MetaQuotes Language 4

MetaQuotes Language 4 (MQL 4) este un nou limbaj integrat pentru programarea strategiilor de tranzactionare. Acest limbaj pemite crearea propriului Expert Advisors care executa procesul de management al procesului de tranzactionare in mod automat si se potriveste perfect pentru implementarea propriilor strategii de tranzactionare. Puteti de asemenea sa va creatii proprii Custom Indicators (indicatori proprii), Scripts (scripturi)si Libraries (biblioteci) de functii cu ajutorul MQL4.

Un numar mare de functii necesare analizei cotatiilor trecute si prezente, aritmeticii de baza si operatiilor logice este inclus in structura MQL4. De asemenea sunt si indicatori de baza integrati, cat si comenzi de plasare si control a acestora.

MetaEditor 4 IDE (Integrated Development Environment) care evidentiaza diferite constructii ale MQL 4 este utilizat pentru a scrie codul de program. Acesta ajuta utilizatorii sa se orienteze in sistemul expert de text cu mai multe usurinta. Folosim MetaQuotes Language Dictionary ca baza de informatii pentru MQL 4. Un ghid succint cuprinde functii impartite pe categorii, operatii, cuvinte rezervate si alte constructii lingvistice si permite gasirea descrierii fiecarui element folosit.

Programele scrise in **MetaQuotes Language 4** au diferite proprietati si scopuri:

- **Expert Advisors** este un sistem de tranzactionare mecanic (mechanical trade system -MTS) conectat la un anumit scenariu. Advisor-ul nu doar va poate informa despre posibilitatea de a face o tranzactie ieftina, cat si poate efectua tranzactii in contul de tranzactionare in mod automat si le poate directiona direct la serverul de tranzactionare. Precum cele mai multe sisteme de tranzactionare, terminalul MetaTrader 4 permite testarea strategiilor cu date istorice afisand pe grafic locurile in care tranzactiile intra si ies.

- **Custom Indicators** (indicatori proprii)este un analog al unui indicator tehnic. Cu alte cuvinte, Custom Indicators permite crearea de indicatori tehnici in plus fata de cei deja integrati in terminalul MetaTrader 4. La fel ca si indicatorii integrati, acestia nu pot efectua tranzactii in mod automat, ci sunt indreptati inspre implementarea functiilor analitice.

- **Scripts** (scripturi) sunt programe destinate unei singure executii a aceleiasi actiuni. Spre deosebire de Expert Advisors, scripturile nu merg in directia miscarii pretului si nu au acces la functiile indicator.

- **Libraries** (biblioteci) sunt biblioteci de functii ale utilizatorului une blocurile de date ale programelor utilizatorului folosite frecvent sunt stocate.

Sintaxa

Format

Simbolurile spatii, taburi, liniile si simbolurile grafice pot fi folosite ca separatoare. Puteti folosi oricate asemenea simboluri in loc de unul. Pentru ca textul sa fie mai usor de citit ar trebui sa folositi simbolurile tab.

Comentarii

Comentariile multilinie incep cu simboluri /* si se incheie cu simbolurile */. Astfel de comentarii nu pot fi suprapuse.
Comentariile simple incep cu simbolurile //, se incheie cu simbolul unei noi linii si pot fi suprapuse in comentarii multilinie.
Comentariile sunt permise acolo unde sunt posibile spatii goale si permit orice numar de spatii.

Exemple:

```
 // comentariu simplu

/*  comentariu
    multi linie  // nested single comment
*/
```

Identificatori

Identificatorii sunt folositi ca nume de variabile, functii si tipuri de date. Lungimea unui identificator nu poate depasi 31 de caractere.
Simbolurile care pot fi utilizate: 0-9, litere latine cu majuscule si litere mici a-z, A-Z (recunoscute ca simboluri diferite), simbolul de subliniat (_). Primul simbol nu poate fi o cifra. Identificatorul nu poate sa coincida cu niciun cuvant rezervat.

Exemple:

```
NUME1 nume1 Total_5 Hartie
```

Cuvinte rezervate

Identificatorii listati mai jos reprezinta cuvinte fixe rezervate. Fiecaruia dintre ele i s-a atribuit o anume actiune si nu poate fi folosit in alte scopuri:

Tipuri de date	Clase de memorie	Operatori	Altele
bool	extern	break	false
color	static	case	true
datetime		continue	
double		default	
int		else	
string		for	
void		if	
		return	
		switch	
		while	

Tipuri de date

Principalele tipuri de date sunt:

- Integer (int) (unitare)

- Boolean (bool) (obiective)
- String (string) (Vector)
- Floating-point number (double) (numar floating-point)
- Color (color) (culoare)
- Datetime (datetime) (data si timp)

Utilizam tipurile Color si Datetime numai pentru a facilita vizualizarea si introducerea acelor parametri pe care i-am setat pentru proprietatile expert advisor sau tabul indicatorul propriu "Input parameters". Datele tipurilor Color si Datetime sunt reprezentate ca valori unitare.

