MetaQuotes Language 4

В търговския терминал **MetaTrader 4** е вграден нов програмен език за писане на търговски стратегии **MetaQuotes Language 4** (**MQL 4**). Този език позволява да се пишат собствени програми – експертни системи (Expert Advisors), които способстват за автоматизирането на търговските процеси и за реализация на собствени търговски стратегии. Освен това, с помощта на MQL 4 могат да се създават собствени технически индикатори (Custom Indicators), скриптове (Scripts) и библиотеки на функциите (Libraries).

Синтаксисът на **MetaQuotes Language 4** до голяма степен прилича на синтаксиса на програмния език C. Той е лесен за изучаване и използване. MQL 4 включва голям брой функции, необходими за анализ на текущите и старите котировки, основните аритметични и логически операции. В езика MQL 4 са вградени също основните индикатори и команди за отваряне на позиции, както и за управлението на позициите.

За писане на кода на програмата се използва текстовият редактор на експертните системи MetaEditor 4, който маркира изразите на езика MQL 4 с различни цветове, като по този начин позволява на потребителя по-лесно да се ориентира в текста на експертната система. MetaQuotes Language Dictionary се използва като информационна система за езика MQL 4. Този кратък указател съдържа разпределени по категории функции, операции, резервирани думи и други програмни изрази, които улесняват разпознаването на всеки отделен елемент на програмния език MQL 4.

Програмите, написани на **MetaQuotes Language 4** имат различни свойства и предназначение:

- **Expert Advisors** — механична търговска система (МТС), свързана с определена графика. Експертната система може да работи не само в режим на информиране за възможността за извършване на сделки, но и да извършва сделки по търговската сметка автоматично, изпращайки ги направо към търговския сървър. Както в повечето информационни системи, търговският терминал MetaTrader 4 поддържа тестване на стратегиите въз основа на историческите данни чрез определяне на входно-изходните точки върху графиката.

- **Custom Indicators** — аналог на технически индикатор. С други думи, Custom Indicators позволяват да се създават технически индикатори в допълнение към вече вградените индикатори в търговския терминал MetaTrader 4. Подобно на вече вградените в терминала индикатори, потребителските индикатори не могат да се използват за автоматична търговия. Те са предназначени само за осъществяване на аналитичните функции.

- **Scripts** — програми, предназначени за еднократно извършване на определени действия. За разлика от експертните системи, скриптовете се стартират не въз основа на тиковете, а въз основа на съответните заявки.

- **Libraries** — библиотеки на потребителските функции, предназначени за съхраняване на често използваните блокове от потребителските програми.

Синтаксис на езика

Формат

Интервалите, символите за табулация, нов ред и преминаване към нова страница се използват като разделители. При писане на собствени експертни системи може да се използва неограничен брой такива символи. Препоръчително е да се използват символите за табулация, за да е по-четлив крайният текст на експертната система.

Коментари

Многоредовите коментари започват със символите /* и завършват с */. Тези коментари не могат да се вграждат в други коментари.
Едноредовите коментари, от своя страна, започват със символите // и завършват със символа за нов ред. Те могат да се вграждат в многоредовите коментари.

Коментарите могат да се слагат навсякъде, където има възможност за вмъкване на интервали. Коментарите допускат неограничен брой интервали.

Примери:

```
// Едноредов коментар

/*  Многоре-
    дов             // Вграден едноредов коментар
    коментар
*/
```

Идентификатори

Идентификаторите се използват като имена на променливите и функциите. Дължината на идентификатора не може да превишава 31 знака.

Допустими символи: числата 0-9, латински големи и малки букви a — z, A - Z, които се разпознават като различни символи, долна черта (_). Първият символ на идентификатора не може да бъде число. Идентификаторът не може да съвпада с резервирана дума.

Примери:

```
NAME1 name1 Total_5 Paper
```

Резервирани думи

Изброените по-долу идентификатори се определят като резервирани думи, на всяка от които съответства определено действие, и по друг начин не могат да се използват:

Типове данни	Класове памет	Оператори	Други
bool	extern	break	false
color	static	case	true
datetime		continue	
double		default	
int		else	
string		for	
void		if	
		return	
		switch	
		while	

Типове данни

Всички данни (променливи и константи) имат свързан с тях тип. Всеки тип определя какви операции са приложими към съответните данни и как тези операции се интерпретират. Към основните типове данни се отнасят:

- цели (int)
- логически (bool)
- редове (string)
- с плаваща точка (double)
- цвят (color)
- дата и време (datetime)

Типовете color и datetime се използват само за представяне и въвеждане на параметрите, които се задават с помощта на таблицата на свойствата на експертната система или потребителския индикатор ("Input parameters"). Типовете данни color и datetime се представят като цели числа. Целите типове, както и типовете данни с плаваща точка се наричат аритметични (числови) типове.

