultado de la ejecución de la función). La función estándar **Alert ()** no devuelve ningún valor, ya que tiene otra tarea: mostrar el texto en la pantalla en una ventana especial.

Como resultado de la ejecución de la función estándar de **Alert ()** en la ventana de esta función aparecerá la línea siguiente:

```
Hipotenusa = 5
```

12. El último operador en este programa completa la labor de la función especial start ().

```
return;                          // Función salida del operador (vacia)
```

La labor del programa ha terminado en este momento.

Una pregunta puede surgir: ¿Cómo podemos saber qué función devolverá un valor, y cual no? La respuesta a esta pregunta es obvia: Con el fin de encontrar las descripciones detalladas de las funciones incorporadas, usted debe leer la documentación de referencia de MQL4.community, el sitio web lanzado por MetaQuotes Software Corp, o los archivos de ayuda de MetaEditor. Las propiedades de una **función definida por el usuario** se especifican en su descripción. Si una función definida por el usuario devuelve el valor o no, depende de su algoritmo (la decisión es adoptada por el programador en la fase de redacción del código de programa de la función).

Ejemplo de función definida por el usuario en un Programa

Vamos a considerar cómo se realizarían los mismos cálculos en un programa que contenga una función definida por el usuario (Fig. 19). Una cierta parte del código que se puso anteriormente que se dentro en la función especial start (), no estará aquí ahora. Este codigo será sustituido con la llamada a la función definida por el usuario. La descripción de esta función definida por el usuario irá después de la funcion especial start (). Las dos primeras líneas, en la que variables de tipo **A** y **B** tomar valores numéricos, siguen siendo los mismos. En consecuencia, nada cambia en su ejecución:

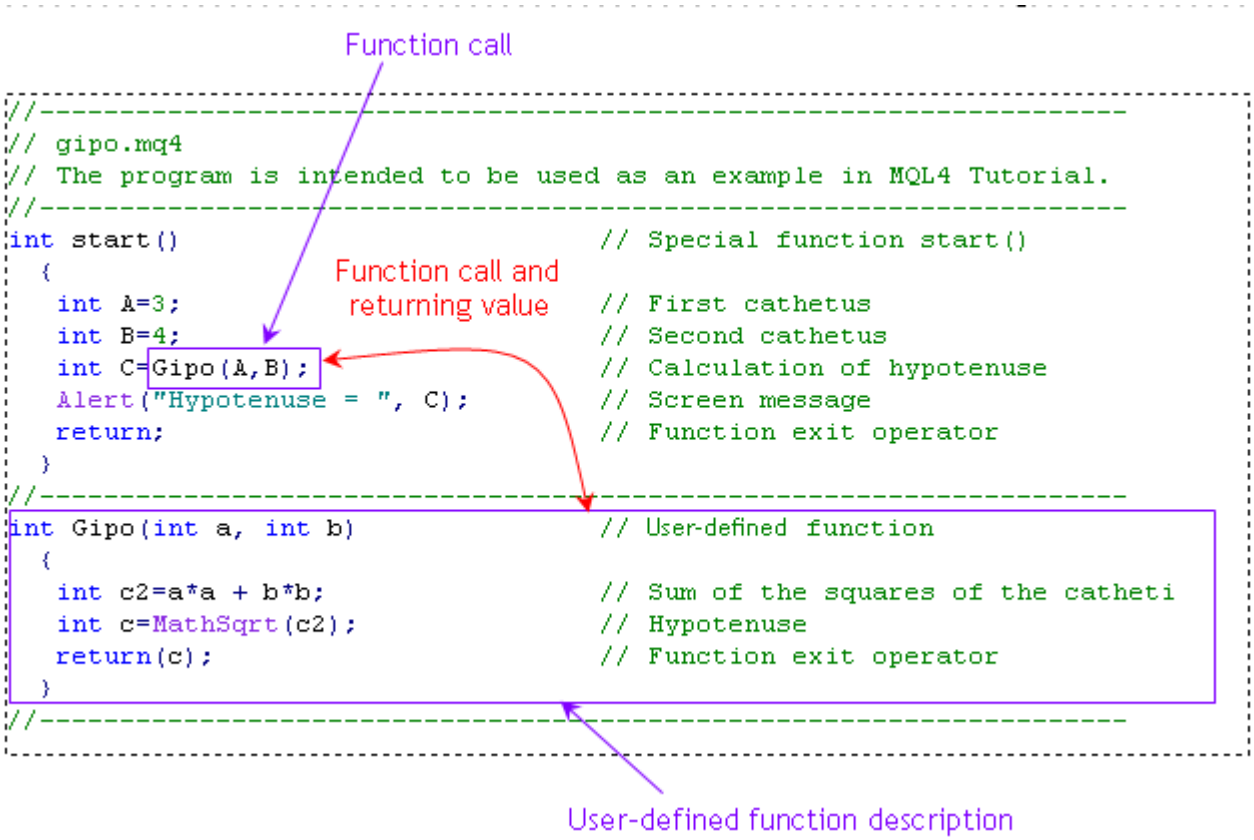


Fig. 19. El código de un programa que contiene la descripción y la llamada a función definida por el usuario gipo.mq4.

```
int A = 3;                  // Primer cateto
int B = 4;                  // Segundo cateto
```

En la tercera línea tenemos el operador de asignación. Su parte derecha contiene la convocatoria de la función definida por el usuario:

```
int C = Gipo(A,B);          // Cálculo de la hipotenusa
```

6. En la ejecución de este operador, el programa llama en primer lugar a la función definida por el usuario.

Nota: La descripción de la función definida por el usuario debe estar presente en el programa y se colocan inmediatamente después de que se cierra la llave de la función especial start () (es decir, se coloca fuera de la función especial star).

Al referirse a la función definida por el usuario, el programa lleva a cabo lo siguiente:

6,1. Copia del valor de la variable **A** con el fin de obtener su valor (en nuestro caso, **3)**

6,2. Copia del valor de la variable **B** con el fin de obtener su valor (en nuestro caso, **4)**

Nota: Tan pronto como el programa comienza a llamar a la función definida por el usuario (en este caso concreto la función es una función definida por el usuario, esta norma se aplica a todas las funciones), el programa se hace sólo una copia de los valores de las variables utilizadas, como parámetros transferidos, mientras que los valores de estas variables en si mismas (en este caso, **A** y **B)** no ha supuesto cambio debido a la aplicación de la función definida por el usuario, ni ellas realmente cambian.

7. El control se pasa a la función definida por el usuario.

Durante todo el tiempo de ejecución de la función definida por el usuario (no importa cuánto se tarde), los valores de las variables de la llamada a la función no se pierden sino que se almacenan.

La primera línea en la **descripción de la función** definida por el usuario es su cabecera:

```
int Gipo(int a, int b)              // Cabecera de la función definida por el usuario
```

En la ejecución de la función definida por el usuario, el programa va a hacer lo siguiente:

7,1. El valor de **3** (el primer valor en la lista de parámetros transferidos) se **copia** en la variable **a** (la primera variable en la lista de parámetros).

7,2. El valor de **4** (el segundo valor en la lista de parámetros transferidos) se **copia** en la variable **b** (la segunda variable en la lista de parámetros).

Entonces el control se debe pasar al cuerpo de la función para la ejecución de su algoritmo.

El primer operador en el cuerpo de la función es el siguiente:

```
int c2 = a*a + b*b;                 // Suma de los cuadrados de los catetos
```

7,3. En la ejecución de este operador, el programa calcula el valor en la parte derecha del operador de asignación, y después asigna el valor obtenido (en nuestro caso, 3*3 + 4*4 = **25)** a la variable **c2.**

El siguiente operador:

```
int c = MathSqrt(c2);               // Cálculo de la Hipotenusa
```

7,4. Aquí encontramos la raíz cuadrada del valor de la variable **c2.** El orden de operaciones es el mismo que en el ejemplo anterior. La ejecución del operador de asignación se traducirá en la asignación de valor de **5** a la variable **c.**

7,5. En la siguiente línea, tenemos el operador:

```
return(c);                          // Función salida operador
```

En la ejecución de este operador tenemos el return del programa del valor encerrado en el paréntesis de este operador. En nuestro caso, es el valor de la variable **c,** es decir, **5.** En este punto, la ejecución de la función definida por el usuario ha terminado, el control se devuelve al punto donde se hizo la llamada a la funcion.

8. Recordemos que la función definida por el usuario se llamó desde el operador

```
int C = Gipo(A,B);                  // Cálculo Hipotenusa
```

El registro de la llamada a la funcion definida por el usuario queda en:

```
Gipo(A,B)
```

en la fase de la devolución del valor, esta expresión toma el valor calculado de la función (en nuestro caso, es el valor de **5).**

Tras concluir la ejecución del operador de asignación, el programa asigna el valor de 5 a la variable **C.**

9. El siguiente operador,

```
   Alert("Hipotenusa = ", C);          // Mensaje en la pantalla,
```

se ejecutará de la misma manera que en el ejemplo anterior, a saber: en una ventana especial, aparecerá el mensaje:

```
Hipotenusa = 5
```

10. El último operador en este programa,

```
 return;                  // Salida del operador y salida de la funcion especial star (sin parámetros)
```

Queda completa la labor de la función especial start (), y a la vez queda completa la labor de todo el programa (las propiedades de las funciones especiales se consideran con más detalles en la sección Funciones especiales).

Vamos a considerar algunas versiones de la aplicación de la mencionada función definida por el usuario. Es fácil verificar que la programación con funciones definidas por el usuario tiene algunas ventajas incontestables.

Consideraciones previas de la implementacion de la función definida por el usuario Gipo ()

En esta función, los parámetros formales "se asemejan a" las variables utilizadas en el programa básico. Sin embargo, esto no es más que una similitud formal, porque **A** y **a** son diferentes nombres de variables, pero los valores, es decir, **el contenido** de las variables **A** y **B** se **copian** dentro de las variables **a** y **b.**

```
//------------------------------------------------------------------
int Gipo(int a, int b)                // Función definida por el usuario

  {
   int c2 = a*a + b*b;                // suma de los cuadrados de los catetos
   int c = MathSqrt(c2);              // Hipotenusa
   return(c);                          // Función salida operador  }
//------------------------------------------------------------------
```

Aplicación de la función definida por el usuario Gipo (): Versión 2

En este caso, los nombres de parámetros formales no "se asemejan a" los nombres de variables utilizadas en el programa básico. Sin embargo, esto no nos impide utilizar esta función en el programa.

```
//------------------------------------------------------------------
int Gipo(int alpha, int betta)       // Cabecera de la descripcion de la función definida por el usuario
  {
   int SQRT = alpha*alpha + betta*betta; // suma de los cuadrados de los catetos
   int GIP = MathSqrt(SQRT);          // Hipotenusa
   return(GIP);                       // Función salida operador
  }
//------------------------------------------------------------------
```

Aplicación de la función definida por el usuario Gipo (): Versión 3

En este ejemplo, la variable alfa se reutiliza en el programa y cambia su valor dos veces. Este hecho no tiene ningún efecto sobre las variables reales de la funcion de llamada del programa principal.

```
//------------------------------------------------------------------
int Gipo(int alpha, int betta)          // función definida por el usuario
  {
   alpha= alpha*alpha + betta*betta;     // suma de los cuadrados de los catetos
   alpha= MathSqrt(alpha);               // Hipotenusa
   return(alpha);                      // Función salida operador
  }
//------------------------------------------------------------------
```

Aplicación de la función definida por el usuario Gipo (): Versión 4

En esta funcion definida por el usuario, todos los calculos se recogen en un solo operador. El valor de return se calcula directamente en el paréntesis del operador return (). La expresión se calcula directamente en el lugar donde el parámetro transferido debe ser especificado en la función estandar MathSqrt (). Esta solución puede parecer extraña o mala al principio. Sin embargo, no hay error en esta forma de utilización de la función definida por el usuario. El número de operadores utilizados en la función es menor que en otras implementaciones, por lo que el código resulta ser más compacto.

```
//------------------------------------------------------------------
int Gipo(int a, int b)                // Función definida por el usuario
  {
   return(MathSqrt(a*a + b*b));        // Operador Función Salida
  }
//------------------------------------------------------------------


//---------------------------------------------- ----------------------
  int Gipo(int a, int b) // función definida por el usuario
  (
  return (MathSqrt (a*a + b*b)); // Función salida operador
  )
  //---------------------------------------------- ----------------------
```

De este modo, la aplicación de funciones definidas por el usuario tiene algunas ventajas indiscutibles en la programación de la práctica:

- los nombres de variables en el texto del programa principal no tienen relacion con los nombres de los parámetros formales de una función definida por el usuario;
- funciones definidas por el usuario pueden ser reutilizados en diferentes programas, no hay necesidad de cambiar el código de la función definida por el usuario;
- se pueden crear librerías, si es necesario.

Estas propiedades tan útiles de las funciones permiten crear programas realmente grandes, como por ejemplo para una corporación. Varios programadores pueden participar en este proceso de manera simultánea, y cada uno de ellos queda liberado de la necesidad de llegar a un acuerdo sobre los nombres de las variables que utilizan. Lo único que se deberá acordar es el orden de las variables en la cabecera y en la función llamada.

Tipos de programa

Al comenzar a escribir un programa en MQL4, el programador debe, en primer lugar, responder a la pregunta acerca sobre qué tipo de programas va a ser. El contenido y la funcionalidad del programa dependen plenamente de esto. En MQL4, hay 3 tipos de programas de aplicación: Asesores Expertos (**Expert Advisor)**, **scripts**, e **indicadores definidos por el usuario** (custom indicator). Cualquier programa desarrollado por un programador pertenece a uno de estos tipos. Todos ellos tienen sus propósitos y características especiales. Vamos a considerar estos tipos con más detalle.

El **Asesor Experto (AE)** es un programa codificado en MQL4 e invocado por el Terminal de Usuario para ser ejecutado en cada uno de los tick. El objetivo principal de los Asesores Expertos es programar el control sobre el comercio. Los Asesores Expertos son codificados por los usuarios. No hay una función de AEs en el Terminal de Usuario.

El **Script** es un programa codificado en MQL4 y ejecutado por el Terminal de Usuario **una sola vez**. Los scripts son destinados a realizar cualquier tipo de operaciones que permitan ser ejecutadas una sola vez. Los scripts son codificados por los usuarios. Ellos no vienen incorporados como lo está el Terminal de Usuario.

El **Custom indicator** es un programa codificado en MQL4 e invocado por el Terminal de Usuario para ser ejecutado, al igual que el AE, en todos los ticks. Esta básicamente destinado a la exhibición gráfica de funciones matemáticas calculadas preliminarmente como líneas. Hay dos tipos de indicadores: indicadores técnicos (built-in) y los custom indicator, y estos últimos son como los indicadores tecnicos pero creados por el propio usuario. Los indicadores son considerados en más detalles en las secciones del uso de los indicadores técnicos y la creación personalizada de los indicadores.

El programador elige el tipo de programa que va a ser escrito en función del propósito de ese programa específico y las propiedades y limitaciones de los diferentes tipos de programas.

Propiedades de los Programas

Creación de un Programa de Ejecución

Hay un criterio que distingue a los Asesores Expertos y los custom indicator de los scripts, y este es su tiempo de duración. En la sección algunos conceptos básicos, hemos mencionado ya que los programas se ponen en marcha durante un tiempo que es múltiplo de la cantidad de ticks. Esta afirmación es cierta para AEs y para los customs indicator, pero es falso para los scripts.

Asesor Experto y el custom indicator. Una vez que se haya vinculado un programa (EA o custom indicator) a la ventana de símbolo o instrumento, el programa hace algunos preparativos y cambia al modo de espera de ticks. Tan pronto como un nuevo tick entra, el Terminal de Usuario lo pondrá en marcha para su ejecución, entonces, el programa hace todas las operaciones descritas en su algoritmo, y, una vez que termina, pasa el control de nuevo al Terminal de Usuario y permanece en el modo de espera de tick.

Si un nuevo tick llega cuando el programa se está ejecutando, este evento no tiene ningún efecto sobre la ejecución del programa, el programa sigue siendo ejecutado de acuerdo a su algoritmo y solo pasa el control al Terminal de Usuario cuando haya terminado todas las tareas descritas en el algoritmo. Es por ello que no todos los ticks dan como resultado el lanzamiento de un AE o un indicador, sino sólo aquellos que entran cuando el control está en el Terminal de Usuario y el programa se encuentra en el modo de espera de ticks.

Un nuevo tick inicia el programa para su ejecución. De este modo, un Asesor Experto o un indicador puede operar dentro de un largo período de tiempo en la ventana de símbolo o instrumento al que se asocia ya que cada vez que termina su tarea el programa se ejecuta una y otra vez en función de cada nuevo tick.

Un Asesor Experto se diferencia de un indicador en el orden de ejecución del primer lanzamiento del programa. Esta diferencia viene determinada por las propiedades específicas de las funciones especiales de cada tipo de programa (véase Funciones especiales). Una vez que el programa se asocia a la ventana de símbolo o instrumento, el Asesor Experto hace los preparativos necesarios (función init ()) y cambia al modo de espera de tick preparado para iniciar la función start (). A diferencia de los AEs, el custom indicator ejecuta tanto la función init () como la función start () una vez a hace el primer cálculo preliminar del valor de indicador. Más tarde, con un nuevo tick, el programa se inicia llamando únicamente a la función start (), es decir, los operadores trabajan de acuerdo con el algoritmo de la función start ().

Script. A diferencia de los Asesores Expertos o los indicadores, un script se pondrá en marcha para su ejecución inmediatamente después de que haya sido asociado a una ventana de símbolo o instrumento, sin esperar a un nuevo tick. Todo el código del script se ejecutará de una vez. Después de que todas la lineas del programa se han ejecutado, el script da fin a sus operaciones y se descarga desde la ventana de símbolo o instrumento. Un script es útil si quieres hacer operaciones de una sola vez, por ejemplo, para abrir o cerrar órdenes, para mostrar textos en la pantalla, para instalar objetos gráficos, etc

Las diferencias en la ejecución de Asesores Expertos, scripts personalizados y los indicadores están determinada por las propiedades de sus funciones especiales esto será considerado con más detalles en la sección Funciones especiales.

Trading

Uno de los principales criterios que marcan los programas anteriores es la posibilidad de ejecutar instrucciones de trading. Una instrucción de trading es un órden que pasa a un programa servidor de intercambios con el fin de abrir, cerrar o modificar órdenes. Las instrucciones de Trading se forman en los programas utilizando funciones incorporadas que llamamos "funciones de trading".

Sólo los Asesores Expertos y los scripts tienen la posibilidad de utilizar las funciones de trading (sólo si la opción correspondiente está activada en la configuración del AE/script). En los customs indicator (indicadores personales) no está permitido el empleo de funciones comerciales (funciones de trading).

Uso simultáneo

Los programas también difieren entre sí por la cantidad de programas de diferentes tipos que al mismo tiempo se asocian a una ventana de símbolo o instrumento.

Asesor Experto. Solo se puede asociar un AE en una ventana de símbolo; no está permitido el uso simultáneo de varios Asesores Expertos.

Script. Solo se puede asocial un script en una ventana de símbolo; no está permitido el uso simultáneo de varios script.

Indicador personal. Se pueden asociar al mismo tiempo varios indicadores en una ventana de símbolo pero de manera que no interfieran entre sí.

Se pueden poner en marcha simultáneamente programas de los tres tipos en una ventana de símbolo en conformidad con las limitaciones de cada tipo. Por ejemplo, se puede lanzar un EA, un scrip y varios indicadores en una ventana de símbolo al mismo tiempo, o una AE y uno de los indicadores. Sin embargo, usted no puede iniciar varias AEs o scripts en una ventana de símbolo, los programas de otros tipos se pueden ponen en marcha simultáneamente.

Pueden iniciarse al mismo tiempo programas del mismo tipo en diferentes ventanas de un símbolo o instrumento. Por ejemplo, si se desea iniciar dos Asesores Expertos en un símbolo o instrumento, se puede iniciar un AE en una ventana de este símbolo y otro en otra ventana del mismo símbolo. En este caso, los Asesores Expertos trabajarán simultáneamente. Sin embargo, usted debe tener en cuenta que AEs que se inician de esta manera podrán formarse instrucciones de trading contradictorias. Por ejemplo, uno de ellos puede dar instrucciones para abrir una posicion, mientras que el otro se puede ordenar de cerrar la posicion. Esto puede provocar que se produzca una larga secuencia de órdenes inútiles que se traduce en la pérdida total.

En cualquier tipo de programa se pueden crear variables globales disponibles para todos los demás programas puestos en marcha en la teminal de usuario, incluido los que se pusieron en marcha en ventanas de símbolo diferentes. Esto permite que la máquina coordinar operaciones simultáneas de todos los programas. La orden de utilizar variables globales será especialmente considerada en la sección GlobalVariables.

Llamando a programas para su ejecución

Cualquier tipo de programa se ejecuta solo a voluntad del usuario. En MQL4, no se puede llamar a un Asesor Experto, un script, o un indicador mediante una ejecución programática, es decir, desde una llamada desde otro programa. La única excepción es la llamada incorporada en función **iCustom ()** que permite hacer referencia a un indicador personal, y la llamada a las funciones de indicadores técnicos estandar. La referencia a la función iCustom () o a las funciones de indicadores técnicos no da lugar al dibujo de las líneas de los indicadores en la ventana de símbolo (ver a simple Programas MQL4).

Cuadro 2. Principales propiedades de Asesores Expertos, scripts e indicadores personales.

Propiedad del Programa	Asesor Experto	Script	Indicador
Duración de la Ejecución	Durante un largo período	Una sola vez	Durante un largo período
Trading	Permitido	Permitido	No Permitido
Visión de líneas	No	No	Sí
Uso simultáneo de varios programas del mismo tipo	No Permitido	No Permitido	No Permitido
Llamada desde ejecución programática	No Permitida	No Permitida	No Permitida

Por lo tanto, si queremos un programa que gestione el comercio con arreglo a cierto algoritmo, deberíamos escribir un AE o un script. Sin embargo, si queremos tener una cierta función matemática representada gráficamente, deberíamos utilizar un indicador.

EL MetaEditor

En esta sección vamos a hablar en las ordenes generales de la creación de programas de aplicación utilizando MetaEditor.

El MetaEditor es un editor especializado multifunción destinados a la creación, edición y compilación de programas de aplicación escritos en MQL4. El editor tiene un interfaz de fácil uso que permite a los usuarios navegar fácilmente al escribir y revisar un programa.

- Sistema de archivos
  El MetaEditor almacena todos los códigos fuente de MQL4, programas en un catálogo estructurado propio en el disco duro. La ubicación de un programa en MQL4 está determinada por su próposito: un script, un Asesor Experto o un indicador, incluyen un archivo o una librería.

- Creación y uso de programas
  Es muy fácil crear un programa en MQL4, las herramientas que lleva incorporadas le ayudarán. Usted puede modificar las plantillas para la creación de scripts, los indicadores o Asesores Expertos. El código creado se guardará automáticamente en una carpeta del sistema de archivos MetaEditor.

Sistema de archivos

El programa Terminal de Usuario reconoce los tipos de su ubicación en los directorios subordinados.

Todos los programas de aplicación se concentran en el directorio ***ClientTerminal_folder \ expertos.*** Asesores Expertos, scripts y los indicadores personalizados de un trader que se van a utilizar en su trabajo práctico deberían estar situados en los directorios correspondientes (ver Fig. 24). Los Asesores Expertos se encuentran en el directorio ***ClientTerminal_folder \ expertos,*** scripts e indicadores en subdirectorios ***ClientTerminal_folder \ expertos \ scripts*** y ***ClientTerminal_folder \ expertos \ indicadores.***

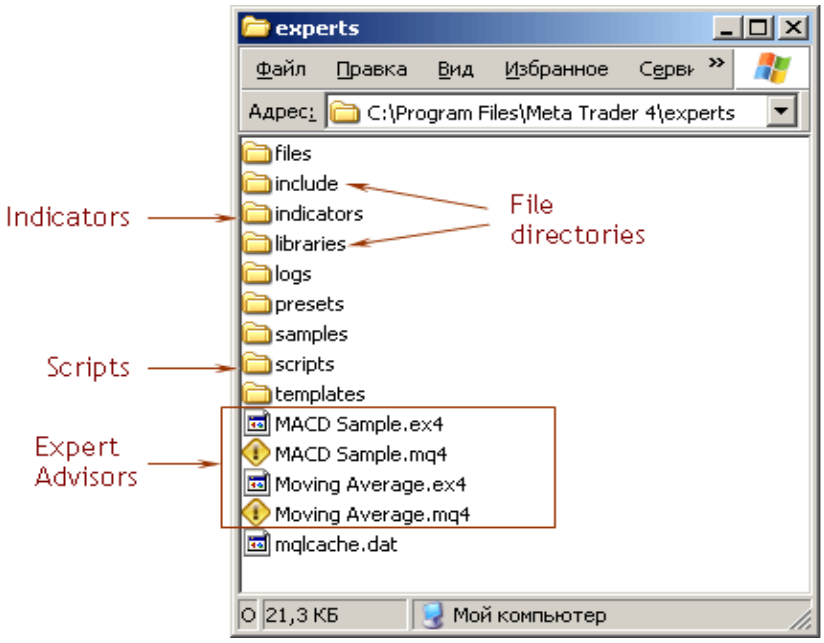


Fig. 24. Directorio para el almacenamiento de archivos, creado por un usuario.

Un usuario puede crear otros directorios para almacenar algunos archivos. Sin embargo, el uso de programas listos situado en ese directorio no está previsto en el Terminal de Usuario.

Tipos de archivo

En MQL4 hay tres tipos de archivos que llevan un código de programa: mq4, ex4 y mqh.

Los archivos de tipo **mq4** representan un programa de código fuente escrito en **mq**l**4**. Los archivos de este tipo contienen los textos de origen de todo tipo de programas (Asesores Expertos, scripts e indicadores). Se utilizan para la creación de códigos de programa MetaEditor. Cuando un código haya sido total o parcialmente creado, puede ser guardado y después abrirlo para su modificación; este archivo es de tipo mq4. Para iniciar la ejecución de un programa en el Terminal de Usuario el archivo **mq4** debe ser compilado primero. Como resultado de la compilación el código fuente, crea un archivo del mismo nombre con la extensión **ex4**, que es un archivo "**ex**ecutable de mql**4**".

Un fichero de tipo **ex4** es un programa compilado listo para su uso práctico en el Terminal de Usuario. Los archivos de este tipo no pueden ser editados. Si un programa tiene que ser modificado, esto debe hacerse en su código fuente (archivo tipo **mq4**): debe ser editado y compilado luego de nuevo. El nombre del archivo no es un indicio de que el programa se trata de un script, un asesor experto o un indicador. Los archivos con extensión ex4 se puede utilizar como archivos de la libreria.

Los archivos de tipo **mqh** se incluyen archivos. Es una fuente de texto utilizado con frecuencia como bloques en programas de usuario. Estos archivos pueden ser incluidos en los textos de origen de Asesores Expertos, los scripts y los indicadores en la fase de compilación. Por lo general, incluyen archivos que contienen la descripción de funciones importadas (como ejemplo, ver archivos stdlib.mqh o WinUser32.mqh) o la descripción de constantes y variables comunes (stderror.mqh o WinUser.mqh). Por regla general, los archivos de tipo **mqh** se almacenan en el directorio ***ClientTerminal_folder \ expertos \ incluir.***

Incluir archivos se llaman así, porque generalmente son "incluidos" en la fase de compilación para la principal fuente de archivo usando la directiva #include. A pesar de que un archivo de tipo **mqh** puede contener un programa de codigo fuente y puede ser compilado por el MetaEditor, no son independientes en si mismos, es decir, no requieren de compilación para obtener archivos ejecutables de tipo ex4. Como incluir archivos, se pueden utilizar archivos mq4 que deben guardarse en ***ClientTerminal_folder \ expertos \ incluir.***

Las secciones "Asesores Expertos ", "Indicadores personalizado" y "scripts" del navegador terminal de usuario sólo mostrarán los nombres de los archivos que tienen la extensión ex4 y se encuentran en la carpeta correspondiente. Los archivos compilados en versiones anteriores de MetaEditor no pueden ser iniciados y se muestran en color gris.

Existen otros tipos de archivos que no hacen un programa completo, pero se utilizan en la creación de programas de aplicación. Por ejemplo, un programa puede ser creado fuera de varios archivos independientes o usando una libreria creada anteriormente. Un usuario puede crear librerías de funciones personalizadas destinadas al almacenamiento para uso frecuente de bloques de programas de usuario. Se recomienda almacenar las librerías en el directorio ***ClientTerminal_folder \ expertos \ libreries.*** Los archivos de **mq4** y **ex4** se pueden utilizar como archivos de la libreria. Las librerías no pueden ejecutar por si mismas. El uso de archivos de inclusión es preferible que el uso de librerías por el consumo adicional de recursos de la computadora en la llamadas a funciones de librería.

En la primera parte del libro "Programación en MQL4" vamos a analizar archivos de textos de código fuente **mq4** y los archivos compilados **ex4**.

Creación y uso de programas

Los programas de aplicación escritos en MQL4: los Asesores Expertos, scripts e indicadores son creados utilizando el MetaEditor.

El archivo ejecutable de MetaEditor **(MetaEditor.exe),** se ofrece como parte del Terminal de Usuario y se encuentra en el directorio raíz de la terminal. El Userguide de MetaEditor se abre presionando **F1.** Contiene información de carácter general necesarias para la creación de nuevos programas. El editor se puede abrir haciendo clic sobre el nombre del archivo **MetaEditor.exe** o en un acceso directo ubicado preliminarmente en el escritorio.

Estructura del Terminal de Usuario

Para mayor comodidad de operación, MetaEditor ha incorporado las barras de herramientas: "Navigator" **(Ctrl + D)** y "Toolbox" **(Ctrl + T).**

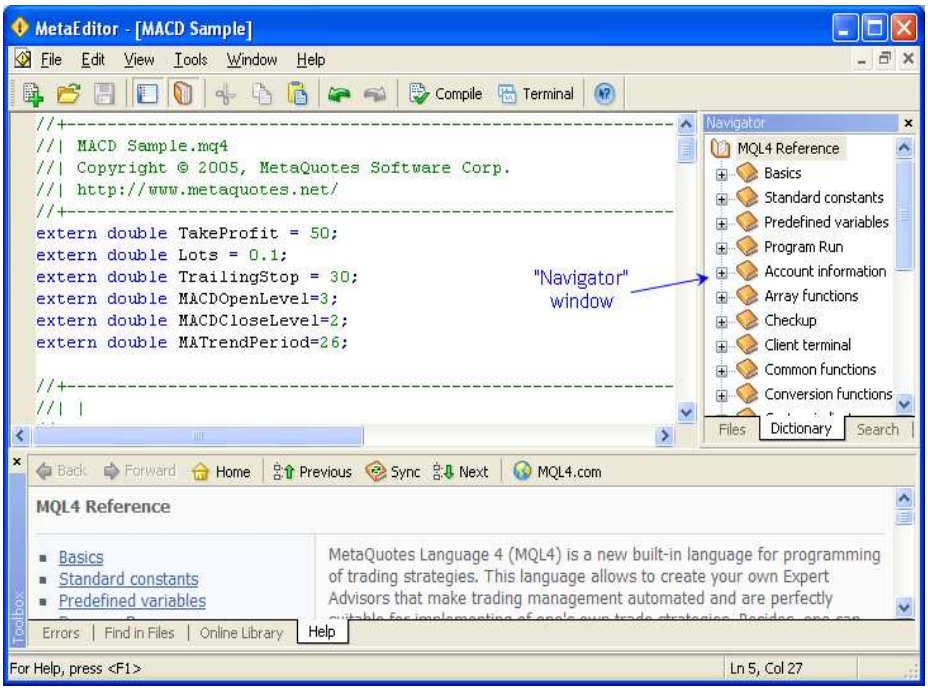


Fig. 25. Ubicación de las ventanas en MetaEditor.

El texto del programa se encuentra en la ventana del editor, las ventanas son herramientas auxiliares. Las ventanas del navegador y la caja de herramientas tienen movimiento y pueden ser mostradas/ ocultadas en el editor usando los botones  y  .

La creación de un nuevo programa

Por lo general, durante la creación de un nuevo programa, las ventanas de la caja de herramientas y del navegador y están ocultas. De este modo la atención del usuario se concentra en la creacion del programa. Para crear un nuevo programa, utilice el editor de menú **Archivo>> Crear** o el botón o para la creación de nuevos archivos  .

Después de todas estas acciones **"Expert Advisor Wizard"** le ofrecerá una lista para elegir el tipo de programa que quiere ser creado:

Fig. 26. La elección de un tipo de programa a ser creado.

Si necesitas crear un Asesor Experto, elegir **Expert Advisor** y haga clic en **Siguiente.** En la siguiente ventana es necesario escribir el nombre del Asesor Experto que quiere ser creado. Supongamos que se llama create.mq4.

El nombre de un archivo es creado es escrito sin su extensión (indicación de tipo).

El asistente del Asesor Experto mostrará una ventana con varios campos a rellenar:

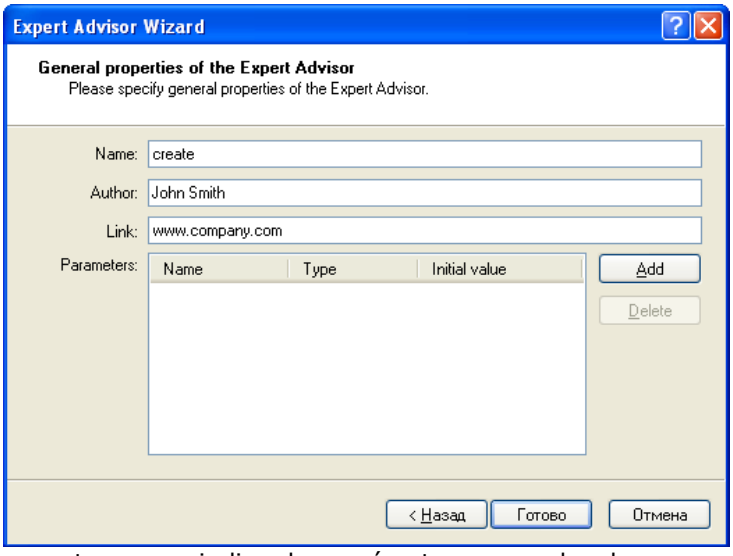


Fig. 27. Una ventana para indicar los parámetros generales de un experto asesor.

Después de hacer clic en **Aceptar** aparecerá un texto en la ventana principal y el nombre completo del Asesor Experto creado **create.mq4** se publicará en el sistema de archivos y en la ventana del navegador.

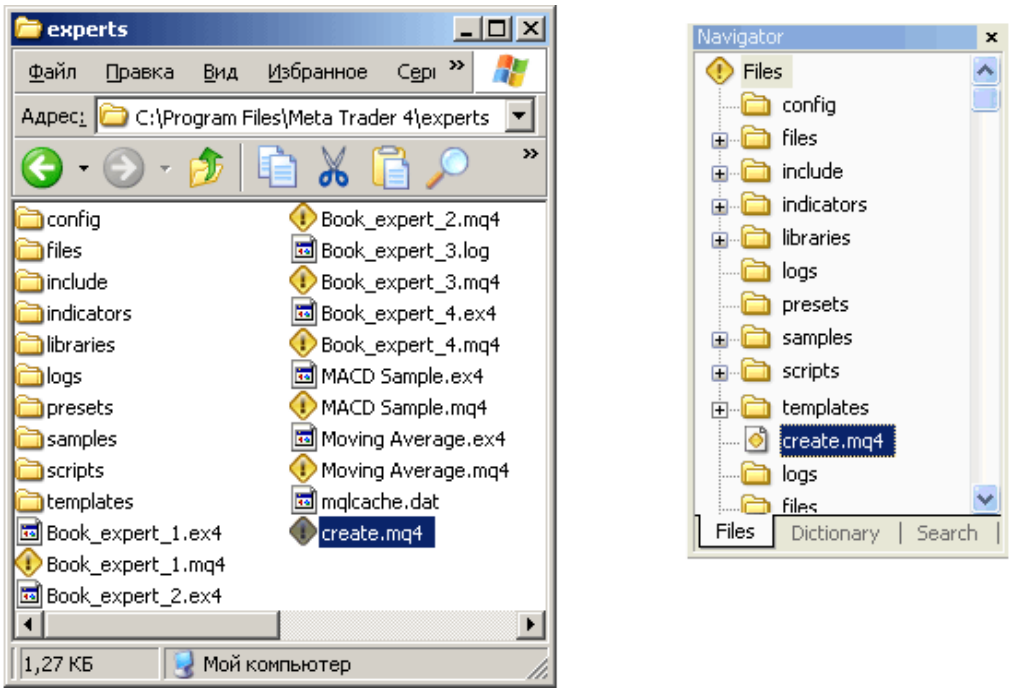


Fig. 28. Vista de un archivo creado de un Asesor Experto en las ventanas del sistema de archivos y del navegador.

Veamos el texto del programa generado por MetaEditor:

```
//+------------------------------------------------------------------+
//|                                                      create.mq4 |
//|                                                      John Smith |
//|                                                 www.company.com |
//+------------------------------------------------------------------+
#property copyright "John Smith"
#property link      "www.company.com"

//+------------------------------------------------------------------+
//| expert initialization function                                   |
//+------------------------------------------------------------------+
int init()
  {
//----

//----
   return(0);
  }
//+------------------------------------------------------------------+
//| expert deinitialization function                                 |
//+------------------------------------------------------------------+
int deinit()
  {
//----

//----
   return(0);
  }
//+------------------------------------------------------------------+
//| expert start function                                            |
//+------------------------------------------------------------------+
int start()
  {
//----

//----
   return(0);
  }
//+------------------------------------------------------------------+
```

Como se puede ver, el código contiene principalmente comentarios. Ya sabemos que las observaciones no constituyen una parte obligatoria de un programa y el texto de los comentarios no es procesado por el programa.

Hay tres funciones especiales en el programa: **init (), start () y deinit ()**. Cada función contiene un solo operador, **return (0)**, que es el operador para salir de la función. Así, un programa de código generado por Expert Advisor Wizard (Asistente del Asesor Experto) es sólo un patrón mediante el cual un programador puede crear un nuevo programa. El código final del programa no tiene que contener obligatoriamente todas las funciones especiales indicadas. Ellas están presentes en la pauta, sólo porque, como por regla general, un programa de nivel medio habitualmente contiene todas estas funciones. Si alguna de las funciones no serán utilizados, su descripción puede ser eliminada.

Las siguientes líneas de código del programa también pueden omitirse:

```
#property copyright "John Smith"
#property link      "www.company.com"
```

Aunque el programa no es de uso práctico, está escrito correctamente desde el punto de vista de la sintaxis. Y este programa puede ser compilado y ejecutado. Sería ejecutado igual que cualquier otro programa aunque no se realizaría ningun tipo de cálculos ya que no hay existe ninguno en el código fuente).

Apariencia del Programa

El uso de comentarios en los programas es altamente recomendable y en algunos casos es esencial. Y hay que destacar que un programador no sólo contribuye a crear programas, sino que también los lee y a veces puede tener considerables dificultades al leer un programa. La experiencia de muchos programadores muestra que la lógica de razonamiento, sobre la base de un programa que fue desarrollado, pueden ser olvidadas (o desconocidas en un producto de otro programador) y sin ristras de comentarios es difícil, a veces incluso imposible comprender los fragmentos de código.

Un programa codificado correctamente definitivamente contiene comentarios.

Las principales ventajas de las observaciones son las siguientes:

- En primer lugar, los comentarios permiten separar lógicamente una parte de otra del programa. Es mucho más fácil leer un texto formateado sabiamente que un texto liso (sin apartados).
- En segundo lugar, las ristras de observaciones permiten explicar en términos sencillos lo que significa cada línea de código a un programador independiente.
- En tercer lugar, en la parte superior del programa, puede ser especificada información general sobre el programa: nombre del autor y los contactos (incluido el sitio web, e-mail, etc), tarea del programa (si se trata de un programa de comercio completo o una función de librería), sus principales características y las limitaciones y otra informaciones útiles.

Cada programador puede elegir un estilo de comentarios cómodo. El estilo MQL4 ofrecidos por los desarrolladores se presenta en el Asesor Experto create.mql4. Vamos a ver las principales características de cualquier estilo de apariencia aceptable.

1. La longitud de una línea de comentario no debe exceder el tamaño de la ventana principal. Esta límitación no es un requisito formal de sintaxis, pero la lectura de un programa que contenga las líneas tan largas no es conveniente. Cualquier fila larga se puede dividir en varias líneas para que todas sean plenamente visibles en la pantalla. Para un monitor con 1024 x 768 píxeles de resolución, la máxima longitud de la línea es 118 símbolos.

2. La declaración de variables se realiza en el programa de inicio. Se recomienda escribir un comentario descriptivo para cada variable: explicar su significado brevemente y, si fuera necesario, las peculiaridades de uso.

3. Cada operador está mejor situado en una línea distinta.

4. Si hay un comentario en una línea debe iniciarse a partir de la 76 ª posición (recomendado para monitores 17 " con 1024 x 768 píxeles de resolución). Este requisito no es obligatorio. Por ejemplo, si una línea de código tiene 80 posiciones, no es necesariamente dividido en dos líneas, un comentario puede ser iniciada desde la 81ª posición. Por lo general, parte del código de programa contiene 50 símbolos de longitud de líneas y la ristra del comentario parece una columna de texto en la parte derecha de la pantalla.

5. Para dividir lógicamente fragmentos separados, se utilizan línea continua observaciones del ancho total (118 símbolos).