Folosim transformarea tipului implicit. Prioritatea tipurilor in cadrul unei transformari in ordine crescatoare este urmatoarea:

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

Inainte de executarea de operatii (cu exceptia operatiei desemnate) datele sunt transferate unui tip de precizie maxima, iar inaintea operatiunilor desemnate, tipului unitare.

Constante unitare

Decimal: numere de la 0 la 9; Zero nu poate fi prima cifra.
Exemple:

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

Hexadecimal: numere de la 0 la 9, litere a-f sau A-F sa reprezinte valorile 10-15; acestea incep cu 0x sau 0X.
Exemple:

```
0x0A, 0x12, 0X12, 0x2f, 0xA3, 0Xa3, 0X7C7
```

Constantele unitare pot lua valori de la -2147483648 la 2147483647. Daca o constanta iese din acest interval, rezultatul nu este definit.

Constante literale

Orice caracter singur intre ghilimele simple sau un cod ASCII hexadecimal al unui caracter care arata in felul '\x10' este o constanta de caracter a unui tip unitar. Anumite caractere precum ghilimele simple ('), ghilimele duble ("), semnul intrebarii (?), bara inversa (\) si caractere control pot fi reprezentate printr-o combinatie de caractere incepand cu bara inversa (\) conform cu tabelul urmator:

Linie	NL (LF)	\n
tab orizontal	HT	\t
capat de rand	CR	\r
bara inversa	\	\
ghilimele simple	'	\'
ghilimele duble	"	\"
hexadecimal cod ASCII	hh	\xhh

Daca un caracter, altul decat cel listat mai sus, urmeaza barei inverse, rezultatul nu este definit.

Exemple:

```
int a = 'A';
int b = '$';
```

```
int c = '©';     // code 0xA9
int d = '\xEA'; // symbol code ®
```

Constante Boolean

Constantele Boolean pot avea valoarea adevarat sau fals, reprezentarea lor numerica este 1, respectiv, 0. Putem folosi si sinonimele True si TRUE (adevarat), False si FALSE (fals).

Exemple:

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

Constantele numar cu punct plutitor

Constantele cu punt plutitor constau intr-o parte unitara, un punct (.) si o parte de fractie. Partile unitare si fractionale sunt o succesiune de numere decimale. O parte neimportanta a partii fractionale poate fi absenta.

Exemple:

```
double a = 12.111;
double b = -956.1007;
double c = 0.0001;
double d = 16;
```

Constantele cu punct plutitor pot lua valori de la 2.2e-308 la 1.8e308. Daca o constanta iese din acest interval, rezultatul nu este definit.

Constantele vector

Constanta vector este o succesiune de caractere cod-ASCII intre ghilimele duble: "Character constant".
O constanta vector este o multitudine de caractere incadrate in ghilimele. Este de tipul vectorial. Fiecare constanta vectoriala, chiar daca este identica cu o alta constanta vectoriala, este salvata intr-un alt loc in memorie. Daca aveti nevoie sa inserati ghilimele duble (”) intr-o linie, trebuie sa puneti o bara inversa (\) inaintea lor. Puteti insera orice constanta cu caracter special in linie daca acesta are o bara inversa (\) inaintea lui. Lungimea unei constante vector este intre 0 si 255 de caractere. In cazul in care constanta vector este mai lunga, caracterele in plus din drepta sunt respinse.

Exemple:

```
"This is a character string"
"Copyright symbol \t\xA9"
"this line with LF symbol \n"
"A" "1234567890" "0" "$"
```

Constante culoare

Constantele culoare pot fi reprezentate in trei moduri: prin reprezentarea caracterului; prin reprezentarea unitara; prin nume (numai pentru culori concrete Web).

Reprezentarea caracterului consta in patru parti care reprezinta gradul de valori numerice ale celor trei culori principale – rosu, verde si albastru. Constanta incepe cu simbolul C si este intre ghilimele simple. Gradul valorilor numerice ale componentei culoare variaza in intervalul 0-255.

Repezentarea valorica unitara este scrisa in forma unui numar hexadecimal sau decimal. Un numar hexadecimal arata ca 0x00BBGGRR unde RR este gradul componentei de culoare rosie, GG – pentru culoarea verde si BB – pentru culoarea albastra. Constantele decimale nu se reflecta direct in RGB. Ele reprezinta valoarea decimala a reprezentarii decimale unitare.
Culorile concrete refleca asa-numitul set de culori Web.