Използва се само неявно преобразуване на типовете. При преобразуване приоритетът на типовете данни се определя по реда на нарастване:

```
int  (bool,color,datetime);
double;
string;
```

Преди извършването на операциите (с изключение на операциите за присвояване), се осъществява преобразуване към онзи тип данни, който има най-голям приоритет. Преди извършването на операциите за присвояване се осъществява преобразуване към целевия тип данни.

Цели константи

Десетични: числата 0—9; първото число не трябва да е 0.

Примери:

```
12, 111, -956 1007
```

Шестнадесетични: числата 0-9, буквите a - f или A - F за стойностите 10-15; започват с Ox или OX.

Примери:

```
0x0A, 0x12, 0X12, Ox2f, OxA3, OXa3, 0X7C7
```

Целите константи могат да приемат стойности от -2147483648 до 2147483647. Ако цялата константа превиши този диапазон, това може да доведе до неочаквани резултати.

Символни константи

Всеки символ, поставен в единични кавички, или шестнадесетичен ASCII-код на символа във вида '\x10' представлява символна константа и има тип int. Някои символи, такива като, единични кавички ('), двойни кавички ("), въпросителна (?), обратна наклонена черта (\), както и управляващите символи могат да се представят с комбинация от символи, започващи с обратна наклонена черта (\), в съответствие с приложената по-долу таблица:

нов ред	NL (LF)	\n
хоризонтална табулация	HT	\t
край на реда	CR	\r
обратна наклонена черта	\	\
единични кавички	'	\'
двойни кавички	"	\"
ASCII-код на символа в шестнадесетична система	hh	\xhh

Ако след обратната наклонена черта следва символ, който се различава от изброените, това може да доведе до неочаквани резултати.

Примери:

```
int a = 'A';
int b = '$';
int c = '©';     // код 0xA9
int d = '\xEA'; // код на символа ®
```

Логически константи

Логическите константи приемат стойностите true или false, числовото съответствие на които съответно е 1 или 0. Могат да се използват синонимите True, TRUE, False и FALSE.

Примери:

```
bool a = true;
bool b = false;
bool c = 1;
```

Константи с плаваща точка

Константите с плаваща точка се състоят от цяла част, точка (.) и дробна част. Цялата част, както и дробната част представляват последователност от десетични числа.

Примери:

```
double a = 12.111;
double b = -956.1007;
double c = 0.0001;
double d = 16.0;
```

Съобразяване с ограниченията, наложени от фирмата Microsoft: типът double съдържа 64 бита - 1 бит за знака, 11 бита за експонентата и 52 бита за мантисата. Пределите на изменението са от -1.7 * e-308 до 1.7 * e308, с точност до 15 знака след точката.

Стрингови константи

Стринговата константа представлява последователност от символи на ASCII-кода, поставена в двойни кавички: "Character constant".

Стринговата константа е масив от символи, поставен в кавички. Типът на стринговата константа е string. Всяка стрингова константа се пази в отделно място на паметта. Ако в съответния ред трябва да се въведе символът двойни кавички ("), то преди него трябва да се сложи обратна наклонена черта (\). Във всеки ред могат да се въвеждат специални стрингови константи, преди които трябва да се сложи обратна наклонена черта (\). Дължината на стринговата константа е от 0 до 255 символа. Ако дължината на стринговата константа превишава максималната, излишните символи отдясно се премахват, а на екрана ще се покаже съответното предупреждение.

Примери:

```
"This is a character string"
"Това е стрингова константа"
"Знак за запазена марка\t\xA9"
"този ред съдържа символа нов ред \n"
"A" "1234567890" "O" "$"
```

Цветови константи

Цветовите константи могат да се представят по три различни начина: литерално представяне; целочислено представяне; име (само за именуваните Web цветове).

Литералното представяне се състои от три части, които представляват числовите стойности на интензивността на трите основни компонента на цветовете - червен (red), зелен (green), син (blue). Константата започва със символа C и се поставя в единични кавички. Числовите значения на интензивността на компонентите на цветовете се намират в диапазона от 0 до 255.

Целочисленото представяне се записва като шестнадесетично или десетично число. Шестнадесетичното число има следния вид 0x00BBGGRR, където RR е стойността на интензивността на червения компонент на цвета, GG – на зеления и BB – на синия. Десетичните константи нямат пряко отражение в RGB. Те представляват десетична стойност на шестнадесетичното целочислено представяне.

Именуваните цветове отразяват така наречения набор от Web цветове.

