MetaQuotes Language 4

MetaQuotes Language 4 (MQL 4) je nový, zabudovaný jazyk pro programování obchodních strategií. Tento jazyk umožňuje vytvoření vlastních „Expert Advisorů“, kteří automaticky obstarávají řízení obchodního procesu a jsou vhodné pro implementaci vašich vlastních obchodních strategií. S pomocí MQL4 můžete vytvořit vlastní uživatelské indikátory, skripta, knihovny funkcí atd.

Velký počet funkcí nutných k analýzám aktuálních i starých cenových nabídek, základní aritmetické a logické operace, jsou obsaženy přímo ve struktuře MQL4. Jsou zde rovněž základní indikátory a příkazy, včetně systému jejich ovládání.

MetaEditor 4 IDE (Integrated Development Environment) který zvýrazňuje různé sestavy MQL4, se používá pro zapisování programových kódů. Napomáhá a zjednodušuje uživateli orientaci v odborných systémových textech. Jako informační příručka pro MQL 4 používáme MetaQuotes jazykový slovník. Stručný průvodce obsahuje funkce rozdělené do kategorií, operací, vyhrazených slov a jiných jazykových struktur, a umožňuje nalezení popisu každéo elementu, který používáme.

Programy zapsané v jazyku MetaQuotes Language mají čtvero různých vlastností a určení:

- „Expert Advisors“ je mechanický obchodní systém (MTS) spojený s určitou Osnovou. Advisor umí nejen informovat o možnostech jak uzavřít obchod, ale i provádět automaticky transakce a směrovat je přímo na server. Jako většina obchodovacích systémů, i MetaTrader 4 terminal podporuje strategie historických dat s vyobrazováním míst v tabulce, odkud obchody přicházejí a kam směřují.;
- „Custom Indicators“ jsou analogie technických indikátorů. Jinými slovy, jedná se o možnost tvorby technických indikátorů navíc k těm, které již byly integrovány v terminálu MetaTrader 4 terminal. Jako u zabudovaných indikátorů, obchody nemohou být prováděna automaticky a jsou určeny pro implementaci analytických funkcí.
- „Scripts“ jsou určené pro jednorázové provedení některých úkonů. Na rozdíl od Expert Advisorů nemají skripta přístup k funkcím indikátorů.
- Libraries jsou knihovny, kde jsou ukládány bloky uživatelských programů.

Syntax

Formát

Volná místa, odrážky, posuny řádků a symboly jsou používány jako oddělovače. Můžete použít jakýkoliv počet těchto symbolů namísto jednoho. K vylepšení čitelnosti se doporučuje používání symbolů.

Komentáře

Mnohořádkové komemtáře začínají /* symboly a jsou zakončeny */ symboly takovéto komentáře nemohou být vkládány.
Jednoduché komentáře začínají // symboly a jsou zakončeny symbolem nového řádku, přičemž mohou být vnořeny i do víceřádkového komentáře.

Komentáře jsou povoleny všude, kde je volné místo a je v nich tolerován jakýkoliv počet mezer.
Příklady:

```
// single comment

/*  multi-
    line          // vnořený jednoduchý komentář
    comment
*/
```

Identifikátory

Identifikátory jsou používány k pojmenování proměnných , funkcí a typů dat. Délka identifikátoru nesmí přesáhnout 31 znaků. Použitelné symboly: číslovky 0-9, velká a malá písmena latinky a-z, A-Z (chápána jako různé symboly), symbol podtržení (_). První symbol nemůže být číslice. Identifikátor nesmí kolidovat s žádným vyhrazeným slovem.