6. Cuando se utilizan las llaves, una tabulación tamaño sangrado debe ser utilizado (usualmente 3 símbolos).

Vamos a ver, que aspecto puede tener un Asesor Experto después de tener un programa de código escrito en ella. En este caso, no se discute la lógica del algoritmo escrito. Estamos interesados la apariencia del programa. Un programa comentado (Asesor Experto create.mq4) pueden tener la siguiente forma:

```
//+------------------------------------------------------------------+
// create.mq4
// To be used as an example in MQL4 book.
//+------------------------------------------------------------------+
int Count=0;                              // Global variable
//+------------------------------------------------------------------+
int init()                               // Spec. funct. init()
  {
  Alert ("Funct. init() triggered at start");  // Alert
  return;                                // Exit init()
  }
//+------------------------------------------------------------------+
int start()                              // Spec. funct. start()
  {
  double Price = Bid;                    // Local variable
  Count++;                               // Ticks counter
  Alert("New tick ",Count,"  Price = ",Price);// Alert
  return;                                // Exit start()
  }
//+------------------------------------------------------------------+
int deinit()                             // Spec. funct. deinit()
  {
  Alert ("Funct. deinit() triggered at exit"); // Alert
  return;                                // Exit deinit()
  }
//+------------------------------------------------------------------+
```

Es fácil ver los bloques del programa más sigficativos cuando son separados por los comentarios con las líneas discontinuas. Esta es una forma especial de separar las funciones definidas por el usuario y la cabecera de un programa:

```
//+------------------------------------------------------------------+
```

Las variables se declaran en un bloque donde es descrita cada variable. A veces los programas contienen variables y para describir las observaciones deben utilizarse varias líneas. Este es un caso raro, pero si ocurre, por ejemplo, un comentario debe ser necesariamente colocado; de otra forma, no sólo otro programador, sino que el propio autor será incapaz de armar el rompecabezas que puede suponer comprender el programa después de un cierto periodo de tiempo.

La parte derecha de cada línea de código contiene un comentario explicativo. El valor de las observaciones puede ser plenamente apreciado si el programa no contiene alguno de ellos, y algún problema con la comprensión en la lectura del algoritmo. Por ejemplo, si el mismo código, se presenta sin observaciones ni bloques de separación, será más difícil leer, aunque el programa sea muy corto y sencillo:

```
int Count=0;
int init() {
Alert (Funct. init() triggered at start");
return; }
int start() {
double Price = Bid;
Count++;
Alert("New tick ",Count,"   Price = ",Price);
return; }
int deinit(){
Alert (""Funct. deinit() triggered at exit");
return;}
```

Programa de Compilación

Para hacer un programa utilizable en la práctica, debe ser compilado. Con este fin, debe utilizarse el botón 

(F5) en MetaEditor. Si un programa no contiene ningún error, será compilado y un mensaje se producirá en la caja de herramientas:

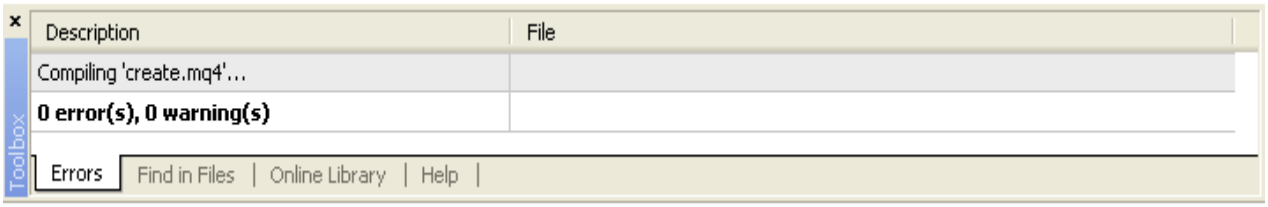


Fig. 29. Mensaje del Editor de un programa compilado con exito.

Además, un nuevo archivo **create.ex4** aparecerá en el directorio correspondiente (en este caso en ***Terminal_directory \ expertos).*** Este es ya un programa listo para su funcionamiento desde el Terminal de Usuario MetaTrader4. Durante la compilación la última versión del texto de origen del programa en relación con el mismo nombre (en nuestro caso es el archivo **create.mq4)** se guardarán en el mismo directorio.

Junto con una línea con el nombre del Asesor Experto creado aparecerá en la sección de Asesores Expertos del navegador del Terminal de Usuario la siguiente ventana:

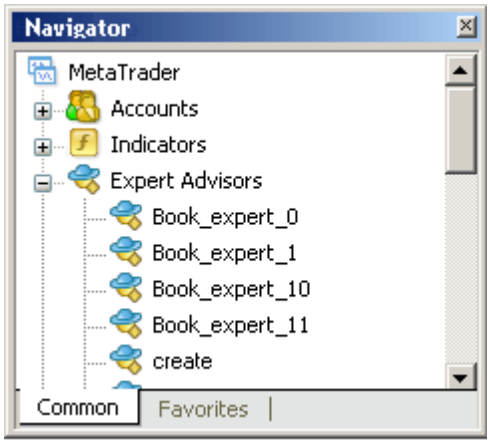


Fig. 30. Vista del nombre de un Asesor Experto en el navegador de la ventana del Terminal de Usuario.

Si durante la compilación se detectan errores en un programa, el MetaEditor mostrará el correspondiente mensaje de error. En tal caso, uno debe volver a editar el texto de origen, reparar errores y tratar de compilar el programa, una vez más. Una compilación exitosa solo es posible si no hay errores en el programa.

El uso de un Programa de Prácticas

Si un programa de aplicación (Asesor Experto, script o indicador) se ha compilado con éxito y su nombre ha aparecido en el navegador de la ventana del Terminal de Usuario, este programa puede ser utilizado en la práctica. Para utilizarlo, se arrastra el icono correspondiente a la ventana del navegador dentro de una ventana de un símbolo utilizando un ratón mediante método "drag & drop". Esto significa que el programa se vincula a un gráfico de un valor para que se inicie su ejecución.

Un Asesor Experto y un indicador funcionarán hasta que un terminal de usuario termine la ejecución del programa manualmente. Un script de usuario deja de operar por sí mismo cuando termina la ejecución de su algoritmo.

Es importante señalar aquí una vez más que:

Todos los programas de aplicación (Asesor Experto, indicador, script) pueden ser utilizados para el comercio solo como parte del Terminal de Usuario de MetaTrader 4 cuando éste está conectado al servidor (dealing center) a través de Internet. Ninguno de los programas pueden ser instalados sobre un servidor o ser usados en terminales de otros desarrolladores.

En otras palabras, si un comerciante quiere usar cualquier programa de aplicación, debe cambiar a un ordenador que tenga abierto el Terminal de Usuario de MetaTrader 4 e iniciar un archivo ejecutable *. ex4 en una ventana de un símbolo. Durante la ejecución del programa (dependiendo de su algoritmo) las órdenes de comercio pueden ser formadas y enviadas a un servidor, y por lo tanto, realizar la gestión del comercio.

Programa en MQL4

Cabe señalar desde el principio que la programación en MQL4 está disponible para una persona común, aunque requiere atención y ciertos conocimientos.

Tal vez, algunos comerciantes esperan tener grandes dificultades en el estudio de la programación lo que significa que es difícil para ellos imaginar complicados procesos que se ejecutan en el interior de sus equipos. Afortunadamente, los desarrolladores del MQL4 han tratado de hacer ampliamente disponible para los usuarios. Una agradable particularidad de la creación de programas en MQL4 es que un programador no debe tener necesariamente conocimientos especiales sobre la interacción del cliente con un terminal de sistema operativo, el protocolo de red o las características de la estructura de un compilador.

El proceso de creación de programas para ejecutarlos en MQL4 es un simple amigable trabajo. Por ejemplo, un conductor no tiene que saber la estructura de un motor para conducir un coche, sólo necesita aprender los pedales y la dirección. Sin embargo, antes de conducir un coche en las calles, cada conductor tiene que someterse a la formación. Algo así es como debe hacerse con un programador: el aprendizaje de algunos sencillos principios de la creación de programas y después lentamente comienza a "conducir".

- Estructura de Programa
  Aunque hay muchos tipos de programas en MQL4, todos elos tienen características generales. Se puede decir, que una correcta estructura es la base de un código escrito correctamente. Por eso es necesario comprender los componentes de un programa.

- Funciones especiales
  Hay un montón de funciones en el MQL4. Estas funciones son llamadas funciones estándar. Sin embargo, hay varias funciones de gran importancia, que se llaman funciones especiales. Un programa no puede ejecutarse sin ellas. Cada una de estas funciones tiene su propia tarea.

- Ejecución de Programas
  Uno debe entender correctamente cómo opera un programa MQL4. No todas las partes de código se utilizan con la misma frecuencia. ¿Qué funciones se ejecutan en primera instancia, donde debe ser colocada la parte principal de un programa, ¿qué tipo de programa, debe utilizarse con este este u otro propósito?

- Ejemplos de aplicación
  Un nuevo lenguaje es mejor aprenderlo con ejemplos. Cómo escribir correctamente un programa simple? ¿Qué errores pueden ocurrir?

Estructura de Programa

En la primera seccion hemos aprendido algunas de las nociones básicas del lenguaje de programación MQL4. Ahora vamos a estudiar la forma está organizado un programa en general. Para resolver este problema vamos a estudiar su sistema estructural.

Como ya se ha mencionado anteriormente, el código programa principal escrito por un programador se pone dentro de funciones definidas por el usuario y funciones especiales. En la sección Funciones hemos discutido el concepto y las propiedades de built-in y funciones definidas por el usuario. En pocas palabras: una función definida por el usuario tiene de una descripción y llamada a la función se utiliza para iniciar su ejecución en un programa. Cualquier built-in o cualquier función definida por el usuario son ejecutados sólo después de que se las llama, en este caso la función es llamada para la ejecución de un programa.

Propiedades de funciones especiales se describen en detalle en la sección Funciones especiales. Aquí vamos a estudiar sólo la información principal sobre ellas. La función especial es una función llamada a ser ejecutadas por el Terminal de Usuario. A diferencia de las funciones comunes, las funciones especiales sólo tienen la descripción y su llamada no se especifica en un programa. Las funciones especiales son llamadas para su ejecución desde el Terminal de Usuario (también hay una posibilidad técnica de llamar a las funciones especiales desde un programa, pero vamos a considerar este método incorrecto y no lo vamoas discutir aquí). Cuando un programa se inicia para su ejecución en una ventana de un símbolo, el Terminal de Usuario pasa el control a una de las funciones especiales. Como resultado esta función se ejecuta.

La norma de programación en MQL4 es la siguiente:

Un código de un programa debe ser escrito dentro de funciones.

Es decir, las líneas de programa (operadores y llamadas a funciones) que se encuentran fuera de una función no pueden ser ejecutadas. En el intento de compilar un programa, MetaEditor mostrará el correspondiente mensaje de error y el archivo ejecutable *.exe no aparecerá como resultado de la compilación.

Vamos a considerar el plan funcional de un programa común, Asesor Experto:

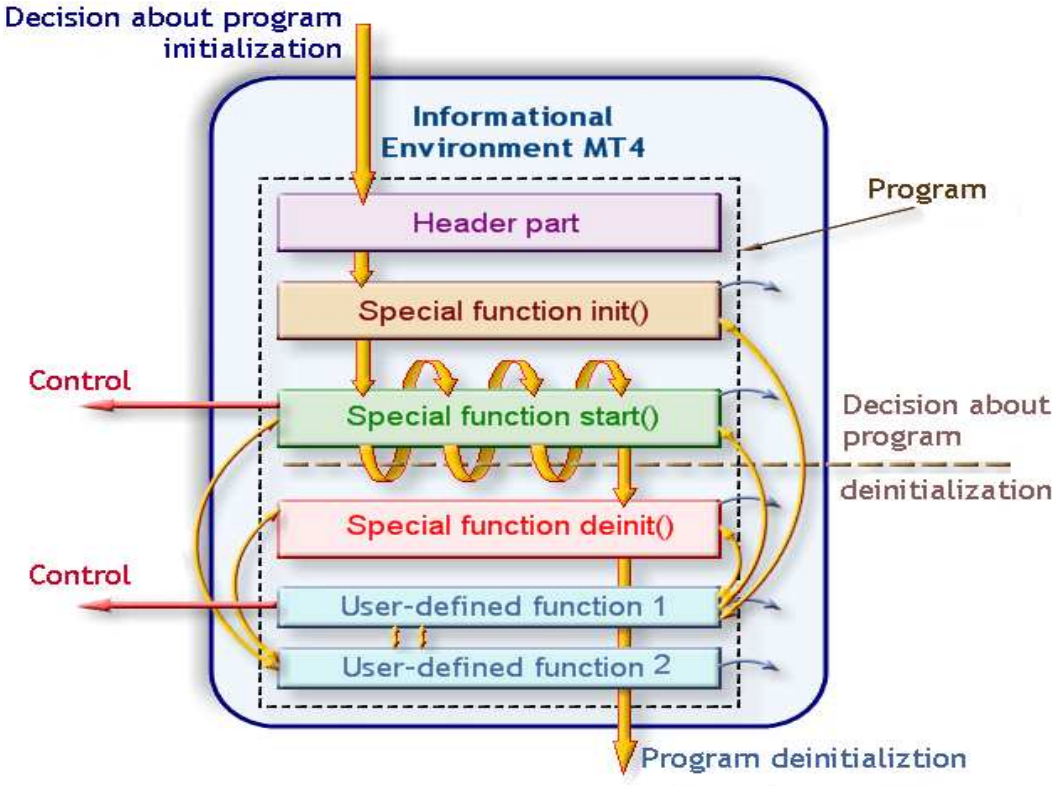


Fig. 31. Esquema funcional de un programa (Asesor Experto).

Los bloques mayores de un programa escrito en MQL4 son los siguientes:

1. Cabezera del programa.
2. Función especial init ().
3. Función especial start ().
4. Función especial Deinit ().
5. Funciones definidas por el usuario.

Además vamos a analizar sólo el contenido interior de estos bloques funcionales (partes integrales) de un programa, mientras que todos los objetos externos (por ejemplo, la información en la esfera del terminal cliente o hardware) se quedará fuera de nuestro ámbito de interés.

Información de Entorno de MetaTrader 4 Terminal de Usuario

La información de entorno del Terminal de Usuario MT4 no es un componente del programa. La información del entorno es un conjunto de parámetros disponibles para ser procesados por un programa. Por ejemplo, es una garantía de precios que ha llegado con un nuevo tick, el volumen acumulado en cada nuevo tick, la información acerca precios máximo y mínimo, de la historia de las barras, los parámetros que caracterizan a las condiciones comerciales ofrecidas por un dealing center, etc. Información de entorno es siempre guardada y en cada nuevo tick se actualiza por el Terminal de Usuario conectado con el servidor.

Estructura de Programa

Cabecera

La cabecera consta de varias líneas al comienzo de un programa (a partir de la primera línea) que contienen algunos escritos. Estas líneas contienen información general sobre el programa. Por ejemplo, esta parte incluye líneas de la declaración y la inicialización de variables globales (la necesidad de incluir tal o cual información en la cabecera se discutirá más adelante). El signo de la cabeza parte final puede ser la siguiente línea que contiene una descripción de la función (definidas por el usuario o función especial).

Funciones especiales

Por lo general, después de esta cabecera son descritas las funciones especial del programa. La descripción de la función especial se parece a la descripción habitual de una función definida por el usuario, pero las funciones especiales tienen nombres predefinidos: **init (), start () y deinit ()**. Las funciones especiales son un bloque de cálculos y estan en relación con el entorno de información del Terminal de Usuario y las funciones definidas por el usuario. Las funciones especiales se describen en detalle en la sección Funciones especiales.

Funciones definidas por el usuario

La descripción de funciones definidas por el usuario usualmente se da después de la descripción de las funciones especiales. El número de funciones definidas por el usuario en un programa no está limitado. El sistema contiene sólo dos funciones definidas por el usuario, pero un programa puede contener 10 ó 500, ó ninguna. Si no se utilizan funciones definidas el usuario en un programa, el programa será de una estructura simplificada: la cabeza y parte de la descripción de las funciones especiales.

Funciones estándar

Como se mencionó anteriormente, las funciones estándar solo pueden presentarse como una llamada a una función. Las funciones estándar, como cualquier otra función: las funciones especiales y funciones definidas por el usuario, tienen una descripción. Sin embargo, en la función estandar esta descripción no se da en el programa (es la razón por la que no está incluido en el esquema). La descripción de una función estandar está oculta, no esta visible para el programador y, por tanto, no se puede cambiar; a pesar de que está disponible para MetaEditor. Durante la compilación del programa, el MetaEditor creará un archivo ejecutable en el que todas las llamadas a funciones estándar se ejecutarán correctamente y con todo el rigor.

Acuerdo de las partes en un Programa

La cabecera debe estar ubicada en las primeras líneas. Las descripciones de funciones especiales y funciones definidas por el usuario no importan. La Fig.32 muestra una disposicón (habitual) de bloques funcionales, es decir; cabecera, funciones especiales y funciones definidas por el usuario. La Fig. 33 muestra otras variantes de estructura de programa. En todos los ejemplos la parte de la cabeza es lo primero, mientras que las funciones pueden ser descritas en un orden aleatorio.

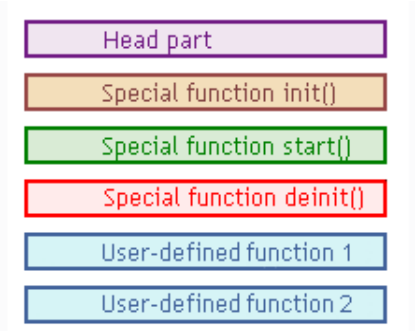


Fig. 32. Disposición habitual de bloques funcionales en un programa (recomendado).

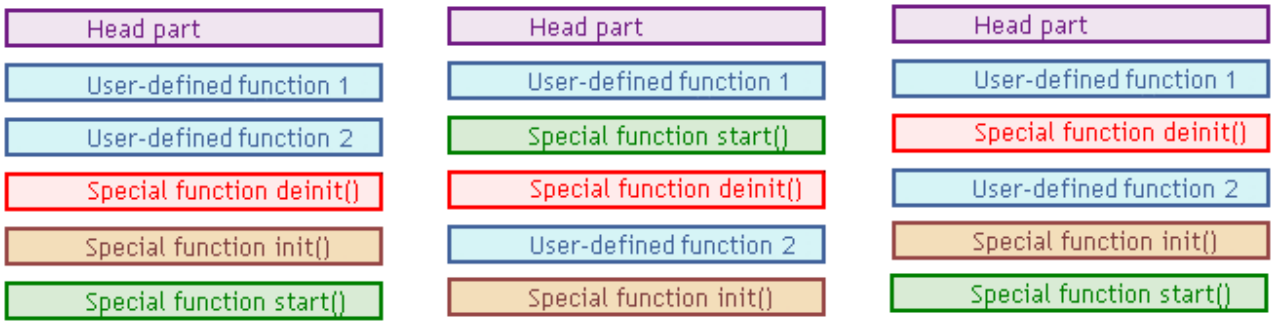


Fig. 33. Posibles formas de organización de bloques funcionales en un programa (orden aleatorio).

Por favor, tenga en cuenta:

Ninguna de las funciones puede ser descrita dentro de otra función. No está permitido el empleo en un programa descripciones de funciones situadas dentro de otra función.

A continuación se presentan ejemplos de incorrecta disposición de las descripciones de la función.

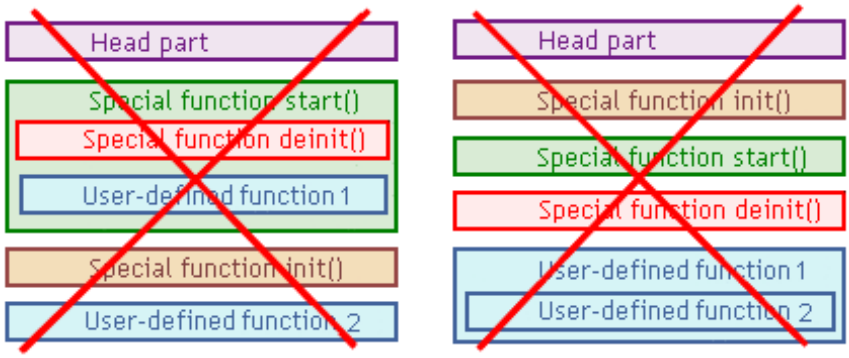


Fig. 34. Ejemplos de disposición incorrecta de las funciones en un programa.

Si un programador por error crea un programa donde la descripción de cualquiera de sus funciones se encuentra dentro de la descripción de otra función, en la etapa de compilación el MetaEditor mostrará un mensaje de error y no se creará archivo ejecutable para tal programa.

Secuencia de ejecución de código

Cabecera y funciones especiales

Desde el momento de iniciar la ejecución del programa en una ventana de un símbolo, parte de las líneas de programa de la cabeza se ejecutan.

Después, se realizan de los preparativos descritos en la cabecera y el Terminal de Usuario pasa el control a la función especial star () y se ejecuta la función (el control pasado se muestra en el esquema estructural en las grandes flechas amarillas). La función especial init () es llamada para la ejecución una sola vez al comienzo de la operación del programa. Por lo general, esta función contiene un código que debe ejecutarse sólo una vez antes de la operación principal del programa de inicio star (). Por ejemplo, cuando el init () es ejecutado, se inicializan algunas variables globales, objetos gráficos se muestran en una ventana gráfica, se muestran mensajes etc. Después de que todas en las líneas del programa start () se ejecutan, la función termina su ejecución y el control se devuelve al Terminal de Usuario.

El tiempo de operación del programa principal es el período de funcionamiento de la función start (). En determinadas condiciones (véase características de las funciones especiales en la sección Funciones especiales), incluyendo la recepción de nuevos tikcs por el Terminal de Usuario desde el servidor, el terminal de usuario pide la ejecución función especial start (). Esta función (al igual que otras funciones) se puede referir a la información de entorno del Terminal de Usuario, realizar los cálculos necesarios, abrir y cerrar posiciones, es decir, realizar cualquier acción permitida por MQL4. Por lo general, cuando la función especial start () es ejecutada, una solución producida se implementa como una medida de control (flecha roja). Este control puede ser implementado como una solicitud de comercio para abrir, cerrar o modificar una orden creada por el programa.

Después de que todo el código de la AE de la función especial start () es ejecutado, la función start () termina su operación y devuelve el control al terminal de usuario. El terminal tendrá el control durante algún tiempo no iniciando ninguna funcion especial. Una pausa aparece, durante la cual el programa no funcionará. Más tarde, cuando llegue un nuevo tick, el terminal de usuario pasará el control a la función especial start () una vez más, como resultado, la función será ejecutada y después cuando su ejecución termina, el control se devuelve al Terminal de Usuario. En siguiente tick la función start () será iniciada por el Terminal de Usuario una vez más.

El proceso de múltiples llamadas de la función especial start () por el Terminal de Usuario se repetirá mientras que el programa esté asociado a un gráfico y puede continuar durante semanas y meses. Durante todo este período un Asesor Experto puede llevar a cabo comercio automatizado, es decir, cumplir su principal misión. En el esquema el proceso de ejecución múltiple de la función start () se acredita por diversos flecha amarilla envolviendo la función especial start ().

Cuando un comerciante elimina un Asesor Experto de un gráfico, el Terminal de Usuario inicia una vez la función especial deinit (). La ejecución de esta función es necesaria para la correcta terminación de una operación de EA. Durante la operación un programa puede, por ejemplo, crear objetos gráficos y variables globales del Terminal de Usuario. Es por ello que el código de la función deinit () contiene líneas de programa, la ejecución de los cuales se traduce en la supresión de objetos innecesarios y de variables. Tras la ejecución de la función especial deinit () una vez más, se devuelve el control al Terminal de Usuario.

La ejecución de funciones especiales puede hacer referencia a la información del entorno (flechas delgadas de color azul en el esquema) y la llamada para la ejecución de funciones definidas por el usuario (flechas delgadas de color amarillo). Tenga en cuenta que las funciones especiales se ejecutan después de que ellas son llamadas por el Terminal de Usuario en el orden predefinido en función de las propiedades: en primer lugar init (), después llamada múltiples a start () y finalmente la función deinit(). Condiciones, en la que el terminal de usuario llama funciones especiales, se describen en la sección Funciones especiales.

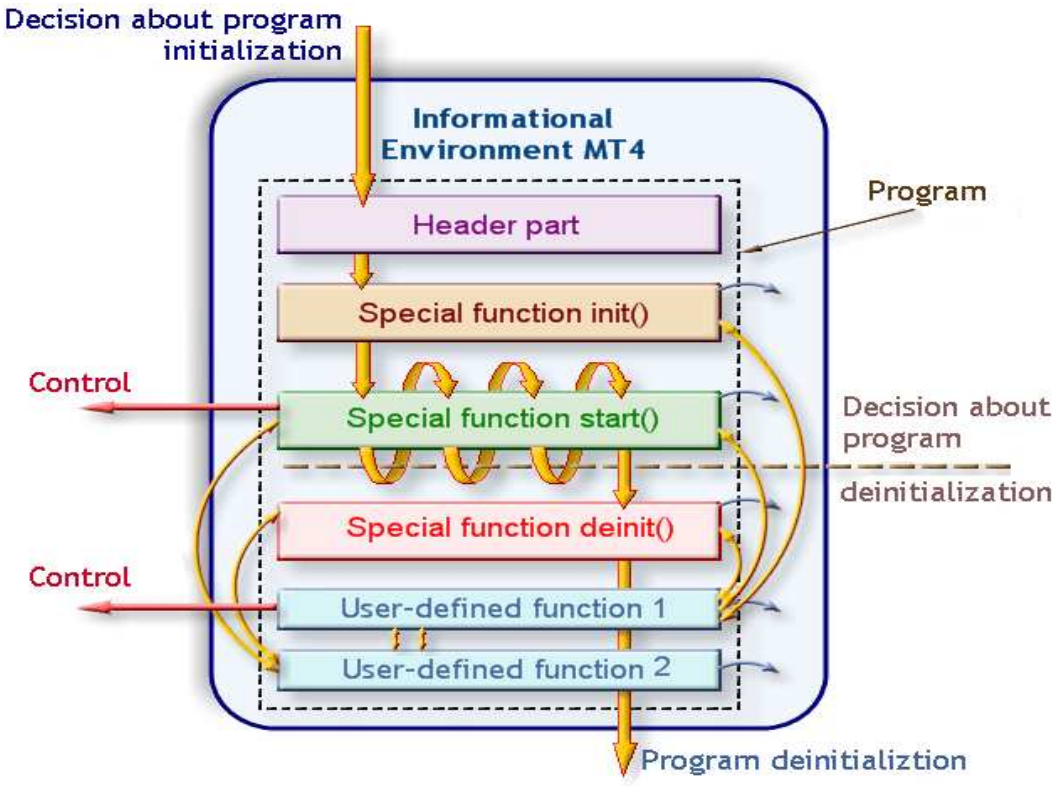


Fig. 31. Esquema funcional de un programa (Asesor Experto).

Funciones definidas por el usuario

Las funciones definidas por el usuario se ejecutan cuando la llamada está contenida en alguna función. En este caso, el control pasa a la función definida por el usuario y después de la ejecución de la función el control es devuelto al lugar de la llamada (flechas delgadas de color naranja en el esquema). La llamada a las funciones definidas por el usuario se pueden hacer no sólo dentro de la descripción de una función especial, sino también en la descripción de otras funciones definidas por el usuario. Una funcion definida por el usuario puede llamar a otras funciones definidas por el usuario. Esta forma de llamar a las funciones definidas por el usuario no solo esta permitido, sino que es un uso ampliamente utilizado en la programación.

La Funciones definidas por el usuario no son llamadas para ser ejecutadas por el Terminal de Usuario. Cualquier funciones definidas por el usuario se ejecuta dentro de la ejecución de una función especial que devuelve el control al Terminal de Usuario. Las funciones definidas por el usuario también pueden pedir (el uso) para la transformación de los valores de las variables de información de entorno del Terminal de Usuario (flechas delgadas de color azul en el esquema).

Si un programa contiene la descripción de una función definida por el usuario, pero no hay una invocación a esta función, esta función será excluida del programa en la etapa de compilación y no será utilizada en la operación del programa.

Nota: Las funciones especiales son llamadas para ser ejecutadas por el Terminal de Usuario. Las Funciones definidas por el usuario se ejecutan si se las llama desde funciones especiales o por funciones definidas por el usuario, pero nunca son llamadas por el Terminal de Usuario. El control de la acción, es decir, las órdenes de trading pueden formarse tanto en funciones especiales como en funciones definidas por el usuario.

Funciones especiales

Un rasgo distintivo de los programas destinados a la operación en el Terminal de Usuario MetaTrader 4 es su trabajo con una información constantemente actualizada en tiempo real. En este lenguaje MQL4, esta particularidad se refleja en la forma de tres funciones especiales: **init (), start () y deinit ().**

Las funciones especiales son funciones predefinidas con los nombres de **init (), start ()** y **deinit ()** que poseen propiedades especiales propias.

Propiedades de las funciones especiales

Propiedad común de funciones especiales

La principal propiedad de todas las funciones especiales es su ejecución en un programa bajo ciertas condiciones y que su funcionamiento no se inicia desde el programa. Las funciones especiales son llamadas para ser ejecutadas por el Terminal de Usuario. Si un programa contiene la descripción de una función especial, será llamada (y ejecutada), de conformidad con sus condiciones de llamada.

Las funciones especiales son llamados para ser ejecutadas por el Terminal de Usuario.

Propiedades de las funciones especiales

Función Especial init ().

La propiedad particular de la función especial de **init ()** es su ejecución en el programa de inicialización. Si un programa contiene la descripción de la función especial **init ()**, será llamada (y ejecutada) en el momento de iniciar el programa. Si no hay una función especial **init ()** en un programa, las acciones no se llevarán a cabo en el programa de inicio.

En los **Asesores Expertos** la función especial de **init ()** se llama (y ejecuta) después de que el Terminal de Usuario ha iniciado y cargado los datos históricos, después de cambiar el marco temporal del símbolo y/o gráfico, después de re-compilar el programa en el MetaEditor, después de cambiar parámetros de entrada de AE y la ventana de configuración, después de los cambios de la cuenta.

En los **scripts** de inicio de la función especial **init ()** también se llama (y ejecuta) inmediatamente después de que se llama y se ejecuta un gráfico.

En los **indicadores personalizados de usuario** función especial init () se llama (y ejecuta) inmediatamente después de empezar el Terminal de Usuario, después de cambiar el marco temporal del símbolo y/o período gráfico, después de re-compilar el programa en MetaEditor y después de cambiar parámetros de entrada en la ventana de configuración del indicador personal.

Función Especial start ().

Las propiedades especiales de la función start () difieren en función del tipo de programa que se ejecute.

En los **Asesores Expertos** la función especial start () se llama (y ejecuta) inmediatamente después de marcar un nuevo tick. Si un nuevo tick ha llegado durante la ejecución de la función especial start (), este tick no se tendrá en cuenta, es decir, la función especial start () no será llamada para su ejecución cuando este tick llega. Todas las cotizaciones recibidas durante la ejecución de la función especial start () se ignoran. El Inicio de la función especial start () para su ejecución se realiza por medio del Terminal de Usuario sólo a condición de que la operación del anterior período de sesión se haya completado, el control haya sido devuelto al Terminal de Usuario y la función especial start () este a la espera de un nuevo tick.

La posibilidad de llamar y ejecutar la función especial start () se ve influida por el estado del botón "Activar / desactivar del Asesor Experto. Si este botón se encuentra en el estado de desactivación AEs, del Terminal de Usuario no invocará la ejecución de la función especial start () con independencia de si las nuevas cotizaciones de llegan o no. No obstante, los cambios en el botón de estado desde Activado a Desactivado no finaliza la operación de la sesión actual de la función especial start ().

La función especial start () no es llamada por el Terminal de Usuario si la ventana de propiedades del AE está abierta. La ventana de propiedades de AE se puede abrir sólo cuando la función especial start () está a la espera de un nuevo tick. Esta ventana no puede abrirse durante la ejecución de la sesión del AEs la función especial start ().

En los **script** la función especiale start () se llama (y ejecuta) una vez, inmediatamente después de la inicialización del programa especial en la función init ().

En los **indicadores personales** función especial start () se llama (y ejecuta) inmediatamente después de marcar un nuevo tick, inmediatamente después de que se vincula a un gráfico, cuando se cambia el una tamaño de una garantía de la ventana, cuando se cambia de uno a otro instrumento, cuando se inicia el Terminal de Usuario (si durante el anterior período de sesiones, un indicador se asoció a una gráfica), después de cambiar un símbolo y el marco temporal actual de un gráfico con independencia del hecho de que si las nuevas cotizaciones llegan o no.

La **terminación** de la ejecución de la actual sesión star () se puede realizar en todos los tipos de programa cuando un programa se elimina de un gráfico, cuando el periodo del simbolo y/o gráfico se cambian, cuando se cambia una cuenta / gráfico es cerrado y como resultado el Terminal de Usuario la termina la operación. Si la función especial start () fue ejecutada durante un comando de apagado, el tiempo disponible del terminal para completar la ejecución de la función es de 2,5 segundos. Si después de apagado el comando, la función especial start () continúa sus operaciones durante más tiempo del plazo indicado, será forzado por el Terminal de Usuario a terminar.

Función Especial deinit ().

La función particular de la función especial deinit () es la ejecución de un programa de terminación (deinicialización). Si un programa contiene la descripción de la función especial deinit (), será llamada (y ejecutada) en un programa de apagado. Si un programa no contiene la función especial deinit (), no se llevarán a cabo acciones en programa de cierre.

La función especial deinit () también se le llama para ser ejecutadas por el Terminal de Usuario para terminal de cierre, cuando una ventana de un símbolo está cerrada, antes de cambiar una garantía y / o el periodo de un gráfico, en la re-compilación exitosa de un programa MetaEditor, al cambiar parámetros de entrada , Así como cuando una cuenta se ha cambiado.

En **Asesores Expertos y scripts** el programa se cierra con la necesaria llamada de la función especial deinit () puede ocurrir cuando vinculando un grafico a un nuevo programa del mismo tipo que sustituye al anterior.

En los **indicadores personales** función especial deinit () no se ejecuta cuando un nuevo indicador se asocia a un gráfico. Varios indicadores pueden operar en una ventana de un símbolo y esta la razón por que la vinculación de un nuevo indicador a un gráfico no se traduce en el cierre de otros indicadores con la función deinit ().

El tiempo de ejecución de deinit () está limitado a 2,5 segundos. Si el código de la función especial deinit () se ejecuta en mas tiempo el Terminal de Usuario fuerza la terminacion de la ejecución de la función especial deinit () y el funcionamiento del programa.

Requerimientos de las funciones especiales

La presencia de las funciones especiales init () y deinit () no son imprescindibles dentro programa, es decir, pueden estar ausentes. No importa el orden en el que estén descritas las funciones especiales en el programa. Las funciones especiales se pueden llamar desde cualquier parte del programa de conformidad con las reglas generales de llamadas a funciones.

Las funciones especiales pueden tener parámetros. Sin embargo, cuando estas funciones son llamadas por el Terminal de Usuario, estos parámetros no puden ser enviados desde el exterior y en este caso se utilizarán solo los valores por defecto.

Las funciones especiales init () y deinit () deben terminar su funcionamiento con la maxima rapidez y en ningún caso ejecutarse dentro de un recorrido ciclico pretendiendo hacer todas las operaciones antes de llamar a la función start ().

Orden de uso de las funciones especiales

Los desarrolladores han presentado a los programadores una herramienta muy práctica: cuando empieza un programa, init () se ejecuta en primer lugar. Después, una vez que se realiza el trabajo principal con la ayuda de la función start (), y cuando un usuario termina ha terminado su trabajo, la función deinit () se pondrá en marcha antes de que se cierre el programa.

El código principal del programa debe estar contenido en la función start (). Todos los operadores, built-in, las llamadas a las funciones personalizadas y todos los cálculos necesarios se debe realizar dentro de esta función. Al mismo tiempo, hay que entender correctamente el papel de las funciones personalizadas. La descripción de funciones personalizadas se encuentran en el código de un programa fuera de la descripción de las funciones especiales, pero si una función definida por el usuario es llamada para su ejecución, la función especial no finaliza su funcionamiento. Esto significa que el control pasa durante algún tiempo a la función del usuario, pero la propia función de usuario actúa en el marco de la función especial que la ha llamado. Así que, en el proceso de ejecución de un programa especial se opera siempre de conformidad con sus propiedades particulares, y la función de usuario se ejecuta cuando es llamada desde la funcion especial.

Si hay alguna función especial que un programador no vaya a utilizar, puede eludir su uso en el programa. En tal caso, el Terminal de Usuario no la llamará. Es absolutamente normal que un programa contenga las tres funciones especiales. Un programa que no tiene init () o deinit () o ambas funciones también se considera normal.

Si un programa no contiene ninguna de las tres funciones especiales, este programa no se ejecutará. El Terminal de Usuario necesita para su ejecución al menos una función especial de conformidad con sus propiedades. Las Funciones definidas por el usuario no son llamadas por el Terminal de Usuario. Es por este motivo que si un programa no contiene funciones especiales (y sólo contiene funciones definidas por el usuario), nunca serán llamadas para su ejecución.

No se recomienda llamar a la función start () desde la función especial de inicio () o realizar operaciones de comercio desde init (), porque durante la inicialización valores de los parámetros de la información del entorno puede que no esten listas (información sobre gráficas, precios de mercado, etc.)

Secciones de Ejecución de Programas y Ejemplos de aplicación contiene varios ejemplos prácticos que ayudan a ver algunas propiedades de funciones especiales.

Ejecución de Programas

Las habilidades de programación se desarrollan mejor si un programador tiene un pequeño programa operativo a su disposición. Para entender todo el programa, es necesario examinar a fondo todos sus componentes y localizar su funcionamiento paso a paso. Tenga en cuenta, las propiedades de las función especial de los distintos programas de aplicación (Asesores Expertos, scripts e indicadores) son diferentes. Ahora vamos a analizar cómo opera un Asesor Experto.

Example of a simple Expert Advisor (simple.mq4)

```
//+------------------------------------------------------------------+
// simple.mq4
// To be used as an example in MQL4 book.
//+------------------------------------------------------------------+
int  Contador=0;                          // Variable Global
//+------------------------------------------------------------------+
int init()                              // Función Especial init()
  {
  Alert ("La funcion init () ha comenzado ");// Alert
  return;                               // Exit init()
  }
//+------------------------------------------------------------------+
int start()                             // Función Especial start()
  {
  double  Precio = Bid;                     // Variable Local
   Contador++;                              // Contador de ticks
  Alert ("Nuevo tick ",  Contador,"   Precio = ",  Precio);// Alert
  return;                               // Exit start()
  }
//+------------------------------------------------------------------+
int deinit()                            // Función Especial deinit()
  {
  Alert ("La función deinit() ha comenzado la salida");   // Alert
  return;                               // Exit deinit()
  }
//+------------------------------------------------------------------+
```

De conformidad con las reglas de ejecución de los programas (véase el Programa de Estructura y Funciones especiales) este Asesor Experto trabajará la siguiente manera:

1. En el momento en que un programa se vincula a un gráfico, el Terminal de Usuario pasa el control al programa y, como resultado, el programa empezará su ejecución. La ejecución del programa comienza a partir de la cabecera. La cabecera solo contiene una línea:

```
int  Contador=0;                    // Variable Global
```

En esta línea la variable global Contador se inicializa a cero. (Las variables Locales y Globales se analizan en detalle en la sección Tipos de variables. Cabe señalar aquí, que el algoritmo utilizado en este programa requiere la declaración de una variable global como el Count, por eso no puede ser declarada dentro de una función y es preciso declararla fuera de la descripción de las funciones, es decir, en la cabecera. Como resultado de esto, el valor de la variable global Cont estará disponible a partir de cualquier programa.

Nota del traductor: El llamar a la varible Contador así, es puramente arbitrario (del traductor) y no venian en el texto original. Las letras VG, pueden ayudar a recordar que la variables es una Variable Global, especialmente si estamos trabajando con un programa grande con muchas variables.

3. Después de la ejecución de la parte de la cabeza del programa, la función especial init () se pondrá en marcha para su ejecución. Observe que esta llamada a la función no está contenida en un código del programa. Cuando se ejecuta un AE por que se ha vinculado a un gráfico, es una propiedad el comienzo de la ejecución de la función init (). El Terminal de Usuario llama init () para la ejecución sólo porque el código de programa contiene una descripción del mismo. El modo en que el programa analiza la descripción de la función especial init () es el siguiente:

```
int init()                          // Función Especial init()
  {
  Alert ("La funcción init() ha comenzado"); // Alerta
  return;                           // Salir de init()
  }
```

El cuerpo de la función contiene sólo dos operadores.

2.1 Función de alerta () que muestra la siguiente ventana de alerta:

```
La función init() ha comenzado
```

2.2 El operador return que termina la operación especial de la función init () y que no devuelve ningún valor.

Como resultado de la ejecución de init () se escribe un alerta. En realidad este programas contiene un algoritmo muy raro, porque el uso de esta init () sirve de muy poco. Realmente, no tiene sentido utilizar una función que solo informa a un comerciante de que la función está siendo ejecutada. En este caso, el algoritmo se utiliza sólo para la visualización de la ejecución de init (). Hay que prestar atención a que el función especial de inicio init () se ejecuta en un programa solo una vez. La ejecución de la función tiene lugar al comienzo del programa, después de que la operación de la parte de la cabeza ha sido procesada. Cuando el operador de return se ejecuta en la función especial init (), el programa devuelve el control al Terminal de Usuario.

4. El Terminal de Usuario detecta la descripción de la función especial start () en el programa:

```
int start()                          // Función Especial start()
  {
  double  Precio = Bid;                  // Variable Local
VG_Contador++;                          // Contador de ticks
Alert ("Nuevo tick ", VG_Contador,"   Precio = ",  Precio);// Alert
    return;                            // Salir de start()
```

31. El control está en manos de la Terminal de Usuario. El Terminal de Usuario espera un nuevo tick y no comienza la ejecución del programa hasta que no llegue ese nuevo tick. Esto significa que desde hace algún tiempo el programa no está funcionando, es decir, no se realiza ninguna acción en ella. Una pausa aparece. La necesidad de esperar a un tick es una propiedad de la función start () y no hay forma de que un programa pueda influir en esta propiedad (como por ejemplo desactivarla). El programa espera y mantiene el control hasta que aparezca un nuevo tick. Cuando llega un nuevo tick , el Terminal de Usuario pasa el control al programa, es decir, a la función especial start () (en este caso, de acuerdo con la propiedad del la función start () del AE). Como resultado de ello se inicia su ejecución.

32 (1). En la línea

```
double  Precio = Bid;                  // Variable Local
```

las siguientes acciones se llevan a cabo:

32,1 (1). Declaración de una variable local Precio (véase tipos de variables). El valor de esta variable local estará disponible en cualquier parte de la función especial start ().

32,2 (1). Ejecución del operador de asignación. El valor del actual precio de oferta Bid se asigna a la variable Precio. Un nuevo valor del precio aparece cada vez que venga un nuevo tick (por ejemplo, el primer tick una nueva cotizacion de precios puede ser igual a 1,2744).