Примери:

```
// символни константи
C'128,128,128'      // сив
C'0x00,0xFF,0xFF'  // светлосин
// именуван цвят
Red
Yellow
Black
// целочислено представяне
0xFFFFFF             // бял
16777215            // бял
0x008000            // зелен
32768               // зелен
```

Константи за дата и време

Константите за дата и време могат да бъдат представени като литерален ред, който се състои от 6 части, представляващи числовите стойности на годината, месеца, датата (или датата, месеца, годината), часа, минутите и секундите. Константата се поставя в единични кавички и започва със символа D. Датата (година, месец, дата) и времето (час, минути и секунди) могат да се пропуснат както заедно, така и поотделно. Константата за дата и време може да приема стойности от 1 януари 1970 година до 31 декември 2037 година.

Примери:

```
D'2004.01.01 00:00'    // нова година
D'1980.07.19 12:30:27'
D'19.07.1980 12:30:27'
D'19.07.1980 12'   //равно на D'1980.07.19 12:00:00'
D'01.01.2004'      //равно на D'01.01.2004 00:00:00'
D'12:30:27'        //равно на D'[дата на компилацията] 12:30:27'
D''                //равно на D'[дата на компилацията] 00:00:00'
```

Операции и изрази

Изрази

Всеки израз се състои от един или повече операнди и символи на операциите. Изразът може да се състои от няколко реда.

Пример:

```
a++; b = 10; x = (y*z)/w;
```

Забележка: Израз, завършващ с точка и запетая представлява оператор.

Аритметически операции

```
Сума на стойностите                              i = j + 2;
Разлика на стойностите                           i = j - 3;
Промяна на знака                                 x = — x;
Произведение на стойностите                      z = 3 * x;
Частно от делението                              i = j / 5;
Остатък от делението                             minutes = time % 60;
Увеличение стойността на променливата с 1        i++;
Намаляване стойността на променливата с 1        k-—;
```

Операциите увеличение/намаляване на стойността на променливата не могат да се използват в изрази.

Пример:

```
int a=3;
a++;            // вярно
int b=(a++)*3; // неверен израз
```

Операции за присвояване

Забележка: Изразите, които включват операциите за присвояване, приемат стойността на левия операнд след присвояването.

```
Присвояване на стойността y на променливата x       y = x;

Увеличение на променливата y с x                    y += x;

Намаляване на променливата y с x                    y -= x;
```

Умножение на променливата y по x	y *= x;
Делене на променливата y на x	y /= x;
Делене на y на абсолютната стойност от x	y %= x;
Логическо изместване на y надясно с x бита	y >>= x;
Логическо изместване на y наляво с x бита	y <<= x;
Побитова операция И	y &= x;
Побитова операция ИЛИ	y |= x;
Побитова операция изключващо ИЛИ	y ^= x;

Забележка: Всеки израз може да съдържа само една операция за присвояване. Побитовите операции се извършват само с помощта на цели числа. При извършване на операцията логическото изместване на y надясно/наляво с x бита се използват 5 младши двоични разряди на стойността x, старшите разряди се отхвърлят, т.е. изместването се осъществява с 0-31 бита. При извършване на операцията %= (делене на y на абсолютната стойност от x) знакът на резултата съвпада със знака на делимото.

Операции за отношение

Логическото значение FALSE представлява цяла нулева стойност, докато значението TRUE представлява ненулева стойност.

Изразите, съдържащи операциите за отношение или логически операции, приемат стойностите FALSE(0) или TRUE(1).

True, ако a е равно на b	a == b;
True, ако a не е равно на b	a != b;
True, ако a е по-малко от b	a < b;
True, ако a е по-голямо от b	a > b;
True, ако a е по-малко или равно на b	a <= b;
True, ако a е по-голямо или равно на b	a >= b;

При извършване на операциите == или != не могат да се използват две ненормализирани числа с плаваща точка. За тази цел едното число трябва да се извади от другото, а нормализираният резултат да се сравни с нула.

Логически операции

Операндът на логическото отрицание НЕ(!) трябва да е от аритметичен тип. Резултатът е равен на TRUE(1), ако стойността на операнда е FALSE(0). Резултатът е равен на FALSE(0), ако операндът не е равен на FALSE(0).

```
// True, ако a е равно на False.

if(!a)

  Print("not 'a'");
```

Логическата операция ИЛИ (||) на стойностите x и y. Изразът приема стойността TRUE(1), ако стойностите x или y не са равни на TRUE(1). В противен случай - FALSE(0).

Пример:

```
if(x<k || x>l)
  Print("out of range");
```

Логическата операция И (&&) на стойностите x и y. Изразът приема стойността TRUE(1), ако стойностите x и y са равни на TRUE(1). В противен случай - FALSE(0).

Пример:

```
if(p!=x && p>y)
  Print("TRUE");
n++;
```

Логическите изрази се изчисляват изцяло, т.е. при тях не се използва схемата на т. нар. кратка оценка.

Побитови операции

Допълване на стойността на променливата до единица. Изразът приема стойността 1 във всички разряди, в които стойността на променливата съдържа 0, и 0 във всички разряди, в които стойността на променливата съдържа 1.

```
b = ~n;
```

Двоичното представяне на x се измества надясно с y разряда. Изместването надясно е логическо, т.е. свободните разряди отляво ще се запълват с нули.