Exemple:

```
// constante simbol
C'128,128,128'      // gri
C'0x00,0xFF,0xFF'  // albastru
// culoarea numita
Rosu
Galben
Negru
// reprezentarea valorii unitare
0xFFFFFF             // alb
16777215           // alb
0x008000           // verde
32768              // verde
```

Constante data si timp

Constantele data si ora pot fi reprezentate printr-o linie caracter constand in 6 parti cu valori pentru an, luna, zi, ora, minute si secunde. Constanta este incadrata de ghilimele simple cu caracterul D.
Constantele data si timp pot varia de la Ian 1, 1970 pana la Dec 31, 2037.

Exemple:

```
D'2004.01.01 00:00'     // Anul Nou
D'1980.07.19 12:30:27'
D'19.07.1980 12:30:27'
D'19.07.1980 12'        //egal cu D'1980.07.19 12:00:00'
D'01.01.2004'           //egal cu D'01.01.2004 00:00:00'
D'12:30:27'             //egal cu D'[compilatie data] 12:30:27'
D''                     //egal cu D'[compilatie data] 00:00:00'
```

Operatii & Expresii

Expresii

O expresie consta intr-unul sau mai multi operanzi si caractere de operare. O expresie poate fi scrisa pe mai multe randuri.

Exemplu::

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

Nota: O expresie care se termina cu punct si virgula este un operator.

Operatii aritmetice

Suma de valori	i = j + 2;
Diferenta de valori	i = j - 3;

Schimbarea semnului operatiei	x = — x;
Inmultire de valori	z = 3 * x;
Divizarea coeficientului	i = j / 5;
Catul impartirii	minutes = time % 60;
Adaugarea 1 la valorile variabilelor	i++;
Extragerea 1 din valorile variabilelor	k-—;

Operatiunile de adaugare si extragere a 1 nu pot fi integrate in expresii.

Exemplu:

```
int a=3;
a++;            // expresie valida
int b=(a++)*3; // expresie invalida
```

Operatia de atribuire

Nota: Valoarea expresiei care contine aceasta operatie este valoarea operandului ramas urmat de caracterul de legatura.

Atribuirea valorii **y** variabilei **x**	y = x;
Adaugarea lui **x** variabilei **y**	y += x;
Extragerea lui **x** din variabila **y**	y -= x;
Inmultirea lui variabilei **y** cu **x**	y *= x;
Impartirea variabilei **y** la **x**	y /= x;
Modul **x** din valoarea lui **y**	y %= x;
Modificarea logica a reprezentarii lui **y** la dreapta cu pasul **x**	y >>= x;
Modificarea logica a reprezentarii lui **y** la stanga cu pasul **x**	y <<= x;
Operatiunea la nivel de bitl **SI**	y &= x;
Operatiunea la nivel de bit **SAU**	y |= x;
Operatiunea la nivel de bit fara **SAU**	y ^= x;

Nota: Poate fi doar o singura operatie de atribuire in expresie. Puteti implementa operatiuni la nivel de bit doar cu numere unitare. Operatiunea de modificare logica foloseste valorile lui x pentru mai putin de 5 cifre binare. Cifrele mai mari sunt respinse, ca atare modificare este doar pentru intervalul de pasi 0-31. Pentru operatiunea %= semnul rezultatului este egal cu semnul numarului impartit.

Operatiuni de relatie

Valoarea logica FALSE este reprezentata cu valoarea unitara zero, in timp ce valoarea logica TRUE este reprezentata cu orice valoare in afara de zero.
Valoarea expresiilor care contin operatii de relatie sau operatii logice ese 0 (FALSE) sau 1 (TRUE).

True daca **a** egal cu **b**	a == b;
True daca **a** nu egal cu **b**	a != b;
True daca **a** mai mic decat **b**	a < b;
True daca **a** mai mare decat **b**	a > b;
True daca **a** este mai mic sau egal cu **b**	a <= b;

True daca **a** este mai mare sau egal cu **b** a >= b;

Doua numere cu punct plutiror ne-normalizate nu pot fi legate prin operatiile == sau !=. Pentru acest scop este necesara extragerea unuia din celalalt si rezultatul normalizat sa fie comparat cu zero.