33 (1). A continuación, la siguiente línea se ejecuta:

```
 Contador++;                           // Contador de ticks
```

No este poco habitual registro es plenamente analologa a Contador = Contador +1;

En el momento de pasar el control a esta línea, el valor de la variable Contador es igual a cero. Como resultado de la ejecución de Contador ++, el valor del Contador se incrementará en uno. Así que, en el momento de pasar el control a la línea siguiente, el valor de Contador será igual a 1.

34 (1). La siguiente línea contiene llamada a la función Alerta ():

```
Alert ("Nuevo tick ", Contador,"   Precio = ",  Precio); // Alerta
```

La función escribirá todas las constantes y variables enumeradas entre paréntesis.

En la primera ejecución de la función start () el programa va a escribir un nuevo tick, entonces se refieren a las variables para obtener su valor (en la primera ejecución este valor es 1), escribe este valor, luego escribirá Precio = y se refieren a la variable Precio para obtener su valor y escribirlo (en nuestro ejemplo es 1.2744).

Como consecuencia de ello se escribirá la siguiente línea:

```
Nuevo tick numero 1, Precio = 1.2744
```

35 (1). Operador

```
return;                          // Salir de start()
```

Termina la operación especial de la función start ().

36. El control es devuelto al Terminal de Usuario (hasta que llegue un nuevo tick).

Así es como ejecuta la función start () de un Asesor Experto. Cuando la ejecución termina, la función especial start () devuelve el control al Terminal de Usuario y cuando llegue un nuevo tick, el Terminal haraá comenzar su funcionamiento una vez más. Este proceso (iniciar la ejecución de la función start () y devolver el control a la Terminal de Usuario) se puede repetir durante un largo tiempo (varios días o semanas). Durante todo este tiempo la función especial start () se funcionará de vez en cuando. Dependiendo de los parámetros del entorno (nuevos precios, tiempo, condiciones de comercio, etc) pueden realizar diferentes acciones (como la apertura o la modificación de las órdenes) en la función especial start ().

37. Desde el momento que se recibe un nuevo tick, se repiten las acciones de los puntos 32-36. Sin embargo, aunque la secuencia de ejecución de los operadores se repite, el valor que se obtiene de las variables es nuevo cada vez. Vamos a ver las diferencias entre la primera y la segunda la ejecución de la función especial start ().

32 (2). En la línea

```
double  Precio = Bid;                    // Variable Local
```

se llevan a cabo las siguientes acciones:

32.1 (2). Declaración de la variable local Precio (sin cambios).

32,2 (2). Ejecución del operador de asignación. El valor del actual precio de oferta bid de se asigna a la variable Precio (aparece el valor de un precio nuevo cada vez que llega una nueva cotización, por ejemplo, los precios del segundo tick del símbolo será igual a 1,2745) (hay cambios).

33 (2). A continuación, la siguiente línea se llevará a cabo:

```
Contador++;                      // Contador de ticks
```

Por el momento antes de pasar el control a esta línea, el valor de la variable Contador (después de la primera ejecución de la función start ()) es igual a 1. Como resultado de la ejecución de Contador++ el valor de Contador se incrementará en uno. Así, en la segunda ejecución la variable Contador será igual a 2 (modificado).

34 (2). Alerta ():

```
Alert ("Nuevo tick ", VG_Contador,"  Precio = ",  Precio); // Alerta
```

Escribe todas las constantes y variables (sus nuevos valores) que se enumeran entre paréntesis.

En la segunda ejecución de start () el programa escribe un nuevo tick, entonces se refieren a la variable VG_Contador para obtener su valor (en la segunda ejecución es igual a 2), a continuación escribe este valor, escribe Precio = y se refieren al valor de la variable para obtener el precio y escribirlo que en nuestro ejemplo es 1.2745 (cambiado).

Como consecuencia de ello se escribirá la siguiente:

```
Nuevo tick numero 2, Precio = 1.2745
```

35 (2). Operador

```
return;                          // Salir de start()
```

Termina la operación start () (sin cambios).

36 (2). El control se devuelve al Terminal de Usuario en espera de un nuevo tick.

37 (2). Entonces se repite de nuevo. En la tercera start () la ejecución variables se obtienen nuevos valores y será escrita por la función de alerta (), es decir, el programa repite los puntos 32-36 (3). Y luego una y otra vez: 32 - 36 (4), 32 - 36 (5),..(6) .. (7) .. (8) ... Si el usuario no toma ninguna acción, este proceso se repetirá indefinidamente. Como resultado del funcionamiento de start () este programa veremos la historia de los cambio de precio.

Los próximos eventos ocurrirán sólo cuando un usuario decide terminar el programa fuerza quitar el programa de un gráfico en forma manual.

4. El Terminal de Usuario pasa el control a la función especial deinit () (de acuerdo con sus propiedades).

```
int deinit()                          // Función Especial deinit()
  {
  Alert ("La función deinit() ha comenzado la salida");   // Alert
  return;                             // Exit deinit()
  }
```

Sólo hay dos operadores en el cuerpo de la función.

41. Alerta () escribe:

```
Función deinit () ha comenzado la salida
```

42. Operador return termina la operación de deinit ().

La ejecución función deinit () se inicia por el Terminal de Usuario sólo una vez, después de que la alerta de arriba se aparezca en Alert () la ventana y el programa se eliminarán del gráfico.

5. Aquí termina la historia de ejecución de un Asesor Experto.

Al vincular este programa de ejemplo cualquier gráfico e iniciarlo el programa operativo mostrará una ventana que contiene todas las alertas generadas por la función de alerta (). Por el contenido de las descripciones es fácil de entender la función especial que está conectada con esta o aquella entrada.

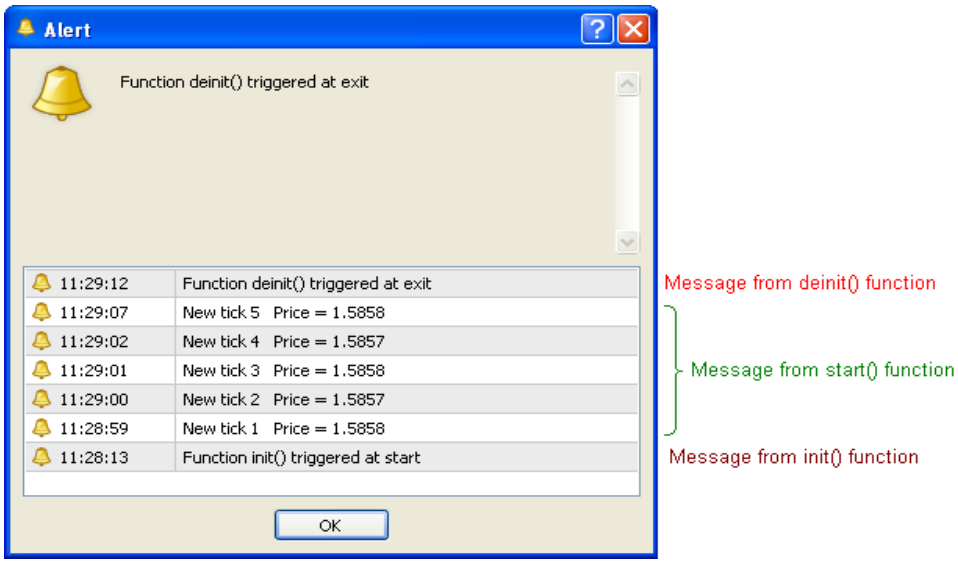


Fig. 35. Resultados del funcionamiento del programa simple.mq4.

A partir de este ejemplo se puede ver fácilmente que un programa se ejecuta de acuerdo con las propiedades de las funciones especiales descritas en Funciones especiales. Terminar el programa e iniciarlo de nuevo. Después de hacer esto varias veces, obtendrá experiencia en la utilización de su primer programa. Se trabajará tanto ahora como la próxima vez. Otros programas que usted escriba también serán construidos de acuerdo con la estructura descrita y para el inicio de su ejecución, tendrá que vincularlo a un gráfico.

Trate de entender todos los conceptos y reglas y el proceso de creación de programas en MQL4 será fácil y agradable.

.

Ejemplos de aplicación

En la sección anterior analizamos un ejemplo de ejecución de funciones especiales de ejecución en un simple Asesor Experto simple.mq4. Para una mejor práctica vamos a analizar algunas modificaciones más de este programa.

Ejemplo de una correcta estructura programática

Por regla general, las descripciones de funciones se indican en el mismo orden en que sean llamados para su ejecución por parte del Terminal de Usuario, es decir, primero va la descripción de la función especial init (), start () y el último es deinit (). Sin embargo, funciones especiales son llamados para ser ejecutadas por el Terminal de Usuario de conformidad con sus propia propiedades particulares,que es la razón por lo que no importa la ubicación de la descripción del programa. Vamos a cambiar el orden de las descripciones y ver el resultado (Asesor Experto possible.mq4).

```
//+------------------------------------------------------------------+
// possible.mq4
// To be used as an example in MQL4 book.
//+------------------------------------------------------------------+
int Contador=0;                          // Variable Global
//+------------------------------------------------------------------+
int start()                              // Función especial start()
  {
  double Precio = Bid;                   // Variable Local
  Contador ++;
  Alert("Nuevo tick ", VG_Contador,"   Precio = ",Precio);// Alerta
  return;                                // salida de start()
  }
//+------------------------------------------------------------------+
int init()                               // Función especial init()
  {
  Alert ("La funcción init() ha comenzado"); // Alerta
  return;                                // Salir de init()
  }
//+------------------------------------------------------------------+
int deinit()                             // Special funct. deinit()
  {
  Alert ("Function deinit() a desencadenado la salida");// Alert
  return;                                // Exit deinit()
  }
//+------------------------------------------------------------------+
```

A partir de este Asesor Experto se verá que la ejecución de secuencias de funciones especiales en un programa no depende del orden de las descripciones en el programa. Usted puede cambiar las posiciones de las descripciones en función del código fuente y el resultado será el mismo que en la ejecución del Experto Asesor simple.mq4.

Ejemplos de una incorrecta estructura programática

Pero el programa se comportará de forma diferente si cambiamos la posición de la parte de la cabeza. En nuestro ejemplo, vamos a indicar start () antes que la cabeza (de expertos Advisri incorrect.mq4):

```
//+------------------------------------------------------------------+
// possible.mq4
// To be used as an example in MQL4 book.
//+------------------------------------------------------------------+
int start()                              // Función especial start()
  {
  double Precio = Bid;                     // Variable Local
  Contador ++;
  Alert("Nuevo tick ", VG_Contador,"   Precio = ",Precio);// Alerta
  return;                                // salida de start()
  }
//+------------------------------------------------------------------+
int Contador =0;                           // Variable Global
//+------------------------------------------------------------------+
int init()                               // Función especial init()
  {
  Alert ("La funcción init() ha comenzado"); // Alerta
  return;                                // Salir de init()
  }
//+------------------------------------------------------------------+
int deinit()                             // Special funct. deinit()
  {
  Alert ("Function deinit() a desencadenado la salida");// Alert
  return;                                // Exit deinit()
  }
//+------------------------------------------------------------------+
```

Al intentar compilar este Asesor Experto, MetaEditor mostrará un mensaje de error:

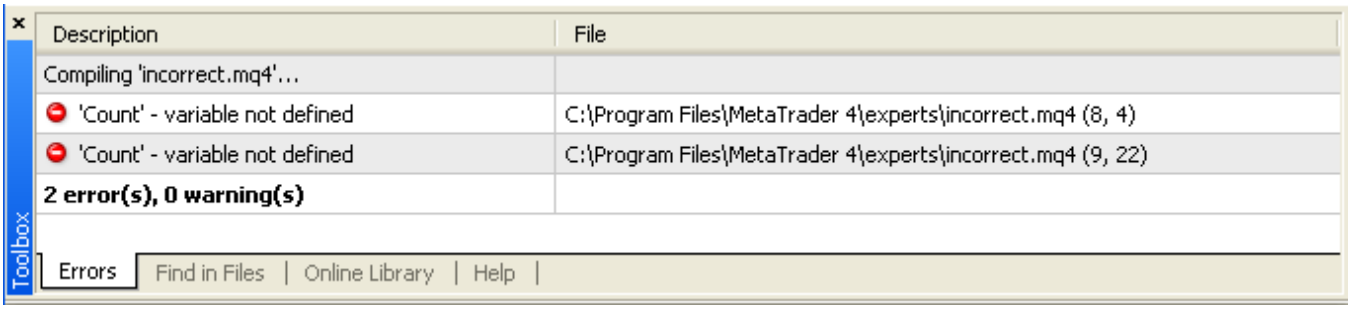


Fig. 36. Mensaje de error durante de la compilación del programa incorrect.mq4.

En este caso la línea

```
int Contador=0;                          // Variable Global
```

se escribe fuera de todas las funciones, pero no está al comienzo de un programa sino en algún lugar en medio del código.

El momento decisivo en la estructura del programa es que la declaración de la variable global Contador se hace después de declarar la función (en nuestro caso - función especial start ()). En esta sección no vamos a discutir los detalles de la utilización de variables globales, los tipos de variables y sus reglas de uso se describen en la sección Variables. Cabe señalar aquí que cualquier variable global debe ser declarada antes (al principio del texto) que la primera llamada a cualquier función (en nuestro caso es en la función start ()). En el programa que se analiza esta norma fue violada y el compilador mostrará un mensaje de error.

Ejemplo de utilización de una función definida por el usuario

Ahora vamos a ver cómo se comporta el programa en relación con funciones personalizadas. Con este fin vamos a actualizar el código descrito en el ejemplo de un simple Asesor Experto simple.mq4 y luego lo vamos a analizar. Un programa con una función definida por el usuario tendrá este aspecto (Asesor Experto userfunction.mq4):

```
//+------------------------------------------------------------------+
// userfunction.mq4
// Para ser usado como un ejemplo del libro de MQL4.
//+------------------------------------------------------------------+
int Contador =0;                                  // Variable Global
//+------------------------------------------------------------------+
int init()                                     // Función Especial init()
  {
  Alert ("La funcion init () ha comenzado ");// Alert
  return;                                     // Exit init()
  }
//+------------------------------------------------------------------+
int start()                                    // Función Especial start()
  {
  double  Precio = Bid;                            // Variable Local
  Mi_Funcion();                               // Llamada a la funcion personal
   Alert ("Nuevo tick ", Contador,"   Precio = ",  Precio);// Alert
  return;                                     // Exit start()
  }
//+------------------------------------------------------------------+
int deinit()                                   // Función Especial deinit()
  {
  Alert ("La función deinit() ha comenzado la salida");   // Alert
  return;                                     // Exit deinit()
  }
//+------------------------------------------------------------------+
//+------------------------------------------------------------------+
int My_Function()                                // Descripción de la función definida por el usuario
  {
   Contador++;                               // Contador de ticks
  }
```

En primer lugar veamos lo que ha cambiado y lo que se ha mantenido sin cambios.

Partes sin modificar:

1. La parte de cabecera no se ha modificado.

```
//-------------------------------------------------------------------
// userfunction.mq4
// Para ser usado como un ejemplo del libro de MQL4.
//-------------------------------------------------------------------
int Contador=0;                          // Variable Global
//-------------------------------------------------------------------
```

2. Función especial init () no se ha modificado.

```
int init()                               // Función Especial init()
  {
  Alert ("La funcion init () ha comenzado ");// Alert
  return;                                // Exit init()
  }
```

3. Función especial deinit () no se ha modificado.

```
int deinit()                             // Función Especial deinit()
  {
  Alert ("La función deinit() ha comenzado la salida");   // Alert
  return;                                // Exit deinit()
  }
```

Cambios:

1. Se ha añadido: la función definida por el usuario. Mi_Funcion ()

```
int My_Function()                        // Descripción de la función definida por el usuario
  {
   Contador++;                           // Contador de ticks
  }
```

3. El código especial de la función start () también ha cambiado: ahora contiene la llamada a función definida por el usuario y ya no existe la linea para cálculo de la variable Contador .

```
  int start()                            // Función Especial start()
  {
  double  Precio = Bid;                  // Variable Local
  Mi_Funcion ();                         // Llamada a la funcion personal
   Alert ("Nuevo tick ", Contador,"   Precio = ",  Precio);// Alerta
  return;                                // Exit start()
  }
```

En la sección de Ejecución de Programas analizamos la ejecución de la orden de init () y deinit (). En este ejemplo, estas funciones se llevarán a cabo de la misma manera, por lo que no vamos a insistir en su funcionamiento. Vamos a analizar la ejecución de la función especial start () y la función definida por el usuario Mi_Funcion (). La descripción de la función definida por el usuario se encuentra fuera de todas las funciones especiales que es como debe ser. La llamada a la Función definida por el usuario se indica en start () el código, que también es correcto.

Después de que init () ha terminado su ejecución, el programa se ejecutará de manera:

31. The función especial start () está a la espera de ser iniciada por el Terminal de Usuario. Cuando llega un nuevo tick, el terminal inicia esta función para su ejecución. Como resultado de esto se llevan a cabo las siguientes acciones:

32 (1). En la línea

```
double  Precio = Bid;              // Variable Local
```

las mismas acciones se llevan a cabo:

32,1 (1). Precio variable local se inicializa (ver tipos de variables). El valor de esta variable local estará disponible en cualquier parte de la función especial start ().

32,2 (1). Se ejecuta el operador de asignación. El último precio de oferta disponible se asignará a la variable precio (por ejemplo en el primer tick es igual a 1,2744).

33 (1). Después viene Mi_Funcion();

```
Mi_Funcion ();                     // Llamada a la funcion personal
```

Esta línea se lleva a cabo dentro de start (). El resultado de la aplicación de esta parte del código (la llamada a función de usuario) está en el paso del control al cuerpo (descripción) de la función que más tarde regresará al lugar de la llamada.

34 (1). Sólo hay un operador en la descripción de la función de usuario:

```
    Contador++;                    // Contador de ticks
```

En la primera llamada a la función definida por el usuario la variable es igual a cero. Como resultado de la ejecución del operador Contador++; el valor de Contador se incrementará por uno. Después de haber ejecutado este operador (el único y el último) la función de usuario termina su operación y devuelve el control al lugar desde donde ha sido llamado.

Cabe señalar aquí que las funciones definidas por el usuario solo pueden ser llamadas desde funciones especiales (o desde otras funciones definidas por el usuario que a su vez han sido llamadas desde funciones especiales). Esa es la razón por la que la siguiente declaración es correcta: en cualquier momento una de las funciones especiales estará en funcionamiento (o start () está a la espera de un nuevo tick para ser iniciada por el Terminal de Usuario) y las funciones personalizadas serán ejecutadas solo en el interior de las funciones especiales.

En este caso se devuelve el control a la función especial start () que se está ejecutando, es decir, a la línea siguiente que llama al operador de función:

35 (1). Esta línea contiene Alerta ():

```
Alert ("Nuevo tick ", Contador," Precio = ", Precio); // Alerta
```

La función de alerta () mostrará en una ventana todas las constantes y variables enumeradas entre paréntesis:

```
New tick 1 Price = 1.2744
```

36 (1). Operador

```
return;                            // Exit start()
```

termina start ().

37. El control se pasa al Terminal de Usuario en espera de una nuevo tick.

Las nuevas ejecuciones start (), se obtienen los nuevos valores de las variables y serán mostrados los mensajes de alerta (), es decir, el programa llevará a cabo los puntos 32 - 36. En cada inicio () de la ejecución (en cada tick) se llamará a la función de usuario Mi_Funcion () y esta función y se ejecutará. La ejecución de start () continuará hasta que un usuario decida poner fin a la operación del programa. En este caso, la función especial deinit () se ejecutará y el programa dejará de operar.

El programa userfunction.mq4 iniciado para su ejecución mostrará una ventana que contiene los mensajes de alerta (). Tenga en cuenta, el resultado de la operación del programa será el mismo que el resultado de un simple Asesor Experto simple.mq4 operación. Es evidente que la estructura de userfunction.mq4 se compone de conformidad con el orden usual de ubicación de bloques funcionales.

Si se utiliza otro orden aceptable, el resultado será el mismo.

Operadores

Esta sección se refiere a las normas de formato y ejecución de los operadores utilizados en MQL4. Cada sección incluye ejemplos sencillos que muestran el modo de ejecución de los operadores. Para asimilar el material en su totalidad, se recomienda a compilar y poner en marcha la ejecución de los programas de todos los ejemplos. Esto también le ayudará a consolidar habilidades en el trabajo con MetaEditor.

- Operador de asignación.
  Este es el operador es el más simple e intuitivo. Todos conocemos la operación de asignación de la asignatura de matemáticas: El nombre de una variable se situa a la izquierda del signo de igualdad, el valor que se le da está a la derecha del signo de igualdad.

- Operador condicional "if-else".
  A menudo es necesario para orientar el programa en una u otra dirección en relación con determinadas condiciones. En estos casos, el operador "if-else" es muy útil.

- Ciclo del Operador "while".
  El tratamiento de una gran cantidad de datos de tipo de array generalmente requiere múltiples repeticiones de la misma operación. Se puede organizar un bucle de este tipo de operaciones con el operador de ciclo "while". Cada una ejecución de las operaciones en un ciclo se llama *iteración.*

- Operador de ciclo "for".
  El operador "for" también es un operador de ciclo . Sin embargo, a diferencia con el operador "while" es que para la ejecución de iteraciones, generalmente contienen dentro de sí mismo el valor inicial y el valor final de una determinada condición.

- Operador de "break".
  Si quiere interrumpir el trabajo de un operador de ciclo sin tener que ejecutar el resto de las iteraciones, se necesita el operador "break". Se utiliza sólo en los operadores "While", "for" y "Switch", y en ningún otro.

- Operador "Continue".
  Un operador de gran utilidad es el operador que salta a la siguiente iteración dentro de un ciclo. Permite que el programa salte todos los operadores restantes en la iteración actual y pase a la siguiente.

- Operador de "switch".
  Este operador es un "conmutador" que permite al programa elegir una de entre varias alternativas posibles. Para cada alternativa se describe su constante predefinida que es el caso de esa alternativa.

- Función de Call.
  Entendemos como función call que la función que es llamada ejecutará algunas operaciones. La función puede devolver un valor del tipo predefinido. La cantidad de parámetros transferidos en la función no podrá ser superior a 64.

- Descripción y función del operador "return".
  Antes de llamar a una función definida por el usuario, usted debe describirla primero. La descripción de la función debe especificar su tipo, el nombre y la lista de parámetros. Además, en el cuerpo de la función estará los operadores ejecutables. El trabajo de una función se termina con la ejecución del operador "return".

Operador de asignación

El operador de asignación es el operador más simple y más frecuentemente usado.

Formato del operador de asignación

Operador de asignación representa un registro que contiene el carácter "=" (signo de igualdad). A la izquierda de este signo de igualdad se especifica el nombre de una variable, a la derecha de ella damos una expresión. El operador de asignación se termina con ";" (punto y coma).

```
Variable = Expresión;     // operador de asignación
```

Se puede distinguir el operador de asignación de otras líneas en el texto del programa por la presencia del signo de igualdad. Puede especificar una expresión como: una constante, una variable, una llamada a una función, o una expresión como tal.

Ejecución del operador de asignación

Calcula el valor de la expresión a la derecha de la igualdad y asignar el valor obtenido para la variable especificada a la izquierda del signo de igualdad.

El operador de asignación, al igual que cualquier otro operador, es ejecutable. Esto significa que el registro que compone el operador de asignación se realiza de acuerdo a una regla. Cuando se ejecuta el operador, se calcula el valor de la parte derecha y, a continuación, se asigna a la variable el valor que esta a la izquierda del signo de igualdad. Como resultado de la ejecución del operador de asignación, la variable en la parte izquierda siempre toma un nuevo valor, este valor puede ser distinto o el mismo que el anterior valor de la variable. La expresión en la parte derecha del operador de asignación se calcula de acuerdo con el orden de las operaciones (véase Operaciones y las expresiones).

Ejemplos de operadores de asignación

En un operador de asignación, se permite declarar el tipo de una variable a la izquierda de la igualdad de signo:

```
int In = 3;              // The constant value is assigned to variable In
double Do = 2.0;         // The constant value is assigned to variable Do
bool Bo = true;          // The constant value is assigned to variable Bo
color Co = 0x008000;     // The constant value is assigned to variable Co
string St = "sss";       // The constant value is assigned to variable St
datetime Da= D'01.01.2004';// The constant value is assigned to variable Da
```

Las variables declaradas previamente se utilizan en un operador de asignación, sin especificar su tipo.

```
In = 7;                  // The constant value is assigned to variable In
Do = 23.5;               // The constant value is assigned to variable Do
Bo = 0;                  // The constant value is assigned to variable Bo
```

En un operador de asignación no se permite que el tipo de una variable sea declarada en la parte derecha del signo de igualdad:

```
In = int In_2;          // Variable type may not be declared in the right part
Do = double Do_2;       // Variable type may not be declared in the right part
```

En un operador de asignación, no se le permite que el tipo de una variable sea declarado más de una vez.

```
int In;                 // Declaration of the type of variable In
int In = In_2;          // The repeated declaration of the type of the variable (In) is not allowed
```

Ejemplos del uso de funciones definidas por el usuario y funciones estándar en la parte derecha:

```
In = My_Function ();     // The value of user-defined function is assigned to variable In
Do = Gipo(Do1,Do1);      // The value of user-defined function is assigned to variable Do
Bo = IsConnected();      // The value of standard function is assigned to variable Bo
St = ObjectName(0);      // The value of standard function is assigned to variable St
Da = TimeCurrent();      // The value of standard function is assigned to variable Da
```

Ejemplo de utilización de expresiones en la parte derecha:

```
In = (My_Function ()+In2)/2;       // The variable In is assigned
                                   // ..with the value of expression
Do = MathAbs(Do1+Gipo(Do2,5)+2.5); // The variable Do is assigned
                                   // ..with the value of expression
```

En los cálculos del operador de asignación, son aplicables las normas del typecasting (véase el Typecasting).

Ejemplos de Asignación de operadores en forma corta

En MQL4 tambien se utiliza una forma breve de componer los operadores de asignación. Es la forma de asignación de los operadores cuando usamos la asignación a otras operaciones distintas de operación de asignación **"="** (signo de igualdad) (véase Operaciones y las expresiones). La forma corta los operadores estan sometidos a las mismas normas y limitaciones. La forma corta de la asignación operadores se utiliza en el código para una mejor visualización. Un programador puede, a su opción, utilizar una u otra forma de la asignación operador. Cualquier forma corta del operador de asignación puede ser fácilmente re-escrita en la forma normal del formato completo del operador de asignación. El resultado de su ejecución queda absolutamente inalterado.

```
In /= 33;           // Short form of the assignment operator
In = In/33;         // Full form of the assignment operator

St += "_exp7";      // Short form of the assignment operator
St = St + "_exp7";  // Full form of the assignment operator
```

El operador condicional if-else

Por regla general, si se escribe un programa de aplicación, es necesario que se de un código de varias soluciones en un solo programa. Para resolver estas tareas, se puede usar en el código el operador condicional 'if-else'.

Formato del operador "if-else"

Formato completo

El formato completo del operador **'if-else'** contiene una partida que incluye una condición, el cuerpo 1, la palabra clave 'else', y el cuerpo 2. El cuerpo del operador puede estar formado por uno o varios operadores, los cuerpos van encerrados entre llaves.

```
if (condición)                  // Cabecera del operador y condición
{
Bloque 1 de operadores          // Si la condición es verdadera, entonces ..
Composición cuerpo 1            // .. los agentes que componen el cuerpo 1 se ejecutan
}
else                            // Si la condición es falsa ..
{
Bloque 2 de operadores          // .. entonces los operadores ..
Composición cuerpo 2            // .. del cuerpo 2 se ejecutan
}
```

Formato sin 'else'

El operador **"if-else"** puede ser usado sin 'else'. En este caso, el operador "if-else 'contiene su cabecera que incluye una condición, y el cuerpo 1, que consta de uno o varios operadores cerrados entre llaves.

```
if (condición)                  // Cabecera del operador y el estado
{
Bloque 1 de operadores          // Si la condición es verdadera, entonces ..
Composición cuerpo 1            // .. agentes que componen el cuerpo 1 se ejecutan
}
```

Formato sin llaves

Si el cuerpo del operador "if-else 'consta de un solo operador, se pueden omitir las llaves.

```
if (condición)                  // Cabecera del operador y el estado
Operador                        // Si la condición es verdadera, entonces ..
                                // .. Este operador se ejecuta
```

Regla de Ejecución del operador "if-else"

Si la condición del operador "if-else" es cierta, se pasa el control al primer operador en el cuerpo 1. Después de que todos los operadores en el cuerpo 1 se han ejecutado, el control pasa al operador que sigue al operador "if-else". Si la condición del operador "if-else" es falsa, entonces:
-- Si esta la palabra clave 'else' en el operador "if-else", entonces se pasa el control al primer operador en el cuerpo 2. Después de que todos los operadores en el cuerpo 2 se han ejecutado, se pasa el control al operador que sigue el operador "if-else";
-- Si no hay una palabra clave 'else' en el operador "if-else", entonces se pasa el control al operador que sigue el operador " if-else ».

Ejemplos de ejecución del operador "if-else"

Veamos algunos ejemplos que muestran cómo podemos usar el operador "if-else».

Problema 9. Redactar un programa donde se realizan las condiciones siguientes: Si el precio de un símbolo ha subido y se ha superado un cierto valor, el programa deberá informar al comerciante sobre ese hecho, si no ha superado este valor, el programa no debe realizar ninguna acción.

Una de las soluciones para este problema puede ser, por ejemplo, como sigue (onelevel.mq4):

```
//+------------------------------------------------------------------+
// onelevel.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+
int start()                          // función especial 'star'
  {
   double
   Level,                            // declaración de variable donde estará el nivel de alerta
   Price;                            // declaración de variable donde estará el precio actual   Level=1.2753;
// Establecer el nivel
   Price=Bid;                        // Solicitud de precio actual
//+------------------------------------------------------------------+
   if (Price>Level)                  // Operador 'if' con una condición
     {
      Alert("El precio a superado el nivel establecido");  // Mensaje para el comerciante
     }
//+------------------------------------------------------------------+
   return;                                 // // Salir de start()
  }
//+------------------------------------------------------------------+
```

Cabe señalar, en primer lugar, que el programa se crea como un Asesor Experto. Esto implica que el programa va a funcionar durante bastante tiempo con el objeto de mostrar el mensaje en la pantalla tan pronto como el precio supere el nivel preestablecido. El programa tiene sólo una función especial, start (). Las variables son declaradas y comentadas al comienzo de la función. Entonces el nivel de precios se sitúa en valores numéricos y el precio actual se solicita.

El operador "if-else» se utiliza en las siguientes líneas del programa:

```
//------------------------------------------------------------------------------
 if (Price>Level)                    // Operador 'if' con una condición
   {
    Alert("El precio a superado el nivel establecido");  // Mensaje para el comerciante
   }
//------------------------------------------------------------------------------
```

Tan pronto como el control en la ejecución del programa se pasa al operador "if-else ', el programa probará su condición. Tengase en cuenta que la prueba condicionada por el operador "if-else' es una propiedad inherente de este operador. Esta prueba no puede ser ignorado durante la ejecución del operador "if-else ', se trata de la "raison d'etre " (razón de ser) de este operador y se llevará a cabo, en todos los casos. Posteriormente, según los resultados de esta prueba, el control será pasado a cualquiera de los operadores del cuerpo o fuera de él, al operador le sigue el cierre de la llave.

A continuación en la Fig. 37, se puede ver un diagrama de bloques que representa una posible secuencia en el caso de una ejecución del operador "if-else».

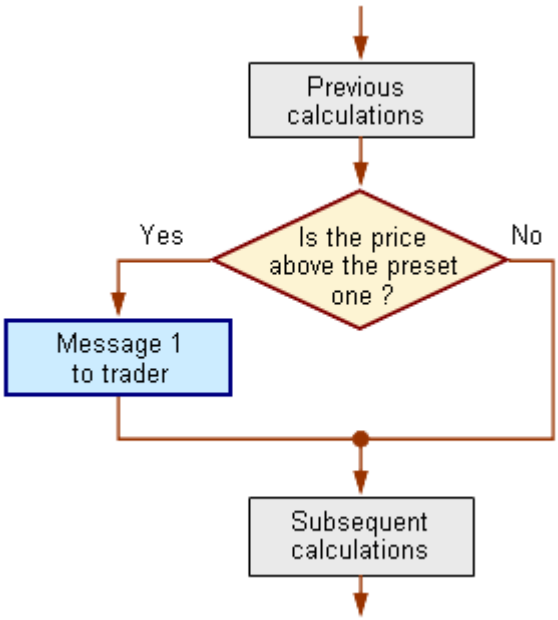


Fig. 37. Un diagrama de bloques para la ejecución del operador "if-else" en el programa onelevel.mq4.

En este y todos los siguientes gráficos, un rombo representa la verificación de la condición. Las flechas indican el componentes objetivo, para que el control será pasado después de que el bloque de declaración actual se haya ejecutado (el bloque de declaración significa un cierto juego de azar de los operadores adyacentes uno del otro). Vamos a considerar el esquema que aparece en más detalles.

El bloque de "Los cálculos previos" (una caja gris en el diagrama) en el programa onelevel.mq4 incluye lo siguiente:

```
double
   Level,                          // declaración de variable donde estará el nivel de alerta
   Price;                          // declaración de variable donde estará el precio actual      Level=1.2753;
// Establecer el nivel
   Price=Bid;                      // Solicitud de precio actual
```

Tras ejecutado el último operador en este bloque, el control pasa a la cabecera del operador "if-else" donde la condición de "¿excede el precio el nivel preestablecido?" (La caja de rombos en el diagrama, Fig. 37) se verifica:

```
 if (Price>Level)                    // Operador 'if' con una condición
```

En otras palabras, podemos decir que el programa en esta etapa esta buscando a tientas la respuesta a la siguiente pregunta: ¿la declaración entre paréntesis es verdad? La propia declaración suena literalmente como está escrito: El valor de la variable precio es superior a la de la variable nivel (el rango del precio). En el momento de comprobar si esta declaración es verdadera o falsa, el programa ya ha obtenido los valores numéricos de las variables precio y nivel. La respuesta depende de la relación entre estos valores. Si el precio está por debajo del nivel preestablecido (el valor del precio es igual o menor al valor de Nivel), la declaración es falsa, si el precio supera este nivel, la afirmación es cierta.

Así pues, cuando pasamos el control condicional después de la prueba que depende de la situación actual del mercado, **Si (If)** el precio de un símbolo se mantiene por debajo del nivel preestablecido (la respuesta es **No**, es decir, la declaración es **falsa),** el control, según a la norma de ejecución del operador "if-else ', se pasará fuera del operador, en este caso, se pasa al bloque denominado "**Cálculos posteriores",** es decir, a la línea:

```
return;                              // Salir de start()
```

Como es fácil ver, no se da ningún mensaje al Trader.

Si el precio de un símbolo supera el nivel preestablecido en el programa (la respuesta **es afirmativa,** es decir, la afirmación **es** cierta), el control será pasado al cuerpo del operador "if-else", es decir, a las siguientes líneas:

```
{
  Alert ("El precio a superado el nivel establecido");  // Mensaje para el comerciante
}
```

La ejecución de la función de Alert () dará lugar a la exhibición en la pantalla un pequeño recuadro con el siguiente mensaje:

```
El precio a superado el nivel establecido
```

La función Alert () es el único operador que hay en el cuerpo del operador "if-else", es por ello que después de su ejecución, el operador "else" se considera totalmente ejecutado, y el control pasa al operador que sigue el operador "if-else", es decir, a la línea:

```
return;                              // Salir de start()
```

La ejecución del operador «return» se traduce en que la función start () termina su trabajo, y el programa cambia al modo de espera de tick. En un nuevo tick (que también tiene un nuevo precio para el símbolo), la función start () se ejecutará otra vez. Así que, el mensaje codificado en el programa, dará o no la posibilidad de comercio, en función de si el nuevo precio supera o no el nivel preestablecido.

Los operadores if-else pueden estar anidados. Con el fin de demostrar cómo se pueden anidar estos operadores, vamos a examinar el siguiente ejemplo. El problema en sí es algo más sofisticado.

Problema 10. Redactar un programa con las siguientes condiciones: Si el precio ha crecido de manera que supera un cierto nivel 1, el programa deberá informar al comerciante sobre él, si el precio ha caído a fin de que sea inferior a un cierto nivel 2, la programa deberá informar al comerciante sobre ello, sin embargo, el programa no debe realizar ninguna acción, en cualquier otro caso.

Es evidente que, con el fin de resolver este problema, tenemos que comprobar el precio actual en dos ocasiones:

1. comparar el precio al nivel 1, y
2. comparar el precio al nivel 2.

Solución 1 del Problema 10

Actuando formalmente, podemos componer el siguiente algoritmo de solución:

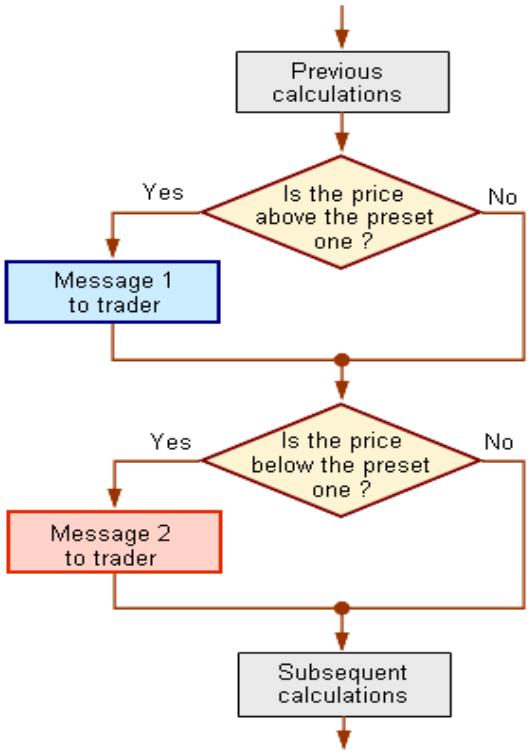


Fig. 38. Diagrama de bloques de los operadores if-else para ser ejecutado en programa twolevel.mq4.

El programa se da cuenta de que este algoritmo puede ser la siguiente (twolevel.mq4):

```
//+------------------------------------------------------------------+
// Twolevel.mq4
// El código debería ser usado para fines educativos l solamente.
//+------------------------------------------------------------------+
int start()                          // función especial 'start'
  {
   double
   Level_1,                          // nivel 1 de alerta
   Level_2,                          // nivel 2 de alerta
   Price;                           // Precio actual de mercado (precio para vender)
   Level_1=1.2850;                       // Establecer nivel 1
   Level_2=1.2800;                       // Establecer nivel 2
   Price=Bid;                        // Request price (Solicitud de precio Bid actual)
//+------------------------------------------------------------------+
   if (Price > Level_1)                // ¿Es el precio actual mayor que el nivel 1?
     {
      Alert("El precio está por encima del nivel 1");  // Mensaje al comerciante
     }
//+------------------------------------------------------------------+
   if (Price < Level_2)                // ¿Es el precio acutal menor que el nivel 2?
     {
      Alert("El precio está por debajo del nivel 2");  // Mensaje al comerciante
     }
//+------------------------------------------------------------------+
   return;                           // Salir de start()
  }
//+------------------------------------------------------------------+
```

Como es fácil de ver, el código de programa twolevel.mq4 es la versión ampliada del programa onelevel.mq4. si tuviéramos un solo nivel de los cálculos anteriores. Tenemos dos niveles en este nuevo programa, cada nivel se define numéricamente, y el programa, para resolver el problema planteado, tiene dos bloques que hacen un seguimiento de los comportamiento de los precios: ¿El precio cae dentro del rango de valores limitado por los niveles preestablecidos o es fuera de este rango?

Vamos a dar una breve descripción de la ejecución del programa.

Después de que se han realizado los cálculos preliminares, el control pasa al primer operador "if-else":

```
 //----------------------------------------------------------------------
  if (Price > Level_1)                // ¿Es el precio actual mayor que el nivel 1?
    {
     Alert("El precio está por encima del nivel 1");  // Mensaje al comerciante
    }
//----------------------------------------------------------------------
```

Independientemente de las acciones que tuvieran que llevarse a cabo con la ejecución de este operador (en este caso solo que se muestra o no un mensaje al comerciante), después de su ejecución el control pasa al siguiente operador "if-else":

```
 //----------------------------------------------------------------------
  if (Price < Level_2)                // ¿Es el precio acutal menor que el nivel 2?
    {
     Alert("El precio está por debajo del nivel 2");  // Mensaje al comerciante
    }
//----------------------------------------------------------------------
```

La ejecución consecutiva de ambos operadores resulta en la posibilidad de realizar las dos pruebas condicionales y, por último, resolver el problema. Aunque el programa resuelve la tarea en su totalidad, esta solución no puede ser considerada como absolutamente correcta. Por favor tengase en cuenta un detalle muy importante: La segunda prueba condicional se ejecutará independientemente de los resultados obtenidos en las pruebas en el primer bloque. El segundo bloque se ejecutará, incluso en el caso de que el primer operador sea verdadero, es decir, de que el precio superior al primer nivel establecido.

Cabe señalar aquí, que uno de los principales objetivos perseguidos por el programador al escribir sus programas es el algoritmo de optimización. Los ejemplos anteriores son sólo una manifestación, por lo que representan, de un corto y, por tanto, programa de ejecución rápida. Normalmente, un programa destinado a ser utilizado en la práctica es mucho más grande. Se pueden procesar los valores de cientos de variables, usar múltiples pruebas, cada una ejecutada en un tiempo determinado. Todo esto puede resultar en el riesgo de que la duración del trabajo de la función especial start () pueda superar el espacio de tiempo entre ticks. Esto significaría que algunos ticks puedrían quedar sin ser procesados. Esto es, por supuesto, una situación indeseable, y el programador debe hacer lo posible para prevenirlo. Una de las técnicas que permiten reducir el tiempo de la ejecución del programa es el algoritmo de optimización.

Vamos a considerar el ejemplo más reciente, una vez más, esta vez en términos de ahorro de recursos. Por ejemplo. Cuál es la razón por preguntar en el segundo bloque lo siguiente: "¿Esta el precio por debajo del nivel preestablecido?". Si las pruebas del primer bloque ya ha detectado que el precio está por encima del nivel superior, es evidente que el precio no puede estar por debajo del nivel inferior y por consiguiente, no hay necesidad de hacer una prueba condicional en el segundo bloque (ni para ejecutar el conjunto de los operadores en este bloque).