Пример:

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

Двоичното представяне на x се измества наляво с y разряда; свободните разряди отдясно ще се запълват с нули.

Пример:

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

Побитова операция И на двоичните представяния на x и y. Изразът приема стойността 1 (TRUE) във всички разряди, в които и x и y не съдържат нула, и 0 (FALSE) във всички останали разряди.

Пример:

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

Побитова операция ИЛИ на двоичните представяния на x и y. Изразът приема стойността 1 във всички разряди, в които x или y не съдържат 0, и 0 във всички останали разряди.

Пример:

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

Побитова операция изключващо ИЛИ на двоичните представяния на x и y. Изразът приема стойността 1 в онези разряди, в които x и y имат различни двоични стойности, и 0 във всички останали разряди.

Пример:

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

Забележка: Побитовите операции се извършват само с цели числа.

Други операции

Индексиране. При обръщане към i-ия елемент от масива, изразът приема стойността на променливата с пореден номер i.

Пример:

```
array[i] = 3;
// Да се присвои стойност 3 на i-ия елемент от масива array.
// Обърнете внимание на това, че първият елемент от масива
// се описва с израза array [0].
```

Извикване на функцията с аргументи x1, x2,..., xn. Изразът приема стойността, която се връща от функцията. Ако типът на върнатата стойност на функцията е void, извикването на тази функция не може да се разполага отдясно в операцията за присвояване. Обърнете внимание на това, че поредността на изпълнението на изразите x1,..., xn е гарантирана.

Пример:

```
double SL=Ask-25*Point;
double TP=Ask+25*Point;
int ticket=OrderSend(Symbol(),OP_BUY,1,Ask,3,SL,TP,
                     "My comment",123,0,Red);
```

Операциите, разделени със запетая се извършват отляво надясно. Два израза разделени със запетая се изчисляват отляво надясно, а значението на левия израз се унищожава. Всички странични ефекти от изчисляването на левия израз могат да възникнат преди изчисляването на десния израз. Типът и стойността на резултата съвпадат с типа и стойността на десния израз.

Приоритети и поредност на извършване на операциите

Всяка група от операции, посочени в таблицата, има еднакъв приоритет. Колкото е по-висок приоритетът на групата от операции, толкова по-нагоре е разположена тя в таблицата. Поредността на извършване определя класификацията на операциите и операндите.

()	Извикване на функция	Отдясно наляво
[]	Маркиране на елемент от масива	
!	Логическо отрицание	Отдясно наляво
~	Побитово отрицание	
-	Промяна на знака	
*	Умножение	Отляво надясно
/	Деление	

%	Деление на абсолютната стойност от	
+	Събиране	Отляво надясно
-	Изваждане	
<<	Изместване наляво	Отляво надясно
>>	Изместване надясно	
<	По-малко от	Отляво надясно
<=	По-малко или равно	
>	По-голямо от	
>=	По-голямо или равно	
==	Равно	Отляво надясно
!=	Не е равно	
&	Побитова операция И	Отляво надясно
^	Побитова операция изключващо ИЛИ	Отляво надясно
|	Побитова операция ИЛИ	Отляво надясно
&&	Логическа операция И	Отляво надясно
||	Логическа операция ИЛИ	Отляво надясно
=	Присвояване	Отдясно наляво
+=	Събиране с присвояване	
-=	Изваждане с присвояване	
*=	Умножение с присвояване	
/=	Деление с присвояване	
%=	Деление на абсолютна стойност с присвояване	
>>=	Изместване надясно с присвояване	
<<=	Изместване наляво с присвояване	
&=	Побитово И с присвояване	
|=	Побитово ИЛИ с присвояване	
^=	Изключващо ИЛИ с присвояване	
,	Запетая	Отляво надясно

За внасяне на промени в поредността на извършване на операциите се използват кръгли скоби.

Оператори

Формат и вмъкване

Формат. Един оператор може да заема един или повече редове. Два или повече оператори могат да се разположат на един ред.

Вмъкване. Един оператор, който управлява поредността на извършване (if, if-else, switch, while и for), може да се вмъква в друг подобен оператор.

Съставен оператор

Съставният оператор (блок) включва един или повече оператори от всякакъв тип, поставени във фигурни скоби { }. След затварящата фигурна скоба не трябва да има точка и запетая (;).

Пример:

```
if(x==0)
  {
   x=1; y=2; z=3;
  }
```

Оператор-израз

Всеки израз, който завършва с точка и запетая (;), е оператор. По-долу са представени примери на оператори-изрази:

Оператор за присвояване.
Идентификатор = израз;

Операторът за присвояване може да се използва само веднъж в един израз.