Příklady:

```
NAME1 name1 Total_5 Paper
```

Vyhrazená slova

Níže uvedené identifikátory jsou pevně stanovenými vyhrazenými slovy. Ke každému z nich je přiřazen určitý úkon a proto nemohou být využívány k jiným účelům:

Datové typy	**Paměťové třídy**	**Operátoři**	**Ostatní**
bool	extern	break	false
color	static	case	true
datetime		continue	
double		default	
int		else	
string		for	
void		if	
		return	
		switch	
		while	

Data Types

Hlavní datové typy jsou:

- Integer (int) – celky (celá čísla)
- Boolean (bool)
- String (string) - řetězce

- Floating-point number (double) plovoucí číselné body
- Color (color) - barva
- Datetime (datetime) – datum a čas

Barvu a datové typy potřebujeme pouze k usnadnění vizualizace a k vložení parametrů, kterými nastavujeme vlastnosti poradců nebo uživatelských indikátorů - "Input parameters" tab. Data barev a datových typů jsou reprezentována hodnotami celých čísel.

Používáme jasný typ transformace. Priorita typů při transformaci ve vzestupném seřazení vypadá takto:

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

Před provedením operací (kromě operací přiřazování) data jsou přenášena do typu s maximální přesností a před přiřazovací operací – do typu celých čísel.

Konstanty celých čísel

Decimální: číslice 0 až 9; Nula by neměla být první číslice.
Příklady:

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

Hexadecimální: číslice 0 až 9, písmena a-f nebo A-F pro reprezentaci hodnot 10-15; začínají 0x nebo 0X.
Příklady:

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

Konstanty pevných čísel mohou zahrnovat hodnoty od -2147483648 do 2147483647. Pokud konstanta přesáhne toto rozmezí, není definována.

Konstanty ve formě písmen

Jakýkoliv prostý znak obsažený jednoduchých apostrofech nebo hexadecimálních ASCII-kódech znaku jako '\x10' je znakem konstanty typu celého čísla. Některé znaky, jako jednoduchý apostrof ('), uvozovky (") otazník (?), obrácené lomítko (\) a řídící znaky , jako jsou kombinace znaků začínající obráceným (\) v souladu s níže uvedenou tabulkou:

```
line feed               NL (LF)   \n
horizontal tab          HT        \t
carriage return         CR        \r
reverse slash           \         \\
single quote            '         \'
double quote            "         \"
hexadecimal ASCII-code  hh        \xhh
```

Pokud se znaky liší od těch uvedených výše, nejsou definovány.

Příklady:

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

Konstanty „Boolean“

Booleanovy konstanty mohou mít hodnotu true nebo false, v číselném pojetí tedy 1 nebo 0. Rovněž můžeme použít symboly se stejným významem True a TRUE, False a FALSE. Příklady:

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

Konstanty bodů plovoucích čísel

Konstanty plovoucích bodů sestávají z části celých číslic, tečky (.) a frakční části. Celá čísla a frakční části jsou pokračovatelem decimálních číslic. Nevýznamná frakční část s tečkou může být vynechána. Příklady:

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

Konstanty plovoucích bodů mohou zahrnovat hodnoty v rozmezí 2.2e-308 až 1.8e308. Pokud konstanta přesáhne toto rozmezí, není definována.

Konstanty - řetězce

Řetězec je pokračováním znaků ASCII-kódu vyznačeného uvozovkami: "Character constant".

Řetězec je seskupení znaků ohraničených apostrofy. Jedná se o řetězcový typ. Každá konstanta – řetězec je uložena v oddělené části paměti. Pokud potřebujete vložit uvozovky (") do řádku, musíte předtím vepsat obrácené lomítko(\). Pokud umístíte nejprve obrácené lomítko (\), můžete za něj vložit zvláštní znak pro konstantu. Délka řetězce sestává z 0 až 255 characters. Pokud je řetězec delší, přebývající znaky směrem vpravo jsou zamítnuty. Příklady:

```
"Toto je znak řetězce"
"Copyright symbol \t\xA9"
"řádek s LF symbolem \n"
"A" "1234567890" "0" "$"
```