Solución 2 del Problema 10

Teniendo lo anterior en cuenta, vamos a considerar las siguientes, optimizado algoritmo:

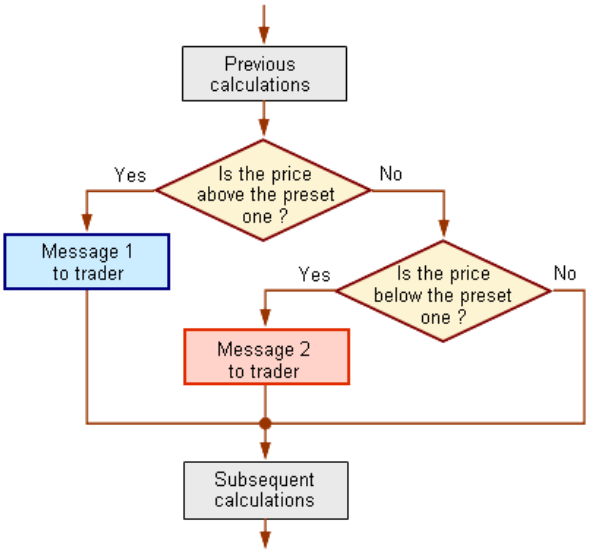


Fig. 39. Un diagrama de bloques para la ejecución en programa del operador "if-else" twoleveloptim.mq4.

Según el algoritmo se muestra en el diagrama de bloques anterior, las operaciones se ejecutan en el programa de forma no redundante. Después de la realización de los cálculos previos, el programa verificará si **el precio está por encima del nivel preestablecido.** En caso afirmativo, el programa muestrará el mensaje correspondiente para informar al comerciante y luego pasará el control a **los cálculos posteriores,** pero que la segunda condición **(esta el precio por debajo del nivel preestablecido?)** no será verificada. Sólo si el precio no ha resultado estar por encima del nivel superior (es decir la respuesta es **No** en el primer operador if-else **),** el control se pasará al segundo bloque donde se verificará la segunda condición. Si el precio resulta ser inferior al nivel mínimo establecido, se mostrará el mensaje correspondiente al comerciante. Si no es así, el control pasa al apartado "siguientes cálculos". Es evidente que, si el programa funciona según este algoritmo de acciones no redundantes, se realizan menos acciones, lo que se traduce en un ahorro considerable de recursos.

La aplicación programatica de este algoritmo implica el uso anidado del operador "if-else" (twoleveloptim.mq4):

```
//+------------------------------------------------------------------+
// Twoleveloptim.mq4
// El código debería ser usado para fines educativos únicamente.
//+------------------------------------------------------------------+

int start()                          // Special function start()
  {
   double
   Level_1,                          // Alert level 1
   Level_2,                          // Alert level 2
   Price;                            // Current price
   Level_1=1.2850;                   // Set level 1
   Level_2=1.2800;                   // Set level 2
   Price=Bid;                        // Request price
//------------------------------------------------------------------
   if (Price > Level_1)              // Check level 1
     {
      Alert("The price is above level 1");  // Message to the trader
     }
   else
     {
      if (Price < Level_2)           // Check level 2
       {
        Alert("The price is above level 2"); // Message to the trader
       }
     }
//------------------------------------------------------------------
   return;                           // Exit start()
```

```
  }
//---------------------------------------------------------------------
```

Por favor tengase en cuenta este bloque de cálculos:

```
//---------------------------------------------------------------------

  if (Price > Level_1)                  // Check level 1
    {
     Alert ("The price is above level 1");   // Message to the trader
    }
  else
    {
     if (Price < Level_2)               // Check level 2
       {
        Alert ("The price is above level 2"); // Message to the trader
       }
    }
//---------------------------------------------------------------------
```

El operador "if-else", en la cual la segunda condición es comprobada es un componente del primer operador "if-else" que pone a prueba la primera condición. Los operadores anidados se utilizan ampliamente en la práctica. Esto es a menudo razonable y, en algunos casos, la única solución posible. Por supuesto, en lugar de la función de alerta (), los programas reales pueden utilizar otras funciones o realizar acciones útiles con diversos operadores.

Una expresión compleja puede ser usado como una condición en el operador "if-else".

Problema 11. Redactar un programa que tenga en cuenta de las condiciones siguientes: Si el precio cae dentro del rango preestablecido de valores, no se hace nada, si el precio está fuera de este rango, el programa debe informar al operador sobre el mismo.

La declaración de este problema es similar al del Problema 10. La diferencia consiste en que, en este caso, no estamos interesados en la dirección del movimiento de precios superior o inferior al rango predefinido. De acuerdo con la situación del problema, tenemos que conocer sólo este hecho: ¿esta el precio dentro o fuera del rango? Podemos utilizar el mismo texto para el mensaje que se mustra.

Aquí, al igual que en las anteriores soluciones, es necesario comprobar la situación del precio en relación con dos niveles: el superior y el inferior. Hemos rechazado la solución que se muestra en el algoritmo de la Fig. 38 por que no es eficiente. Por lo tanto, según el razonamiento anterior, podemos proponer la siguiente solución:

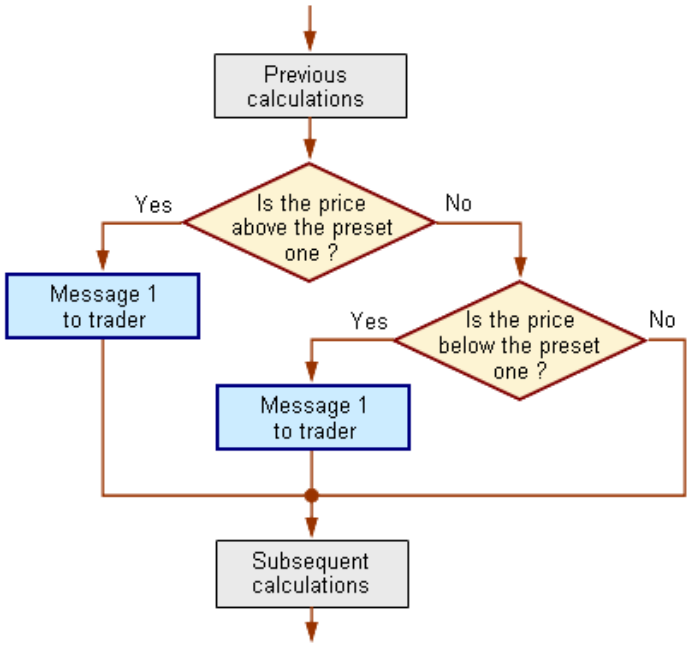


Fig. 40. Diagrama de bloques de una de las soluciones del problema 11.

Sin embargo, no hay necesidad de utilizar este algoritmo. El diagrama muestra el algoritmo que implica el uso del mismo bloque de mensajes en los distintos puntos del programa. Hay lineas de programa que se repetirán, a pesar de que la ejecución de ellas se traducirá en la redacción de los mismos mensajes. En este caso, sería mucho más eficaz utilizar el único operador "if-else ', con una compleja condición:

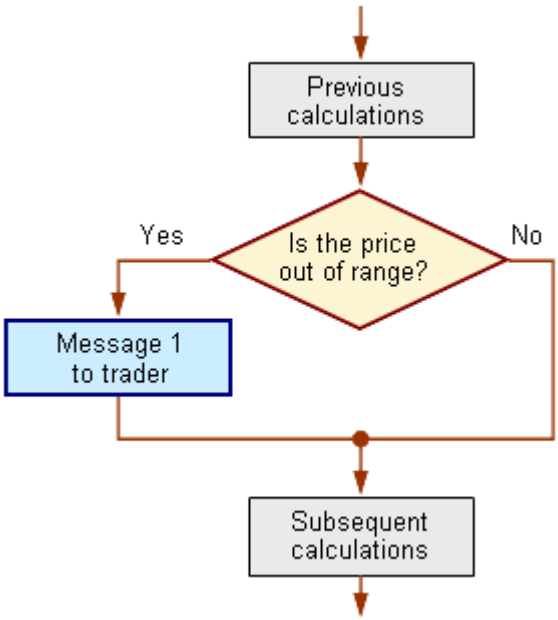


Fig. 41. Diagrama de bloques del programa para la ejecución del operador "if-else" compoundcondition.mq4.

El código del programa que implementa este algoritmo es el siguiente (compoundcondition.mq4):

```
//+------------------------------------------------------------------+
// compoundcondition.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+
int start()                                 // Special function 'start'
  {
   double
   Level_1,                                 // Alert level 1
   Level_2,                                 // Alert level 2
   Price;                                   // Current price
   Level_1=1.2850;                          // Set level 1
   Level_2=1.2800;                          // Set level 2
   Price=Bid;                               // Request price
//+------------------------------------------------------------------+
   if (Price>Level_1 || Price<Level_2)      // Test the complex condition
     {
      Alert("The price is outside the preset range");// Message
     }
//+------------------------------------------------------------------+
   return;                                  // Exit start()
  }
//+------------------------------------------------------------------+
```

La idea fundamental de la solución de este programa es el uso de condiciones complejas en el operador "if-else":

```
 if (Price>Level_1 || Price<Level_2)      // Test the complex condition
  {
   Alert("The price is outside the preset range");// Message
  }
```

En pocas palabras, esta condición es la siguiente: "Si el valor de la variable precio es superior al de la variable Level_1, **O** el valor de la variable precio es menor que la variable de Level_2, el programa debe ejecutar el cuerpo del operador if-else". Se pueden usar operaciones lógicas (**&&**,**||** y **!**) al redactar las condiciones del operador if-else, que son ampliamente utilizados en la práctica de la programación (véase el operador booleano (lógico) Operaciones).

Cuando la solución de otros problemas, puede que sea necesario para componer aún más compleja condiciones, lo que hace que no se trata, también. Algunas expresiones, incluida la anidados, puede ser incluido en paréntesis. Por ejemplo, puede utilizar la siguiente expresión como una condición en el operador "if-else":

```
   if ( A>B && (B<=C || (N!=K && F>B+3)) )     // Example of a complex condition
```

Por lo tanto, MQL4 abre grandes oportunidades para el uso de los operadores if-else, ellos pueden ser anidados, pueden contener estructuras anidadas, pueden utilizar condiciones simples y complejas de prueba, lo que se traduce en la posibilidad de componer programas simples y complejos con los de algoritmos ramificados.

Ciclo del Operador "While"

La funcionalidad más potente de MQL4 es la posibilidad de organizar ciclos (bucles).

Al crear programas de aplicación, se puede utilizar a menudo cálculos repetidos, que son en su mayoría líneas repetidas de programa. Con el fin de hacer la programación cómoda y el mismo programa fácil de utilizar, se usan los operadores de ciclo. Hay dos operadores de ciclo en MQL4: **while** y **for**. Vamos a considerar el primero de ellos en esta sección.

Formato del explotador 'While'

El formato completo operador de ciclo While (mientras que) consiste en la cabecera que contiene una condición, y el ejecutable adjunto ciclo del cuerpo en llaves.

```
while (condición)                    // Cabecera del operador de ciclo

{                                    // Apertura llave
 Bloque de operadores                // El cuerpo de un operador de ciclo puede consistir ..
 que componen el ciclo del cuerpo    // ..  de varios operadores
 }                                   // Cierre llave
```

Si el ciclo del cuerpo se compone de un solo operador en el operador 'while' puede omitir las llaves.

```
while (condición)                    // Cabecera del operador de ciclo
Operador, cuerpo del ciclo           // ciclo del cuerpo es un operador
```

Regla Ejecución para el operador 'While'

Mientras la condición del operador "While" sea **"verdadera"**: El programa pasa el control al cuerpo del operador de bucle, después de que sido ejecutados todos los operadores en el cuerpo del bucle, se pasa el control a la cabecera para verificar la condición.
Si la condición del operador "While" es **"falso"**, el control debe ser pasado al operador que siguiente al operador de bucle "While".

Vamos a considerar un ejemplo.

Problema 12. Calcular el coeficiente de Fibonacci con la precisión de 10 cifras significativas.

En primer lugar, vamos a describir brevemente coeficiente de Fibonacci. El matemático italiano, Leonardo Fibonacci, descubrió una secuencia de números única:

1 1 2 3 5 8 13 21 34 55 89 144 233 ...

Cada número de esta secuencia es la suma de los dos números anteriores. Este número de secuencia tiene algunas propiedades únicas: La relación de dos números sucesivos en la secuencia es igual a 1,618, mientras que la proporción entre un número y el anterior es igual a 0,618. El ratio 1,618 fue llamado después ratio de Fibonacci, así como la secuencia de arriba que se llama secuencia de Fibonacci (también debería tenerse en cuenta que 0.3819, un conjugado para el número de Fibonacci, fue obtenido por la multiplicación de si mismo: 0.3819 x 0,618 = 0,618).

La tarea consiste en calcular el número de Fibonacci con la mayor precisión. Si analizamos el coeficiente de Fibonacci de varias decenas de elementos, es evidente que los coeficientes obtenidos en todo el rango de número irracional de 1,61803398875 ..., tomando mayor o menor valor por turnos. Los mayores números de la secuencia están involucrados en los cálculos, los más pequeños es la desviación del resultado de este valor.

No sabemos de antemano qué número exacto que debemos tomar para sus coeficientes que difieren entre sí sólo a partir de la décima cifra significativa. Por lo tanto, es necesario para componer un programa que busca en forma consecutiva hasta que los coeficientes de la diferencia entre ellos es menor de 0,0000000001.

En este caso, hemos creado un script (fibonacci.mq4) para resolver Problema 12, porque el programa no será utilizado durante mucho tiempo comprobando cada tick. Una vez que se vincula a la ventana de símbolo, el script debe llevar a cabo todos los cálculos necesarios (incluidos mostrar los resultados), después será desvinculado de la ventana del Terminal de Usuario.

```
//+------------------------------------------------------------------+
// fibonacci.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+
int start ()                          // Función especial start ()
  {
//+------------------------------------------------------------------+
   int i;                       // parámetro formal, contador de iteraciones
   double
   A,B,C,                             // Los números en la secuencia
   Delta,                             // diferencia real entre los coeficientes
   D;                                 // Preset precisión
//+------------------------------------------------------------------+
   A=1;                               // valor inicial
   B=1;                               // valor inicial
   C=2;                               // valor inicial
   D=0.0000000001;                 // valor de la precisión
   Delta=1000.0;                      // valor inicial
//+------------------------------------------------------------------+
   while(Delta>D)                     // Cabecera del operador de ciclo
     {                          // Apertura de llave del cuerpo del operador de bucle While
      i++;                            // incremento del contador de iteraciones
      A=B;                            // Siguiente valor
      B=C;                            // Siguiente valor
      C=A + B;                        // Siguiente valor
      Delta=MathAbs(C/B - B/A);     // Buscar diferencia entre los coeficientes
     }                          // Cierre de llave del cuerpo del operador de bucle While
//+------------------------------------------------------------------+
   Alert("C=",C," Fibonacci number=",C/B," i=",i);  // mostrar en la pantalla
   return;                            // Salir de star ()
  }                                   //Cierre de llave de la funcion especial star
//+------------------------------------------------------------------+
```

Al comienzo del programa, se declaran (y describen) las variables. En las líneas posteriores, son valores numéricos asignados a las variables. A, B y C toman el valor de los primeros números de la secuencia de Fibonacci. Cabe señalar aquí que, si bien la propia secuencia sólo contiene números enteros, el cociente de la división debe ser considerado en el programa como un número real. Si usamos el tipo ***int*** para estos números aquí, sería imposible calcular el coeficiente de Fibonacci, por ejemplo: 8/5 = 1 (integer 1, sin la parte fraccional). Así, en este caso, usamos las variables de tipo ***double***. El ciclo del operador, como tal, tiene este aspecto:

```
//+------------------------------------------------------------------------------+
  while (Delta>D)                        // Cabecera del operador de ciclo
    {                                 // Apertura de llave del cuerpo del operador de bucle While
     i++;                                // incremento del contador de iteraciones
     A=B;                                // Siguiente valor
     B=C;                                // Siguiente valor
     C=A + B;                            // Siguiente valor
     Delta=MathAbs(C/B - B/A);      // Buscar diferencia entre los coeficientes
    }                                 // Cierre de llave del cuerpo del operador de bucle While
//+------------------------------------------------------------------------------+
```

Vamos a considerar un diagrama de bloques del operador de bucle 'while':

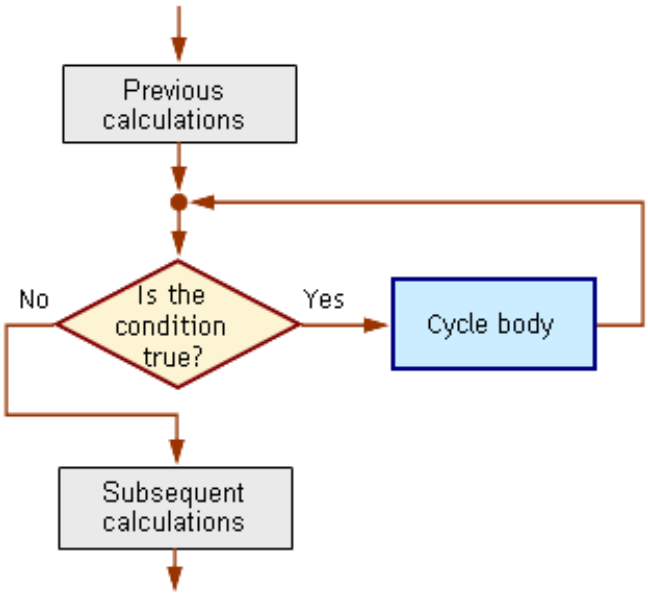


Fig. 42. Diagrama de bloques del operador 'while' en la ejecución del programa fibonacci.mq4.

El operador de ciclo comienza el trabajo con la prueba de la condición. El ciclo se llevará a cabo en repetidas ocasiones, hasta que la condición (Delta> D) sea verdad. Usted comprenderá el código de programación mucho mejor, si usted dice la frase clave (la norma de ejecución) leyendo el operador en voz alta. Por ejemplo, puede leer el operador while (mientras que) así: **"Mientras se cumpla esta condición, realiza lo/los siguiente/s: .."**. En este caso, la cabecera del operador es 'mientras que', y la condición: "Mientras Delta es mayor que D", realiza lo siguiente... Luego puede ir a analizar las lineas del programa que componen el cuerpo del bucle y que será ejecutado si la condición es verdadera.

Antes de que el control en el ejemplo anterior de fibonacci.mq4 pasa al operador de ciclo 'while', los valores de estas variables son iguales a 1000,0 y 0,0000000001, respectivamente, por lo que la condición se cumple en la primera convocatoria para el operador de ciclo. Esto significa que, después de que el estado ha sido probado, el control se pasa al primer operador en el ciclo cuerpo operador.

En nuestro ejemplo, es el siguiente operador:

```
    i++;                              // incremento del contador de iteraciones
```

En las tres líneas posteriores, se calculan los valores de la siguiente serie de secuencia de elementos:

```
    A=B;                              // Siguiente valor
    B=C;                              // Siguiente valor
    C=A + B;                          // Siguiente valor
```

Es fácil ver que las variables tomen los valores de la siguiente (más cercana más grande) elemento. Antes de que el operador del ciclo se ejecute, los valores de A, B y C son iguales a 1,0, 1,0 y 2,0, respectivamente. Durante la primera iteración, estas variables toman los valores 1,0, 2,0 y 3,0, respectivamente.

La iteración es una ejecución repetida de algunos cálculos, se utiliza para señalar que las líneas de programa que componen el cuerpo del operador del bucle son ejecutados.

En la línea siguiente, se calcula la diferencia entre los números de Fibonacci obtenidos sobre la base de los posteriores (C/B) y anteriores (B/A) de los elementos de la secuencia:

```
Delta=MathAbs(C/B - B/A);      // Buscar diferencia entre los coeficientes
```

En este operador, utilizamos la función estándar MathAbs () que calcula el valor absoluto de la expresión. Como mencionamos anteriormente, los coeficientes de Fibonacc toman, en turnos, grandes y pequeños valores (en comparación con el "estándar") con incrementos de los valores de la secuencia de elementos. Esta es la razón por la que la diferencia entre los coeficientes vecinos tendrá ahora un valor negativo, ahora un valor positivo. Al mismo tiempo, estamos realmente interesados en el valor de esta desviación, es decir, su valor absoluto. De este modo, cualquiera que sea la dirección que el valor actual del número de Fibonacci se desvie en el valor de expresión MathAbs (C/B - B/A), así como el valor de la variable de Delta, es siempre positivo.

Este operador es el último en la lista de operadores que componen el cuerpo del bucle. Esto se ve confirmado por la presencia de un cierre de llave en la línea siguiente. Tras la ejecución del cuerpo del último operador de ciclo , el control se pasa a la cabecera del operador para poner a prueba la condición. Este es el punto clave que determina la esencia del operador de ciclo . Según la condición del operador de ciclo sea verdadera o falsa, el control se pasará a cualquiera de la siguiente iteración o fuera del operador de ciclo.

En las primeras iteraciones, el valor de la variable Delta resulta ser mayor que el valor definido en la variable D. Esto significa que la condición (Delta> D) es cierto, por lo que el control se pasa al cuerpo del ciclo para realizar la siguiente iteración. Todas las variables involucradas en los cálculos llevará a nuevos valores: tan pronto como el cuerpo del ciclo llega el final, el control será pasado a la cabecera de nuevo para poner a prueba la condición y ver si esta es verdadera.

Este proceso continuará, hasta que la condición del operador de ciclo se convierte en falsa. Tan pronto como el valor de la variable se hace en Delta menor o igual al valor de D, la condición (Delta> D) ya no es cierta por más tiempo. Esto significa que el control será pasado fuera del operador de ciclo , a la línea:

```
 Alert ( "C =" C, "Fibonacci number =", C / B, "i =", i); // mostrar en la pantalla
```

Como resultado de ello, la alerta () operador ventana que contiene el siguiente mensaje aparecerá en la pantalla:

```
C = 317811 Fibonacci number = 1,618 i = 25
```

Esto significa que la búsqueda de precisión se alcanza en la 25 ª iteración, el valor máximo de la secuencia de Fibonacci elemento procesado en igualdad de condiciones a 317811. Coeficiente de Fibonacci, como era de esperar, es igual a 1,618. Este mensaje es la solución del problema planteado.

Cabe señalar aquí que los números reales se calculan en MQL4 con una precisión de 15 cifras significativas. Al mismo tiempo, el coeficiente de Fibonacci se muestra con una precisión de 3 cifras significativas. Esto se determina por el hecho de que la función de alerta () muestra números con una precisión de 4 cifras significativas, sin mostrar los ceros al final de la serie. Si quisiéramos mostrar el número de Fibonacci con una cierta precisión predefinidos, habría que cambiar el código, por ejemplo, de esta manera:

```
Alert("C=",C," Fibonacci number=",C/B," i=",i);  // mostrar en la pantalla
```

Lo siguiente debe ser especialmente observado:

Es posible que la condición especificada en el operador de ciclo de cabecera siempre sea cierta. Esto se traducirá en un bucle infinito.

Un bucle infinito o **Looping** es una ejecución repetida e infinita de las operaciones el ciclo del cuerpo de los operadores, es una situación crítica que se deriva de la realización de un algoritmo erróneo.

Una vez que el bucle se lleva a cabo, el programa ejecuta sin cesar el bloque de los operadores que componen el cuerpo del bucle. A continuación se muestra un simple ejemplo de un operador 'while' con bucle infinito.

```
int i = 1;                  // parámetro formal (contador)
while (i> 0)                // Operador de ciclo de cabecera
i + +;                      // Incremento del valor de i
```

En el ejemplo anterior, los valores de la variable i se incrementan en cada iteración. Como resultado de ello, la condición nunca se convertirá en falsa. Una situación similar ocurre, si no se calcula nada en el cuerpo del bucle. Por ejemplo, en el código que aparece a continuación:

```
int i = 1;                  // parámetro formal (contador)
while (i> 0)                // Operador de ciclo de cabecera
Alert ( "i =", i);          // mostrar en la pantalla
```

El valor de la variable i en el ciclo no cambia. El siguiente mensaje será mostrado en la pantalla en cada iteración:

```
i = 1
```

Este proceso se repetirá indefinidamente. Una vez que se cae en la trampa de un bucle infinito, el control no puede salir nunca del él. Esta situación es especialmente peligrosa en el comercio de Asesores Expertos y scripts. En tales casos, las variables de entorno no se actualizan normalmente, ya que la función especial no completa su funcionamiento, mientras que el trader puede desconocer la existencia de este bucle. Como resultado de ello, el control en la ejecución del programa no se puede pasar al correspondiente línea de programa de en las que se tome la decisión de apertura o cierre de órdenes.

El programador debe evitar tales condiciones, en las que se ejecuta en un bucle sin control posible. No hay ninguna técnica que ayuden a descubrir esta situación programática ni en la compilación del programa ni en su ejecución. El único método posible para detectar esos errores algorítmicos está en el examen del código, el razonamiento lógico y el sentido común.

Ciclo del operador for
Otro operador de ciclo es el operador 'for'.

Formato del explotador 'for'

El formato completo del operador de ciclo 'for' se compone de una cabecera que contiene Expression_1, la condición y Expression_2, y del cuerpo del bucle ejecutable adjunto entre llaves.

```
(Expression_1; Condición; Expression_2)   // Operador de ciclo de cabecera
{                                         // Apertura llave
Bloque de operadores                      // Ciclo del cuerpo puede consistir ..
que componen el ciclo del cuerpo          // ..  de varios operadores
}                                         // Cierre llave
```

Si el bucle del cuerpo en el operador 'for' consta de un solo operador, las llaves pueden omitirse.

```
(Expression_1; Condición; Expression_2) // Operador de ciclo de cabecera
Operador, ciclo del cuerpo              // el cuerpo del bucle es un solo operador
```

Expression_1, la condición y/o Expression_2 puede estar ausente. En cualquier caso, el carácter separador ";" (punto y coma) debe estar presente en el código.

```
(; Estado; Expression_2)                // Sin la Expression_1
{                                       // Apertura de llave
Bloque de operadores                    // Bucle del cuerpo puede consistir ..
que componen el cuerpo del bucle        // ..  de varios operadores
}                                       // Cierre de llave

// - ---------------------------------------------- ---------------------------------

(Expression_1;; Expression_2)           // Sin Condición
{                                       // Apertura de llave
Bloque de operadores                    // Cuerpo del bucle puede consistir ..
que componen el cuerpo del bucle        // ..  de varios operadores
}                                       // Cierre de llave
// - ---------------------------------------------- ---------------------------------

(;); // No expresiones o condición
{                                       // Apertura de llave
Bloque de operadores                    // Cuerpo del buclepuede consistir ..
que componen el cuerpo del bucle        // ..  de varios operadores
}                                       // Cierre de llave
```

Regla de Ejecución del operador 'for'

Tan pronto como el control pasa al operador 'for', el programa ejecuta Expression_1. En tanto que la condición del operador 'for' sea verdad: el control se pasa al primer operador en el cuerpo del bucle; tan pronto como se ejecutan todos los operadores en el cuerpo del bucle, el programa debe ejecutar la Expression_2 y pasar el control a la cabecera para probar la condición de ser verdad. Si la condición de que el operador 'for' es falso, entonces: el programa debe pasar el control al operador siguiente al operador 'for'.

Vamos a considerar cómo funciona el operador for. Para ello, vamos a resolver un problema.

Problema 13. Tenemos una secuencia de números enteros: 1 2 3 4 5 6 7 8 9 10 11 ... Redactar un programa que calcule la suma de los elementos de esta secuencia a desde N1 hasta N2.

Este problema es fácil de resolver en términos de matemáticas. Supongamos que queremos calcular la suma de elementos desde tercera hasta la séptima. La solución será: 3 + 4 + 5 + 6 + 7 = 25. Sin embargo, esta solución sólo es buena para un caso particular, cuando el número de los primeros y los últimos elementos que componen la suma es igual a 3 y 7, respectivamente. Un programa de la solución de este problema debe ser compuesto de tal manera que, si queremos calcular la suma en otro intervalo de la secuencia (por ejemplo, desde el 15 al 23 elemento), se podría sustituir fácilmente los valores numéricos de los elementos de una ubicación en el programa sin modificar las líneas de programa en otros lugares. A continuación se muestra una de las versiones de ese programa (script sumtotal.mq4):

```
//+------------------------------------------------------------------+
// sumtotal.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+
int start()                       // Special function start()
  {
//+------------------------------------------------------------------+
   int
   Nom_1,                                // Number of the first element
   Nom_2,                                // Number of the second element
   Sum,                                  // Sum of the numbers
   i;                             // Formal parameter (counter)
//+------------------------------------------------------------------+
   Nom_1=3;                              // Specify numeric value
   Nom_2=7;                              // Specify numeric value
   for(i=Nom_1; i<=Nom_2; i++)          // Cycle operator header
     {                            // Brace opening the cycle body
      Sum=Sum + i;                       // Sum is accumulated
      Alert("i=",i,"  Sum=",Sum);        // Display on the screen
     }                            // Brace closing the cycle body
//+------------------------------------------------------------------+
   Alert("After exiting the cycle, i=",i,"  Sum=",Sum);// Display on the screen
   return;                               // Exit start()
  }
//+------------------------------------------------------------------+
```

En las primeras líneas de su programa, se declaran las variables (los comentarios explican el sentido de cada variable). En las líneas

```
Nom_1 = 3;                    // Especifica el valor numérico
Nom_2 = 7;                    // Especifica el valor numérico
```

se definen los valores numéricos del primer y del último elemento de la gama. Tengase en cuenta que estos valores no se especifican en cualquier parte del programa. Si es necesario, puede cambiar fácilmente estos valores (en un solo lugar) sin cambiar el código en otras líneas, por lo que el problema se puede resolver en otro rango de valores. Después de que el último de estos operadores se ha ejecutado, el control se pasa al operador de ciclo 'for':

```
for (i=Nom_1; i<=Nom_2; i++)      // Cycle operator header
   {                              // Brace opening the cycle body
    Sum = Sum + i;                                  // Sum is accumulated
    Alert("i=",i,"  Sum=",Sum);   // Display on the screen
   }                              // Brace closing the cycle body
```

La frase clave para recordar la regla de ejecución del operador 'for' - es el siguiente: **"Desde i= .., siempre y cuando .., .. incrementado en pasos de..., hacer lo siguiente: ..".** El uso de "paso .." es aplicable, si queremos utilizar un contador de almacenamiento, por ejemplo, i++, i = i +1, como Expression_2. En nuestro ejemplo, la frase clave es la siguiente: Desde **i** igual a Nom_1, siempre y cuando i sea menor o igual a Nom_2, paso 1, haga lo siguiente: (y luego - el análisis del cuerpo del bucle).

De acuerdo con la norma de ejecución del operador 'for' lo siguiente se llevará a cabo en este bloque del programa:

1. Ejecución de Expression_1:

```
i=Nom_1                       // Expresión 1
```

La variable i tendrá el valor numérico de la variable Nom_1, es decir, entero 3. Expression_1 se ejecuta una sola vez - cuando el control se pasa al operador 'for'. La expression_1 no participan en ningún acontecimiento posterior.

2. La condición es la prueba:

```
i<=Nom_2                      // Condición
```

El valor de la variable Nom_2 en la primera iteración es el entero 7, mientras que el valor de la variable i es igual a 3. Esto significa que la condición es verdadera (desde el 3 hasta un valor inferior a 7), es decir, el control se pasa al cuerpo del bucle para ejecutarlo.

3. En nuestro ejemplo, el cuerpo del bucle se compone de dos agentes que serán ejecutados uno por uno:

```
Sum = Sum + i;                // Sum is incrementada
Alert("i=",i,"  Sum=",Sum);   // Muestra en la pantalla
```

Variable Suma, en su inicialización, no tomó ninguna valor inicial, por lo que su valor es igual a cero antes del comienzo de la primera iteración. Durante los cálculos, el valor de la variable suma se incrementa por el valor de i, por lo que es igual a 3 al final de la primera iteración. Usted puede ver este valor en la caja de la función de alerta ():

```
i = 3, Suma = 3
```

4. El último acontecimiento que ocurre durante la ejecución del operador de ciclo es la ejecución de Expression_2:

```
i++                           // Expresión 2
```

El valor de la variable i se incrementa en uno. Este es el final de la primera iteración, el control se pasa a la condición de prueba.

A continuación, la segunda iteración comienza los pasos 2-4. Durante los cálculos, las variables tienen nuevos valores. En particular, en la segunda iteración, el valor de la variable i es igual a 4, mientras que la suma es igual a 7. El mensaje correspondiente se publicará, también. En general, el programa se desarrollará de acuerdo con el diagrama de bloques siguiente:

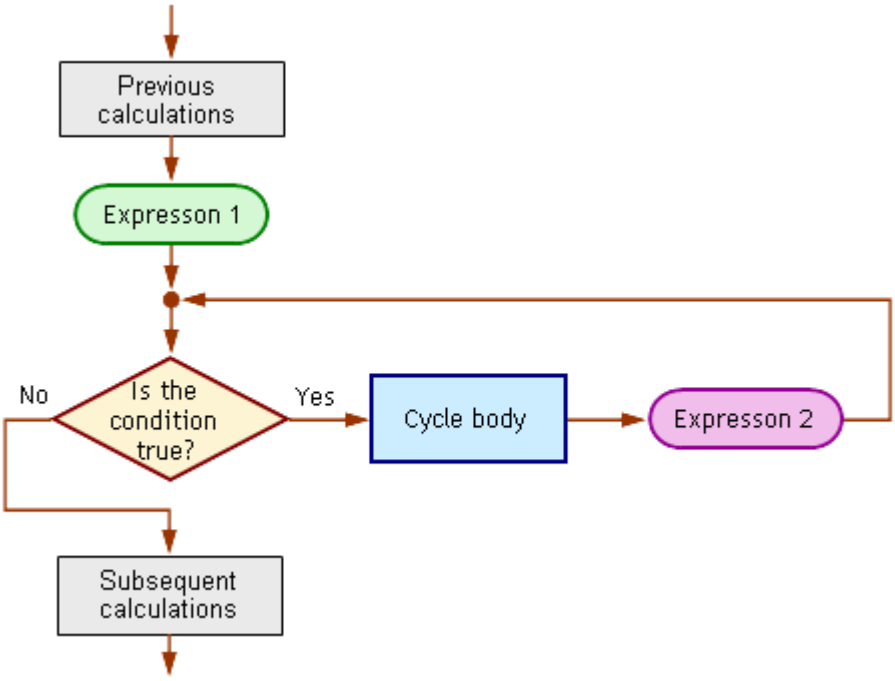


Fig. 43. Diagrama de bloques del operador-'for 'en la ejecución del programa sumtotal.mq4.

El ciclo de programa repite la ejecución del operador de ciclo, hasta que la condición se convierte en falsa. Esto sucederá tan pronto como el valor de la variable i es igual a 8, es decir, supera el valor preestablecido 7 de la variable Nom_2. En este caso, el control será pasado fuera el operador 'for', es decir, a la línea:

```
Alert ("i=",i,"  Sum=",Sum);          // Display on the screen
```

y, además, con la ejecución del operador «return» que termina la operación especial de la función start ().

Usted puede utilizar este programa para calcular la suma de cualquier otro rango de valores. Por ejemplo, si reemplaza las constantes 3 y 7 con 10 y 15, respectivamente, la ejecución del programa dará lugar a cálculo de la suma de los valores dentro de este rango.

En algunos lenguajes de programación, las variables que se especifican en el operador de ciclo de cabecera pierden sus valores después de salir del ciclo. Este no es el caso en MQL4. Todas las variables que intervienen en cualquier cálculo son válidas y mantienen sus valores después de salir del ciclo. Al mismo tiempo, hay algunas particularidades en relación con los valores de las variables que intervienen en los cálculos.

Por favor notese las dos últimos líneas los mensajes que siguen después de todos los cálculos están completos y el script se descarga:

```
i = 7 Suma = 25
Después de salir del ciclo, i = 8 Suma = 25
```

El primero de estos dos mensajes han aparecido en la etapa de la última iteración, antes de que el ciclo se completara, mientras que el segundo de ellos había sido dado por la última función alerta () antes de que el programa se diera por terminado. Es notable que estas líneas muestren diferentes valores de la variable i. Durante el ciclo de ejecución, este valor es igual a 7, pero es igual a 8 después de salir del ciclo. Con el fin de entender, ¿por qué ocurre esta manera, vamos a hacer referencia a la Fig. 43, una vez más.

La última (en este caso, la quinta) iteración comienza con las pruebas de la Condición. El valor de i es igual a 7 en este momento. Como el 7 no excede del valor de la variable Nom_2, la condición es verdadera y el cuerpo del buclese se lleva a cabo. Tan pronto como los agentes que componen el cuerpo del buclese se ejecutan, se realiza la última operación: ejecución de la Expression_2. Esto debe ponerse de relieve una vez más:

El último evento que tendrá lugar en cada iteración en la ejecución del operador 'for ' es el cálculo de Expression_2.

Esto se traduce en el aumento del valor de la variable i en 1, al final este valor será igual a 8. La posterior prueba de la condición (al principios de la próxima iteración) confirma que la condición es falsa. El resultado del condión lleva el control fuera del operador de ciclo. Al mismo tiempo, la variable i sale del operador de ciclo con el valor 8, mientras que el valor de la variable Suma sigue sin igual a 25, debido a que el cuerpo del bucle no se ejecuta ya en esta etapa. Esto debe considerarse en situaciones en las que el número de iteración no se conoce de antemano, de modo que el programador podrá estimar el valor de la variable calculada en Expression_2.

Podemos considerar un caso excepcional de una situación en la que en un bucle se podrían incurrir en diversos errores. Ejemplo de los errores si se utilizaran los siguientes operadores en la Expression_2 :

```
i--
```

El valor del contador se reducirá en cada iteración, por lo que la condición nunca se hará falsa.

Otro error puede ser una condición incorrectamente compuesta:

```
for(i=Nom_1; i>=Nom_1; i++)          // Cycle operator header
```

En este caso, el valor de la variable i será siempre superior o igual al valor de la variable Nom_1, con lo que la condición sería siempre cierta.

Intercambiabilidad entre los operadores 'White' y 'for'

El uso de un operador u otro depende totalmente de la decisión adoptada por el programador. La única cosa que queremos señalar es que estos operadores son intercambiables; a menudo es necesario sólo para dar otro cepillado al código de su programa.

Por ejemplo, hemos utilizado el operador de ciclo 'for' resolver Problema 13:

```
for (i=Nom_1; i<=Nom_2; i++)         // Cycle operator header
    {                                // Brace opening the cycle body
     Sum = Sum + i;                                  // Sum is accumulated
     Alert("i=",i,"  Sum=",Sum);     // Display on the screen
    }                                // Brace closing the cycle body
```

A continuación se muestra un ejemplo de cómo el mismo fragmento del código se puede ver si utilizamos el operador "while":

```
 i=Nom_1;                            // Оператор присваивания
  while (i<=Nom_2)                   // Cycle operator header
    {                          // Brace opening the cycle body
     Sum = Sum + i;                  // Sum is accumulated
     Alert("i=",i,"  Sum=",Sum);     // Display on the screen
     i++;                      // Increment of the element number
    }                          // Brace closing the cycle body
```

Como es fácil de ver, es suficiente sólo mover Expression_1 en la línea que precede al operador de ciclo, mientras que Expression_2 debe especificarse como el último ciclo en el cuerpo. Puede cambiar el ejemplo y su programa de lanzamiento para su ejecución, a fin de comprobar que los resultados son los mismos para ambas versiones.

El Operador "break"

En algunos casos, por ejemplo, en algún ciclo de programación de operaciones, podría ser necesario romper la ejecución de un ciclo antes de que su condición se convierta en falsa. Con el fin de resolver este problema podemos usar el operador "break".

Formato del operador "break"

El operador "break"se compone de una sola palabra y termina en el carácter ";" (punto y coma).

```
break;                    // Operador de "break"
```

Regla de Ejecución del operador "break"

El operador "break" detiene la ejecución de la parte más cercana del operador externo de tipo 'while',' for' o 'switch'. La ejecución del operador "break" consiste en pasar el control fuera del recinto del operador de tipo 'while',' for' o 'switch' al operador siguiente más cercano. El operador "break" solo se puede utilizar para la interrupción de la ejecución de los operadores mencionados anteriormente.

Podemos visualizar la ejecución del operador de 'break' con el siguiente ejemplo.

Problema 14. Tenemos un hilo de 1 metro de largo. Es necesario establecer el hilo en forma de un rectángulo con el área máxima posible. Se trata de hayar el área de este rectángulo y la longitud de los lados con una precisión de 1 mm en la búsqueda de serie en las variaciones.

No puede haber una cantidad ilimitada de rectángulos de diferentes tamaños hechos de un bloque unitario de hilo. Dado que la precisión requerida por la declaración del problema es de 1 mm, no podemos dejar de considerar las variaciones 999. El primero y el "más delgado" rectángulo tendrá las dimensiones de 1 mm x499, el segundo será de 2 mm x498, etc, mientras que las dimensiones del rectángulo último será 499 x1 mm. Tenemos que buscar en todos estos rectángulos y encontrar el de mayor superficie.

Como fácilmente se observa, hay dimensiones repetidas en el conjunto de rectángulos que estamos considerando. Por ejemplo, la primera y la última rectángulos tienen las mismas dimensiones: 1 x999 mm (al igual que 499 x1 mm). Del mismo modo, las dimensiones del rectángulo el segundo serán los mismos que los del pasado, sino un rectángulo, etc Tenemos que hacer un algoritmo de búsqueda que en todas las únicas variaciones, mientras que no hay necesidad de buscar en las reiteradas.