Пример:

```
x=3;
y=x=3;   //false
```

Оператор за извикване на функция
Име_на_функция (аргумент1,..., аргументN);

Пример:

```
fclose(file);
```

Празен оператор

Празният оператор се състои само от точка и запетая (;). Той не съдържа никакви символи и не извършва никакви действия.

Оператор break

Операторът break; прекратява извършването на най-близкия вмъкнат външен оператор switch, while или for. Управлението се предава на оператора, който е разположен след завършващия. Една от функциите на този оператор се състои в това да завърши изпълнението на цикъла при присвояване на определена стойност на дадена променлива.

Пример:

```
for(int i=0;i<n;i++)
  if(a[i]==b[i])
    break;
```

Оператор continue

Операторът continue; предава управлението на най-близкия външен оператор на цикъла while или for, като извиква началото на следващата итерация. Този оператор е противоположен по действие на оператора break.

Пример:

```
for(i=0, k=9;i<10; i++, k--)
  {
   if(a[i]!=0) continue;
   a[i]=b[k];
   }
```

Оператор return

Операторът return; прекратява извършването на текущата функция и връща управлението на извикващата програма. Операторът return(израз); прекратява извършването на текущата функция и връща управлението на извикващата програма с предаване на стойността на израза. Изразът на оператора се поставя в кръгли скоби. Изразът не трябва да съдържа оператор за присвояване.

Пример:

```
return(x+y);
```

Във функциите от типа void операторът return трябва да се използва без израз.

Пример:

```
return;
```

Затварящата фигурна скоба на функцията предполага неявно изпълнение на оператора return без израз.

Условен оператор if

```
if (израз)
   оператор;
```

Ако изразът е верен, операторът се изпълнява. Ако изразът не е верен, управлението се предава на израза, който е разположен след оператора.

Пример:

```
  if(a==x)
    temp*=3;
   temp=MathAbs(temp);
```

Условен оператор if-else

if (израз)
оператор1
else
оператор2

Ако изразът е верен, изпълнява се оператор1, а управлението се предава на оператора, който е разположен след оператор2 (т. е. оператор2 не се изпълнява). Ако изразът не е верен, се изпълнява оператор2.

Частта else на оператора if не е задължителна. Затова значението на вмъкнатите оператори if с пропусната част else може да бъде неоднозначно. В този случай else се свързва с най-близкия предишен оператор if в същия блок, където липсва частта else.

Пример:

```
  // Частта else се отнася към втория оператор if:
  if(x>1)
     if(y==2)
        z=5;
  else z=6;

   // Частта else се отнася към първия оператор if
  if(x>l)
    {
  if(y==2)
       z=5;
    }
  else z=6;

   // Вмъкнати оператори
  if(x=='a')
     y=1;
  else if(x=='b')
    {
     y=2;
     z=3;
    }
  else if(x=='c')
    y = 4;
  else
    Print("ERROR");
```

Оператор switch

switch (израз)
{
case константа: оператори
case константа: оператори
...
default: оператори
}

Този оператор сравнява стойността на израза с константите във всички варианти case и предава управлението на оператора, който съответства на стойността на израза. Всеки вариант case може да бъде маркиран с цяла или със символна константа или константен израз. Константният израз не може да включва променливи или извикване на функции.

Пример:

```
case 3+4: //правилно
case X+Y: //неправилно
```

Операторите, свързани с default, се изпълняват, ако нито една от константите в операторите case не е равна на стойността на израза. Вариантът default може да не бъде последен. Ако нито една константа не съответства на стойността на израза и вариантът default липсва, тогава не се извършват никакви действия. Ключовата дума case заедно с константата служи просто за маркиране, и ако ще се изпълняват оператори за някой от вариантите case, то по-нататък ще се извършват операторите на всички следващи варианти дотогава, докато програмата не срещне оператора break, което, от своя страна, позволява да се свърже една последователност от оператори с няколко варианта. Константният израз се изчислява по време на компилиране. Нито една от двете константи в оператора switch не могат да имат еднакви стойности.

Пример:

```
switch(x)
   {
   case 'A':
     Print("CASE A\n");
     break;
  case 'B':
  case 'C':
     Print("CASE B or C\n");
     break;
  default:
     Print("NOT A, B or C\n");
     break;
  }
```

Оператор while

while (израз) оператор

Ако изразът е верен, операторът се изпълнява дотогава, докато изразът не стане неверен. Ако изразът не е верен, тогава управлението се предава на следващия оператор.

Забележка: Стойността на израза се определя преди изпълнението на оператора. Следователно, ако изразът не е верен от самото начало, тогава операторът изобщо не се изпълнява.

Пример:

```
while(k<n)
 {
   y=y*x;
    k++;
 }
```

Оператор for

for (израз1; израз2; израз3)
оператор;

Израз 1 описва инициализацията на цикъла. Израз 2 служи за проверка на условията за завършване на цикъла. Ако той е верен, изпълнява се операторът for, след което се извършва израз 3. Операцията се повтаря дотогава, докато израз 2 не стане неверен.