Operatii Boolean

Operandul negatiei NU (!) trebuie sa fie de tip aritmetic; rezultatul este egal cu TRUE (1) daca valoarea operandului este FALSE (0); rezultatul este egal cu FALSE (0) daca operandul eeste diferit de FALSE (0).

```
// True daca a este false.
 if(!a)
    Print("not 'a'");
```

Expresia logica OR (||) a valorilor x si y. Valoarea expresiei este TRUE (1) daca x sau y sunt TRUE. In alt caz, valoarea expresiei este FALSE (0).

Exemplu:

```
if(x<k || x>l)

   Print("out of range");
```

Expresia logica AND (&&) a valorilor x si y. Valoarea expresie este TRUE (1) daca valorile x si y sunt TRUE. In alt caz, valoarea expresiei este FALSE (0).

Exemplu:

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

Operatii la nivel de bit

Complementul valorilor variabile. Valoarea expresiei contine 1 in toate cifrele unde n contine 0; valoarea expresiei contine 0 in toate cifrele unde n contine 1.

```
b = ~n;
```

Reprezentarea in codul binar a lui x este mutata la drepta cu y cifre. Mutarea la drepta este o modificare logica, bit-ii ramasi liberi de la stanga vor fi inlocuiti cu zerouri.

Exemplu:

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

Reprezentarea in codul binar a lui x este mutata la stanga cu y cifre; bit-ii ramasi liberi de la dreapta vor fi inlocuiti cu zerouri.

Exemplu:

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

Operatia la nivel de bit SI a reprezentarilor lui x si codate binar. Valoarea expresiei contine 1 (TRUE) in toti bit-ii unde atat x cat si y nu sunt egali cu zero; valoarea expresiei contine 0 (FALSE) in toti ceilalti biti.

Exemplu:

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

Operatia la nivel de bit SAU a reprezentarilor lui x si y codate binar. Valoarea expresiei contine 1 (TRUE) in toti bit-ii unde unul dintre x sau y nu este egal cu zero; valoarea expresiei contine 0 (FALSE) in toti ceilalti biti.

Exemplu:

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

Operatia la nivel de bit FARA SAU a reprezentarilor lui x si y codate binar. Valoarea expresiei contine 1 (TRUE) in toti bitii unde x si y au valori binare diferite; valoarea expresiei contine 0 (FALSE) in toti ceilalti biti.

Exemplu:

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

Nota: Operatiile la nivel de bit se executa numai cu unitari.

Alte operatii

Indexarea. La adresarea matricei elementului i, valoarea expresiei este egala cu variabila de sub numarul i.

Exemplu:

```
array[i] = 3;
 //Atribuie valoarea 3 elementului matrice cu indicele i.
//Nu uitati ca primul element din matrice
//este descris cu expresia de matrice [0].
```

Apelul functiei cu argumente x1,x2,…,xn. Expresia accepta valoarea returnata de functie. Daca valoarea returnata este de tip nul, nu puteti plasa o astfel de functie de apel in operatia de atribuire. Nu uitati ca expresiile x1,x2,…,xn vor fi executate in aceasta ordine.

Exemplu:

```
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);
```

Operatiunea „virgula” este executata de la stanga la dreapta. O pereche de expresii separate de virgula este calculata de la stanga la dreapta cu o stergere secventiala a valorii expresiei de la stanga. Toate efectele secundata ale calcularii expresiei de la stanga vor aparea inainte de calcularea expresiei de la dreapta. Tipul si valoarea rezultatului coincide cu tipul si valoarea expresiei de la dreapta.

Reguli de prioritate

Fiecare grup de operatii din tabel are aceeasi prioritate. Cu cat este mai mare prioritatea, cu atat este mai inalta pozitia in grupul din tabel. Ordinea de executie determina gruparea operatiilor si operanzilor.

()	Functie de apel	De la stanga la dreapta
[]	Selectarea unui element matrice	
!	Negatie	De la stanga la dreapta
~	Negatie la nivel de bit	
-	Operatie de schimbare de semn	
*	Inmultire	De la stanga la dreapta
/	Impartire	
%	Impartire in modul	
+	Adaugare	De la stanga la dreapta
-	Extragere	
<<	Mutare la stange	De la stanga la dreapta
>>	Mutare la dreapta	
<	Mai putin de cat	De la stanga la dreapta
<=	Mai putin sau egal cu	
>	Mai mare decat	
>=	Mai mare sau egal cu	
==	Egal cu	De la stanga la dreapta
!=	Nu e egal cu	
&	Operatia la nivel de bit SI	De la stanga la dreapta
^	Operatia la nivel de bit fara SAU	De la stanga la dreapta
|	Operatia la nivel de bit SAU	De la stanga la dreapta
&&	SI logic	De la stanga la dreapta
||	SAU logic	De la stanga la dreapta
=	Atribuire	De la dreapta la stanga
+=	Atribuire adaugare	
-=	Atribuire extragere	
*=	Atribuire inmultire	
/=	Atribuire impartire	
%=	Atribuire modul	
>>=	Atribuire modificare dreapta	
<<=	Atribuire modificare stanga	
&=	Atribuire nivel de bit SI	
|=	Atribuire nivel de bit SAU	
^=	Atribuire fara SAU	
,	Virgula	De la stanga la dreapta

Utilizati parantezele pentru a schimba ordinea de executare a operatiilor.