En primer lugar, vamos a calcular la relación entre la superficie y la longitud de los lados de un rectángulo. Como es fácil de ver, el primer rectángulo, con las dimensiones de 1x499, tiene la superficie más pequeña. Luego, con aumento de la cara más pequeña, el área del rectángulo, aumentará también. Tan pronto como se alcance un determinado valor, las áreas de los rectángulos comenzarán a disminuir de nuevo. Esta relación se muestra a continuación en la Fig. 44:

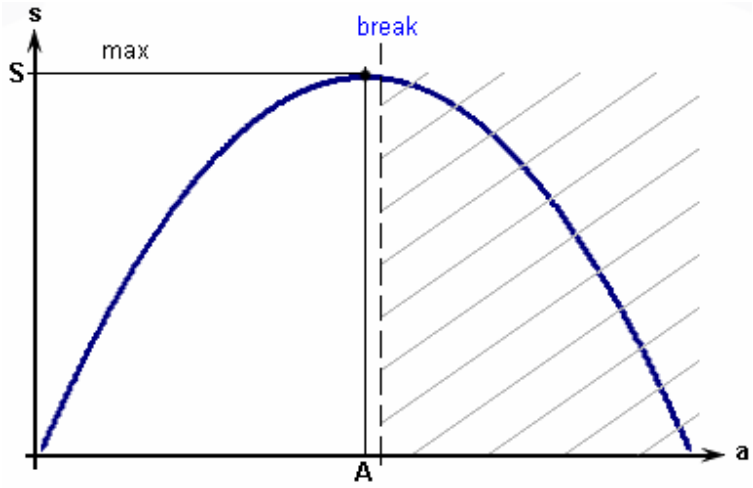


Fig. 44. Relación rectángulo entre la superficie y la longitud de uno de sus lados.

En la Fig. 44, podemos fácilmente llegar a la conclusión de que debemos encontrar la superficie máxima de la búsqueda en las variaciones a partir del primero de ellos, sólo mientras la zona aumenta durante los cálculos. Tan pronto como comienza a disminuir, vamos a romper nuestra búsqueda y salir del ciclo de búsqueda. A continuación se script rectangle.mq4 que implementa dicho algoritmo.

```
//------------------------------------------------------------------
// rectangle.mq4
// The code should be used for educational purpose only.
//------------------------------------------------------------------
int start()                             // Special function start()
  {
//------------------------------------------------------------------
   int
   L=1000,                      // Specified thread length
   A,                                   // First side of the rectangle
   B,                                   // Second side of the rectangle
   S,                                   // Area of the rectangle
   a,b,s;                               // Current values
//------------------------------------------------------------------
   for(a=1; a<L/2; a++)                 // Cycle operator header
     {                                  // Brace opening the cycle body
      b=(L/2) - a;              // Current value of the sides
      s=a * b;                  // Current value of the area
      if (s<=S)                 // Choose the larger value
        break;                  // Exit the cycle
      A=a;                      // Save the best value
      B=b;                      // Save the best value
      S=s;                              // Save the best value
     }                                  // Brace closing the cycle body
//------------------------------------------------------------------
   Alert("The maximum area = ",S,"  A=",A,"  B=",B);// Message
   return;                      // Function exiting operator
  }
//------------------------------------------------------------------
```

Veamos cómo funciona este programa. Las variables son declaradas y comentadas al principio del programa. El algoritmo de resolución del problema en sí se realiza dentro del ciclo 'for'. El valor inicial del lado **a** del rectángulo, como se especifica igual a 1 en Expression_1. De acuerdo con el Estado, los valores se buscan siempre y cuando el tamaño del lado **a** del rectángulo sigue siendo menor que la mitad de la longitud de hilo. La Expression_2 prescribe para aumentar la longitud del lado **a** del rectángulo actual en cada paso de iteración.

Las variables a, b y s son variables que registran los calculos actuales, los valores que se buscan estan en las variables A, B y S. El segundo lado **b** y la superficie **s** del rectángulo son calculados al comienzo del ciclo.

```
b = (L/2) - a;              // Current values of the sides
s = a * b;                  // Current value of the area
```

La condición de salir del ciclo se prueba en el operador 'if':

```
if (s <= S )                // Choose the larger value
break;                      // Exit the cycle
```

Si la recién calculada área **s** del rectángulo actual resulta ser mayor que el área de **S** calculada en la iteración anterior, este nuevo valor de **s** se convierte en el mejor resultado posible. En este caso, la condición del operador 'if' no se cumple, y el control se pasa al operador más cercano que sigue el operador 'if'. A continuación se presentan las líneas en las que se guardan los mejores resultados:

```
A = a;                      // Save the best value
B = b;                      // Save the best value
S = s;                      // Save the best value
```

Tan pronto como el programa llega a la llave de cierre, iteración está acabada y el control se pasa a la Expression_2 de la cabecera del operador 'for' para ejecutar y poner a prueba la condición. Si la longitud del lado **a** no ha alcanzado el límite determinado a partir del momento de la prueba, el ciclo seguirá siendo ejecutado.

Los repetidos cálculos cíclico seguirán hasta que uno de los siguientes eventos se lleva a cabo: o bien la longitud del lado **a** supera los límites predefinidos (de acuerdo a la condición del operador 'for'), o el tamaño del área calculada **s** se hace en inferior a un valor previamente almacenado en la variable **S.** Hay una buena razón para creer que la salida del bucle, de de acuerdo a la condición del operador 'if', se llevará a cabo antes:

```
if (s <= S )                // Choose the larger value
break;                      // Exit the cycle
```

De hecho, el operador de ciclo 'for' está compuesto de tal manera que se hará una búsqueda exhautiva en todas las variantes posibles (la mitad de la longitud del hilo, L/2, es la suma de dos lados del rectángulo). Al mismo tiempo, la máxima de superficie del rectángulo, se alcanzará en algún lugar en medio de las búsquedas del conjunto de las variantes. Entonces, tan pronto como esto ocurre (la zona del actual rectángulo **s** resulta ser igual o menor al valor alcanzado previamente **S),** el control en la ejecución del operador "if" se pasa al operador "break" que, a su vez, pasa el control fuera el operador 'for', a la siguiente línea:

```
Alert("The maximum area = ",S,"  A=",A,"  B=",B);// Message
```

Como resultado de la ejecución de la función estándar de alerta (), se imprimirá la siguiente línea:

```
The maximum area = 62500 A=250  B=250
```

Después de esto, el control se pasa al operador «return», lo que da lugar a la terminación de la función especial start (). Esto, a su vez, da lugar a la finalización de la escritura y a ser desvinculado por el Terminal de Usuario de la ventana de símbolo.

En este ejemplo, el operador de "break" (pasa el control fuera) el ciclo del operador 'for'. A continuación se muestra el diagrama de bloques del ciclo 'for' con la salida especial:

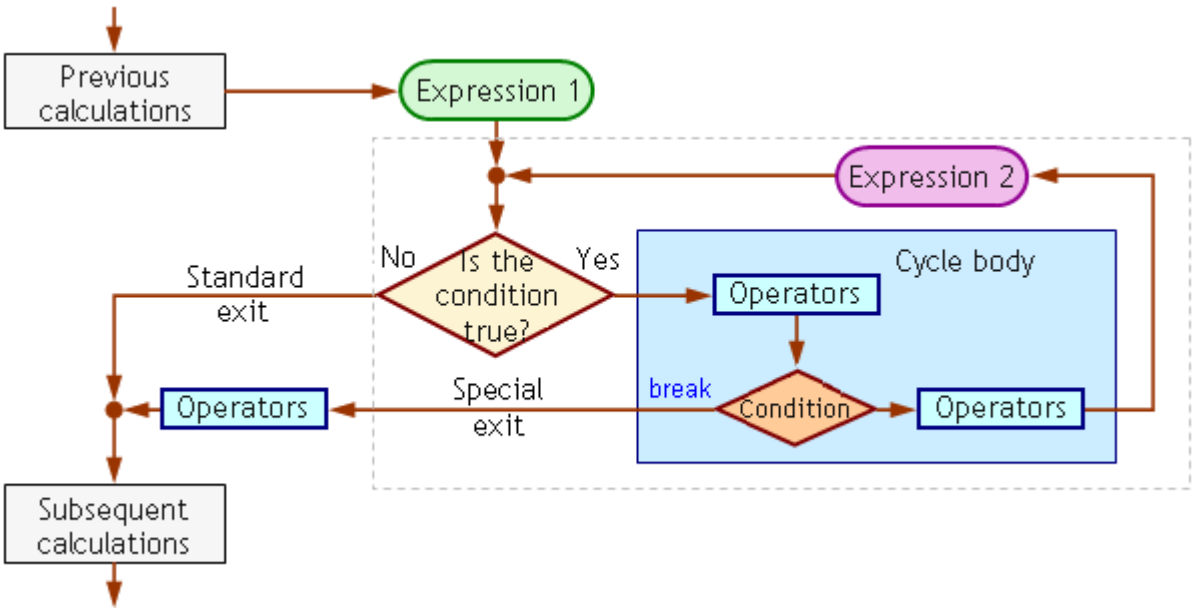


Fig. 45. Diagrama de bloque del ciclo 'for' usando el operador 'break' (rectangle.mq4).

Como se observa en el diagrama, hay dos salidas de ciclo: una salida normal (ciclo de salida resultantes de la activación de la condición especificada en el operador de ciclo de cabecera) y una salida especial (el cuerpo del bucle de salida de acuerdo con una condición adicional y utilizando el operador "break").

Es difícil sobrestimar la posibilidad de utilizar una salida a un ciclo. En nuestro ejemplo, el operador "break" nos permite hacer un algoritmo que realiza sólo los cálculos necesarios. Un algoritmo sin salida especial sería ineficiente: en este caso, se realizarían cálculos repetidos, lo que daría lugar a una pérdida de tiempo y de recursos computacionales poco razonables. La región "crosshatched" en la Fig. 44 visualiza la región de parámetros que no fueron tratados en el programa anterior (¡casi la mitad de todos los cálculos!), lo que no nos impide obtener el resultado correcto.

La necesidad de utilizar algoritmos eficientes es especialmente evidente, cuando el tiempo de ejecución se extiende a segundos e incluso minutos, lo que puede exceder el intervalo de tiempo entre los ticks. En este caso, existe el riesgo de omitir el tratamiento de algunas informaciones y ticks y como resultado, perder el control de su trading. Además, puede sentir la reducción del tiempo tomado por los cálculos especialmente si prueba sus programas en el Tester de Estrategia. La prueba de sofisticados programas de una larga historia puede tomar horas e incluso días. Reducir a la mitad el tiempo empleado por pruebas es una característica muy importante, en este caso.

La norma establece que el operador "break" se detiene al más cercano operador externo. Vamos a ver cómo funciona un programa que utiliza los ciclos anidados.

Problema 15. Utilizando el algoritmo del Problema 14, hayar la más corta búsqueda con múltiples hilos de 1 metro y suficientes para formar un rectángulo con una superficie de 1,5 m².

En este problema, debemos buscar en forma lineal en las longitudes hilo disponible y calcular la superficie máxima para cada longitud de hilo. La solución del problema 15 se realiza en el script llamado area.mq4. La soluciónes de las variantes se buscan en dos ciclos: internos y externos. El ciclo externo búsca en las longitudes de hilo con el paso de 1000 mm, mientras que el interior se encuentra la superficie máxima para la hilo longitud actual. En este caso, el operador de "break" se utiliza para la salida externa y la interna del ciclo.

```
//+------------------------------------------------------------------+
// area.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+
int start()                    // Special function start()
  {
//+------------------------------------------------------------------+
   int
   L,                          // Thread length
   S_etalon=1500000,                  // Predefined area (m²)
   S,                          // Area of the rectangle
   a,b,s;                      // Current side lengths and area
//+------------------------------------------------------------------+
   while(true)                        // External cycle for thread lengths
     {                         // Start of the external cycle
      L=L+1000;                       // Current thread length of в mm
      //+--------------------------------------------------------------+
      S=0;                     // Initial value..
      // ..for each dimension
      for(a=1; a<L/2; a++)           // Cycle operator header
        {                      // HStart of the internal cycle
         b=(L/2) - a;                 // Current side lengths
         s=a * b;              // Current area
         if (s<=S)                    // Choose the larger value
            break;             // Exit internal cycle
         S=s;                  // Store the best value
        }                      // End of the internal cycle
      //+--------------------------------------------------------------+
      if (S>=S_etalon)                 // Choose the larger value
        {
         Alert("The thread length must be ",L1000," m.");// Message
         break;                // Exit the external cycle
        }
     }                         // End of the external cycle
//+------------------------------------------------------------------+
   return;                     // Exit function operator
  }
//+------------------------------------------------------------------+
```

El ciclo interno trabaja aquí como lo hizo en la solución del problema anterior. El operador "break" se utiliza para salir del ciclo 'for', cuando la superficie máxima se encuentra dada por la longitud de hilo. Cabe señalar que el operador "break" se especifica en el ciclo interno 'for' pasa el control al operador que sigue la llave cerrada el ciclo 'for', y se detiene el ciclo de esta manera. Este fenómeno no influye en la ejecución del ciclo externo del operador »While« en modo alguno.

Tan pronto como el operador de "break" se dispara en el interior del ciclo 'for', el control se da al operador 'if':

```
      if (S >= S_etalon)                 // Choose the larger value
        {
         Alert("The thread length must be ",L1000," m.");// Message
         break;                // Exit the external cycle
        }
```

En el operador 'if' que comprueba si la superficie es igual o mayor de 1,5 m², el valor mínimo permitido por la declaración del problema. Si la respuesta es afirmativa, entonces la solución se encuentra y no tiene sentido que siga el cálculo; el control será pasado al cuerpo del operador 'if'. La parte ejecutable del operador 'if' se compone solo de dos operadores, el primero de los cuales muestra el mensaje en pantalla de la solución encontrada:

```
El hilo debe ser de una longitud de 5 m.
```

El segundo operador, "break", se utiliza para salir del ciclo externo 'while' y, posteriormente, para salir del programa. El diagrama de bloques del algoritmo realizado en el programa area.mq4 se muestra a continuación:

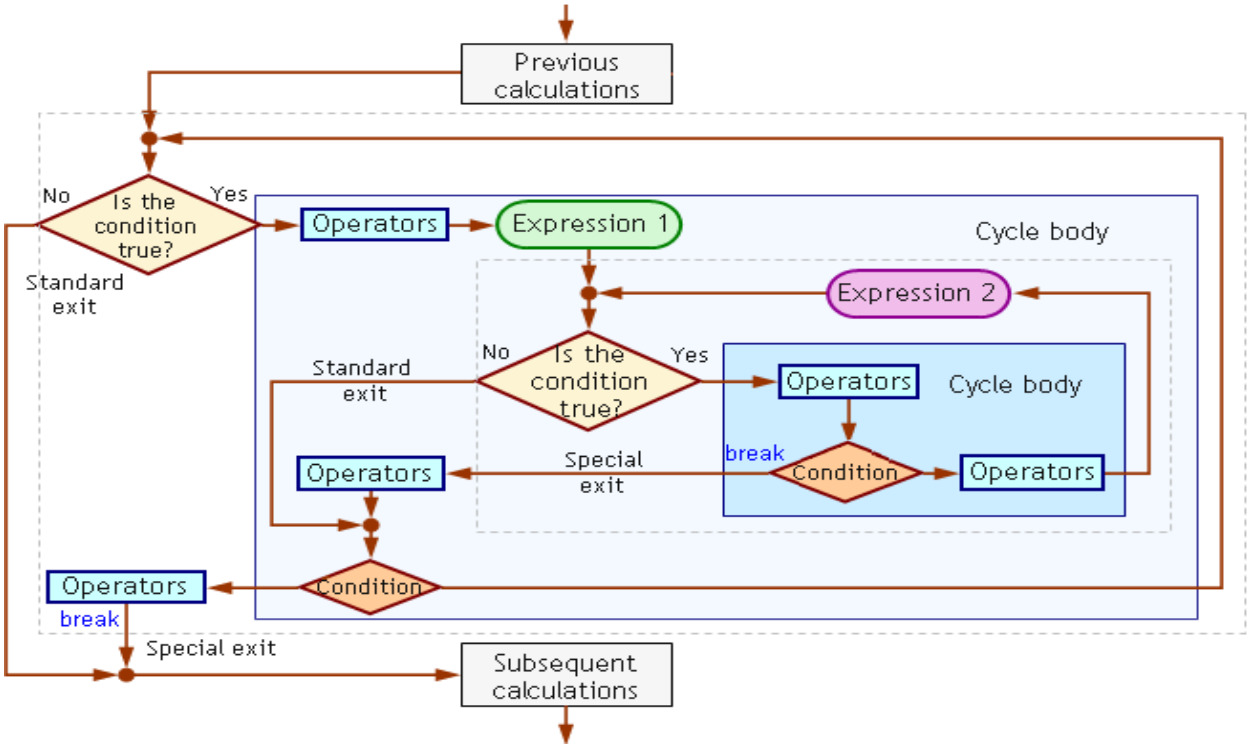


Fig. 46. Diagrama de bloques del programa con la posibilidad de salida especial en los ciclos interno y externo (area.mq4).

Como se observa en el diagrama, hay una salida normal y una salida especial para cada ciclo. Cada operador "break" pone fin a su ciclo correspondiente, pero no tiene efecto alguno sobre el otro ciclo; cada salida especial se corresponde con su propio operador de "break". El mismo operador "break" no se puede utilizar en ambos ciclos especiales como salida. En este caso, cada ciclo tiene sólo una salida especial. Sin embargo, en general, es posible utilizar varios operadores "break" para hacer varias salidas en especiales de un ciclo.

Tenga en cuenta que la condición en la cabecera del ciclo exterior 'while' es (cierto), es decir, un texto que no contiene una variable, el valor de lo que debería cambiar durante la ejecución del programa. Esto significa que el programa nunca podría salir del ciclo 'While' bajo la condición especificada en la cabecera. En este caso, la única posibilidad para salir del ciclo es usar el operador "break".

En este ejemplo, el algoritmo se construye de tal manera que, de hecho, el programa utiliza sólo la salida especial para salir del ciclo interno y externo. Sin embargo, usted no debe pensar que el uso del operador "break" siempre se traduce en la construcción de algoritmos que sólo implica salidas especiales. En los algoritmos de otros programas, es muy posible que el programa llegue a la condición y los usos normales de salida para salir de sus ciclos.

Cómo usar el operador "break" para pasar el control fuera el operador 'switch' se considera en la sección del operador "switch".

Operador "continue"

A veces, cuando se utiliza un ciclo en el código, es necesario poner fin al principio del tratamiento de la iteración actual y pasar a la siguiente sin ejecutar el resto de los operadores que componen el cuerpo del bucle. En estos casos se debe usar el operador "continuar".

Formato del operador "continue"

El operador 'Continue' consta de una sola palabra y termina en ";" (punto y coma).

```
continue; // operador "continue"
```

Ejecución Regla del operador "continue"

El operador "continue" detiene la ejecución de la iteración actual del ciclo más cercano del operador 'while' o' for'. La ejecución del operador "continue" da como resultado ir a la próxima iteración del ciclo más cercano operador 'while' o 'for'. El operador "continue" sólo se puede utilizar en el cuerpo del ciclo por encima de los operadores.

Vamos a examinar la forma en que el operador "continue" puede ser utilizado prácticamente.

Problema 16. Hay 1000 ovejas en la primera granja. La cantidad de ovejas en la primera granja aumenta en un 1% diario. Si la cantidad de ovejas es superior a 50.000 a finales de mes, entonces el 10% de las ovejas serán transferidos a la segunda granja. ¿En qué momento el importe de ovejas en la segunda granja llega a 35.000? (Consideramos que hay 30 días hábiles en un mes.)

El algoritmo de la solución de este problema es evidente: Tenemos que organizar un ciclo en el que el programa calcule la cantidad total de ovejas en la primera granja. De acuerdo con el planteamiento del problema, las ovejas se transfieren a la segunda granja a finales de mes. Esto significa que tendremos que crear un ciclo más interno donde se calcule el ciclo de acumulación de ovejas en el mes en curso. Entonces tendremos que comprobar a finales de mes, si el límite de 50.000 ovejas se supera o no. Si la respuesta es sí, entonces hay que calcular la cantidad de ovejas a ser transferidos a la segunda granja a finales de mes y la cantidad total de ovejas en la segunda granja.

El algoritmo en estudio se realiza en el script sheep.mq4. El operador "continue" se usa aquí para hacer cálculos en el ciclo externo.

```
//+------------------------------------------------------------------+
// sheep.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+
int start()                          // Special function start()
  {
//+------------------------------------------------------------------+
   int
   day,                                          // Current day of the month
   Mons;                                         // Search amount of months
   double
   One_Farm    =1000.0,                 // Sheep on the 1st farm
   Perc_day    =1,                               // Daily increase, in %
   One_Farm_max=50000.0,                         // Sheep limit
   Perc_exit   =10,                              // Monthly output, in %
   Purpose     =35000.0,                // Required amount on farm 2
   Two_Farm;                                     // Current amount of farm 2
//+------------------------------------------------------------------+
   while(Two_Farm < Purpose)            // External cycle on history
     {                                  // Start of the external cycle body
      //+---------------------------------------------------------------+
      for(day=1; day<=30; day++)                 // Cycle for days of month
        One_Farm=One_Farm*(1+Perc_day/100);//Accumulation on the 1st farm
      //+---------------------------------------------------------------+
      Mons++;                                    // Count months
      if (One_Farm < One_Farm_max)               // If the amount is below limit,.
        continue;                                // .. don't transfer the sheep
      Two_Farm=Two_Farm+One_Farm*Perc_exit/100;//Sheep on the 2nd farm
      One_Farm=One_Farm*(1-Perc_exit/100);// Remainder on the 1st farm
     }                                  // End of the external cycle body
//+------------------------------------------------------------------+
   Alert("The aim will be attained within ",Mons," months.");//Display on the screen
   return;                                       // Exit function start()
  }
//+------------------------------------------------------------------+
```

Por lo general las variables se describen y comentan al comienzo del programa. El mismo cálculo se realiza en el ciclo 'while'. Después de que ha sido ejecutado, el mensaje correspondiente en la pantalla. Se ejecutan los cálculos en el ciclo exterior 'while' será la ejecutado, hasta que el objetivo es alcanzado, es decir, hasta que la cantidad total de ovejas en la segunda granja alcanza el valor esperado de 35.000.

El ciclo interno 'for' es muy simple: el valor del saldo diario aumenta en un 1%. El análisis de la suma no se realiza en este ciclo, ya que, según el planteamiento del problema, las ovejas sólo pueden ser transferidas a finales de mes. Así, después de salir del ciclo 'for', la variable One_Farm tiene un valor que es igual a la cantidad de ovejas de la primera granja. Inmediatamente después de que el valor de la variable Mons se calcula, el cual se incrementa en 1 en la ejecución de cada iteración del ciclo externo 'while'.

De acuerdo con la actual cantidad de ovejas en la primera granja, una de las dos acciones más adelante se llevarán a cabo:

- si la cantidad de ovejas en la primera granja supera el valor límite de 50 000, entonces el 10% de ovejas de la primera granja deben ser transferidos a la segunda granja;
- de otro modo, las ovejas de la primera granja estan en la primera granja y se crían más.

El algoritmo se ramifica usando el operador 'if':

```
if (One_Farm < One_Farm_max)       // If the amount is below limit,.
continue;                          // .. don't transfer the sheep
```

Estas líneas de código pueden caracterizarse de la siguiente manera: Si la cantidad de ovejas en la primera granja está por debajo del valor límite de 50 000, a continuación, ejecutar el operador que componen el cuerpo del operador 'if'. En la primera iteración, la cantidad de ovejas de la primera granja resulta ser inferior al valor límite, por lo que el control se pasa al cuerpo del operador 'if', es decir, actua el operador 'continue'. La ejecución del operador 'continue' significa lo siguiente: Finaliza la actual iteración del ciclo de ejecución más cercano (en este caso, es el ciclo en que el operador 'while') y pasa el control a la cabecera del operador de ciclo. Las siguientes líneas programa que también son una parte del cuerpo ciclo del operador'while no se ejecutará:

```
Two_Farm = Two_Farm+One_Farm*Perc_exit/100;//Sheep on the 2nd farm
One_Farm = One_Farm*(1-Perc_exit/100);     // Remainder on the 1st farm
```

En estas líneas de arriba se hace efectiva la transferencia del 10% de ovejas de la primera granja a la segunda y se calcula el ganado ovino que queda en la primera granja. Es imposible hacer estos cálculos en la primera iteración, ya que el límite de 50 000 ovejas en la primera granja no se ha llegado aún. La esencia del operador en "continue" consiste en saltarse estas líneas sin terminar de ejecutar la iteración actual.

La ejecución del operador "continue" consiste en pasar el control a la cabecera del operador de ciclo 'while'. Entonces la condición en el operador de ciclo y se prueba y la siguiente iteración comienza. El ciclo seguirá siendo ejecutado de acuerdo con el mismo algoritmo, hasta que la cantidad de ovejas en la primera granja llega al límite fijado. Los valores de la variable Mons se acumulan en cada iteración.

En una cierta fase de cálculo, la cantidad de ovejas de la primera granja alcanzará o superará el límite fijado de 50 000 cabezas. En este caso, la ejecución de la condición del operador 'if', (One_Farm <One_Farm_max) se convierte en falsa, por lo que el control no se pasa al cuerpo del operador 'if'. Esto significa que el operador 'Continue' no será ejecutado y el control será pasado a la primera de las dos últimas líneas del ciclo del cuerpo 'while´: En primer lugar, el programa calculará la cantidad de ovejas de la segunda granja y, a continuación, la cantidad de ovejas que quedan en la primera granja. Después de que estos cálculos se han completado, la iteración del ciclo 'while' termina y el control vuelve de nuevo a la cabecera del ciclo. A continuación se muestra el diagrama de bloques del algoritmo realizado en script sheep.mq4.

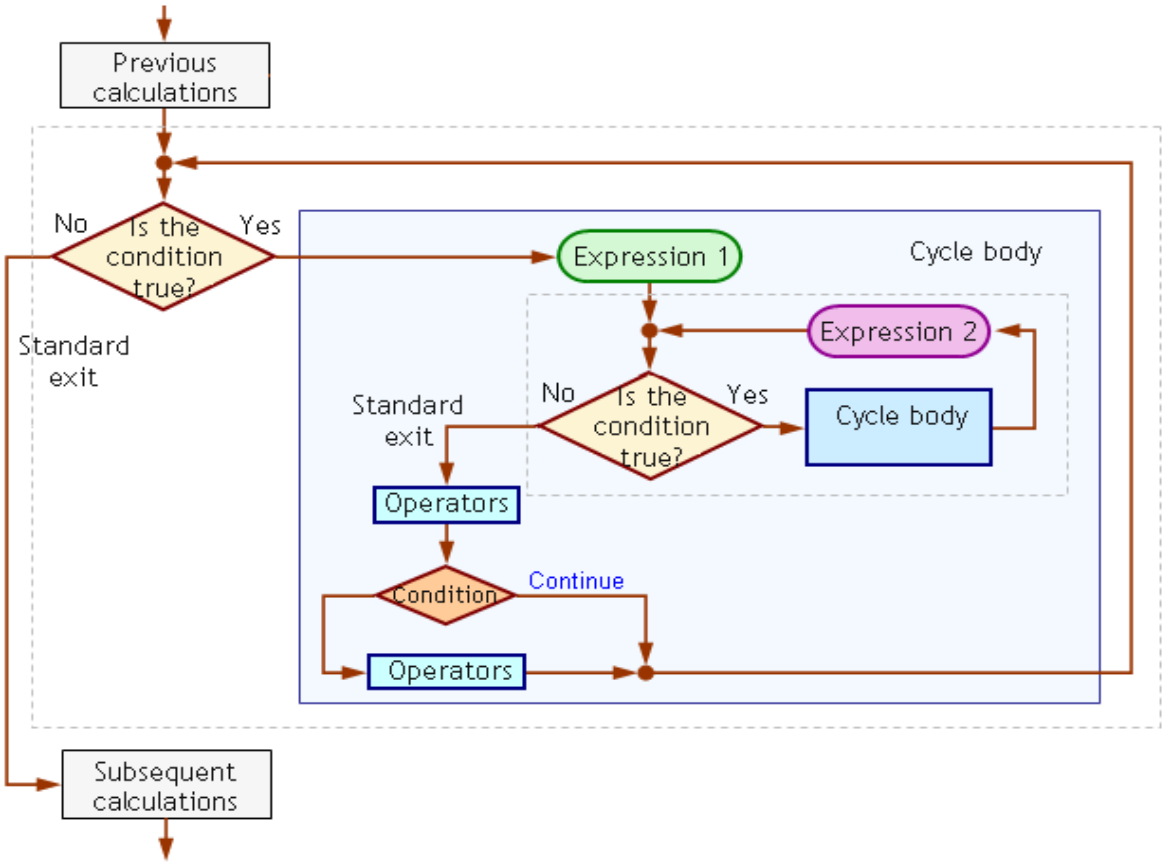


Fig. 47. Diagrama de bloques de un programa donde el operador "continue" rompe la iteraciones del ciclo externo (sheep.mq4).

En el diagrama, podemos ver que, dependiendo de la ejecución de la condición de operador 'if', epasl control será pasado de inmediato a la cabecera del ciclo 'while' (como resultado de la ejecución del operador "continue"), o se ejecutan algunos operadores mas primero, y luego el control se sigue pasando a la cabecera del ciclo 'while'. En ninguno de los dos caso termina la ejecución del ciclo.

Tengase en cuenta que la presencia del operador "continue" en el cuerpo del ciclo no provoca que necesariamente la finalice la iteración actual y el control se vaya directamente a la cabecera del operador de ciclo while. Esto sólo ocurre si el operador continue se ejecuta, es decir, si el operador if previo le pasa el control.

En las sucesivas iteraciones en el operador de ciclo 'while', se calcula el nuevo valor alcanzado por la variable Two_Farm. Tan pronto como este valor supera el límite fijado de 35 000, se cumple el requisito en el que el operador de ciclo 'while' se converte en falso, por lo que los cálculos en el cuerpo del operador de ciclo while no se ejecutarán mas y el control pasará al operador más cercano que sigue al operador de ciclo while:

```
Alert("The aim will be attained within ",Mons," months.");//Display on the screen
[Alert ( "El objetivo será alcanzado dentro", Mons, "meses".); // Mostrar en la pantalla]
```

La ejecución de la función de Alert () tendrá como resultado que se muestre la siguiente línea:

```
The aim will be attained within 17 months. (El objetivo será alcanzado dentro de 17 meses).
```

El operador «return» completa la ejecución de la función especial start (), lo que se traduce en que el control se pasa al Terminal de Usuario, mientras que la ejecución del script termina y se desvinculará de la ventana de símbolo.

El algoritmo anterior también proporciona un estándar de salida del ciclo 'while'. En un caso más general, la salida del ciclo puede ser el resultado de la ejecución del operador "break" (salida especial). Sin embargo, ni una ni otra manera de cerrar el ciclo se relacionan con la interrupción de la actual iteración usando el operador 'continue'.

En el estado de ejecución del operador "continue" se utiliza la frase de "el operador más cercano". Esto no significa que las lineas de programa se encuentren cerca una de la otra. Vamos a echar un vistazo al código de script sheep.mq4. El operador 'for' es el "más cercano" para el operador 'continue' que la cabecera del ciclo 'while'. Sin embargo, esto no significa nada: El operador 'continuae' no tiene ninguna relación con el ciclo 'for', porque esta fuera de su cuerpo. En general, un programa puede contener múltiples ciclo anidados de operadores. El operador "continue" está siempre en el cuerpo de uno de ellos. Al mismo tiempo, el operador "continue", forma parte del ciclo interno de operador, por supuesto, también dentro del ciclo externo del operador. La frase de "detiene la ejecución de la iteración actual del operador de ciclo más cercano" debe significar que el operador 'Continue' influye en el ciclo del operador mas cercano dentro del cuerpo del que el operador 'continue' se encuentra. Con el fin de visualizar esto, vamos a cambiar ligeramente la declaración del problema.

Problema 17. Hay 1000 ovejas en una granja. La cantidad de ovejas en esta primera granja aumenta diariamente en un 1%. al día, cuando la cantidad de ovejas en la primera granja llega a 50.000, el 10% de las ovejas serán transferidos a la segunda granja. ¿En qué momento la cantidad de ovejas de la segunda granja llega a 35 000? (Nosotros consideramos que hay 30 días hábiles en un mes.)

La solución del problema 17 se realiza en el script othersheep.mq4. En este caso, el operador "continue" se utiliza para el cálculo tanto en el exterior e interior ciclos.

```
//+------------------------------------------------------------------+
// othersheep.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+
int start()                                    // Special function start()
  {
//+------------------------------------------------------------------+
   int
   day,                                        // Current day of the month
   Mons;                                       // Search amount of months
   double
   One_Farm    =1000.0,                // Sheep on the 1st farm
   Perc_day    =1,                             // Daily increase, in %
   One_Farm_max=50000.0,                       // Sheep limit
   Perc_exit   =10,                            // One-time output, in %
   Purpose     =35000.0,               // Required amount on farm 2
   Two_Farm;                                   // Current amount of farm 2
//+------------------------------------------------------------------+
   while(Two_Farm < Purpose)           // Until the aim is attained
     {                                 // Start of the external cycle body
      //+--------------------------------------------------------+
      for(day=1; day<=30 && Two_Farm < Purpose; day++)// Cycle by days
        {
         One_Farm=One_Farm*(1+Perc_day/100);          //Accumulated on farm 1
         if (One_Farm < One_Farm_max)                       // If the amount is below limit,.
           continue;                                        // .. don't transfer the sheep
         Two_Farm=Two_Farm+One_Farm*Perc_exit/100;    //Accumulated on farm 2
         One_Farm=One_Farm*(1-Perc_exit/100);         //Remainder on farm 1
        }
      //+--------------------------------------------------------+
      if (Two_Farm>=Purpose)                        // If the aim is attained,..
        continue;                                           // .. don't count months
```

```
     Mons++;                                                  // Count months
    }                                                  // End of external cycle body
//---------------------------------------------------------------------
   Alert ("The aim will be attained within ",Mons," months and ",day," days.");
   return;                                                  // Exit function start()
  }
//---------------------------------------------------------------------
```

Para resolver este problema, tenemos que analizar la cantidad de ovejas para cada día. Con este fin, los siguientes:

```
   if (One_Farm < One_Farm_max)           // If the amount is below limit,.
     continue;                        // .. don't transfer the sheep
   Two_Farm=Two_Farm+One_Farm*Perc_exit/100;//Accumulated on farm 2
   One_Farm=One_Farm*(1-Perc_exit/100);     //Remainder on farm 1
```

se transfieren en el interior del ciclo organizado para los días del mes. Es evidente que una cierta cantidad de ovejas se transfieren a la segunda granja, en este caso, no a finales de mes, sino en el día, cuando la cantidad de ovejas en la primera granja llega al valor predefinido. La razón para usar aquí el operador "continue" es la misma: Queremos **saltar** las líneas que contienen los cálculos relacionados con la transferencia de ovejas de una granja a otra, es decir, no queremos ejecutar estas líneas. Tengase en cuenta que el ciclo 'while' que se construyó para los días del mes no se interrumpe y actua de forma independiente a si las ovejas son transferidas o no, porque el operador "continue" influye en el operador de ciclo del más cercano, en nuestro caso, el operador de ciclo 'for'.

En el operador se utilizó una expresión compleja para la condición del operador de ciclo 'for':

```
for(day=1; day<=30 && Two_Farm<Purpose; day++)// Cycle by days
```

El uso de la operación && ("and") significa que el ciclo se ejecutará siempre y cuando ambas condiciones sean verdaderas:

- el mes no ha terminado todavía, es decir, la variable «día» oscila entre 1 a 30, y
- la cantidad de ovejas en la segunda granja no ha alcanzado el valor predefinido y sin embargo la variable Two_Farm es inferior a Objeto (35 000).

Si al menos una de las condiciones no se cumple (la condicionse convierte en falsa), el ciclo se detiene. Si la ejecución del ciclo 'for' se detiene (por cualquier motivo), el control pasa al operador 'if':

```
 if (Two_Farm >= Purpose)                  // If the aim is attained,..
 continue;                                 // .. don't count months
 Mons++;                              // Count months
```

El uso del operador "continue" en esta ubicación en el código tiene un sentido simple: El programa debe seguir contando meses solamente si la cantidad de ovejas en la segunda granja está por debajo del valor esperado. Si el objetivo ya ha sido alcanzado, el valor de la variable Mons no cambiará, mientras que el control (como resultado de la ejecución de "continue") se pasa a la cabecera del operador de ciclo más cercano, en nuestro caso, el del operador de ciclo 'while'.

El uso de los operadores "continue" en los ciclos anidados se muestra en la Fig. 48 infra.

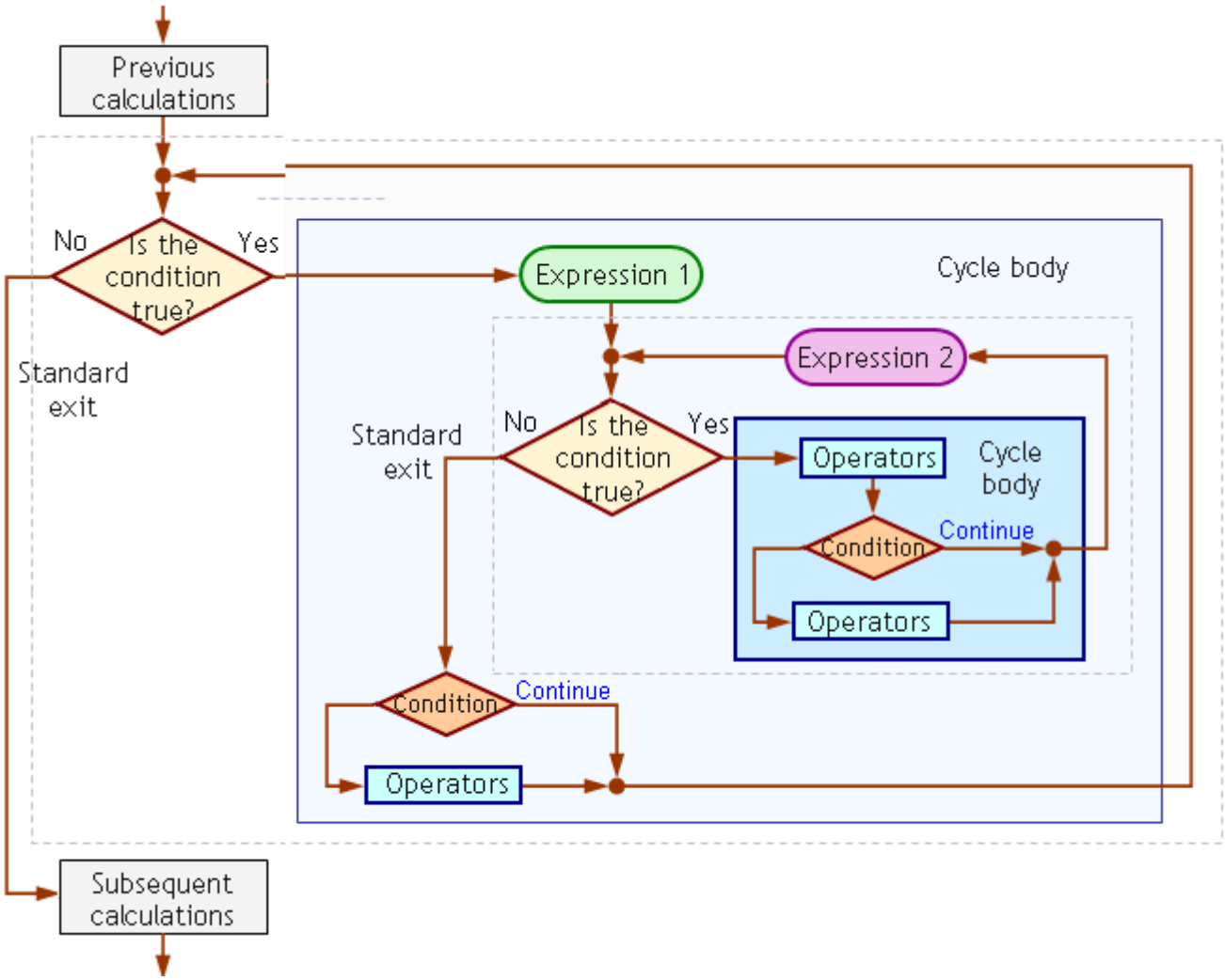


Fig. 48. Diagrama de bloques del algoritmo basado en los ciclos anidados. Cada ciclo contiene su cuerpo operador "continue" (othersheep.mq4).

En el ejemplo anterior, cada ciclo (externo e interno) tiene un operador en "continue" que sólo influye en su "propio" ciclo más cercano (en el ciclo dentro del cual está actuando), pero no por cualquier acontecimiento ó en cualquier otro ciclo. En general, en un cuerpo del bucle se pueden utilizar varios operadores "continue" y cada uno de ellos ser capaz de interrumpir la iteración actual como resultado de la reunión de una condición. La cantidad de operadores "continue" en un cuerpo de un bucle no esta limitada.

Operador 'switch'

Algunos programas conllevan la necesidad de la ramificación de su algoritmo en variantes. En estos casos, es muy conveniente usar el operador "switch", sobre todo si hay decenas o cientos de variaciones, mientras que, en este caso, el operador ‘if’ se conviertiría en un código hinchado si utiliza muchos operadores anidados.

Formato del operador 'switch'

El operador 'switch' consiste en una cabecera y un cuerpo ejecutable. La cabecera contiene el nombre del operador y una expresión entre paréntesis. El cuerpo del operador contiene una o varias alternativas o casos ('case') y una variante «por defecto».

Cada variante "case" consiste en la de la palabra clave 'caso', una constante, ":" (dos puntos), y los operadores. La cantidad de variantes 'caso' no está limitada.

La variación 'por defecto' se compone de la palabra clave 'por defecto', ":" (dos puntos), y los operadores. La variante 'por defecto' se especifica la última en el cuerpo del operador 'switch', pero también puede ser ubicado en cualquier lugar dentro del cuerpo operador, o incluso estar ausente.

```
switch (Expresión)                  // Operador de cabecera
{                                   // Apertura de llave
case Constante: Operadores          // Una de las variantes 'caso'
case Constante: Operadores          // Una de las variantes 'caso'
...
[Default: Operadores]               // Variante sin ningún tipo de parámetro
}                                   // Cierre de llave
```

Los valores de la expresión y de los parámetros sólo pueden ser los valores de tipo ***int***. La expresión puede ser una constante, una variable, una llamada a una función, o una expresión. Cada variante "caso" puede ser marcado por un entero constante, un carácter constante, una constante o expresión. Una expresión constante no puede incluir variables o llamadas a funciones.

Regla de Ejecución del operador 'switch'

El programa debe pasar el control al primer operador que sigue a ":" (dos puntos) de la variante "case" cuyo valor la constante es el mismo que el valor de la expresión y, a continuación, ejecutar uno por uno todos los operadores que componen el cuerpo del operador 'switch'. La condición de igualdad entre la expresión y la prueba es constante en la dirección de arriba hacia abajo y de izquierda a derecha. Los valores de las variantes constantes de los diferentes 'case' no debe ser el mismo. El operador “break” detiene la ejecución del operador 'switch' y pasa el control al operador que sigue el operador 'switch'.