Ако израз 2 не е верен, цикълът завършва и управлението се предава на следващия оператор. Израз 3 се изчислява след всяка итерация. Операторът for е еквивалентен на следната последователност от оператори:

```
израз1;
while (израз2)
{
    оператор;
    израз 3;
};
```

Пример:

```
for(x=1;x<=7;x++)
  Print(MathPower(x,2));
```

Някой от трите израза, включени в оператора for или и трите заедно могат да липсват, но разделящите ги точки и запетаи (;) са задължителни.

Ако е пропуснат израз2, то тогава се смята, че той е винаги верен. Операторът for (;;) представлява безкраен цикъл, еквивалентен на оператора while(1).

Всеки от изразите 1 и 3 може да включва няколко израза, обединени от оператора запетая ','.

Пример:

```
for(i=0,j=n-l;i<n;i++,j--)
   a[i]=a[J];
```

Функции

Определяне на функции

Функцията се определя не само от описанието на типа на връщаната стойност, но и от формалните параметри, както и от съставния оператор (блок), който описва действията, които се извършват от съответната функция.

Пример:

```
double                          // типът на връщаната стойност
linfunc (double a, double b) // име на функцията и списък от параметри
  {                             // съставен оператор
   return (a + b);              // връщана стойност
  }
```

Операторът return може да връща стойността на израза, който е разположен в този оператор. Ако е необходимо, стойността на израза може да се преобразува до типа на резултата на функцията. Функцията, която не връща стойност, трябва да бъде описана като void.

Пример:

```
void errmesg(string s)
  {
   Print("error: "+s);
  }
```

Извикване на функция

име_на_функция (x1, x2,..., xn)

Аргументите (фактическите параметри) се предават според стойността, т. е. всеки израз xl, . . . , xn се изчислява, и стойността се предава на функцията. Поредността на изчисляване на изразите, както и поредността на зареждане на стойностите са гарантирани. По време на извършване се прави проверка на броя или типа на аргументите, предадени на функцията. Този метод за обръщане към функцията се нарича извикване по стойност. Съществува и друг метод, който се нарича извикване чрез връзка. Извикването на функция е израз, стойността на който представлява връщаната от функцията стойност. Описаният тип функция трябва да съответства на типа връщана стойност. Функцията може да се обяви или да се опише на всяко място в програмата:

```
int somefunc()
  {
   double a=linfunc(0.3, 10.5, 8);
  }
double linfunc(double x, double a, double b)
  {
   return (a*x + b);
  }
```

Специални функции init, deinit и start

След свързване с графиката програмата започва работата си с функцията init(). Тази функция се стартира също и веднага след стартиране на клиентския терминал или при смяна на финансовия инструмент и/или на периода на графиката.

Всяка свързана с графиката програма завършва работата си с функцията deinit(). Тази функция се стартира и при изключване на клиентския терминал, при затваряне на графиката или непосредствено преди смяната на финансовия инструмент и/или на периода на графиката.

При получаване на нови котировки се изпълнява функцията start() на съответните експертни системи и потребителски индикатори. Ако по време на получаване на новите котировки се е изпълнявала функцията start(), която е била стартирана при постъпване на предишната котировка, то поредното извикване на функцията start() ще се изпълни само след инструкцията return(). Всички получени по време на изпълнението на програмата нови котировки се игнорират.

Отделянето на програмата от графиката, смяната на финансовия инструмент и/или на периода на графиката, затварянето на графиката, както и изключването на клиентския терминал прекъсва изпълнението на програмата.

Изпълнението на скриптовете не зависи от идващите котировки.

Променливи

Описания

Описанията се използват не само за определяне на променливите, но и за обявяване на типовете променливи, както и на функциите, определени на друго място. Описанията не са оператори. Променливите трябва да се обявяват преди използването им. За идентифициране на променливите се използват уникални имена.

Основните им видове са:

- string - символен ред;
- int - цяло число;
- double - число с плаваща точка (с двойна точност);
- bool - логическа стойност true или false.

Пример:

```
string  sMessage;
int     nOrders;
double dSymbolPrice;
bool    bLog;
```

Допълнителни видове:

- datatime - дата и време, беззнаково цяло число, което съдържа броя на секундите, изминали от 0 часа на 1 януари 1970 година.
- color - цяло число, което представлява съвкупност от трите цветови компонента.

Тези типове данни имат смисъл само при обявяване на входните параметри за тяхното представяне в прозореца Properties.