Operatori

Format si inserare

Format. Un operator poate ocupa unul sau mai multe randuri. Doi sau mai multi operatori pot fi situati pe acelasi rand.
Inserare. Operatorii de control al ordinii executarii (*if, if-else, switch, while* si *for*) pot fi inserati unul in celalalt.

Operator compus

Un operator compus (un bloc) consta intr-unul sau mai multi operatori de orice tip aflati intre acolade {}. Acolada care se inchide nu trebuie urmata de punct si virgula (;).

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

Operator expresie

Orice expresie urmata de punct si virgula (;) este un operator. Iata cateva exemple de operatori expresie

Operator de atribuire

Identificator=expresie;

Puteti folosi un operator de atribuire intr-o expresie doar o singura data.

Exemplu:
```
x=3;
y=x=3; // error
```

Operatorul functie de apel

Nume_functie (argument1,..., argumentN);

Exemplu:
```
fclose(file);
```

Operator nul

Consta doar intr-un punct si virgula (;). Utilizat pentru a indica corpul nul al unui operator control.

Operator Break

Operatorul *break* ;termina executarea operatorului cel mai apropiat inserat in exterior *switch, while* sau *for*. Controlul este dat operatorului care urmeaza celui incheiat. Unul dintre scopurile acestui operator este sa incheie executia in bucla atunci cand o anumita valoare este atribuita unei variabile.

Exemplu:

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

Operatorul Continue

Operatorul *continue*; Operatorul da control celui mai apropiat operator din ciclu de la exterior *while* sau *for*, pentru urmatoare iteratie. Scopul acestui operator este opus celui break.

Exemplu:

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

Operator Return

Operatorul return; operatorul incheie executia functiei curente si intoarce controlul programului in asteptare.
Un return (expresie); operaturlui incheie executia functiei curente si intoarce controlul programului in asteptare impreuna cu valoarea expresiei. Operatorul expresie este intre paranteze. Expresia nu trebuie sa contina un operator atribuit.

Exemplu:

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

Functiile void trebuie completate prin „return;” – operator fara expresie.

Exemplu:

```
return;
```

Inchiderea acoladei functiei presupune realizarea implicita a operatorului return fara expresie.

Operatorul conditional if

if (expresie)
operator;

Daca expresia este adevarata, operatorul este executat. Daca expresia este falsa, controlul este dat expresiei ce urmeaza operatorului.

Exemplu:

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

Operatorul conditional if-else

if (expresie)
operator1

```
else
   operator2
```

Daca expresia este adevarata, operator1 este executat si controlul este dat operatorului care urmeaza dupa operator2 (operator2 nu este executat). Daca expresia este falsa, operator2 este executat.
Partea „else” a operatorului „if” poate fi omisa. Asadar, pot aparea ambiguitati in cadrul operatorilor inserati „if” care au omisa partea „else”. Daca se intampla asta, „else” se adreseaza celui mai apropiat operator „if” din blocul care nu are o parte „else”.

Exemplu:

```
  //   Partea "else" se refera la al doilea operator "if":
  if(x>1)
    if(y==2)
      z=5;
  else z=6;

   //   Partea "else" se refera la primul operator "if":
  if(x>l)
    {
     if(y==2)
       z=5;
    }
  else z=6;

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

Operatorul Switch

```
switch (expresie)
{
    case constant: operatori
    case constant: operatori
    ...
    default: operatori
}
```

Compara valorile expresiei cu constante in toate variantele „case” si da controlul operatorului care seamana cu valoarea expresiei. Fiecare varianta a „case” poate fi marcata cu o constanta unitara sau caracter sau cu o expresie constanta. Expresia constanta nu trebuie sa includa variabile si functii de apel.

Exemplu:

```
case 3+4: //valid
case X+Y: //invalid
```

Operatorii conectati cu etichete „default”sunt executati daca nici una dintre constantele operatorilor „case” nu este egala cu valoarea expresiei. Varianta „Default” nu este obligatoriu finala. Daca nici una dintre constante nu seamana cu valoarea expresiei, iar varianta „default” lipseste, nu este executata nici o functie. Cuvantul cheie „case” si constanta sunt doar etichete, iar daca operatorii sunt executati pentru o varianta a „case”, programul va executa in continuare operatorii tuturor variantelor care urmeaza pana in momentul cand ajung la operatorul „break”. Acesta permite legarea unei succesiuni de operatori cu cateva variante. O expresie constanta este calculata in timpul compilarii.
Nici una dintre cele doua constante dintr-un operator switch nu pot avea aceeasi valoare.

Exemplu:

```
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;
  }
```

Operatorul ciclic while

while (expresie) operator

Daca expresia este adevarata, operatorul este executat pana expresia devina falsa. Daca expresia este falsa, controlul este dat urmatorului operator.

Nota: O valoare a expresiei este definita inainte de executarea operatorului. Asadar, daca expresia este falsa de la inceput, operatorul nu mai este executa deloc.

Exemplu:

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

Operatorul ciclu for

for (expresie1; expresie2; expresie3)
operator;

Expresie1 descrie initializarea ciclului. Expresie2 este verificarea terminarii ciclului. Daca este adevarat,

este executata bucla operatorului din corp, Expresie3 este executat. Ciclul se repeta pana Expresie2 devine falsa.
Daca este fals, ciclul se termina iar controlul este dat urmatorului operator. Expresie3 este calculat dupa fiecare iteratie. Operatorul for este echivalent cu urmatoarea succesiune de operatori:

```
expresie1;
while (expresie2)
{
    operator;
    expresie3;
};
```

Exemplu:

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

Oricare dintre cele trei expresii poate fi absenta din operatorul „for”, dar nu trebuie sa omiteti punctul si virgulele (;) care le separa.

Daca Expresie2 este omisa este considerata adevarata constant. Operatorul "for(;;)" este un ciclu continuu echivalent cu operatorul "while(l)".
Fiecare dintre expresiile 1-3 poate consta in diferite expresii unite prin operatorul comma „,”.

Exemplu:

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

Functii

Definitia unei functii

O functie este definita printr-o declaratie de tip valoare returnata, prin parametri formali si operatori compusi (bloc) care descriu actiuni pe care functia le executa.

Exemplu:

```
double                          // tip
 linfunc (double a, double b)  // nume functie si
                               // lista parametri
  {                            // operatori inserati
   return (a + b);             // valoare returnata
  }
```

Operatorul return poare returna valoarea expresiei inclusa in acest operator. In caz de nevoie, valoarea expresiei isi asuma rezultatul tipului de functie. O functie care nu returneaza o valoare trebuie sa fie de tip void.

Exemplu:

```
void errmesg(string s)
  {
```

```
  Print("error: "+s);
 }
```

Functia de apel

functie_nume (x1,x2,...,xn)

Argumentele (parametri propriu-zisi) sunt transferati in functie de valoarea lor. Fiecare expresie x1,...,xn este calculata iar valoarea este trimisa functiei. Ordinea calcularii expresiilor si ordinea incarcarii valorilor este garantata. In timpul executie, sistemul verifica numarul si tipul de argumente date functiei. Un astfel de mod de adresare al functiei este denumit valoare de apel. Mai exista o alta modalitate – apel prin legatura. O functie apel este o expresie care isi asuma valoarea returnata de functie. Acest tip de functie trebuie sa corespunda tipului valorii returnate. Functia poate fi declarata sau descrisa in orice parte a programului:

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

Functii speciale *init, deinit* si *start*

Fiecare program isi incepe lucrul cu functia "init()". Functia "init()", atasata la tabele, este de asemenea lansata si dupa pornirea terminalului client, precum si in cazul schimbarii simbolurilor financiare si/sau periodicitatii graficelor.

Fiecare program isi termina lucrul cu functia "deinit()". Functia "deinit()" este de asemenea lansata de inchiderea terminalului client, inchiderea ferestrei de grafice, si la schimbarea simbolurilor financiare si/sau periodicitatii graficelor.

La cotatii noi, functia "start()" a expert advisori atasata si a programelor indicatorilor proprii este executata. Daca la cotatii noi, functia "start()", lansata la cotatiile precedente, a fost indeplinita, urmatoarea functie "start()" va fi executata doar dupa instructiunea "return()". Toate cotatiile noi primite de program in timpul executiei sunt ignorate.

Detasarea programului de grafice, schimbarea simbolurilor financiare si/sau a periodicitatii graficelor, inchiderea graficelor si de asemenea existenta terminalului client intrerup executarea programului.
Executarea scripturilor nu depinde de cotatiile provenite.

Variabile

Definitii

Definitiile sunt folosite pentru a defini variabile si pentru a exprima tipuri de variabile si functii definite in alt loc. O definitie nu este un operator. Variabilele trebuie declarate inainte de folosire. Numai constantele pot fi utilizate pentru a initializa variabile .

Tipuri de baza sunt:

- string – un sir de caractere;
- int – un unitar;
- double – variabila floating-point (precizie dubla);
- bool – un numar boolean "true" sau "false".

Exemplu:

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

Tipurile aditionale sunt:

- datatime - data si timp, integer nesemnat, continand secunde incepand cu ora 0, 1 ianuarie 1970.
- color – integer ce reflecta o colectie de componente in trei culori.

Tipurile aditionale de date sunt folositoare doar la declararea datelor de input pentru o reprezentare mai convenabila a acestora intr-o foaie de proprietati.

Exemplu:

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

Variabile sir de caractere

O variabila sir de caractere reprezinta o secventa indexata a datelor cu tipuri identice.