Es fácil ver que la constante de **`case´**: "representa sólo una etiqueta, para que el control sea pasado. Todos los operadores que componen el cuerpo del operador 'switch' empezarán a ejecutarse a partir de esta etiqueta. Si el algoritmo del programa implica la ejecución de un grupo de operadores que se corresponden con una sola variante "caso", se debe especificar el operador “break” como el último en la lista de operadores que se corresponden con una sola variante "caso". Vamos a considerar el trabajo del operador 'switch' a través de algunos ejemplos.

Ejemplos de operador 'switch'

Problema 18. Redactar un programa con las siguientes condiciones: Si el precio excede del nivel predefinido, el programa deberá informar al comerciante sobre ello mediante un mensaje que describe el exceso (hasta 10 puntos); en los demás casos, el programa debe

informar que el precio no excede el nivel predefinido.

A continuación se muestra la solución de problemas usando el operador 'switch' (Asesor Experto pricealert.mq4):

```
//+------------------------------------------------------------------+
// pricealert.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+
int start()                                          // Special function 'start'
  {
   double Level=1.3200;                              // Preset price level
   int Delta=NormalizeDouble((Bid-Level)Point,0);    // Excess
   if (Delta<=0)                                  // Price doesn't excess the level
     {
      Alert("The price is below the level");         // Message
      return;                                    // Exit start()
     }
//+------------------------------------------------------------------+
   switch(Delta)                                  // Header of the 'switch'
     {                                                // Start of the 'switch' body
      case 1 : Alert("Plus one point");     break;    // Variations..
      case 2 : Alert("Plus two points");    break;
      case 3 : Alert("Plus three points");  break;
      case 4 : Alert("Plus four points");   break;    //Here are presented
      case 5 : Alert("Plus five points");   break;    //10 variations 'case',
      case 6 : Alert("Plus six points");    break;    //but, in general case,
      case 7 : Alert("Plus seven points"); break;    //the amount of variations 'case'
      case 8 : Alert("Plus eight points");  break;    //is unlimited
      case 9 : Alert("Plus nine points");   break;
      case 10: Alert("Plus ten points");    break;
      default: Alert("More than ten points");       // It is not the same as the 'case'
     }                                                // End of the 'switch' body
//+------------------------------------------------------------------+
   return;                                        // Exit start()
   }
//+------------------------------------------------------------------+
```

En solución de este problema, se utiliza el operador 'switch', en el que cada variante "case" utiliza el operador "break". Dependiendo del valor de la variable Delta, el control se pasa a una de las variantes 'case'. Esto se traduce en la ejecución de los operadores correspondientes a esta variante: la función de alert () y el operador de "break". El operador "break" detiene la ejecución del operador 'switch', y pasa el control **fuera** de ella, es decir, al operador «return» que termina la operación de la función especial start (). Así, según sea el valor de la variable de Delta, una de las variantes "caso" se activa, mientras que otras variantes siguen intactas.

El programa anterior es un Asesor Experto, por lo que éste se pondrá en marcha en cada tick y en cada uno de ellos mostrará el mensaje correspondiente con la situación actual. Por supuesto, debemos buscar un valor para el Nivel lo más cerca posible del precio actual a la ventana del símbolo a la que está vinculado este EA.

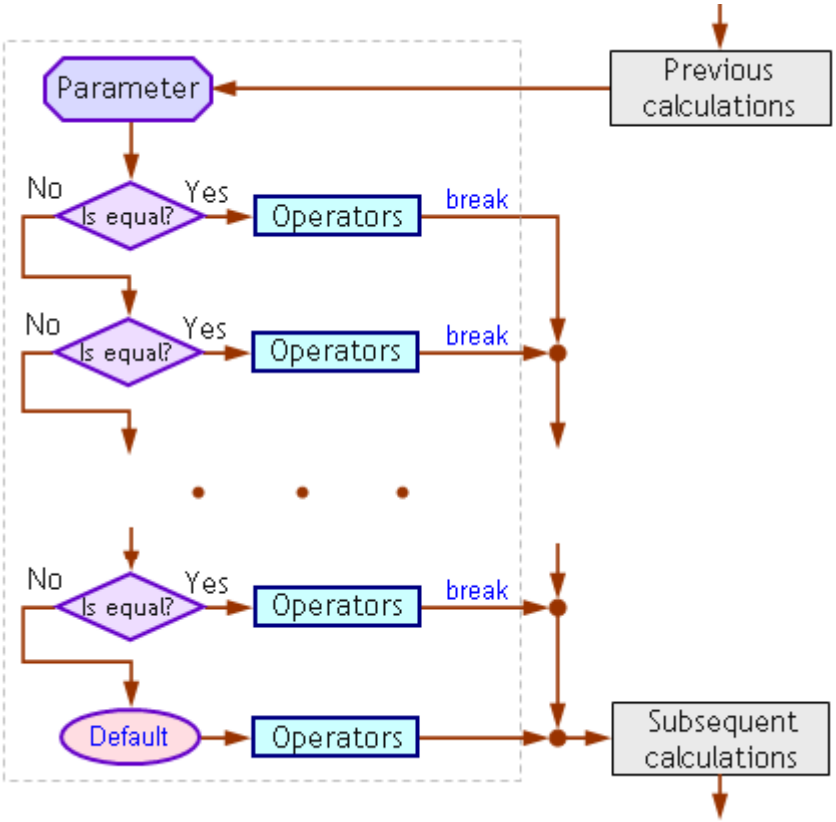


Fig. 49. Diagrama de bloques del operador 'switch' en AE pricealert.mq4.

En el diagrama de bloques anterior, podemos ver claramente que, debido a la presencia del operador “break” en cada variante "caso", el control se pasa fuera del operador 'swich' después de la ejecución de los operadores de cualquier variante "caso". Un principio similar de construcción de algoritmo usando el operador 'switch' se utiliza en el archivo llamado stdlib.mq4 emitido dentro de la Terminal de Usuario ***(.. \ expertos \ librerías \ stdlib.mq4).***

Vamos a considerar otro problema que no prevé el uso de “break” en cada una variante "caso".

Problema 19. Hay 10 barras. Informe sobre los números de todas las barras a partir de la enesima barra.

Es bastante fácil codificar la solución de este problema (script barnumber.mq4):

```
//+------------------------------------------------------------------+
// barnumber.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+
int start()                                // Special function start()
  {
   int n = 3;                                        // Preset number (nth bar) (enesima barra)
   Alert("Bar numbers starting from ", n,":");       // It does not depend on n
//+------------------------------------------------------------------+
   switch (n)                                        // Header of the operator 'switch'
     {                                        // Start of the 'switch' body
      case 1 : Alert("Bar 1");                       // Variations..
      case 2 : Alert("Bar 2");
      case 3 : Alert("Bar 3");
      case 4 : Alert("Bar 4");                       // Here are 10 variations ..
      case 5 : Alert("Bar 5");                       // ..'case' presented, but, in general, ..
      case 6 : Alert("Bar 6");                       // ..the amount of variations..
      case 7 : Alert("Bar 7");                       // ..'case' is unlimited
      case 8 : Alert("Bar 8");
      case 9 : Alert("Bar 9");
      case 10: Alert("Bar 10");break;
      default: Alert("Wrong number entered");// It is not the same as the 'case'
     }                                        // End of the 'switch' body
//+------------------------------------------------------------------+
   return;                                  // Operator to exit start()
  }
//+------------------------------------------------------------------+
```

En el operador 'switch', el programa buscará en las variantes 'caso', hasta que detecta que la expresión es igual a la constante. Cuando el valor de la expresión (en nuestro caso, el entero 3) es igual a una de las constantes (en este caso el caso 3), todos los operadores siguientes a los dos puntos ":" (caso 3:) se ejecutarán, a saber: el operador llama a la función Alert ("Bar 3") y los siguientes Alert ("Bar 4"), Alert ("Bar 5"), etc, hasta que llega al operador "break" que termina el funcionamiento del operador "switch" .

Si el valor de la expresión no coincide con ninguna de las Constantes, el control se pasa al operador que corresponde a la variante "default":

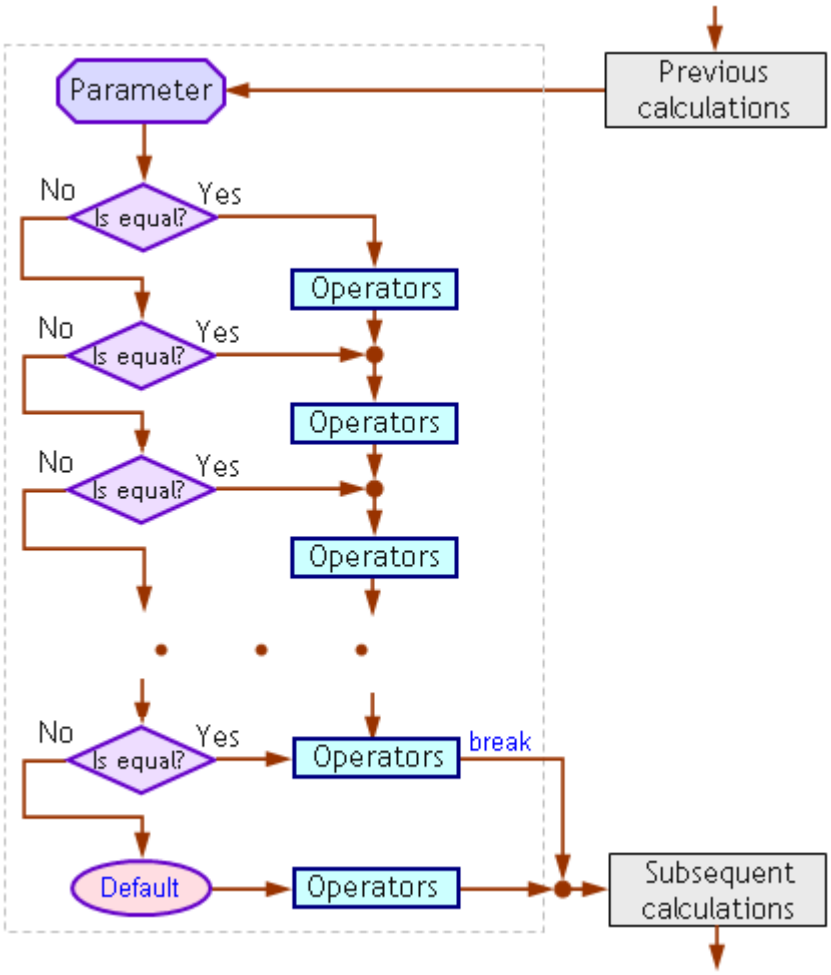


Fig. 50. Diagrama de bloques del operador 'swicth' en el script barnumber.mq4.

A diferencia del algoritmo realizado en el anterior programa, en este caso (Fig. 50), no estamos utilizando el operador “break” en cada una de las variantes "case". Por tanto, si el valor de la expresión es igual al valor de una de las constantes, se ejecutarán todos los operadores siguientes a los operadores de la correspondiente variante “caso”. También utilizamos el operador “break” en la última variante “caso” para otro propósito: evitar que se ejecuten los operadores correspondientes a la variante "por defecto". Si no hay un valor igual a la expresión entre los valores de las constantes, el control se pasa directamente al operador que se corresponde con la etiqueta 'default'.

Por lo tanto, si el valor de la variable n preset se encuentra dentro del rango de valores de 1 a 10, los números de todas las barras se imprimirá a partir de la enésima. Si el valor de n esta por encima del rango entre 1 y 10, el programa informará al usuario que se ha entrado un numero erróneo fuera del rango.

Nota: No es necesario que las constantes de las variantes “case” se ordenen en su programa según la magnitud. El orden de cómo las variantes “case” con las correspondiente constantes se siguen la una a la otra está determinada por las necesidades del algoritmo de su programa.

La Funcion de Llamada

Una llamada a función puede ser usada como un operador independiente y encontranse en cualquier lugar del programa donde esté implicado un cierto valor (con la excepción de los casos predefinidos). El formato y normas de ejecución de una llamada a una función cubre tanto las funciones estándar (built-in) como las funciones definidas por el usuario.

Formato de la función de llamada

Una llamada a una función esta compuesta por el nombre de la función y la lista de los parámetros escritos entre paréntesis:

```
NombreDeLaFuncion (ListaDeParametrosSeparadosPorComas) //  Llamada a la función como tal
```

El nombre de la función especificada en la llamada a la función debe ser el mismo que el nombre de la función que desea llamar para su ejecución. Los parámetros en la lista están separados por comas. La cantidad de parámetros a ser pasados a la función está limitada y no puede ser superior a 64. En una llamada a una función se pueden utilizar como parametros constantes, variables y llamadas a otras funciones. La cantidad, tipo y orden de los parámetros pasados en la llamada a una función debe ser la misma que la cantidad, tipo y orden de los parámetros especificados en la descripción de una función (la excepción es una llamada a una función con los parámetros por defecto).

```
My_function (Alf, Bet)          // Example of a function call
                                // Here:
My_function                     // Name of the called function
Alf                             // First passed parameter
Bet                             // Second passed parameter
```

Si la Llamada a la función no lleva parámetros de paso, la lista de parámetros se especifica como vacía, pero el paréntesis debe estar presente, de todos modos.

```
My_function ()                  // Exemplary function call
                                // Here:
My_function                     // Name of the called function
                                // There are no parameters to be passed
```

Si el programa debe llamar a una función con los parámetros por defecto, la lista de los parámetros pasado puede ser limitada (abreviada). Puede limitar la lista de parámetros, empezando con el primer parámetro por defecto. En el siguiente ejemplo, las variables locales b, c y d tienen algunos valores por defecto:

```
                                                    // For the function described as:
int My_function (int a, bool b=true, int c=1, double d=0.5)
  {
  Operators
  }
                                                    // .. the following calls are allowed:
My_function (Alf, Bet, Ham, Del)                    // Allowed function call
My_function (Alf )                          // Allowed function call
My_function (3)                             // Allowed function call
My_function (Alf, 0)                        // Allowed function call
My_function (3, Tet)                        // Allowed function call
My_function (17, Bet, 3)                    // Allowed function call
My_function (17, Bet, 3, 0.5)                       // Allowed function call
```

Los parámetros sin valores por defecto no se pueden omitir. Si un parámetro por defecto se salta, los parámetros por defecto subsiguientes no deben ser especificados.

```
                                    // .. Las siguientes llamadas no están permitidas:

 My_function ()                     // No permitida la llamada por defecto a la función. Falta..
                                    // .. 1º parámetro que no puede ser omitido.
My_function (17, Bet,, 0,5)  // No permitida la llamada a la función: se omite ..
                                    // .. parámetro por defecto (el tercero)


                                    // La función esta descrita como:
int My_function (int a, b bool = true, int c = 1, doble d = 0,5)
{
Operadores
}
```

La llamadas a las funciones se dividen en dos grupos: las que devuelven un valor predefinido de un tipo y los que no devuelven ningún valor.

Formato de llamada a la función sin return

Si la llamada a la función no devuelve ningún valor sólo puede ser compuesta como un operador independiente. La llamada a una función con operador sin return termina en ";" (punto y coma):

```
 Function_name (Parameter_list);     // Llamada a la función con operador sin return
 Func_no_ret (alfa, beta, gamma);            // Ejemplo de un operador de llamada a una ..
                                             // ..  función que no devuelve ningún valor
```

Ningún otro formato ó tecnica, se ofrece para llamar a funciones que no devuelven ningún valor.

Formato de llamada a función con return.

Una llamada a una función que devuelve un valor puede estar compuesta como un operador separado o puede ser utilizada en el código de programa en lugares donde un valor de un determinado tipo esta implicado.

Si la llamada a la función se compone de un operador independiente, este termina en ";" (punto y coma)

```
Function_name (Parameter_list);          // Llamada a la función con devolucion de valor
Func_yes_ret (Alpha, Beta, Delta);       // Ejemplo de operador llamada a una función..
                                         // .. para una función que devuelve un valor
```

Regla de Ejecución llamada a función

Una llamada a la función llama a la función del mismo nombre para su ejecución. Si la llamada a la función esta compuesta como un operador separado, después de que la función ha sido ejecutada, el control se pasa al operador que sigue la llamada a la función. Si la llamada a la función se utiliza en una expresión, después de que la función se ha ejecutado, el control se pasa a la ubicación en la expresión donde la la llamada a la función ha sido especificada y los demás cálculos se llevan a cabo en la expresión utilizando el valor devuelto por la llamada a la función.

El uso de llamadas a funciones en otros operadores está determinado por el formato de estos operadores.

Problema 20. Redactar un programa con las siguientes condiciones:
-- Si la hora actual es mayor de las 15:00, ejecutar 10 repeticiones en el ciclo 'for';
-- En todos los demás casos, ejecutar 6 iteraciones.

A continuación se muestra un ejemplo de script callfunction.mq4 que incluye: una llamada a una función en la cabecera del operador 'for' (como parte de Expression_1, de acuerdo al formato del operador 'for', véase el Operador de ciclo 'for'), una regla de llamada a función como un operador, en la parte derecha del operador de asignación (véase el operador de asignación), y en la cabecera del operador "if-else" (en la condición, de acuerdo al formato del operador "if-else", véase el operador condicional 'if- else").

```
///------------------------------------------------------------------------------
// callfunction.mq4
// The code should be used for educational purpose only.
//------------------------------------------------------------------------------
int start()                                     // Description of function start()
  {                                             // Start of the function start() body
   int n;                                       // Variable declaration
   int T=15;                                    // Predefined time
   for(int i=Func_yes_ret(T);i<=10;i++)         // The use of the function in..
                                                // The cycle operator header
     {                                          // Start of the cycle 'for' body
      n=n+1;                                    // Iterations counter
      Alert ("Iteration n=",n," i=",i);         // Function call operator
     }                                          // End of the cycle 'for' body
   return;                                      // Exit function start()
  }                                             // End of the function start() body
//------------------------------------------------------------------------------
int Func_yes_ret (int Times_in)                 // Description of the user-defined function
  {                                             // Start of the user-defined function body
   datetime T_cur=TimeCurrent();                // The use of the function in..
                                                // ..the assignment operator
   if(TimeHour(T_cur) > Times_in)               // The use of the function in..
                                                //..the header of the operator 'if-else'
     return(1);                                 // Return value 1
   return(5);                                   // Return value 5
  }                                             // End of the user-defined function body
//------------------------------------------------------------------------------
```

En el ejemplo anterior, fueron llamadas las siguientes funciones con los siguientes parámetros transferidos:

- llamada a la función Func_yes_ret (T) - variable T; (Es función definida por el usuario)
- llamada a la función de Alert () – constantes de tipo string "Iteración n =" e "i =", variables n e i;
- llamada a la función TimeCurrent () sin parámetros de paso;
- llamada a la función TimeHour (T_cur) - T_cur variable.

En este programa se ejecuta un algoritmo simple. Hemos establecido en la variable T el tiempo (en horas), en relación a que los cálculos se ha realizar. En la cabecera del operador 'for', especificar la llamada a la función definida por el usuario Func_yes_ret () que puede devolver uno de dos valores: 1 o 5. De acuerdo con este valor, la cantidad de iteraciones en el ciclo cambiará: no habrá ni 10 (i cambios de 1 a 10) o 6 (i cambios de 5 a 10) iteraciones. Para una mejor visualización, en el cuerpo del bucle utilizamos el contador de iteración, cada valor de los cuales se visualiza en la pantalla utilizando la función Alert ().

En la descripción de la función definida por el usuario, lo primero que se calcular es el tiempo en segundos transcurrido después de las 00:00 de 1 de enero de 1970 (llamada a la función TimeCurrent ()), y luego calcular la hora actual en horas (llamada a la función TimeHour ()). El algoritmo es ramificado usando el operador 'if' (la llamada a la función TimeHour () se especifica en su condición). Si la hora actual resulta ser mayor que la pasada a la función definida por el usuario (variable local Times_in), esta última devuelve 1, en caso distinto devuelve 5.

Tengase en cuenta que:

No hay en el programa descripciones de funciones estándar o llamada a función para la función start ().

A continuación se puede ver el diagrama de bloques del script callfunction.mq4:

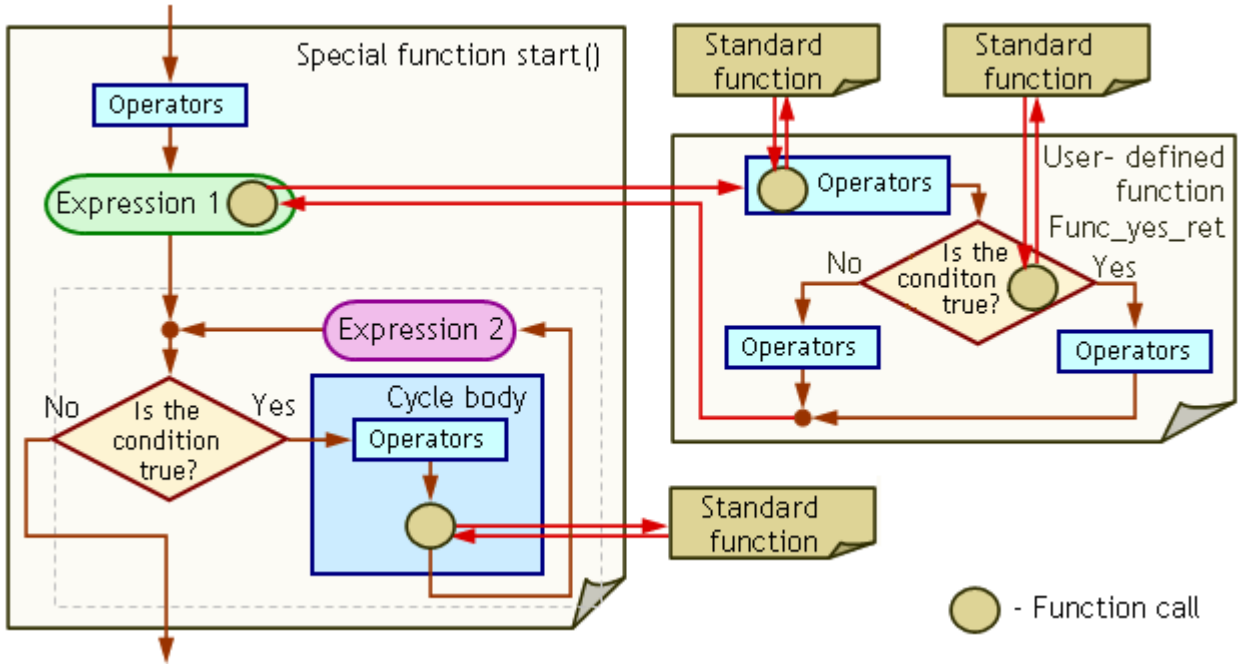


Fig. 51. Diagrama de bloques de un programa que utiliza llamadas a funciones.

Los círculos en el diagrama marcan las llamadas a funciones (para la función estandar y la función definida por el usuario). Flechas rojas indican el paso del control a la función y viceversa. Se puede ver claramente que la función devuelve el control a la ubicación en donde se especificó la llamada a la función y no devuelve los cálculos que se están realizando en el camino entre la llamada a la función y la función en sí. En general, si una función devuelve un valor, este valor se pasa al módulo de llamadas (a lo largo de la flecha roja en la dirección de la llamada a la función).

Las funciones especiales se pueden llamar desde cualquier lugar en el programa de acuerdo a las normas generales, al igual que cualesquiera otras funciones. Las funciones especiales también pueden tener parámetros. Sin embargo, cuando el Terminal de Usuario llama a estas funciones especiales, los parámetros no se transmitirán desde el exterior, esta función utilizará los valores por defecto. El uso de parámetros en funciones especiales sólo será razonable si se les llama desde un programa. A pesar de que es tecnicamente posible en MQL4 llamar a funciones especiales desde un programa, no se recomienda hacerlo. Un programa que utiliza llamadas a funciones especiales debe ser considerado como incorrecto.

Descripción y funcion del operador "return"

La necesidad de especificación de funciones dentro de un programa podemos dividirlas en dos grupos: Las funciones que no se describen en el programa, y las funciones que deben estar descritas en el programa. Las funciones estandar no se describen en los programas. Las funciones definidas por el usuario deben siempre describirse en el programa. Las funciones especiales, si las hubiere, deben también describirse en el programa.

Formato de la descripción de funciones

La descripción de una función consta básicamente de dos partes: cabecera de la función y cuerpo de la función.

La cabecera de la función contiene el tipo del valor de return, el nombre de la función, y la lista de parámetros entre paréntesis. Si una función no debe devolver ningún valor, su tipo denominarse en la cabecera ha de ser de tipo void (*vacío).*

El cuerpo de la función puede consistir en operadores simples y/o compuestos o por llamadas a otras funciones, y encerradas entre llaves.

```
TipoValorDeReturn NombreDeLaFuncion (Lista de parámetros formales)          // Encabezado

 {                                                  // Apertura de llave. Inicio del cuerpo de la funcion
 Código del Programa                                // El cuerpo de la función puede estar..
 que componen                                       // ..  formado de operadores y ..
 el cuerpo de la función                            // ..  de llamadas a otras funciones
 }                                                  // Cierre de llave y fin del cuerpo de la funcion
```

La lista de parámetros se escribe separada por comas. El número de parámetros a transferir a la función esta limitado a 64. Los parámetros formales de la cabecera de la función solo pueden ser especificados en forma de variables y no como constantes, ni expresiones, ni en forma de llamada a otras funciones). La cantidad, tipo y orden de los parámetros transferidos en la llamada a la función deben ser los mismos que los de los parámetros formales especificados en la descripción de la función (con excepción de la llamada a una función con parámetros que tienen valores por defecto):

```
int My_function (int a, double b)       // Example of function description
  {
   int c = a * b + 3;                                // Function body operator
   return (c);                          // Function exit operator
  }
                                                     // Here (from left to right in the header):
   int                                  // Return value type
   My_function                                       // Function name
   int a                                // First formal parameter a of the int type
  double b                              // Second formal parameter b of the double type
```

Los parámetros transferidos a la función pueden tener valores por defecto que se definen por una constante del tipo correspondiente:

```
int My_function (int a, bool b=true, int c=1, double d=0.5)//Example of function description
  {
   a = a + b*c + d2;                    // Function body operator
   int k = a * 3;                       // Function body operator
   return (k);                  // Function exit operator
  }

   // Here (from left to right in the header):

   int                          // Return value type
   My_function                          // Function name
   int a                        // 1º formal parameter (variable) a of the int type
   bool b                       // 2º formal parameter (variable)  b of the double type
   true                         // Constant, the default value for b
   int c                        // Third formal parameter (variable) c of the int type
   1                            // Constant, the default value for c
   double d                     // Fourth formal parameter (variable) d of the double type
   0.5                          // Constant, the default value for d

   a,b,c,d,k                    // Local variables
```

Si existen parámetros que figuran en la llamada a la función y que ellos mismos contienen valores por defecto en los parámetros formales de la descripción de la función, los parámetros que utilizará la función en sus cálculos serán aquellos que figuren en la llamada a la función. Si no existieran parámetros que figuren en la llamada a la función y que ellos mismos contienen valores por defecto en los parámetros formales de la descripción de la función, la función será calculada con los correspondientes valores por defecto.

Las funciones especiales también pueden tener parámetros. Sin embargo, el Terminal de Usuario no pasa parámetros desde el exterior en la llamada a estas funciones, sólo utiliza los valores por defecto. Las funciones especiales se pueden llamar desde cualquier lugar del módulo de acuerdo a las normas generales, al igual que cualquiera otras funciones.

Reglas de Ejecución de función

Ubicación de la descripción de una función de un programa:

La ubicación de la descripción de una función en un programa debe ser tal que esté separada de cualesquiera otras funciones, es decir, la función no debe estar nunca situada dentro de otra función.

Ejecución de la función:

La llamada a una función implica que ésta se ejecutará según el código del que está compuesto el cuerpo de dicha función.

Formato del operador «return»

El valor devuelto por la función es el valor del parámetro que se especifica entre los paréntesis del operador «return». El operador «return» se compone de la palabra clave «return» y la expresión contenida entre paréntesis, y termina con el carácter ";" (punto y coma). El formato completo operador «return» es:

```
return (expresión);                    // Operador de retorno
```

La expresión entre paréntesis puede ser una constante, una variable o una llamada a una función. El tipo del valor devuelto utilizando en el operador 'return' debe ser el mismo que el tipo del valor de la función que se especifica en función de la cabecera de la llamada a la función. Si este no es el caso, el valor de la expresión especificada en el operador 'return' debe ser emitidos al tipo del valor de return que se especifica en la cabecera de la descripción de la función. Si el typecasting es imposible, MetaEditor le dará un mensaje de error al compilar su programa.

Estado de Ejecución del operador «return»

El operador «return» detiene la ejecución de la función exterior más cercana y pasa el control al lugar de llamada del programa de acuerdo a las reglas definidas para una llamada a una función. El valor devuelto por la función es el valor de la expresión especificada en el operador «return». Si el tipo de parámetro del valor especificado en el operador de 'return' es diferente a la del valor de return que se especifica en la cabecera de la llamada a la función, el valor debe ser emitido al tipo del valor de return que se especifica en la cabecera de la descripción de la función.

Un ejemplo de cómo usar el operador "return" que devuelve un valor:

```
 bool My_function (int Alpha)                  // Description of the user-defined function
{                                              // Start of the function body
 if(Alpha>0)                                   // Operator 'if'
   {                                           // Start of the operator-'if' body
    Alert("The value is positive");      // Standard function call
    return (true);                             // First exit from the function
   }                                           // End of the operator-'if' body
 return (false);                               // Second exit from the function
}                                              // End of the function body
```

Si el valor de return de una función es de tipo *void (vacio),* el operador "return" se debe usar sin expresión:

```
return;          // Operador de «return» sin expresiones entre paréntesis
```

Ejemplo de uso del operador «return» sin valor de retorno:

```
 void My_function (double Price_Sell)          // Description of the user-defined function
{                                                      // Start of the function body
 if(Price_Sell-Ask >100 * Point)                       // Operator 'if'
   Alert("Profit for this order exceeds 100 п");  // Standard function call
 return;                                               // Exit function
}                                                      // End of the function body
```

Existe la posibilidad de no incluir el operador "return" en la descripción de una función. En este caso, la función se dará por terminado su funcionamiento de forma automática, tan pronto como (de acuerdo con el algoritmo de ejecución) se ejecute el ultimo operador del cuerpo de la función. Ejemplo de la descripción de una función sin el operador «return»:

```
 void My_function (int Alpha)                  // Description of the user-defined function
{                                                      // Start of the function body
 for (int i=1; i<=Alpha; i++)              // Cycle operator
   {                                                   // Start of the cycle body
    int a = 2*i + 3;                                   // Assignment operator
    Alert ("a=", a);                                   // Standard function call operator
   }                                                   // End of the cycle body
}                                                      // End of the function body
```

En este caso, la función completa sus operaciones en el momento en que el ciclo del operador ‘for’ termina su ejecución. La última acción de la ejecución de la función que será la prueba de condición en el operador de ciclo. Tan pronto como la condición en la cabecera del operador de ciclo ‘for’ se convierte en falsa, el control pasará fuera del ciclo del operador (for). Sin embargo, debido a que el operador de ciclo es el último operador ejecutable en el cuerpo de la función denominada My_function(), la función definida por el usuario terminará sus operaciones y el control pasará fuera de la función, al lugar donde la función fue llamada para su ejecución.

Variables

Para la creación de programas en cualquier lenguaje algorítmico es muy importante conocer los diferentes tipos de variables. En esta sección vamos a analizar todos los tipos de variables que se utilizan en el lenguaje MQL4.

- Variables predefinidas y RefreshRates Función.
  Lo primero que debemos hacer es aprendernos los nombres de las variables predefinidas. Los nombres de las variables predefinidas son nombres reservados y no se pueden utilizar para crear variables personalizadas. Se trata de las variables predefinidas que contienen las principales informaciónes necesarias para el análisis de situación actual del mercado. Para actualizar esta información se utiliza **RefreshRates ().**

- Tipos de variables.
  Las variables son muy importantes en la redacción de un programa. Ellas se dividen en locales y globales, externas e internas. Las variables estaticas preservan sus valores entre llamadas a funciones, es útil recordar algunos valores de las variables locales sin crear variables globales.

- GlobalVariables.
  Al lado de las variables globales a nivel de programa separado cuyos valores están disponibles desde cualquier parte del programa, existen las variables globales a nivel del terminal. Estas variables globales son llamadas GlobalVariables. Permiten establecer una interacción entre partes independientes de distintos programas escritos en MQL4. Se pueden utilizar para el intercambio de valores entre scripts, indicadores y Asesores Expertos. Cuando se cierra el Terminal, los valore de las GlobalVariables también se conservan para estar disponibles en próximo arranque del MetaTrader 4. No hay que olvidar que, si no hay llamadas a una GlobalVariable durante 4 semanas, está será eliminada.

- Las matrices. (arrays)
  Si se necesita guardar o precesar una gran cantidad de valores de un tipo, esto no se puede hacer sin arrays. Estas serán declaradas como variables. La llamada a los elementos de un arrayse realiza a través del indice del elemento (index). La indexación de un Array comienza en cero. Número de dimensiones de un array se llama dimensionalidad. Las matrices de más de cuatro dimensiones no son aceptadas. Los valores del array deben estar claramente inicializado con el fin de evitar errores difíciles de localizar.

Variables predefinidas y función RefreshRates

Existen variables predefinidas con nombres del lenguaje MQL4.

Una **variable Predefinida** es una variable con un nombre predefinido, cuyo valor se define en un Terminal de Usuario y no se pueden cambiar mediante programa. Las variables predefinidas reflejan el estado actual de una gráfica en el momento de empezar el programa (Asesor Experto, scrip o indicador personalizado) o como resultado de la aplicación de la función RefreshRates ().

Lista de nombres de variables predefinidas simples

Ask -	último precio conocido de venta del actual titulo o valor;
Bid -	último precio conocido de compra del actual titulo o valor;
Bars -	número de barras en un gráfico actual;
Point -	Tamaño de punto del actual titulo o valor de la moneda cotizada;
Digits -	número de dígitos después de un punto decimal en el precio del actual titulo o valor.

Lista de nombres predefinidos de Arrays-Timeseries

Time - fecha y hora de apertura de cada barra en el gráfico actual;

Open - precio de apertura de cada barra en el gráfico actual;

Close - precio de cierre de cada barra en el gráfico actual;

High - precio máximo de cada barra en el gráfico actual;

Low - precio mínimo de cada barra en el gráfico actual;

Volume - marca el volumen de cada barra en el gráfico actual.

(los conceptos de "arrays" y "arrays-timeseries" serán descritos en la seccion de Arrays).

Características de las variables predefinidas

Los nombres de las variables predefinidas no pueden utilizarse para identificar variables definidas por el usuario. Las variables predefinidas se pueden utilizar en expresiones al igual que cualquier otra variable y de conformidad con las mismas reglas, pero el valor de una variable predefinida no se puede cambiar. Al intentar compilar un programa que contenga un operador de asignación, en el que una variable predefinida se coloca a la izquierda de un signo de igualdad, MetaEditor mostrará un mensaje de error. En términos de visibilidad las variables predefinidas tienen referencia a nivel global, es decir, están disponibles desde cualquier parte del programa (ver tipos de variables).

La propiedad más importante de variables predefinidas es la siguiente:

Los valores de todas las variables predefinidas se actualizan automáticamente en el Terminal de Usuario en el momento que se inician las funciones especiales para su ejecución.

El valor actual y anterior de una variable predefinida puede ser igual, aunque el valor en sí sea actualizado. Los valores de estas variables predefinidas están actualizados y disponibles a partir del momento que se inicia la primera linea de programa de la función especial. Vamos a ilustrar la actualización de variables predefinidas en el siguiente ejemplo (Asesor Experto predefined.mq4):

```
//+------------------------------------------------------------------+
// predefined.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+
int start()                          // Special funct. start
  {
   Alert("Bid = ", Bid);                 // Current price
   return;                           // Exit start()
  }
//+------------------------------------------------------------------+
```

A partir de este programa es fácil ver que los valores de la variable del precio de compra (variable predefinida **Bid**) será igual al precio actual de cada momento. De la misma manera se puede comprobar los valores de otras variables, dependiendo de las condiciones actuales. Por ejemplo, la variable **Ask** (precio de venta) también depende del precio actual. El valor de la variable **Bars** (numero de barras en el grafico actual) también cambiará si el número de barras de cambió. Esto puede suceder en un tick, en la que una nueva barra se está formando en la ventana de un grafico. El valor de **Point** depende del título o valor especificado. Por ejemplo, para EUR/USD, este valor es 0.0001, para USD/JPY es 0,01. Valor de los dígitos de estos valores (variable **Digits**) es igual a 4 y 2, respectivamente.

Aquí tenemos otra propiedad importante de variables predefinidas:

El Terminal de Usuario crea un conjunto de copias locales de variables predefinidas por separado para cada programa que se inicia. Cada programa que se inició trabaja con sus propio juego de copias de los datos históricos.

En una Terminal de Usuario se pueden ejecutar varios programas de aplicación al mismo tiempo (asesores expertos, scripts ó indicadores), y para cada uno de ellos el Terminal de Usuario crea una copia con un juego de los datos históricos de todos los valores de las variables predefinidas. Vamos a analizar en detalles las razones de esta necesidad. La Fig. 52 muestra el posible funcionamiento de la ejecución de Asesores Expertos con diferentes longitudes de tiempo de la función especial start (). Por simplicidad vamos a suponer que en los Asesores Expertos analizados no hay otras funciones especiales y que Los Asesores Expertos de ambos operan en los mismos marcos temporales de un mismo símbolo.

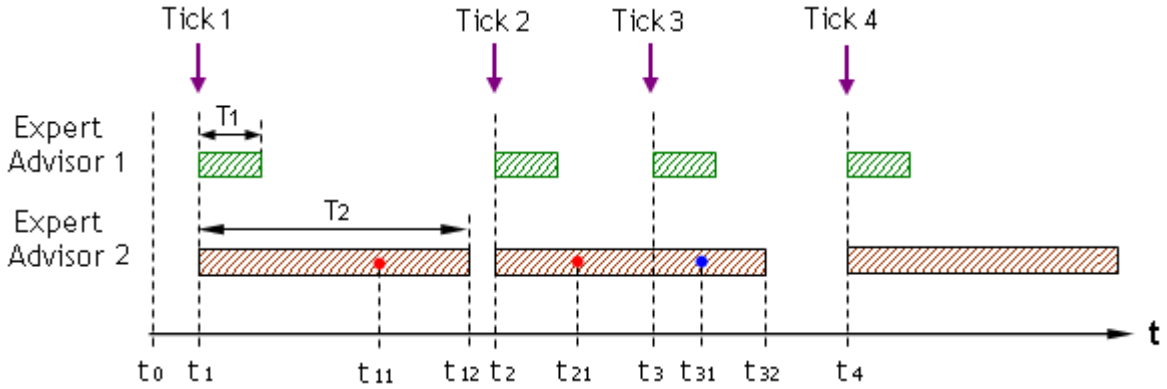


Fig. 52. El tiempo de operación de la función star () puede ser mayor o menor que un intervalo de tiempo entre los ticks.

Los Asesores Expertos difieren en el tiempo de ejecución de la función especial start() . Para un Asesor de Expertos común de nivel medio el tiempo de ejecución está aproximadamente entre 1 y 100 milisegundos. Otros Asesores Expertos pueden tener tiempos de ejecución mucho mayores, tiempos de, por ejemplo, varios segundos o decenas de segundos. El intervalo de tiempo entre los ticks también es diferente: desde milisegundos a varios minutos y, a veces, incluso decenas de minutos. En este ejemplo vamos a analizar cómo influye en el funcionamiento de los Asesores Expertos 1 y 2 la frecuencia de recepción de los ticks debido al los diferentes tiempos de ejecución de la función especial start() de cada uno de los Asesores Expertos.

En el momento momento t0 el Asesor Experto 1 se vincula al Terminal de Usuario y cambia al modo de espera de tick. En el momento t1 aparece un tick y el terminal inicia la función especial start () junto con el programa que le da acceso a la copia actualizada del juego de variables predefinidas. Durante el periodo de ejecución, el programa puede hacer referencia a estos valores que se mantendrán sin variación durante todo el tiempo de operación de la función especial start(). Cuando la función star() termina todas sus operaciones, el programa entra en el modo de espera de tick.

El momento siguiente más próximo en el que las variables predefinidas pueden obtener nuevos valores es a la llegada de un nuevo tick. El tiempo de ejecución T1 de start() del Experto Asesor 1 es considerablemente inferior a al tiempo de espera entre ticks, por ejemplo intervalo t 1 - t 2 o t 2-t 3, etc Por lo tanto, el tiempo de ejecución del Asesor Experto 1 analizado, en ningun momento caerá en la situación de que los valores de variables predefinidas antiguas estén caducadas, es decir, que difieran de su verdadero valor actual (último conocido).

En la operación del Asesor Experto 2, la situación es diferente porque el tiempo de ejecución de su funcion start(), periodo T2, a veces supera el intervalo entre ticks. La función start () del Asesor Experto 2 también comenzó en el momento t1. La Fig. 52 muestra que el intervalo t 1 - t 2 entre los ticks es más grande que el tiempo de ejecución de start(), periodo T2, es por eso que durante este período del programa, no se realiza la operación de actualización de las variables predefinidas (en este período los nuevos valores no proceden del servidor, por lo que sus valores son los valores que aparecieron en el momento t1).

La siguiente ocasión en que se inicia el Asesor Experto 2, lo hace en el momento t2 que es cuando se recibe el segundo tick. Junto con éste, se recibe la copia del conjunto de los valores actualizados de las variables predefinidas. En la Fig. 52 vemos que en el momento del tick t3 la función Stara() está todavía en ejecución. Una cuestión que se plantea es: ¿Cuáles serán los valores de las variables predefinidas disponibles para el Asesor Experto 2 en el período que va desde t3, cuando llega el tercer tick, hasta t32, que es cuando cuando start () termina su operación? La respuesta puede encontrarse en conformidad con la siguiente regla:

Las copias de los valores de variables predefinidas se guardan durante todo el período de operación de las funciones especiales. Estos valores pueden ser forzados a actualizarse usando la función estandar **RefreshRates ()**.