Пример:

```
extern datatime tBegin_Data   = D'2004.01.01 00:00';
extern color    cModify_Color = C'0x44,0xB9,0xE6';
```

Масиви

Масивът представлява индексирана съвкупност от еднотипни данни.

```
int       a[50];      //Едномерен масив от 50 цели числа.
double m[7][50];      //Двумерен масив от седем масива,
                      //всеки от които включва 50 числа.
```

Индексът на масива може да бъде само цяло число. Допускат се не повече от четиримерни масиви.

Определяне на локални променливи

Променлива, обявена във функция представлява локална променлива. Областта на видимост на локалната променлива е ограничена от границите на функцията, в която тя е обявена. Локалната

променлива може да бъде инициализирана с всякакъв израз. Инициализацията на локалната променлива се осъществява всеки път при извикване на съответната функция. Локалните променливи се разполагат във времевата област на паметта на съответната функция.

Формални параметри

Пример:

```
void func(int x, double y, bool z)
{
...
}
```

Формалните параметри са локални. Областта на видимост на формалните параметри е блокът на функцията. Формалните параметри трябва да се различават по име от външните и локалните променливи, определени във функцията. На формалните параметри могат да се присвоят определени стойности в блока на функцията. Формалните параметри могат да бъдат инициализирани с константи. В този случай началната стойност е стойността по подразбиране. Параметрите, разположени след началния параметър трябва също да бъдат инициализирани.

Пример:

```
void func(int x, double y = 0.0, bool z = true)
{
...
}
```

При извикване на такава функция инициализираните параметри не са задължителни, вместо тях ще бъдат представени стойности по подразбиране.

Пример:

```
 func(123, 0.5);
```

Параметрите се предават по стойност, т.е. промените на съответната локална променлива в извикваната функция по никакъв начин няма да се отразят в извикващата функция. В качеството на параметри могат да се предават масиви, въпреки че когато един масив се предава като параметър, елементите му не могат да се променят.

Съществува възможност за предаване на параметрите чрез връзка. В този случай модификацията на тези параметри ще се отрази на съответните променливи в извикваната функция. За да се поясни, че параметърът се предава чрез връзка, трябва да се постави модификатор & след типа данни.

Пример:

```
void func(int& x, double& y, double& z[])
{
...
}
```

Масивите също могат да се предават чрез връзка. В този случай всички промени ще се отразят в изходния масив. Параметрите, предавани чрез връзка, не трябва да се инициализират със стойности по подразбиране.

Статични променливи

Статична променлива се определя от класа на паметта static. Модификаторът static се поставя преди типовете данни.

Пример:

```
{
 static int flag;
}
```

Статичните променливи се съхраняват в постоянна област от паметта на програмата, стойностите им не се губят при изход от функцията. Всички променливи в блока, освен формалните параметри на функцията, могат да бъдат определени като статични. Статичната променлива може да бъде инициализирана със съответстваща на типа ѝ константа, за разлика от простата локална променлива, началната стойност на която може да бъде всякакъв израз. При липса на явна инициализация, статичната променлива се инициализира с нула. Статичните променливи се инициализират еднократно преди извикването на специализираната функция init(), т.е. при изход от функцията, в която е обявена статичната променлива. В този случай стойността на тази променлива не се губи.

Глобални променливи

Глобалните променливи се определят на същото ниво както и функциите, т.е. те не са локални в блоковете.

Пример:

```
int Global_flag;

int start()
{
...
}
```

Областта на видимост на глобалните променливи е цялата програма, глобалните променливи са достъпни от всички функции, определени в програмата. Те се инициализират с нула, освен ако не е зададена друга начална стойност. Глобалната променлива може да бъде инициализирана само със съответстваща на типа ѝ константа. Инициализацията на глобалните променливи се осъществява еднократно преди извършването на функцията init().

Забележка: не трябва да се бъркат променливите, обявени на глобално ниво, с глобалните променливи на клиентския терминал, достъпът към които се осъществява с помощта на функциите GlobalVariable...().

Външни променливи

Външната променлива се определя от класа на паметта extern. Модификаторът extern се поставя преди типовете данни.

Пример:

```
extern double InputParameter1 = 1.0;
int init()
{
...
}
```

Външните променливи определят входните параметри на програмата, те са достъпни от прозореца Properties на програмата. В скриптовете външни променливи могат да не се определят. Масивите не могат да изпълняват ролята на външни променливи.

Инициализация на променливите

При определяне на променливите, всяка от тях може да бъде инициализирана. Всички променливи се инициализират с нула (0), освен ако не е зададена друга начална стойност. Глобалните и

статичните променливи могат да бъдат инициализирани само с константа от съответния тип. Локалните променливи могат да бъдат инициализирани с всякакъв израз, не само с константа. Инициализацията на глобалните и статичните променливи се осъществява еднократно. Инициализацията на локалните променливи се осъществява всеки път при извикване на съответните функции.