Konstanty barev

Konstanty barev mohou být reprezentovány třemi způsoby: reprezentace formou znaku; celým číslem; jménem (pouze u konkrétních webových barev).
Reprezentace znakem sestává ze čtyř částí prezentujících poměr numerických hodnot třech hlavních barev – červené, zelené, modré. Konstanty začínají písmenem C a jsou ohraničeny apostrofy. Numerické hodnoty barev se nacházejí v rozmezí 0 – 255.
Hodnoty celých čísel jsou zapsány hexadecimální nebo decimální formou. Hexadecimální forma vypadá takto: 0x00BBGGRR, přičemž RR je poměr červené složky, GG – zelené BB - modré. Decimální konstanty nejsou přímo definovány do RGB (barev). Reprezentují

decimální hodnotu hexadecimální prezentace. Konkrétní barvy reflektují tzv. webovou sestavu barev.
Příklady:

```
// symboly konstant
C'128,128,128'    // šedá
C'0x00,0xFF,0xFF' // modrá
// pojmenování barvy
Red
Yellow
Black
// reprezentace hodnoty celého čísla
0xFFFFFF          // bílá
16777215          // bílá
0x008000          // zelená

32768             // zelená
```

Konstanty data a času

Tyto konstanty mohou být reprezentovány jako řádky znaků sestávající z šesti částí pro hodnoty roku, měsíce, dnes, hodiny, minuty a sekundy. Konstanta je ohraničena apostrofy a začíná písmenem D. Konstanta se může pohybovat v rozmezí od 1.ledna, 1970 31.prosince, 2037.
Příklady:

```
D'2004.01.01 00:00'     // Nový rok
D'1980.07.19 12:30:27'
D'19.07.1980 12:30:27'
D'19.07.1980 12'   //rovno D'1980.07.19 12:00:00'
D'01.01.2004'      // rovno D'01.01.2004 00:00:00'
D'12:30:27'        // rovno  D'[compilation date] 12:30:27'
D''                // rovno D'[compilation date] 00:00:00'
```

Operations & Expressions

Výrazy

Výraz se skládá z jednoho nebo více operandů a operačních znaků. Výraz může být zapsán v několika řádcích.

Příklad:

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

Poznámka: Výraz zakončený středníkem je operátor.

Aritmetické operace

```
Součet hodnot                          i = j + 2;
Rozdíl hodnot                  i = j - 3;
```

```
Změna operačních znaků              x = – x;
Produkt hodnot                        z = 3 * x;
Rozdělení kvocientu                         i = j / 5;
Zůstatek dělení                      minutes = time % 60;
Přičtení 1 k proměnné hodnotě          i++;
Odečet 1 od proměnné hodnoty          k––;
```

Operace sčítání/odečítání 1 nemohou být implementovány do výrazů.
Příklad:

```
int a=3;
a++;           // platný výraz
int b=(a++)*3; // neplatný výraz
```

Přiřazení operací

Poznámka: Hodnota výrazu obsažena v této operaci je hodnotou levého operandu následujícího přiřazovací znak.

```
Přiřazení hodnoty y k proměnné x                    y = x;
Přičtení x k proměnné y                                y += x;
Odečtení x od proměnné y                          y -= x;
Násobení y proměnnou x                            y *= x;
Dělení y proměnnou x                               y /= x;
Modul x hodnota y                                      y %= x;
Logický přenos y prezentace vpravo přes x bit   y >>= x;
Logický přenos y prezentace vlevo přes x bit    y <<= x;
Bitová operace AND                                    y &= x;
Bitová operace OR                                     y |= x;
Bitová operace  exclusive OR                           y ^= x;
```

Poznámka: K výrazu může být přiřazena pouze jedna operace. Můžete implementovat bitovou operaci výhradně s celými čísly. Operace logického přenosu používá hodnoty x s méně než pěti binárními číslicemi. Vyšší číslice jsou zamítnuty, takže přenos slouží rozsahu 0-31 bitů. U %= výsledný znak je roven znaku děleného čísla.