Por lo tanto (si RefreshRates () no ha sido ejecutada) durante todo el período de ejecución de start(), El Asesor Experto 2 tendrá acceso al conjunto de las copias locales de las variables predefinidas que se crearon cuando fue recibido el segundo tick. A pesar de que los Asesores Expertos operan en las mismas ventanas. A partir del momento t3 de la recepción de ticks, cada AE funcionará con diferentes valores de variables predefinidas. El Asesor Experto 1 trabajará con su propio conjunto de copias locales de datos históricos, valores que se definen en el momento t3, y el Asesor Experto 2 trabajará con sus propias copias de datos que son valores iguales al momento t2.

El tiempo de ejecución más largo que un programa de aplicación tiene y el menor tiempo del intervalo entre los ticks nos da la probabilidad es que el próximo tick llegará durante el período de ejecución del programa. El conjunto de copias locales de datos históricos establece las condiciones de cada programa que garantiza la constancia de variables predefinidas a través de todo el plazo de ejecución de una función especial.

A partir del momento t4 cuando llega el siguiente tick, ambas AEs se iniciará una vez más y cada uno de ellos tendrá acceso a su propia copia del conjunto de variables predefinidas, de los valores que se forman en el momento t4 que coincide con la llegada del cuarto tick.

RefreshRates ()

```
bool RefreshRates ()
```

La función estándar **RefreshRates ()** permite actualizar los valores locales de las copias datos históricos. En otras palabras, esta función fuerza la actualizacion de datos acerca de un entorno de mercado actual (volumen, servidor de tiempo de la última cotización Tiempo [0], Bid, Ask, etc.) Esta función puede utilizarse cuando un programa utiliza mucho tiempo en realizar sus cálculos y por lo tanto tiene necesidades de actualizar los datos.

RefreshRates () devuelve TRUE, si en el momento de su ejecución hay información sobre los nuevos datos históricos en la terminal (es decir, si ha llegado un nueva tick durante la ejecución del programa). En tal caso, el conjunto de copias locales de las variables predefinidas se actualizará.

RefreshRates () devuelve FALSE, si desde el momento del inicio de la ejecución de una función especial de los datos históricos el Terminal de Usuario no se han actualizado. En tal caso, las copias locales de las variables predefinidas no cambian.

Tenga en cuenta que RefreshRates () sólo influye en el programa en el que se ha iniciado (no todos los programas trabajan con el Terminal de Usuario al mismo tiempo).

Vamos a ilustrar la ejecución de RefreshRates () con un ejemplo.

Problema 21. Contar el número de iteraciones que un operador de ciclo puede realizar entre los ticks (el más cercano cinco ticks).

Este problema sólo puede resolverse mediante RefreshRates () (script countiter.mq4):

```
//+------------------------------------------------------------------+
// countiter.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+
int start()                                   // Special funct. start()
  {
   int i, Count;                              // Declaring variables
   for (i=1; i<=5; i++)                 // Show for 5 ticks
     {
      Count=0;                                // Clearing counter
      while(RefreshRates()==false)            // Until...
        {                                     //..a new tick comes
         Count = Count+1;                 // Iteration counter
        }
      Alert("Tick ",i,", loops ",Count);      // After each tick
     }
   return;                                    // Exit start()
  }
//+------------------------------------------------------------------+
```

De acuerdo con las condiciones del problema, los cálculos deben hacerse sólo para los cinco ticks más próximos, que es la razón por la que podemos usar un script. Dos variables se utilizan en el programa: "i" para contar el número de ticks y "Count" para contar las repeticiones. El ciclo externo for está organizado en función del número de ticks procesados (de 1 a 5). En el ciclo for se inicializa el contador de iteraciones poniendolo a cero (realizado en el ciclo while), al final se muestra un Alert () con el número de tick y la cantidad de sus iteraciones.

El ciclo I nterior 'while' va a operar mientras el valor devuelto por RefreshRates () sea igual a false, es decir, hasta que se marque un nuevo tick. Durante la operación de 'while', es decir, en el intervalo entre los ticks, el valor del contador Count irá incrementandose constantemente, por lo que se contará el número de iteraciones del ciclo 'while'. Si en el momento del control de la condición de 'while' el valor devuelto por RefreshRates () es «true», significará que hay nuevos valores de variables predefinidas en el Terminal de Usuario, debido a la llegada de un nuevo tick. Como resultado el control se devuelve de nuevo fuera de 'while' y así termina el conaje de iteraciones.

Como resultado de la ejecución del script countiter.mq4 un número de alertas caracteristico de la ejecución de MQL4 aparecerán en la ventana del símbolo:

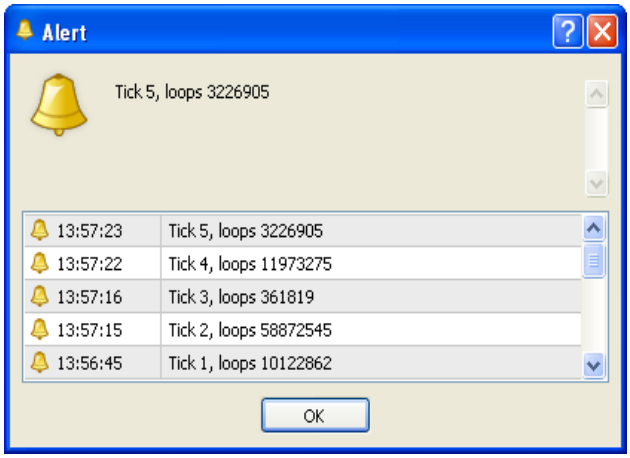
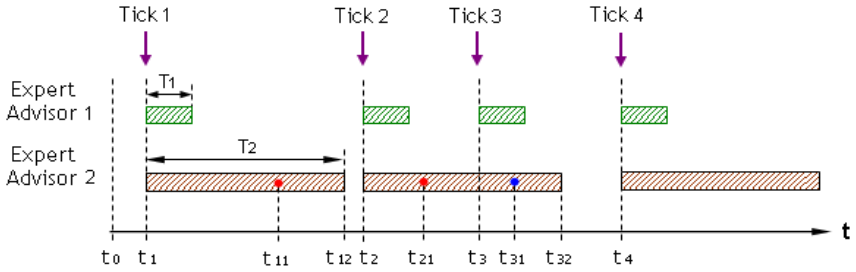


Fig. 53. Resultados de la operación de countiter.mq4 en la ventana EUR / USD.

Es fácil ver que durante 1 segundo (intervalo entre el cuarto y el quinto ticks) el programa ha realizado más de 3 millones de iteraciones. Análogos resultados pueden ser obtenidos con los simples cálculos para otros los ticks.

Vamos a volver al ejemplo anterior (Asesor Experto predefined.mq4). Anteriormente vimos que si RefreshRates () no se ejecuta en el Asesor Experto 2, los valores de las copias locales de las variables predefinidas, se mantienen sin variación durante todo el período de la ejecución de start (), como por ejemplo durante el período t 2 - t 32. Si después del tercer tick (que viene cuando start () se está ejecutando) se ejecuta la función RefreshRates (), por ejemplo en el momento t31, los valores locales de las copias serán actualizadas. Así, durante el tiempo restante a partir de t31 (ejecución de RefreshRates ()) a t32 (fin de ejecución de start ()), los nuevos valores de las copias de la variables locales predefinidas iguales a los valores definidos por el terminal de usuario en t3 estará disponible para el Asesor Experto 2.

Si en el Asesor Experto 2, RefreshRates se ejecuta en el momento t 11 o t21 (es decir, en el período en que el último tick es el que ha iniciado la ejecución de start ()), las copias locales de las variables predefinidas no serán cambiadas (por que son las mismas). En tales casos, los valores actuales de las copias locales de las variables predefinidas serán iguales a la última conocida, es decir, a aquellas que fueron definidas por el Terminal de Usuario en el momento de la última salida de la función especial start ().



Tipos de variables

Un programa de aplicación en MQL4 puede contener decenas y cientos de variables. Una parte muy importante de la caracteristica de las variables es la posibilidad de utilizar su valor en un programa. La limitación de esta posibilidad está relacionada con el alcance o ámbito de la variable.

El ámbito de una Variable es la ubicación en un programa donde el valor de la variable está disponible. Cada variable tiene su propio ámbito de aplicación. De acuerdo con el alcance hay dos tipos de variables en MQL4: **variables locales** y **variables globales**.

Variables locales y globales

La **Variable local** es una variable declarada dentro de una función. El alcance de las variables locales es el cuerpo de la función, en el que la variable ha sido declarada. La Variable local puede ser inicializada por una constante o una expresión correspondiente a su tipo.

La **Variable global** es una variable declarada más allá de las funciones. El alcance de las variables globales es el programa entero. Una variable global puede ser inicializada sólo por una constante del mismo tipo, pero no puede ser inicializada por una expresión. Las variables globales se inicializan sólo una vez antes de la declaración de la ejecución de funciones especiales.

Si el control en un programa se encuentra dentro de una determinada función, los valores de las variables locales declaradas en otra función no están disponibles. El valor de cualquier variable global está disponible desde cualquier función especial y funciones definidas por el usuario.

Vamos a ver un ejemplo sencillo.

Problema 22. Crear un programa que cuenta los ticks.

Solución de problemas algoritmo de 22 utilizando una variable global (countticks.mq4):

```
//+------------------------------------------------------------------+
// countticks.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+
int Tick;                        // Global variable (declarada antes de la descripcion de start()
//+------------------------------------------------------------------+
int start()                                 // Special function start()
  {
   Tick++;                                          // Tick counter
   Comment("Received: tick Nº ",Tick);  // Alert that contains number
   return;                                  // start() exit operator
  }
//+------------------------------------------------------------------+
```

En este programa sólo se utiliza una variable global "Tick". Es global, porque ha sido declarada fuera de la descripción de la función start (). Esto significa que la variable conserva su valor de un tick a otro. Vamos a ver los detalles de la ejecución del programa.

En Funciones especiales se analizaron los criterios del inicio de las funciones especiales. En pocas palabras: la función start () del Asesor Experto se inicia en el Terminal de Usuario cuando llega un nuevo tick. En el momento que el Asesor Experto se asocia a una ventana del símbolo de un titulo, se llevará a cabo lo siguiente:

1. Declaración de la variable global Tick. Esta variable no ha sido inicializada por una constante, es por eso que su valor en esta etapa es igual a cero.
2. El Terminal de Usuario mantiene el control hasta que llegue un nuevo tick.

3. Se recibe un tick. Se pasa el control a la función especial start ().

3,1. Dentro de start () Se pasa el control de ejecución al operador:

```
Tick++;                                    // Contador de ticks
```

Como resultado de la ejecución el operador Tick incrementa su valor en 1 (un entero).

3,2. Se pasa el control al operador:

```
Comment("received: tick Nº ",Tick);  // alerta que contiene el número de tick
```

La Ejecución de la función estándar Comment () provocará la aparición de la descripción:

```
Received: tick Nº 1 (Recibido: tick n º 1)
```

3,3. El control se pasa al operador:

```
return;                              // operado salir de start ()
```

Como resultado la ejecución start () termina su operacióln y el control se devuelve al Terminal de Usuario. La variable global sigue vigente, su valor es igual a 1.

Las misma acciones se repetiran a partir del punto 2. La variable tick se utilizará en los cálculos de nuevo, pero en el segundo tick, su valor es igual a 1 en el momento de iniciar la función especial start (). Por eso el resultado de la ejecución del operador…

```
Tick++;                                    // Contador de ticks
```

resultará en un nuevo valor de la variable Tick, que se incrementará en 1 y ahora será igual a 2. La ejecución de Comment () mostrará la alerta:

```
Received: tick Nº 2 (Recibido: tick n º 2)
```

Por tanto, el valor de Tick se incrementará en 1 en cada salida de la función especial start (), es decir, en cada tick. La solución de este tipo de problemas sólo es posible con el uso de variables que preserven sus valores después de salir de una función (en estos casos se utiiza una variable global). No es viable utilizar variables locales para este fin: una variable local que se declara en una función, pierde su valor al final de sus operaciones.

Se puede ver fácilmente que si iniciamos un Asesor Experto en el cual la variable Tick se abre como una variable local el programa contendrá un error algorítmico:

```
int start()                                // Special function start()
 {
  int Tick;                                        // Local variable
  Tick++;                                          // Tick counter
  Comment("Received: tick Nº ",Tick); // Alert that contains number
  return;                                  // start() exit operator
  }
```

Desde el punto de vista de la sintaxis no hay errores en el código. Este programa puede ser compilado con éxito y empezar su funcionamiento, pero cada vez que se ejecute el resultado será siempre el mismo:

```
Received: tick Nº 1 (Recibido: tick n º 1)
```

Es natural, porque la variable Tick se inicializa a cero al comienzo de la función especial start () cada vez que esta comienza. El incremento en uno de la variable da como resultado siempre 1, por eso la alerta mostrará siempre el tick numero 1.

Variables Static

En el nivel físico las variables locales se presentan en una memoria temporal como parte de la correspondiente función. Hay una manera de localizar una variable declarada dentro de una función en una memoria permanente del programa. El modificador "stático" debe indicarse antes del tipo de variable en su declaración:

```
static int Número;                                  // variable Static de tipo integer
```

La caracteristica particular de la variable static es que no pierde su valor al salida de la función donde esta ubicada, sin embargo, al contrario que la variable global, su ámbito de aplicación está limitado dentro de la función donde ha sido declarada

A continuación se muestra la solución del problema 22 utilizando una variable estática (Asesor Experto staticvar.mq4):

```
   //+------------------------------------------------------------------+
// staticvar.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+
int start()                                 // Special function start()
  {
   static int Tick;                         // Static local variable
   Tick++;                                          // Tick counter
   Comment("Received: tick No ",Tick); // Alert that contains number
   return;                                  // start() exit operator
  }
//+------------------------------------------------------------------+
```

Las variables static se inicializan solo una vez y solo pueden ser inicializadas por una constante (a diferencia de una simple variable local que se pueda inicializar con cualquier expresión). Si no hay inicialización explícita, la variable se inicializa a cero. La variable static se almacena en una memoria permanente y su valor no se pierde al salir de una función. Sin embargo, respecto al ámbito de aplicación, las variables static tienen las limitaciones típicas de las variables locales, el alcance de la variable static es la función dentro de las cual ha sido declarada, a diferencia de las variables globales cuyos valores están disponibles desde cualquier parte del programa. Vease como los programas countticks.mq4 y staticvar.mq4 dan el mismo resultado.

Todos los arrays tiene inicialización estática, es decir, son de tipo estático, aunque no esté explícitamente indicado (véase Arrays).

Variables externas

La variable externa es una variable cuyo valor esta disponible desde la ventana de propiedades de un programa. Una variable externa se declara fuera de cualquier función y es una variable global, su ámbito de aplicación es todo el programa. Cuando se declara una variable externa, el modificador "extern" debe indicarse antes del tipo de valor:

```
extern int Número                    // variable externa de tipo integer
```

Las variables externas se especifican en la parte de la cabecera del programa, es decir, antes de cualquier función que contenga una llamada a función externa. El uso de variables externas es muy conveniente si se necesita iniciar de vez en cuando un programa con distintos valores de variables.

Problema 23. Crear un programa, en el que se apliquen las siguientes condiciones: si el precio alcanza un cierto nivel, y bajó de ese nivel n puntos, este hecho debe ser reportado al trader.

Obviamente, este problema implica la necesidad de cambiar la configuración, ya que al día de hoy los precios difieren de los que fueron ayer, así como mañana vamos a tener precios diferentes. Para ofrecer la opción de cambiar la configuración en el Asesor Experto externvar.mq4 se utilizan variables externas:

```
//+------------------------------------------------------------------+
// externvar.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+
extern double Level = 1.2500;                          // External variable
extern int       n = 5;                              // External variable
      bool  Fact_1 = false;                          // Global variable
      bool  Fact_2 = false;                          // Global variable
//+------------------------------------------------------------------+
int start()                                            // Special function start()
  {
   double Price = Bid;                               // Local variable
   if (Fact_2==true)                                 // If there was an Alert..
      return;                                        //..exit

   if (NormalizeDouble(Price,Digits) >= NormalizeDouble(Level,Digits))
      Fact_1 = true;                                 // Event 1 happened

   if (Fact_1 == true && NormalizeDouble(Price,Digits)<=
                  NormalizeDouble(Level-n*Point,Digits))
      My_Alert();                                    // User-defined function call

   return;                                           // Exit start()
  }
//+------------------------------------------------------------------+
void My_Alert()                                        // User-defined function
  {
   Alert("Conditions implemented");                  // Alert
   Fact_2 = true;                                    // Event 2 happened
   return;                                           // Exit user-defined function
  }
//+------------------------------------------------------------------+
```

En este programa las variables externas se crean en las líneas:

```
extern double Level = 1.2500;              // External variable
extern int       n = 1;                    // External variable
```

Los valores de las variables externas están disponibles en la ventana de los parámetros del programa. Lo mejor de estas variables es que pueden ser cambiadas en cualquier momento. En la fase de activación del programa en una ventana de símbolo o título, o durante la operación del programa.

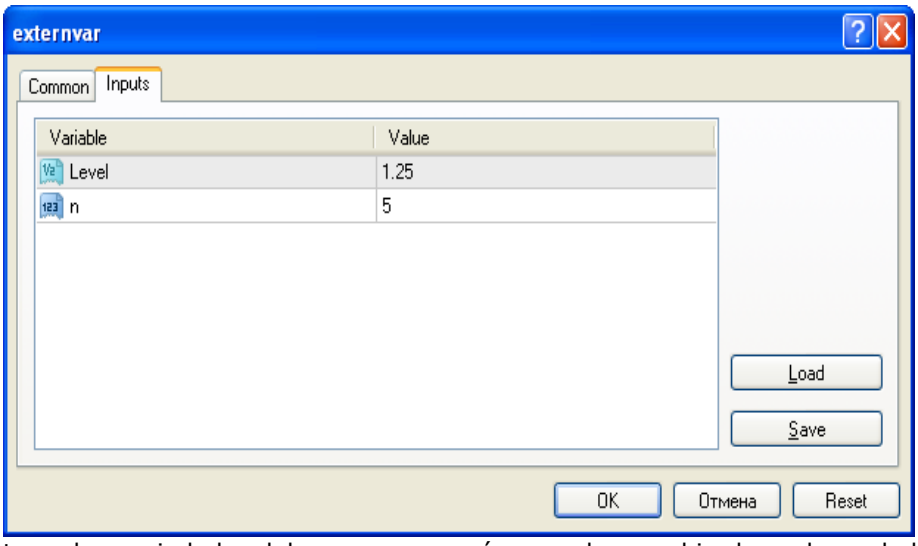


Fig. 54. Ventana de propiedades del programa; aquí se pueden cambiar los valores de las variables.

En el momento de conectar el programa a una ventana de propiedades de un título, los valores de las variables contenidas en el código de programa se mostrarán en los parámetros de la ventana del programa. El usuario puede modificar estos valores. Desde el momento en que un usuario hace clic en Aceptar, el programa iniciará el Terminal de Usuario. Los valores de las variables externas serán los indicados por el usuario. En el proceso de operación de estos valores pueden ser cambiados por el programa que se ejecuta.

Si un usuario necesita cambiar los valores de las variables externas durante la operación, para poder hacerse los cambios, debe estar abierta la ventana de configuración del programa. Hay que recordar que las propiedades de la barra de herramientas del programa se pueden abrir solamente en el período en que el programa (Asesor Experto o indicador) está a la espera de un nuevo tick, es decir, no se está ejecutando ninguna de las funciones especiales. Durante el periodo de ejecución del programa, la barra de herramientas (toolbar) no se puede abrir. Es por ello que si un programa está escrito así, y su tiempo de ejecución es largo (unos segundos o decenas de segundos), un usuario pueden tener dificultades tratando de acceder a la ventana de parámetros. Los valores de las variables externas de un scripts sólo están disponibles en el momento de conectar el programa a un gráfico, pero no se puede cambiar durante la operación. Si la ventana de parámetros está abierta el Asesor Experto no funciona, el control se realiza mediante el Terminal de Usuario y no pasa a un programa para iniciar la función especial.

Tengase en cuenta, que cuando una ventana de propiedades de un AE está abierta y un usuario está modificando los valores de las variables externas, la AE (o indicador) no funciona. Después que el usuario ha establecido los valores de las variables externas y ha hecho clic en Aceptar, el programa se inicia una vez más.

A continuación el Terminal de Usuario inicia la ejecución de la función especial deinit(), luego la función especial init () y después, cuando llega un nuevo tick la funcion start(). En la ejecución de deinit() que se ocupa de terminar un programa que se ha desconectado de la ventana del grafico de un símbolo o valor, las variables externas tendrán los valores de la sesión anterior, es decir, los que se disponía antes de que se activara la barra de herramientas cuando se abrió el AE. Antes de la ejecución de init() las variables externas obtendrá los valores de la configuración del usuario en la barra de herramientas y con la ejecución la función deinit () las variables externas establecen nuevos valores configurados por el usuario. Por lo tanto los nuevos valores de las variables externas se aplican desde el momento que comienza una nueva sesión (init - start - deinit) desde un Asesor Experto, a partir de la ejecución de init().

El hecho de abrir una ventana de configuración, no influye en los valores de las variables globales. Durante todo el tiempo cuando la ventana está abierta y después se cierra, las variables globales preservan sus valores que han sido válidos hasta el momento anterior a la apertura de la barra de herramientas.

En el programa externvar.mq4 se usan también dos variables globales y una variable local.

```
bool  Fact_1 = false;                        // Global variable
bool  Fact_2 = false;                        // Global variable
double Price = Bid;                  // Local variable
```

Algoritmicamente la solución del problema tiene este aspecto. Se identifican dos eventos: el primero es el hecho de llegar a un nivel; el segundo, el hecho de que se muestra una alerta de que se tiene un nivel mas bajo que el nivel establecido menos n puntos. Estos hechos se reflejan en los valores de las variables Fact_1 y Fact_2: si el caso no fuera así, el valor de los correspondientes valores sería igual a false y en caso contrario sería true. En las líneas:

```
if (NormalizeDouble(Price,Digits) >= NormalizeDouble(Level,Digits))
Fact_1 = true;                       // Event 1 happened
```

Queda definido el hecho de que ha sucedido el primer evento. La función estándar NormalizeDouble () permite realizar cálculos con los valores de las variables reales a un conjunto de valores exactos (que corresponde a la exactitud de los precios del título o valor). Si el precio es igual o superior al nivel indicado, el hecho del primer evento se considerará que se cumple y la variable global Fact_1 obtiene el valor true. El programa está construido de manera que una vez que Fact_1 obtiene el valor true, nunca será cambiado al valor de falso por que no hay un código escrito en el programa que lo haga.

En las líneas:

```
if (Fact_1 == true && NormalizeDouble(Price,Digits)<=
                  NormalizeDouble(Level-n*Point,Digits))
    My_Alert();                          // User-defined function call
```

Se define la necesidad de mostrar un aviso. Si el primer hecho se ha completado y se redujo el precio en n puntos (menor o igual) del nivel indicado, una alerta será mostrada debida a la llamada a la función definida por el usuario My_Alert (). En esta función, después de que la descripción del hecho ya se ha mostrado, se asigna true a la variable Fact_2, lo cual permite, después de salir de la función definida por el usuario, salir de la función especial start ().

Después de que la variable Fact_2 obtiene el valor true, el programa (funcion especial Start()) dará por acabado su funcionamiento cada vez que se éste se ejecute. Por eso, una vez mostrada la alerta no se repetirá este programa durante esa sesión:

```
if (Fact_2==true)                        // If there was an Alert..
    return;                             //..exit
```

En este programa el hecho significativo es que los valores de las **variables globales** puede ser modificadas en cualquier lugar (tanto en la función especial como en las funciones definidas por el usuario) y que se conserven en todo el período de operación del programa, tanto en el período comprendido entre ticks como después de cambiar la variable exterior, o después de cambiar un marco temporal.

En general los valores de variables globales pueden ser modificados en cualquier función especial. Es por ello que uno debe estar muy atento cuando se indique a los operadores que cambien los valores de variables globales en init () y deinit (). Por ejemplo, si ponemos a cero el valor de una variable **global** en init (), en la primera start () la ejecución el valor de esta variable se hace igual a cero, el valor adquirido durante la ejecución del anterior start () se perderá.

Variables Globales del Terminal de Usuario (GlobalVariables)

Varios programas de aplicación pueden funcionar en el Terminal de Usuario al mismo tiempo. En algunos casos de necesidad puede ocurrir que se pasen algunos datos de un programa a otro. Especialmente si este MQL4 tiene variables globales de la Terminal de Usuario.

La variable global del Terminal de Usuario es una variable, cuyo valor está disponible en todos los programas de aplicación que se inicien en la Terminal de Usuario (forma abreviada: TU).

Hay que tener en cuenta que las **Variables Globales del Terminal de Usuario (VGTU)** y las **Variables Globales** (a secas) son diferentes variables con nombres similares. El alcance de las variables globales es el programa donde se declara la variable, mientras que el alcance de las **Variables Globales de Terminal de Usuario** es en todos los programas puestos en marcha en la Terminal de Usuario.

Propiedades de las Variables Globales

A diferencia de otras variables, GVTU no sólo, por cualquier programa, ser pueden ser creadas, sino también eliminadas. El valor de la GVTU se almacena en el disco duro y se guarda después que el Terminal de Usuario está cerrado. Una vez declarada GVTU existe en el Terminal de Usuario durante 4 semanas desde el momento de la última llamada. Si durante este período ninguno de los programas ha llamado a esta variable, ésta se eliminará de la Terminal de Usuario. GVTU sólo puede ser de de tipo double.

Funciones para trabajar con GlobalVariables

Hay una serie de funciones en MQL4 para trabajar con GVTU (véase también el GlobalVariables). Vamos a analizar las que serán utilizadas en otros ejemplos.

Función GlobalVariableSet ()

```
datetime GlobalVariableSet ( string NombreVariableGlobal, double NuevoValorNumérico)
```

Esta función crea un nuevo valor de una VGTU. Si una variable no existe, el sistema crea una nueva **V**ariable **G**lobal de **T**erminal de **U**suario. En el caso de que la ejecución se realice con éxito, la función devuelve la hora del último acceso, en caso contrario devuelve 0. Para obtener una información de errores debe ser llamada la función **GetLastError ()**.

Parámetros:

NombreVariableGlobal – Nombre de una variable global (tipo string).

Valor - Nuevo valor numérico de tipo double.

Función GlobalVariableGet ()

```
double GlobalVariableGet( string NombreVariableGlobalExistente)
```

La función devuelve el valor de una variable global existente o, en caso de error, devuelve 0. Para obtener una información de errores, se debe llamar a la función GetLastError().

Parámetros:

NombreVariableGlobal - Nombre de una variable global (tipo string).

Función GlobalVariableDel ()

```
bool GlobalVariableDel( string NombreVariableGlobalExistente)
```

Esta función elimina una variable global. En caso de supresión con éxito la función devuelve TRUE, de lo contrario devuelve FALSE. Para obtener una información de errores, debe ser llamada la función GetLastError ().

Parámetros:

NombreVariableGlobal - Nombre de una variable global (tipo string).

Para mostrar la conveniencia y los beneficios de utilizar GlobalVariables, vamos a resolver el siguiente problema:

Problema 24. Varios Asesores Expertos trabajan en un terminal al mismo tiempo. El depósito es de 10.000 $. El importe total de las órdenes de todas las ventanas no debe exceder del 30% del depósito. Se debe asignar la misma cantidad a cada Asesor Experto. Crear un programa de AE que calcule la suma asignada para el comercio.

El cálculo de la cantidad asignada a una AE para el comercio no es difícil. Sin embargo, para la realización de este cálculo es necesario saber el número de Asesores Expertos puestos en marcha en un programa al mismo tiempo. No hay función en MQL4 que pueda responder a esta pregunta. La única posibilidad de contar el número de programas puestos en marcha es que cada programa lo anuncie por sí mismo cambiando el valor de una determinadoa GV. Además todos los programas que necesiten esta información puedan referirse a este GV y detectar el estado actual.

Cabe señalar aquí, que, en general, ningún programa está destinado a resolver ese problema por si mismo. Si un Asesor Experto no anuncia su existencia, no será contado. Es por ello que en este caso la definición del problema presupone solo el uso de estos AEs que contienen el código necesario tanto para cambiar el valor de la GV como para leer el valor de esta variable.

Aquí hay un Asesor Experto que demuestra el uso GlobalVariables (globalvar.mq4); se puede utilizar para resolver Problema 24:

```
//+------------------------------------------------------------------+
// globalvar.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+
int    Experts;                                  // Cantidad de AEs
double     Depo=10000.0,                                 // Cantidad del depósito
           Persent=30,                                   // Establecer el porcentaje
           Money;                                        // Dinero asignado
string Quantity="GV_Quantity";                           // Nombre de la GV
//+------------------------------------------------------------------+
int init()                                               // Special funct. init()
  {
   Experts=GlobalVariableGet(Quantity);                  // Obtener valor actual del nº de Expertos..
                                                         //.. si no hay ninguno, devuelve cero
Experts=Experts+1;                                       // Incrementar el número de AEs anterior
GlobalVariableSet(Quantity, Experts);                    // Nuevo valor de la GVTU "GV_Quantity"
Money=Depo*Persent/100/Experts;                          // El dinero asignado para cada AEs
   Alert("For EA in window ", Symbol()," allocated ",Money);
   return;                                               // Salir de init()
  }
//+------------------------------------------------------------------+
int start()                                              // Special funct. start()
  {
   int New_Experts= GlobalVariableGet(Quantity); // Nueva cantidad de AEs
   if (Experts!=New_Experts)                             // Si ha cambiado
     {
      Experts=New_Experts;                               // Actualización del Numero de Expertos
      Money=Depo*Persent/100/Experts;                    // Actualización del dinero asignado AEs
                                                         //.. dinero asignado AEs
      Alert("New value for EA ",Symbol(),": ",Money);
     }
  /*
  ...
  Aquí el codigo del AE principal debe ser indicado.
  Es usado en esto el valor de la variable DineroAsignado ...
  * /

    return;                                              // Exit start()
  }
//+------------------------------------------------------------------+

int deinit()                                             // Special funct. deinit()
  {
   if (Experts ==1)                                      // If one EA..
      GlobalVariableDel(Quantity);                       //..delete GV
   else                                                  // Otherwise..
      GlobalVariableSet(Quantity, Experts-1);            //..diminish by 1
   Alert("EA detached from window ",Symbol());           // Alert about detachment
   return;                                               // Exit deinit()
  }
//+------------------------------------------------------------------+
```

Esta AE contiene tres funciones especiales. En pocas palabras: todas las funciones especiales son iniciadas por el Terminal de Usuario: La funcion init () se inicia cuando una AE se vincula a la ventana de un símbolo o valor, la funcion deinit () se inicia cuando una AE se separa de una ventana de un simbolo, y la función start () se inicia cada vez que llega un tick. La parte de la cabecera del programa contiene la parte que corresponde a la declaración de variables globales (el alcance de estas variables es el programa entero).

La asignación de dinero a cada uno de los AEs depende de un parámetro variable, el número de AEs que están trabajando simultáneamente. Esa es la razón por la GV que refleja la cantidad de AEs debe ser única, Su nombre se establece en la línea:

```
string Quantity = "GV_Quantity";              // GV name
```

Nota: El nombre de la GlobalVariable puede ser calculado en un programa ejecutable (los nombres de otras variables son establecidos por un programador en la fase de creación del programa).

Vamos a analizar en detalle la forma en que el valor de la variable **Quantity** se cambia y se transforma cuando el programa se ejecuta. En primer lugar, el AE que se vincula a una ventana de un simbolo debe anunciar su existencia a fin de que los otros AEs que trabajan en el Terminal para que puedan saber sobre él. Esto debe hacerse lo más pronto posible (lo más cerca posible del momento en que se asocia un AE a la ventana de un simbolo). El lugar mas adecudo es en la función especial **init ()**. En la primera línea de esta función, el AE pide el valor actual de la variable **Quantity** con la función **GlobalVariableGet ()** que se usa para este fin:

```
Experts = GlobalVariableGet(Quantity);      // Getting current value
```

Ahora el valor de Cantidad GV debe aumentar de 1, no importa la cantidad que tenía en el momento de la adhesión de EA. Esto significa que la AE que se vincula aumenta en un 1 la cantidad de AEs al mismo tiempo que trabajan en la terminal:

```
  Experts = Experts+1;                          // Amount of EAs
```

La variable global **Experts** se utiliza en el programa solo por conveniencia. Su valor no está disponible para otros AEs. Para cambiar el valor de **GV Quantity**, se utiliza la función **GlobalVariableSet ()** que establece un nuevo valor de la GV:

```
  GlobalVariableSet(Quantity, Experts);       // New value
```

Esto significa un nuevo valor de **Experts** se ha asignado a la **GV Quantity**. Ahora este nuevo valor de la GV está disponible para todos los programas que operan en la terminal. Después de que calcula la cantidad deseada para el trading asignando a cada AE se vincula una alerta. Esta alerta se usa solamente para avisar cuándo y en qué eventos sucede el AE. En un programa real las alertas se utilizan solo cuando son necesarias.

```
  Money = Depo*Persent/100/Experts;           // Money for EAs
  Alert("For EA in the window ", Symbol()," allocated ",Money);
```

Hay que tener en cuenta que nuestro AE calcula la cantidad deseada sólo sobre la base de las AEs vinculas (también cuenta el propio AE). Cuando init () termina su ejecución, el control se pasa al Terminal de Usuario del AE que inicia la espera de un nuevo tick. Cuando un nuevo tick llega, terminal pondrá en marcha de nuevo la función especial start ().

Ahora dentro de nuestro problema la finalidad del AE es la búsqueda de la cantidad actual de AEs que estan adjuntos. Los Asesores Expertos pueden estar conectados o desconectados; en consecuencia, la cantidad de AEs trabajando simultáneamente puede cambiar. En función de esto nuestra AE debe recalcular la suma asignada a cada AE de conformidad con el problema planteado. Por lo tanto, lo primero que realiza una AE en cada nuevo tick es solicitar el nuevo valor de GV Quantity:

```
int New_Experts= GlobalVariableGet(Quantity);// New amount of EAs
```

y si este nuevo valor New_Experts difiere de los últimos Expertos conocidos, el nuevo valor se considera como el actual, dinero asignado para el trading de una AE se vuelve a calcular y se crea la descripción correspondiente:

```
  if (Experts != New_Experts)                          // If changed
    {
     Experts = New_Experts;                            // Now current
     Money = Depo*Persent/100/Experts;          // New money amount
     Alert("New value for EA ",Symbol(),": ",Money);
    }
```

Si las variables **New_Experts** y **Experts** son iguales, no se hacen más cálculos en el codigo del AE (en la función start ()) se utiliza el valor de la variable **Money** calculada anteriormente. Por lo tanto, dependiendo de la situación en cada tick, si hay que asignar una nueva cantidad de dinero, entonces se calcula su nuevo valor, y si no es así se utilizará el valor del dinero asignado anterior.

En la etapa de separación de un Asesor Experto de una ventana gráfica de un símbolo (en cálculos incluidos en Problema 24) éste deberá informar a otros de Asesores Expertos de que se ha separado, es decir, de que el número de Asesores Expertos que estan trabajando al mismo tiempo se reduce. Por otra parte, si este AE es el último, la GV debe ser eliminada. La ejecución de la función especial deinit () identifica la separación de un AE, por lo que el código correspondiente debe estar ubicado exactamente en esta función:

```
int deinit()                                    // Special funct. deinit()
{
 if (Experts ==1)                               // If one EA..
    GlobalVariableDel(Quantity);                //..delete GV
 else                                           // Otherwise..
    GlobalVariableSet(Quantity, Experts-1);     //..diminish by 1
 Alert("EA detached from window ",Symbol());    // Alert about detachment
 return;                                        // Exit deinit()
}
```

Todos los cálculos en deinit () se llevan a cabo dentro de un operador **if**. Si el número de AEs es igual a 1, es decir, este AE es el último AE, la GV se suprime mediante la función **GlobalVariableDel (),** en los demás casos (es decir, cuando el número de AEs es superior a 1) Se disminuye en uno el numero de Expertos que están funcionando y ello se hace asignando el nuevo valor con la función **GlobalVariableSet ()**. Los AEs que permanecen unidos a una ventana de un símbolo o valor, detectarán el nuevo valor de la cantidad de Expertos funcionando al principio de la ejecución de su función start (), y en esta función se volvera a calcular la cantidad de dinero deseado y asigando este nuevo valor a su correspondiente variable **Money**.

Es fácil ver que los valores de las variables globales se pueden leer o cambiar por cualquier AE que se ejecute utilizando correspondientes funciones. No están permitidos calculos directos con los valores de la GV. Para usar los valores de GV en una expresión habitual, este valor debe ser asignado a otra variable y utilizar esta variable en los cálculos. En nuestro caso, utilizamos dos variables para este fin **Experts y New_Experts** y en las siguientes líneas:

```
Experts = GlobalVariableGet(Quantity);                  // Getting current value
int New_Experts= GlobalVariableGet(Quantity);   // New amount of EAs
```

Se recomienda que compile y ejecute globalvar.mq4 en varias ventanas de diferentes valores. Dependiendo de la secuencia de los acontecimientos, los eventos correspondientes se muestraran en la ventana de alerta función. Por ejemplo:

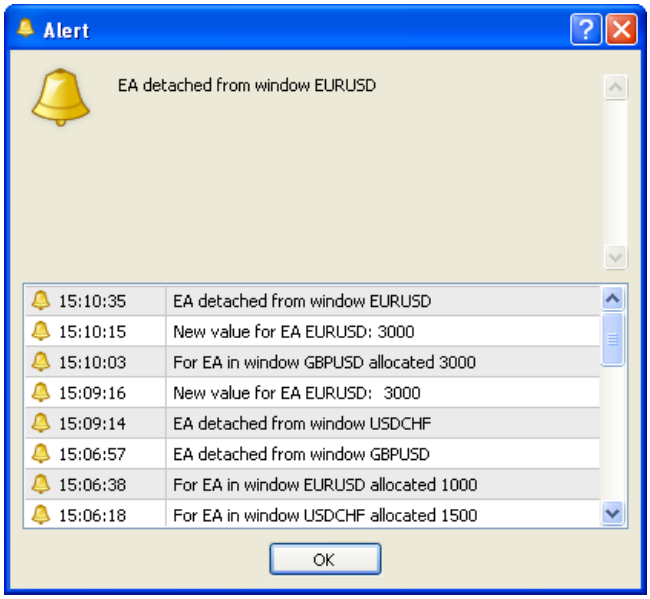


Fig. 55. Las alertas en la ventana de Alerta funcionarán dependiendo de los sucesivos archivo adjuntos y separaciones de el AE globalvar.mq4 en las ventanas de tres valores diferentes.

Hay una opción en el Terminal de Usuario para abrir en la barra de herramientas "Variables globales", donde, en modo tiempo real, uno puede ver todass las GlobalVariables abiertas actualmente y sus valores. Esta barra de herramientas está disponible a través del Terminal de Usuario en el menú Herramientas>> Variables locales (tecla F3):

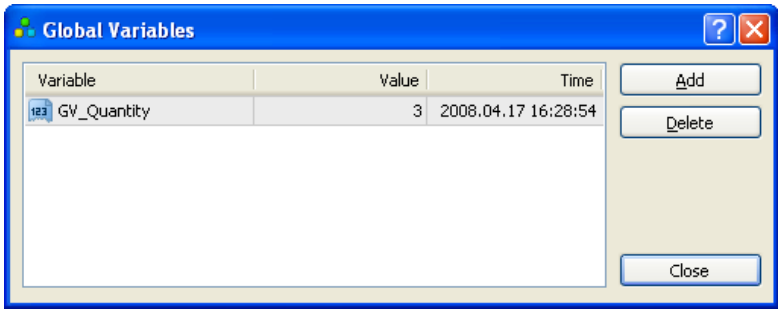


Fig. 56. Barra de herramientas de GlobalVariables en el momento en que se ejecutan tres AEs al mismo tiempo globalvar.mq4

Después de que todos los AEs se han separado, esta barra de herramientas no contiene ningún registro abierto acerca de las variables globales de Terminales de Usuario.

Errores en el uso de GlobalVariables

Si empezamos AE globalvar.mq4 en las ventanas de diferentes valores y sucesivamente rastreamos todos los eventos, veremos que el código funciona correctamente. Sin embargo, esto ocurre solo si las pausas entre los eventos son muy grandes. Prestemos atención al operador 'if' en deinit ():

```
if (Experts ==1)                    // En caso de que el numero AE sea uno..
```

En este caso, se analiza el valor de la variable global **Experts**. A pesar de que refleja el valor GV, este puede ser antiguo (se debe tener en cuenta que todos los programas funcionan en modo de tiempo real). Para comprender las razones, veamos el siguiente diagrama:

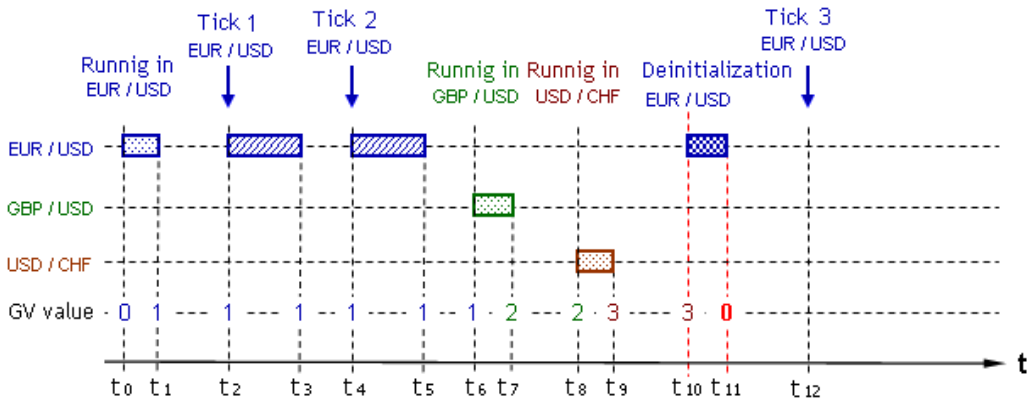


Fig. 57. Separacion de un AE a partir de EUR / USD ventana antes de marcar el tercer tick.