Примери:

```
// инициализация на целочислена променлива
int mt = 1;
// инициализация на променлива с плаваща точка (двойна точност)
double p = MarketInfo(Symbol(),MODE_POINT);
// инициализация на ред
string s = "hello";
```

Масиви

Пример:

```
int mta[6] = {1,4,9,16,25,36};
```

Списъкът на стойностите на елементите от масива трябва да бъде поставен във фигурни скоби. Ако е зададен размерът на масива, то стойностите са равни на 0, освен ако не са зададени други стойности.

Описание на външните функции

Типът на външните функции, определени в друг компонент на програмата, трябва да бъде явно описан. Липсата на такова описание може да доведе до грешки при компилиране, асемблиране или изпълнение на програмата. При описание на външен обект използвайте ключовата дума #import със съответен модул.

Примери:

```
#import "user32.dll"
     int        MessageBoxA(int hWnd ,string lpText,
                            string lpCaption,int uType);
     int        SendMessageA(int hWnd,int Msg,int wParam,int lParam);
#import "lib.ex4"
     double   round(double value);
#import
```

Препроцесор

Ако символът # се използва като първи символ в реда на програмата, този ред представлява команда на компилатора. Командите на компилатора завършват със символа за нов ред.

Обявяване на константа

#define идентификатор стойност

Идентификаторът на константата се подчинява на същите правила, на които се подчиняват и имената на променливите. Стойността му може да бъде от всякакъв тип.

Пример:

```
#define ABC                100
#define PI                 0.314
#define COMPANY_NAME "MyCompany Ltd."
```

Компилаторът ще замени всяко влизане на идентификатора в текста на програмата със съответната стойност.

Управление на компилацията

#property идентификатор стойност

Списък на предварително определени идентификатори на константите.

Пример:

```
#property link          "www.mycompany.com"
#property copyright     "My Company®"
#property stacksize     1024
```

Константа	Тип	Описание
link	string	Линк към сайта на фирмата-производител
copyright	string	Името на фирмата-производител
stacksize	int	Размер на стека за рекурсивни извиквания
indicator_chart_window	void	Извеждане на индикатора в прозореца на графиката
indicator_separate_window	void	Извеждане на индикатора в отделен прозорец
indicator_buffers	int	Броят на буферите за изчисляване, не трябва да превишава 8
indicator_minimum	int	Минимумът за графиката
indicator_maximum	int	Максимумът за графиката
indicator_colorN	color	Цвят на линията N, където N е от 1 до 8
indicator_levelN	int	Нивото N в отделния прозорец на индикатора, където N е от 1 до 8
show_confirm	void	Извеждане на прозореца за потвърждение преди стартирането на скрипта
show_inputs	void	Извеждане на прозореца Properties преди стартирането на скрипта, и забрана на извеждането на прозореца за потвърждение

Компилаторът ще запише обявените стойности в настройките на извършвания модул.

Определяне на библиотека

#define library

Конструкция #define library определя набора от библиотечни функции, групирани във файл. Библиотеката се различава от програмата по това, че тя няма входна точка. Функциите ѝ могат да бъдат достъпни и от друга програма, написана на езика MQL4.

Пример:

```
#define library
```

Включване на файлове

Забележка: Командният ред #include може да се среща навсякъде в програмата, но обикновено всичките включвания се разполагат в началото на файла с изходния текст.
#include <име_файла>

Пример:

```
#include <win32.h>
```

Препроцесорът замества този ред със съдържанието на файла win32.h. Ъгловите скоби означават, че файлът win32.h ще бъде взет от стандартния каталог (обикновено това е каталогът на _терминала\experts\include). Текущият каталог не се преглежда.

#include "имя_файла"

Пример:

```
#include "mylib.h"
```

Компилаторът замества този ред със съдържанието на файла mylib.h. Тъй като името на файла е поставено в кавички, търсенето се осъществява в текущия каталог, който съдържа основния файл с изходния текст. Ако текущият каталог не съдържа този файл, търсенето се осъществява в каталозите, които са определени с име в опциите на компилатора. Ако тези каталози също не съдържат този файл, тогава се преглежда стандартният каталог.

Импортиране на функции и други модули

```
#import "имя_файла"
   func1();
   func2();
#import
```

Пример:

```
#import "user32.dll"
     int MessageBoxA(int hWnd,string lpText,string lpCaption,
                              int uType);
     int MessageBoxExA(int hWnd,string lpText,string lpCaption,
                                 int uType,int wLanguageId);
#import "melib.ex4"
#import "gdi32.dll"
     int       GetDC(int hWnd);
     int       ReleaseDC(int hWnd,int hDC);
#import
```

Импортирането на функции се осъществява от компилираните модули на MQL4 (файловете с разширение *.ex4), както и от модулите на операционната система (файловете с разширение *.dll). В последния случай се обявяват също и импортираните функции. Командата #import (може и без параметри) завършва описанието на импортираните функции.