```
int a [50];             //O variabila sir de caractere cu o singura dimensiune de 50 de integeri.
double m[7][50];   //O variabila sir de caractere cu doua dimensiune a sapte variabile sir de caractere,
                        //fiecare dintre ele continand 50 de integer.
```

Doar un integer poate fi un index de variabila sir de caractere. Nu pot fi declarate variabile sir de caractere cu mai mult de patru dimensiuni.

Definirea variabilelor locale

Variabila declarata in interiorul oricarei functii este locala. Scopul unei variabile locale este limitat la limita functie in care o variabila este declarata. Variabila locala poate fi initializata de rezultatul oricarei expresii. Orice apel de functie executa initializarea variabilelor locale. Variabilele locale sunt stocate in zona memoriei corespunzatoare functiei.

Parametri formali

Exemplu:

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

Parametrii formali sunt locali. Scopul este blocarea functiei. Parametrii normali trebuie sa aiba nume diferite de cele ale variabilelor externe si variabilelor locale definite intr-o singura functie. Intr-un bloc al

functiei cu parametrii formali pot fi atribuite anumite valori. Parametrii formali pot fi initializati de constante. In acest caz, valoarea initializata este considerata valoare default.
Parametrii care urmeaza parametrului initializat trebuie initializati de asemenea.
Prin apelul la aceasta functie, parametrii initializati pot fi omisi, in locul lor substituindu-se cei default.

Exemplu:

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

Parametrii sunt trecuti prin valoare. Ceea ce inseamna ca modificari ale variabilelor locale corespunzatoare din interiorul functiei apel nu vor fi reflectate in nici un fel in functia apel. Ca parametri este posibil sa treaca variabile sir de caractere. Cu toate acestea, pentru o variabila sir de caracter sa treaca drept parametru, este imposibila schimbarea elementelor variabilei sir de caracter.

Exista posibilitatea de a trece parametrii prin referinta. In acest caz, modificarea acestor parametri va fi reflectata in variabilele corespunzatoare in apelul functiei. Pentru a sublinia ca parametrul este trecut prin referinta, dupa un tip de data este necesar sa fie pus modificatorul &.

Exemplu:

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

Variabilele sir de caractere pot trece ca referinte, toate modificarile vor fi reflecate in variabila sir de caractere initiala. Parametrii care au trecut ca referinte, nu pot initializati ca valori default.

Variabile statice

Clasa de memorie „static” defineste o variabila statica. Specificatorul „static” este declarat inainte de un tip de date.

Exemplu:

```
{
 static int flag
}
```

Variabilele statice sunt variabile constante din moment ce valoarea lor nu se pierde cand se iese din functie. Orice variabila din bloc, cu exceptia parametrilor formali ai functiei, pot fi definiti ca statici. Variabila statica poate fi initializata prin tipul constantei corespunzatoare, spre deosebire de o variabila locala simpla care poate fi initializata prin orice expresie. Daca nu exista o initializare explicita, variabila statica este initializata drept nula. Variabilele statice sunt initializate ca valori-singulare inainte de apelul functiei „init()”. Aceasta se intampla la iesirea dintr-o functie in care este declarata variabila statica, valoare acestei variabile nu se pierde.

Definirea variabilelor globale

Sunt definite la acelasi nivel precum functiile, adica nu sunt locale in nici un bloc.

Exemplu:

```
int Global_flag;

 int start()
{
```

```
...
}
```

Scopul variabilelor globale il reprezinta intregul program. Variabilele globale sunt accesibile din toate functiile definite in program. Sunt initializate cu zero daca nici o alta valoare initiala nu este explicit definita. Variabilele globale pot fi initializate numai prin tipul corespunzator de constanta. Initializarea variabilelor globale este efectuata cu fiecare valoare in parte inainte de executarea functiei „init()".

Nota: nu este necesar sa se amestece variabilele declarate la nivel global, cu variabile globale ale terminalului client, la care accesul se face prin functia by GlobalVariable...().

Definirea variabilelor externe

Clasa de memorie „extern" defineste o variabila externa. Specificarotul „extern " este declarat inaintea unui tip de date.

Exemplu:

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

Variabilele externe definesc datele de input ale programului, si sunt accesibile dintr-o foaie de proprietate a programului. Nu este semnificativa definirea variabilelor externe in scripturi. Variabilele sir de caractere nu se pot reprezenta ca variabile externe.

Initializarea variabilelor

Orice variabila poate fi initializata in timpul definirii acesteia. Orice variabila localizata permanent va fi initializata cu zero (0) in cazul in care nici o alta valoare initiala nu este definita in mod explicit. Variabilelel globale si statice pot fi initializate doar prin constanta tipului corespunzator. Variabilele locale pot fi initializate prin orice expresie, nu doar o constanta. Initializarea variabilelor globale si statice este efectuata cu fiecare valoare in parte.
Initializarea variabilelor locale este efectuata de fiecare data prin apelul functiei corespondente.

Tipuri de baza

Exemple:

```
  int mt = 1;           // initializarea integer
  // initializarea numarului floating-point (precizie dubla)
  double p = MarketInfo(Symbol(),MODE_POINT);
  // initializarea string
  string s = "hello";
```

Sir de caractere

Exemplu:

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

Lista elementelor sir de caractere trebuie inclusa in acolade. Daca marimea sirului de caractere este definita, valorile care nu sunt definite in mod explicit sunt egale cu 0.

Definirea functiilor externe

Tipurile functiilor externe definite intr-o alta componenta a unui program trebuie definite in mod explicit. Absenta unei astfel de definitii poate duce la erori in compilare, asamblare sau executare a programului nostru. In timpul descrierii unui obiect extern, folosinti cuvantul cheie #import cu referire la modul.

Exemple:

```
#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
```

Preprocesor

Daca primul semn dintr-un rand de program este #, inseamna ca aceast rand este o comanda de compilare. O astfel de comanda de compilare se termina cu un nou rand.

Declararea unei constante

#define identificator_valoare

Identificatorul unei constante se supune acelorasi reguli precum numele variabilelor. Valoarea poate fi de orice tip.

Exemplu:

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

Compilatorul va inlocui fiecare aparitie a unui identificator in codul sursa al valorii corespunzatoare.

Compilatia de control

#property identifcator_valoare

Lista identificatorilor constanti predefiniti.

Exemplu:

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

Constanta	Tip	Descriere
link	string	link catre pagina de internet a companiei
copyright	string	numele companiei
stacksize	int	Dimensiunea stack-ului
indicator_chart_window	void	Arata indicatorul in fereastra grafic
indicator_separate_window	void	Arata indicatorulintr-o fereastra separata

indicator_buffers	int	Numarul memoriei tampon pentru calcul, pana la 8
indicator_minimum	int	Marginea de jos a graficului
indicator_maximum	int	Marginea de sus a graficului
indicator_color X	color	Culoarea pentru afisarea liniei X, unde X e de la 1 la 8

Compilatorul va scrie valorile declarate ale setarilor modulului executabil.

Fisierele inclusive

Nota: Randul comanda #include poate fi plasat oriunde in program, dar, in general, toate incluziunile sunt plasate la inceputul codului sursa.

#include <nume_fisier>

Exemplu:

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

Preprocesorul inlocuieste acest rand cu continutul fisieerului win32.h. Semnele < si > inseamna ca fisierul win32.h va fi luat din directorul default (in general, terminal_director\experti\include). Directorul curent nu este cautat.

#include "nume_fisier"

Exemplu:

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

Compilatorul inlocuieste randul cu continutul fisierului mylib.h. Din moment ce acest nume este alaturat de semne de intrebare, cautarea se desfasoara in directorul curent (unde fisierul principal al codului sursa este localizat). Daca fisierul nu este gasit in directorul sursa, sunt cautati directori definiti in setarile compilatorului. In cazul in care cautarea nu a fost gasita nici acolo, directorul default este cercetat.

Importul de functii si alte module

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

Exemplu:

```
#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
```

Functiile sunt importata din modulele compilate MQL4 (fisiere *.ex4) si din modulele sistemului de operare (fisiere *.dll). In cazul celui din urma, functiile importate sunt de asemenea declarate. O noua comanda #import (poate fi fara parametri) incheie descrierea functiilor importate.