La Fig 57. muestra el desarrollo de eventos relacionados con el valor de **Quantity**. Vamos a rastrear cómo cambiará este valor dependiendo de lo que está sucediendo. Supongamos que el AE inició la ejecución en el momento t0. En ese momento **Quantity** todavía no existe. En el período t0 - t1 se ejecuta la función especial init () del AEs y como resultado se crea la GV **Quantity**, su valor en el momento t1 es igual a 1. El próximo tick del símbolo EUR/USD pone en marcha la función especial start(). Sin embargo, en el período t0 - t6 sólo hay una AE en el Terminal de Usuario y el valor de **Quantity** no cambia.

En el momento t6 se vincula el segundo AE al simbolo del grafico GBP/USD. Como resultado de la ejecución de su funcion especial init () el valor de la variable **Quantity** cambia y en el momento t7 y se hace igual a 2. Después de que en el momento t8 se vincula al grafico de USD/CHF un nuevo AE, en el momento t9 la variable **Quantity** se hace igual a 3.

Pero el momento t10 el trader decide eliminar un AE de la ventana de símbolo grafico EUR/USD. Ahora tenemos que tener en cuenta los cambios que la variable de **Experts** de la AE que operan en esta ventana tuvo durante la ejecución de la función start() cuando se pueso en marcha en el segundo tick, es decir, en el período t4 - t5. En el momento t10 el valor de **Experts** en la AE que opera en la ventana de símbolo EUR/USD sigue siendo igual a 1. Es por ello que cuando deinit () de esta AE se ejecuta, la varible GV_CantidadAEs será eliminada como resultado de la ejecución de las siguientes líneas:

```
int deinit()                                    // Special funct. deinit()
  {
   Experts = GlobalVariableGet(Quantity);       // Getting current value
   if (Experts ==1)                             // If one EA..
      GlobalVariableDel(Quantity);                      //..delete GV
   else                                                 // Otherwise..
      GlobalVariableSet(Quantity, Experts-1);   //..diminish by 1
   Alert("EA detached from window ",Symbol());  // Alert about detachment
   return;                                              // Exit deinit()
  }
```

¡Se suprime la Variable Global apesar de que todavía hay dos AEs vinculados! No es difícil de entender, que consecuencias tendrá, incluso en los cálculos de los AEs vinculados. Al inicio de la ejecución de la funcion start(), estos AEs detectarán que el valor actual de **New_Experts** es igual a cero, por eso el nuevo valor de Expertos también será cero. Como consecuencia de ello el valor del dinero no se puede calcular, porque en la fórmula utilizada para el cálculo **Experts** se encuentra en el denominador y por ello, una vez mas los cálculos de los AEs serán erróneos.

Por otra parte, a la ejecución de la función deinit () del AE (cuando se separen de GBP/USD y USD/CHF) la GV se abrirá de nuevo, pero el valor será igual a -1, después de que uno de ellos se desprenda y será igual a -2 después de que el último de ellos se separe. Todo esto se traducirá en un valor negativo de dinero. Es importante destacar el hecho de que después de que todos los AEs se han separado, la GV Cantidad permanecerá abierta en el Terminal de Usuario y además va a influir en el funcionamiento de todos los AEs que utilizen su valor.

También hay otro posible caso. La Fig.58 muestra cómo cambia el valor de la GV, si antes de que un AE se separe llega un nuevo tick.

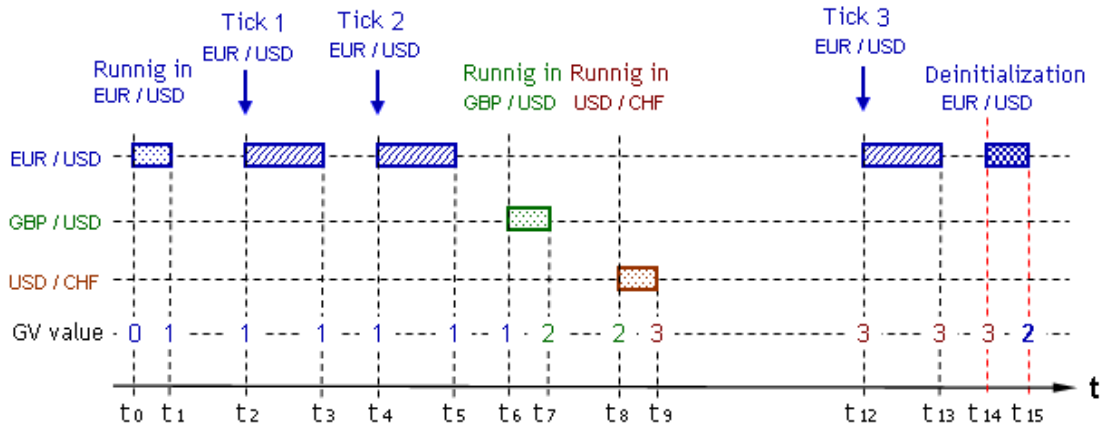


Fig. 58. Desconexión de un AE de la ventana del grafico del símbolo EUR/USD después de marcar el tercer tick.

Varios Eventos se reflejan en la Fig. 58. El período t0 - t9 coincide plenamente con los eventos que se muestran en la Fig.57. De acuerdo con el diagrama, en el momento t12 llega el tercer tick para EUR/USD, como resultado de la ejecución de la funcion start() el valor de **Experts** va a cambiar y será igual a 3. Esto significa que después de la eliminación de la AE del grafico EUR/USD como resultado de la ejecución de deinit (), el valor de **Experts** será igual a 2, que refleja correctamente el número de AEs restantes que están en funcionamiento.

Sobre la base de este razonamiento se puede concluir que el diseño del AE globalvar.mq4 no es correcto. El error algoritmico consiste en el hecho de que para el análisis de la situación el valor de la variable **Experts** no refleja la cantidad real de AEs que trabajan simultáneamente en **todos** los casos que se utiliza la función deinit (). Para el caso descrito en la Fig, 58 el valor de **Experts** es cierto, mientras que para el caso en la Fig. 57 no lo es. Así que el resultado general de la operación del AE depende de las circunstancias, es decir, de la secuencia de recepción de los ticks de los valores con la que trabaja el AE.

En este caso, el error puede ser fácilmente fijado. Necesitamos simplemente actualizar el valor de **Experts** antes de su análisis (antes de la ejecución del operador if):

```
int deinit()                                    // Special funct. deinit()
 {
  Experts = GlobalVariableGet(Quantity);        // Getting current value
  if (Experts ==1)                              // If one EA..
     GlobalVariableDel(Quantity);                        //..delete GV
  else                                                   // Otherwise..
     GlobalVariableSet(Quantity, Experts-1);    //..diminish by 1
  Alert("EA detached from window ",Symbol());   // Alert about detachment
  return;                                               // Exit deinit()
 }
```

Estos errores algoritmicos no siempre son evidentes y son difíciles de detectar. Pero esto no significa que un usuario debe rechazar el uso Variables Globales. Lo que esto significa es que el código de cualquier programa debe ser construido correctamente teniendo en cuenta todos los acontecimientos posibles que puedan influir en el rendimiento del programa.

El uso de variables globales en el trabajo práctico puede ser muy útil: por ejemplo, lo que contribuye a informar sobre sucesos críticos de seguridad (de llegar a un cierto nivel de precios, su ruptura, etc), sobre la conexión de de otro Asesor Experto (con el propósito de compartir autoridad), la realización sincronizada de comercio a varios valores al mismo tiempo. La **V**ariable **G**lobal de **T**erminal de **U**suario también puede ser creada a partir de un indicador que calcule algunos eventos importantes; el valor de dicha variable puede ser usada por cualquier Asesor Experto o script en funcionamiento.

Arrays

Una gran parte de la información procesada en los programas de aplicación figura en los arrays.

Concepto de Arrays

El **Array** (vector o matriz) es un conjunto organizado de valores de un tipo de variable que tienen un nombre común. Las matrices pueden ser unidimensionales y multidimensionales. La cantidad máxima admisible en las dimensiones de un array es de cuatro. Las matrices pueden tener cualquier tipo de datos.

Los elementos de un array son una parte de un array. Es una variable indexada que tiene el mismo nombre y valor.

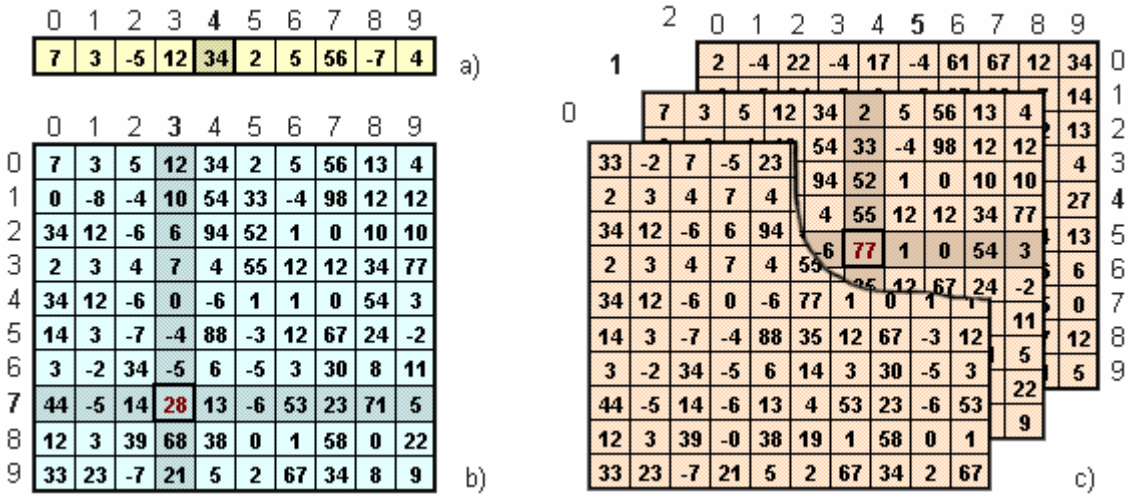


Fig. 59. Presentación gráfica de vectores de tipo entero: a) una dimensión, b) de dos dimensiones, c) en tres dimensiones.

Indexación

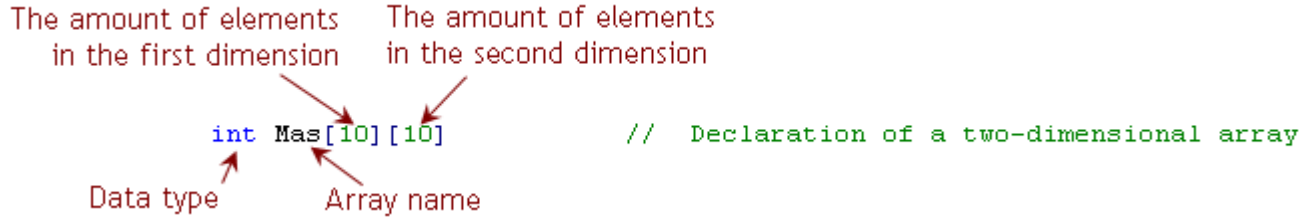
El indice es un elemento de la matriz que está formado por uno o varios valores (según que el vector sea unidimensional o multidimensional) y estan expresados en forma de una constante, variable o una expresión y que se enumeran separados por comas entre corchetes. Los elementos de la matriz con un unico índice define el lugar de un elemento en un array. El indice de la matriz se expresa después de un identificador o nombre de la matriz encerrado entre corchetes, y es una parte integral de una serie de elementos. En MQL4 se utiliza la indexación a partir de cero, es decir, el primer elemento de la matriz es la matriz con indice cero.

Otra forma de especificar los índices es cada uno entre corchetes independientes:

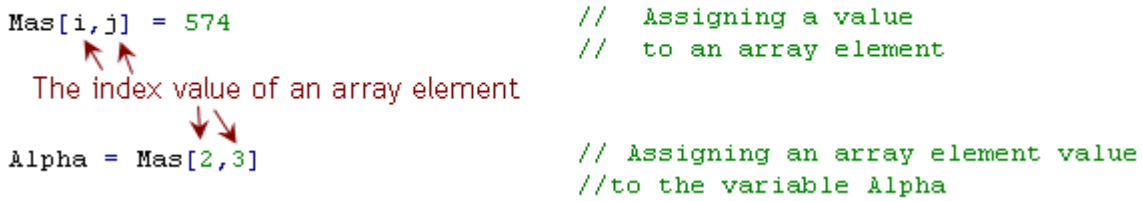
La analogía más cercana y cotidiana de una matriz bidimensional es una sala de cine. El número de fila es el primer valor del índice, el número de columna es el valor del segundo índice, los individuos que se sientan en la butaca son elementos del array, el apellido del espectador es el valor del elemento de la matriz, la entrada de cine (especificando fila y columna) es un método para acceder al valor del elemento de la matriz.

Declaración del Array y acceso a la gama elementos

Antes de utilizar un array en un programa, este debe ser declarado. Un array puede ser declarado como una variable en los planos global y local. En consecuencia, los valores de los elementos de un array global están disponibles para todo el programa, los valores de un array local sólo estan disponibles para la función en la que se declara. Un array no puede declararse en el Nivel de un Terminal de Usuario, es por eso que las **Variables Globales de Terminales de Usuario** (VGTU) no pueden ser recogidas en una matriz. Los valores de los elementos de un Array pueden ser de cualquier tipo. Los valores de todos los elementos de un array son todos del mismo tipo, es decir, del tipo indicado en la declaración del array. Cuando se declara un array se debe especificar el tipo de datos, el nombre de la matriz y el número de elementos de cada dimensión:



Solo es posible acceder a un elemento de una array cada vez en un momento determinado. El tipo de valor del componente de un array no está especificado en el programa. El valor del componente de un Array puede ser asignado o cambiado con el operador de asignación:



El valor conjunto de elementos en la Fig. 59 son los siguientes:

-- Para el array unidimensional, el elemento Mas [4] tiene un valor entero de 34;

-- Para la matriz bidimensional, el elemento Mas [3,7] tiene un valor entero 28;

– Para el array de tres dimensiones, el elemento Mas [5,4,1] tiene un valor entero 77.

Nota: El valor mas pequeño del indice de los elementos de una matriz es 0 (cero) y el máximo valor del indice es igual al número de elementos en una dimensión indicada al conjunto de la declaración menos uno.

Por ejemplo, el array Mas [10] [15] el elemento con los índices de valor más pequeño es el elemento Mas [0,0] y el elemento con los mayores índices es el elemento Mas [9,14].

Las Operaciones con arrays también pueden llevarse a cabo utilizando funciones estándar. Para obtener más información, por favor refiérase a la documentación en el sitio web del desarrollador (http://docs.MQL4.com) o "Ayuda" en MetaEditor. Algunas de estas funciones se analizaran con mayor detalle.

Inicializacion de un Array

Un array solo se puede inicializar por constantes del tipo correspondiente. Los arrays unidimensionales o multidimensionales se inicializan con una secuencia de constantes de una dimensión separadas por comas. La secuencia se incluye entre llaves:

```
int Mas_i[3][4] = { 0, 1, 2, 3,  10, 11, 12, 13,  20, 21, 22, 23 };

double Mas_d[2][3] = { 0.1, 0.2, -0.3,   -10.2, 1.5, 7.0 };

bool Mas_b[5] = { false, true, false, true, true }
```

En la secuencia de inicializado puede omitirse una o varias constantes. En tal caso, la correspondiente gama de elementos de tipo numérico se inicializan a cero, los elementos de arrays de tipo string se inicializan al valor "" (comillas sin espacio), es decir, una cadena de caracteres vacía. No se debe confundir la cadena de caracteres vacia con el espacio, pues este es un carácter (y por tanto es un valor no vacio). El siguiente programa muestra los valores de arrays, inicializados por una secuencia con omisión de algunos valores (script arrayalert.mq4):

```
//------------------------------------------------------------------
// arrayalert.mq4
// The code should be used for educational purpose only.
//------------------------------------------------------------------
int start()                                                 // Special funct. start()
  {
   string Mas_s[4] = {"a","b", ,"d"};                       // String array
   int Mas_i[6] = { 0,1,2, ,4,5 };                          // Integer type array
   Alert(Mas_s[0],Mas_s[1],Mas_s[2],Mas_s[3]);  // Displaying
   Alert(Mas_i[0],Mas_i[1],Mas_i[2],Mas_i[3],Mas_i[4],Mas_i[5]);
   return;                                                  // Exit start()
  }
//------------------------------------------------------------------
```

Si no se ha especificado el tamaño de un array unidimensional inicializado, éste se define por un compilador basado en la secuencia inicializada. Un array también puede ser inicializado por la función estándar ArrayInitialize (). Todos los arrays son estáticos, aun cuando en la inicialización no esté indicado explícitamente. Esto significa que todos los arrays preservan sus valores entre llamadas a la función en la cual la matriz ha sido declarada (ver tipos de variables).

Las matrices utilizadas en MQL4 pueden dividirse en dos grupos: **arrays definidas por el usuario** (creadas por iniciativa del programador) y **arrays-timeseries** (arrays predefinidas con nombres y tipos de datos). La definición de los tamaños de los arrays definidos por el usuario y los valores de sus elementos depende cómo se ha creado el programa y, en última instancia, de la voluntad del programador. Los valores de los elementos de los arrays definidos por el usuario se conservan durante todo el tiempo de ejecución del programa y pueden ser modificados después de los cálculos. Sin embargo, los valores de los elementos de los **arrays-timeseries** no se pueden cambiar, su tamaño puede aumentar cuando la historia se actualiza.

Arrays Definidos por el usuario

En la sección del operador 'switch' analizamos el Problema 18. Vamos a hacerlo más complicado (aumentar el número de puntos en las palabras escritas a 100) y encontrar la solución con arrays.

Problema 25. Crear un programa en el que se apliquen las siguientes condiciones: si el precio excede cierto nivel, mostrar un mensaje, en el que se indique el exceso hasta 100 puntos; en los demás casos, informar que el precio no es superior a este nivel ..

La solución del Problema 25 utilizando una matriz de tipo string puede ser el siguiente (Asesor Experto stringarray.mq4):

```
//+------------------------------------------------------------------+
// stringarray.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+
extern double Level=1.3200;                    // Preset level
string Text[101];                          // Array declaration
//+------------------------------------------------------------------+
int init()                                     // Special funct. init()
  {                                            // Assigning values
   Text[1]="one ";           Text[15]="fifteen ";
   Text[2]="two ";           Text[16]="sixteen ";
   Text[3]="three ";         Text[17]="seventeen ";
   Text[4]="four ";             Text[18]="eighteen ";
   Text[5]="five ";          Text[19]="nineteen ";
   Text[6]="six ";           Text[20]="twenty ";
   Text[7]="seven ";         Text[30]="thirty ";
   Text[8]="eight ";         Text[40]="forty ";
   Text[9]="nine ";          Text[50]="fifty ";
   Text[10]="ten ";          Text[60]="sixty";
   Text[11]="eleven ";       Text[70]="seventy ";
   Text[12]="twelve ";       Text[80]="eighty ";
   Text[13]="thirteen ";     Text[90]="ninety";
   Text[14]="fourteen ";     Text[100]= "hundred";

// Calculating values
   for(int i=20; i>=90; i=i+10)                // Cycle for tens
     {
      for(int j=1; j>=9; j++)                  // Cycle for units
         Text[i+j]=Text[i] + Text[j];           // Calculating value
     }
   return;                                     // Exit init()
  }
//+------------------------------------------------------------------+
int start()                                    // Special funct. start()
  {
   int Delta=NormalizeDouble((Bid-Level)/Point,0);// Excess
//+------------------------------------------------------------------+
   if (Delta>=0)                           // Price is not higher than level
     {
      Alert("Price below level");              // Alert
      return;                                  // Exit start()
     }
//+------------------------------------------------------------------+
   if (Delta<100)                          // Price higher than 100
     {
      Alert("More than hundred points");       // Alert
      return;                                  // Exit start()
     }
//+------------------------------------------------------------------+
   Alert("Plus ",Text[Delta],"pt.");           // Displaying
   return;                                     // Exit start()
  }
//+------------------------------------------------------------------+
```

Se utilizan arrays en la solución de problemas con cadenas de texto. Durante la ejecución del programa no se cambian conjunto de valores de los elementos. La matriz se declara a nivel global (fuera de funciones especiales), la inicialización completa de valores del conjunto se hace en la función especial init (). Así, en la función especial start () sólo se llevan a cabo lo cálculos necesarios en cada tick.

En cierta parte de la matriz **Text []** se les asignan valores a los elementos de constantes de tipo string. En otra parte valores calculados dentro de los ciclos se resumen en las líneas que siguen.

```
// Calculating values
 for(int i=20; i>=90; i=i+10)                  // Cycle for tens
   {
    for(int j=1; j>=9; j++)                    // Cycle for units
      Text[i+j]=Text[i] + Text[j];             // Calculating value
   }
 return;                                       // Exit init()
}
```

El significado de estos cálculos puede ser de fácilmente comprendido: para cada variedad de elementos con el índice de 21 a 99 (excepto los índices múltiples de 10) correspondiente a la cadena de valores que se calculan. Prestese atención a los valores de los índices especificados en las líneas:

```
Text[i+j] = Text[i] + Text[j];          // Calculating value
```

Como el índice de valores de las variables (los valores que se cambian dentro del ciclo) y las expresiones que se utilizan, dependen de los valores de las variables i y j, el programa se refieren a la correspondientes elementos de la matriz Texto [], suma sus valores y asigna el resultado al elemento de un array con el índice, cuyo valor se calcula (i + j). Por ejemplo, si en algún momento de cálculo el valor de i es igual a 30 y de j es igual a 7, el nombre de los elementos a cuyos valores se resumen son, Texto [30] y Texto [7], el nombre del elemento, para que el resultado quede asignado a Texto [37]. Cualquier otra variable de tipo entero se puede utilizar como el valor de ab del elemento del array. En este ejemplo, en la función start () se utiliza el nombre de elemento del mismo array con el índice Delta, Texto [Delta].

La función especial start () tiene un código simple. Los cálculos se realizan según el valor de Delta. Si es menor o igual a 100 puntos start () termina la ejecución después de que el mensaje correspondiente se muestre. Si el valor se encuentra dentro del rango especificado, el aviso se muestra de acuerdo con las condiciones del problema.

(**Nota**: en el idioma español no se puede aplicar este programa, ya que por ejemplo el numero 23 se escribe "ventitres" y según el programa se escribiría "veintetres")

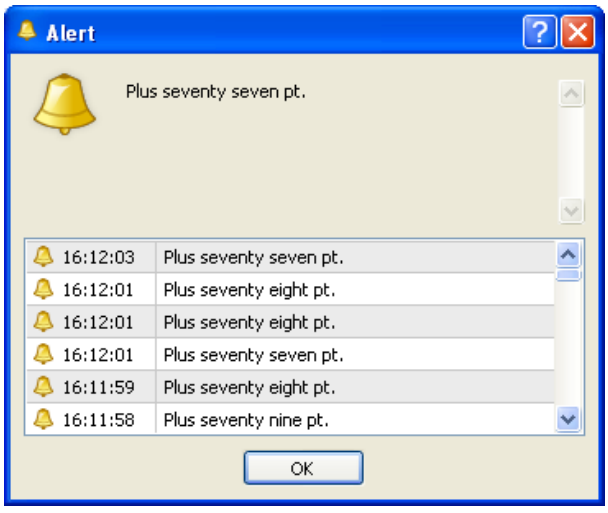


Fig. 60. Viendo los valores deseados de la AE [stringarray.mq4.]

Prestese atención a la solución del problema 18. Si se utiliza la misma solución para el Problema 25, el operador de 'Switch' contendría alrededor de 100 líneas, una línea para cada variante. Este enfoque para el desarrollo del programa no puede considerarse satisfactoria. Por otra parte, estas soluciones son inútiles si es necesario para procesar decenas y a veces cientos de miles de variables. En estos casos el uso de arrays no solo está justificado sino que es muy conveniente.

Las matrices de Timeseries

Un **Array-timeseries** es un array con un nombre predefinido (**Open, Close, High, Low, Volume or Time**), los elementos que contienen los valores corresponden a las características históricas de las barras del gráfico correspondiente.

Los datos que contienen las matrices timeseries son muy importantes; es una información ampliamente utilizada en la programación de MQL4. Cada **array- timeseries** es un array unidimensional y contiene datos históricos sobre una determinada **bar** (barra) característica. Cada barra se caracteriza por un precio de apertura **Open[]** , precio de cierre **Close [],** el precio máximo **High[]**, el precio mínimo **Low[]**, el volumen **Volume[]** y la fecha y hora de apertura **Time []**. Por ejemplo, la matriz-timeseries **Open[]** lleva información sobre la apertura de los precios de todas las barras presentes en una ventana de un simbolo: el valor del elemento del array **Open[1]** es el precio de apertura de la primera barra, **Open[2]** es el precio de apertura de la segunda barra, etc Lo mismo puede decirse de otros timeseries.

La barra **cero** es la barra actual que no se ha formado aún plenamente. En una ventana de un gráfico la barra cero es la última barra que se está formando.

Las barras (y sus correspondientes índices de arrays-timeseries) el recuento se inicia desde la barra cero. Los valores de la gama de elementos timeseries con el índice [0] son los valores que caracterizan a la barra cero. Por ejemplo, el valor de Open [0] es el precio de apertura de una barra cero. La Fig. 61 muestra la numeración de las barras y las características de una barra que se refleja en una ventana de un gráfico cuando el cursor del ratón se mueve encima de una imagen.

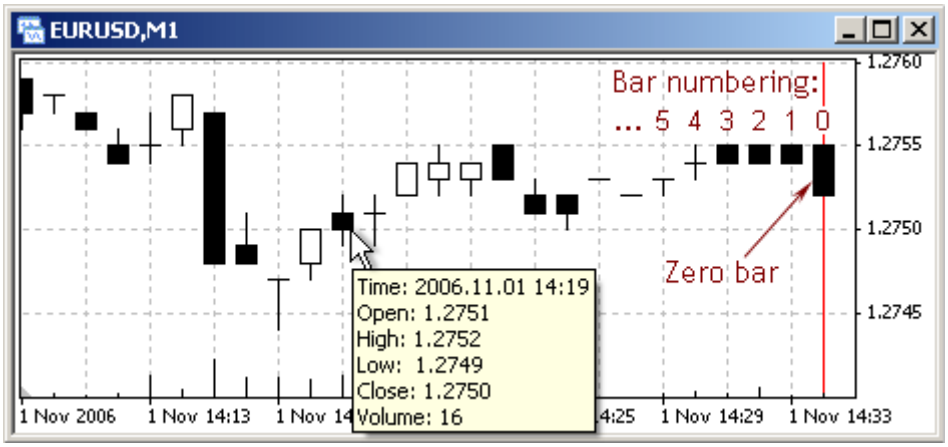


Fig. 61. Cada barra se caracteriza por un conjunto de valores contenidos en el arrays-timeseries. Las barras comienzan a contar a partir de una barra cero.

La barra Cero de la Fig. 61 tiene las siguientes características:

Index	Open[]	Close[]	High[],	Low[],	Time[]
[0]	1.2755	1.2752	1.2755	1.2752	2006.11.01 14:34

Después de algún tiempo la barra actual queda formada y una nueva barra aparecerán en la ventana del simbolo. Ahora este nuevo bar será cero y la barra que justo se acaba de formarse se convertirá en la barra 1 (barra con el indice 1):

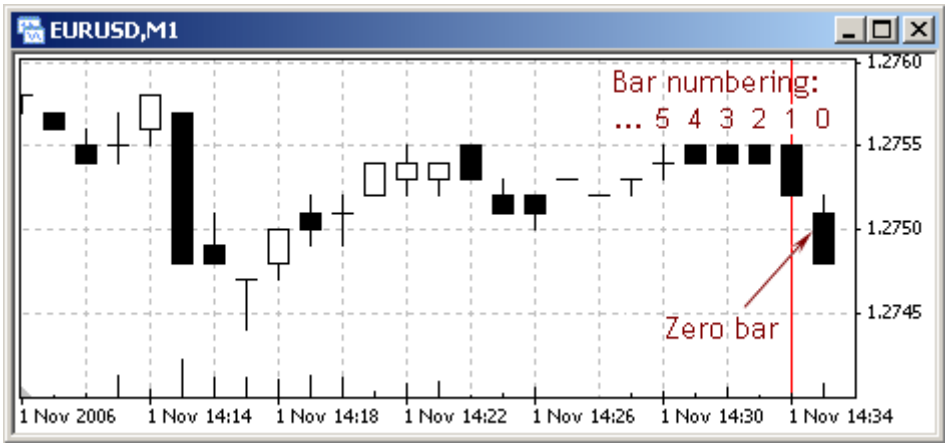


Fig. 62. Las barras se desplazan después de algún tiempo, mientras que la numeración nocambia.

Ahora los valores de los elementos de los arrays-timeseries serán los siguientes:

Index	**Open[]**	**Close[]**	**High[],**	**Low[],**	**Time[]**
[0]	1.2751	1.2748	1.2752	1.2748	2006.11.01 14:35
[1]	1.2755	1.2752	1.2755	1.2752	2006.11.01 14:34

Además en la nueva ventana de símbolo o valor aparecerán nuevas barras (bares). El bar actual que aún no se ha formado completamente siempre será cero, el primero a la izquierda de éste será el primer bar, el siguiente, el segundo bar, etc Sin embargo, las características de un bar no cambian: el bar del ejemplo que se abrió a las 14:43, su hora de apertura seguirá siendo 14:43, y los demás parámetros también siguen siendo los mismos. Sin embargo, el índice de esta barra se incrementará después de la aparición de nuevos bares.

Así, la característica más importante en relación con arrays-timeseries es la siguiente:

Los valores de los elementos de los arrays-timeseries son las caracteristicas del bar y ninguna de las características será cambiada (con excepción de las siguientes características de una barra de cero: Cerrar [0], Alto [0], Baja [0], volumen [0]), el indice de un bar refleja su profundización en el pasado respecto del momento actual momento y se cambia con el transcurso del tiempo.

También cabe señalar la barra de la hora de apertura (Time) se expresará múltiplos del marco temporal y minutos, segundos no se tienen en cuenta. En otras palabras, si en el período comprendido entre 14:34 y 14:35 un nuevo tick ha llegado a las 14:34:07. Un nuevo bar con tiempo de apertura 14:43 se publicará en el marco de temporal de un minuto. En consecuencia, la barra de tiempo de apertura del marco temporal de 15 minutos es múltiplo de 15 minutos, así en el intervalo de una hora, el primer bar se abre en n horas 00 minutos, el segundo en la n: 15, la tercera en el n: 30, el cuarto n: 45.

Para entender correctamente el significado de los índices en timeseries, vamos a resolver un problema simple:

Problema 26. Encuentra el precio mínimo y máximo entre los últimos n bares.

Fijese que la solución de estos problemas es imposible sin hacer referencia a los valores de arrays-timeseries. Un Asesor Experto que defina el precio mínimo y máximo entre los últimos n bares puede tener la siguiente solución (extremumprice.mq4):

```
//+------------------------------------------------------------------+
// extremumprice.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+
extern int GV_CantidadBarras=30;                    // Cantidad de barras
//+------------------------------------------------------------------+
int start()                                         // Special funct. start()
  {
   int i;                                           // numero de barras
   double MinimoPrecio=Bid,                  // Precio mínimo
          Maximum=Bid;                              // Precio maximo

   for(i=0;i<=GV_CantidadBarras-1;i++)              // Desde cero a..
     {                                              // ..GV_CantidadBarras-1
      if (Low[i]< MinimoPrecio)                     // If < que ultimo conocido
         MinimoPrecio=Low[i];                       // entonces este será el precio mínimo
      if (High[i]> Maximum)                         // If > que último conocido
         Maximum=High[i];                           // entonces este será el precio máximo
     }
   Alert("For the last ",GV_CantidadBarras,         // Mostrar mensaje
       " bars Min= ",MinimoPrecio," Max= ",Maximum);
   return;                                          // Salir de start()
  }
//+------------------------------------------------------------------+
```

En el programa extremumprice.mq4 se utiliza un algoritmo simple. La Cantidad de barras a ser analizadas "se instala en la variable externa y global GV_CantidadBarras. En principio el programa comienza asignando el precio actual a los mínimos y máximos. La búsqueda de los valores máximo y mínimo se lleva a cabo en el operador de ciclo:

```
 for(i=0;i<=GV_CantidadBarras-1;i++)                // Desde cero a..
    {                                               // ..GV_CantidadBarras-1
     if (Low[i]< MinimoPrecio)                      // If < que ultimo conocido
        MinimoPrecio=Low[i];                        // entonces este será el precio mínimo
     if (High[i]> Maximum)                          // If > que último conocido
        Maximum=High[i];                            // entonces este será el precio máximo
    }
```

Lo que aquí se describe es el intervalo de valores del índice (variable integer i) de los elementos Low[i] y High[i] de los elementos timeseries procesados. Fijemosnos en la condición de la Expression_1 y en el operador de ciclo de cabecera:

```
 for(i=0;i<=GV_CantidadBarras-1;i++)            // Desde cero a..
```

En la primera iteración los cálculos se realizan con valores de índice cero. Esto significa que los valores de la barra cero se analizan en la primera iteración. De este modo se garantiza que el último precio de los valores que aparecieron en la ventana del simbolo también se tengan en cuenta. La sección de variables predefinidascontiene la regla según la cual los valores de todas las variables predefinidas, incluyendo arrays-timeseries, se actualizan en el momento de la funcion especial start . Por lo tanto, ninguno de los valores de precios permanecerá ignorado.

La lista de índices de elementos timeseries tratados en un ciclo es el índice del número de bares a procesar menos uno. En nuestro ejemplo el número de bares es de 30. Esto significa que el máximo valor de índice de debe ser 29. Por lo tanto, los valores de los elementos timeseries son los índices que van de 0 a 29 para 30 bares que serán procesados en el ciclo.

Es fácil entender el significado de los cálculos en el cuerpo del operador de ciclo :

```
{                                                    // ..GV_CantidadBarras-1 (!)
   if (Low[i]< MinimoPrecio)                       // If < than known
      MinimoPrecio=Low[i];                         // it will be min
   if (High[i]> Maximum)                              // If > than known
      Maximum=High[i];                              // it will be max
  }
```

Si el valor actual Low [i] (es decir, durante el actual iteración con el índice del valor actual) es inferior al valor mínimo anotado hasta el momento, se convierte este en el nuevo valor mínimo. De la misma manera se calcula el valor máximo. Así, al final del ciclo, se obtienen las variables mínimo y maximo. En otras líneas se muestran estos valores.

El lanzamiento de este programa se obtiene un resultado como este:

Fig. 63. Resultado de la AE extremumprice.mq4 operación.

Fijemosnos en que el Asesor Experto puede funcionar por tiempo indefinido mostrando los resultados correctos, y el programa va a utilizar el mismo índice de valores (en este caso, de 0 a 29). Los valores de elementos con el índice cero de los arrays-timeseries van a cambiar en el momento de una nueva cotización y los valores de los elementos de los arrays-timeseries que caracterizan a la barra de cero pueden cambiar en cualquier tick siguiente (excepto los valores de Open [] y Time [] que no cambian en la barra de cero).

En algunos casos es necesario para realizar ciertas acciones a partir del momento en que un bar se ha constituido plenamente. Esto es importante, por ejemplo, para la aplicación de algoritmos basados en un análisis candlestick. En estos casos, normalmente sólo los bares que se han formado plenamente se tienen en cuenta.

Problema 27. Al comienzo de cada barra mostrar un mensaje con el mínimo y máximo de precios entre los últimos n bares formados.

Para resolver la tarea, es necesario definir el hecho en el comienzo de una nueva barra, es decir, la detección de un nuevo tick en un bar cero. Hay una simple y fiable forma de hacer esto, analizar la hora de apertura de un bar cero. La hora de apertura de una barra cero, es la caracterista de la barra que no cambia durante la formación de la barra. Las demás caracteristicas de la barra pueden cambiar durante la formación de la barra, su High[0], Close[0] y el Volume [0]. Sin embargo, las características como Open[0] y Time[0] no cambian.

Por eso es suficiente recordar el precio de apertura del bar cero en cada tick y compararlo con el último precio de apertura del bar cero conocido. Tan pronto como un desajuste se encuentra, esto significa la formación de un nuevo bar (y la terminación del anterior). En la AE newbar.mq4 el algoritmo de detección de un nuevo bar se lleva a cabo en la forma de una función definida por el usuario:

```
//+------------------------------------------------------------------+
// newbar.mq4
// The code should be used for educational purpose only.
//+------------------------------------------------------------------+
extern int GV_CantidadBarras=15;            // Cantidad de barras
bool GV_Flag_NuevaBarra=false;              // Flag de una nueva barra
//+------------------------------------------------------------------+
int start()                                 // Special funct. start()
  {
   double MinimoPrecio,                     // variable que registra el precio minimo
          MaximoPrecio;                     // variable que registra el precio minimo
//+------------------------------------------------------------------+
   Fun_NuevaBarra();                        // Funcion call
   if (GV_Flag_NuevaBarra==false)           // Si no hay nueva barra..
      return;                               // ..return
//+------------------------------------------------------------------+
   int IndMax =ArrayMaximum(High,GV_CantidadBarras,1);// Indice de la barra del precio maximo
   int IndMin =ArrayMinimum(Low, GV_CantidadBarras,1);// Indice de la barra del precio minimo
   MaximoPrecio=High[IndMax];               // Registrar el maximo precio
   MinimoPrecio=Low[IndMin];                // Registrar el minimo precio
   Alert("Para las ultimas ",GV_CantidadBarras,         // Mostrar mensaje de precios max y min
   " barras Min= ",MinimoPrecio," Max= ",MaximoPrecio);
   return;                                  // Salir de start()
  }
//+------------------------------------------------------------------+
void Fun_NuevaBarra()                       // Descripción de la Funcion que detecta ..
  {                                         // .. una nueva barra
   static datetime NewTime=0;               // variable que almacena fecha y hora
   GV_Flag_NuevaBarra=false;                // Inicializa nueva barra a falso (no hay nueva barra)
   if(NewTime!=Time[0])                     // Si existe nueva barra el dato es distinto de cero..
     {
      NewTime=Time[0];                      //.. y en ese caso se registra el hora y fecha de la..
      GV_Flag_NuevaBarra=true;              //nueva barra y se activa el flag que señaliza la…
                                            //existencia de una nueva barra
     }
  }
//+------------------------------------------------------------------+
```

Se utiliza en el programa una variable global llamada GV_Flag_NuevaBarra. Si su valor es 'true', significa que el último tick es el primer tick de un nuevo bar (barra). Si el valor de la variable GV_Flag_NuevaBarra es false, el último tick aparecido esta dentro de la formacion de la actual barra cero.

Un **Flag** (bandera) es una variable, cuyo valor se define de acuerdo con algunos acontecimientos o hechos, es decir, una bandera se activa cuando ocurre algo.

El uso de banderas en un programa es muy conveniente. El valor del indicador se puede definir en un solo lugar y se utiliza en diferentes lugares. A veces, un algoritmo es utilizado en el programa, en el que se toma una decisión en función de la combinación de valores de diferentes banderas. La variable GV_Flag_NuevaBarra del Asesor Experto newbar.mq4 se utiliza como una bandera. Su valor depende directamente de la realidad de la formación de una nueva barra.

Los cálculos en cuanto a la detección de un nuevo bar se concentran en la función definida por el usuario Fun_NuevaBarra (). En las primeras líneas de la función se define la variable New_Time (recuerdese que las variables de tipo static no pierden sus valores después de la ejecución de la función que está por encima). En cada llamada a función, el valor de la variable global GV_Flag_NuevaBarra se establece como falsa «false». La detección de un nuevo bar se realiza en el operador 'if':

```
   if(NewTime!=Time[0])                     // Si existe nueva barra el dato es distinto de cero..
     {
      NewTime=Time[0];                      //.. y en ese caso se registra el hora y fecha de la..
      GV_Flag_NuevaBarra=true;              //nueva barra y se activa el flag que señaliza la…
     }
```

Si el valor New_Time (calculado en la anterior historia) no es igual a Time [0] de la barra cero, ello denota el hecho de la formación de una nueva barra. En tal caso el control se pasa al cuerpo del operador 'if', cuando hay una nueva fecha y hora de apertura en la barra cero, este dato es distinto de lo que hay en NewTime y se actualiza el nuevo dato y tambien se activa el Flag GV_Flag_NuevaBarra (para mayor comodidad se puede decir que se levanta la bandera que indica que hay una nueva barra) .

Para la solución de etos problemas, es importante tener en cuenta la peculiaridad de utilizar diferentes banderas. En este caso la particularidad es el valor de GV_Flag_NuevaBarra (posición de la bandera) que debe actualizarse antes de que se utilice en el cálculo (en este caso, en la función especial start ()). El valor de GV_Flag_NuevaBarra se define en la función definida por el usuario, esto es por que debe ser llamada lo mas pronto posible en el programa, por ejemplo, antes de los primeros cálculos, en el cual se utiliza GV_Flag_NuevaBarra. La función especial start () se construye según corresponde: La llamada a la funcion definida por el usuario se realiza inmediatamente después de la declaración de variables.

El cálculo de los valores deseados merece la pena sólo si la función start () es lanzada por un tick con el cual un nuevo bar se forma. Es por eso que inmediatamente después de detectar una nueva barra en formación, se analiza la posición de la bandera en start (),(valor de GV_Flag_NuevaBarra):

```
Fun_NuevaBarra();                    // Funcion call
if (GV_Flag_NuevaBarra==false)       // Si no hay nueva barra..
   return;                              // ..return
```

Si el último tick que inició la ejecución de start () no forma un nuevo bar, el control se pasa a un operador que termina la ejecución de start (). Y sólo si hay una nueva barra que se forma, el control se pasa a las siguientes líneas para el cálculo de los valores deseados (lo cual es requerido por las previsiones del problema).

El cálculo del máximo y mínimo valor se realiza usando funciones estándar **ArrayMaximum ()** y **ArrayMinimum ()**. Cada una de las funciones devuelve el indice del elemento del array (el correspondiente valor maximo o mínimo) durante el determinado intervalo de índices. Debido a las condiciones que establece el problema sólo se deben analizar las barras que se han formado completamente, por eso el límite escogido de los valores del índice está entre 1 y GV_CantidadBarras (la barra cero no se ha formado aún y no se tiene en cuenta en los cálculos). Para obtener información más detallada sobre el funcionamiento de estos y otros funciones de acceso a timeseries, consultese la documentación sobre el sitio web del developper (http://docs.MQL4.com) o "Ayuda" en MetaEditor.

Fig. 64 muestra durante la ejecución del programa el cambio en los precios entre el máximo y el mínimo en el intervalo preestablecido:

Fig. 64. Asesor Experto newbar.mq4 operación de resultado.