Vztažné operace

Logická hodnota FALSE je reprezentována celým číslem nulové hodnoty, přičemž TRUE je prezentována jakoukoliv hodnotou jinou než 0. Hodnoty výrazů obsahujících vztažné operace nebo logické operace jsou 0 (FALSE) nebo 1 (TRUE).

```
True if a equals b                          a == b;
True if a doesn't equal b                   a != b;
True if a is less than b                    a < b;
True if a is greater than b                 a > b;
True if a is less than or equals b          a <= b;
True if a is greater than or equals b       a >= b;
```

Dva plovoucí body, které nejsou normalizované, nemohou být přiřazeny operacemi == nebo != . Pro tento účel je nutné odečíst jeden od druhého a normalizovaný výsledek srovnat s nulou.

Booleanovy operace

Operand of negace NOT (!) musí být aritmetického typu; výsledek je true(1) pokud je hodnota operandu false(0);

výsledek je false(0), pokud je hodnota operandu jiná než false(0).

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

Logická operace OR (||) hodnot x a y. Hodnota výrazu je true(1), pokud je x nebo y true. Jinak bude hodnota výrazu false(0).
Příklad:

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

Logická operace AND (&&) hodnot x a y. Hodnota výrazu je true(1), pokud x a y hodnoty jsou true. Jinak bude hodnota výrazu false(0).
Příklad:

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

Veškeré Booleanovy operace jsou spočítány, jinými slovy, booleanovo vyhodnocení není zkratováno.

Bitové operace

Doplnění proměnných hodnot. Hodnota výrazu obsahuje 1 ve všech číslicích, kde n obsahuje 0; Doplnění proměnných hodnot. Hodnota výrazu obsahuje 0 ve všech číslicích, kde n obsahuje 1

b = ~n;

Reprezentace x pomocí binárních kódů je posunuta doprava o y číslic. Přesunutí vpravo je logický přesun, kdy bity na levé straně budou vyplněny nulami.

Příklad:

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

Reprezentace x pomocí binárních kódů je posunuta doprava o y číslic. Přesunutí vpravo je logický přesun, kdy bity na levé straně budou vyplněny nulami.

Příklad:

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

Bitová operace AND binárně kódované prezentace x a y. Hodnota výrazu obsahuje 1 (TRUE) u všech bitů, kde jak x tak y nejsou rovny nule; Hodnota výrazu obsahuje 0 (FALSE) u všech

ostatních bitů.
Příklad:

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

Bitová operace OR binárně kódované prezentace x a y. Hodnota výrazu obsahuje 1 (TRUE) u všech bitů, kde jedno z x nebo y není rovno nule; Hodnota výrazu obsahuje 0 (FALSE) u všech ostatních bitů.
Příklad:

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

Bitová operace EXCLUSIVE OR binárně kódované prezentace x a y. Hodnota výrazu obsahuje 1 u všech bitů, kde x a y nají rozdílnou binární hodnotu; hodnota výrazu obsahuje 0 u všech ostatních.
Příklad:

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

Poznámka: Bitové operace jsou prováděny pouze celými čísly.

Ostatní operace

Indexing. Při adresování k i' elementu seskupení je hodnota výrazu rovna proměnné pod hodnotou i.
Příklad:

```
array[i] = 3;
//Přiřazení hodnoty 3 k seskupení elementů s indexem i.
//Mějte na paměti, že první element seskupení
//je popsán výrazem seskupení [0].
```

Vyvolání funkce x1,x2,…,xn argumenty. Výraz akceptuje hodnotu vrácenou funkcí. Pokud je vrácená hodnota typu „void“, nemůžete umístit vyvolání funkce vpravo v přiřazovací operaci. Mějte na paměti, že výrazy x1,x2,…,xn Mmají být provedeny v tomto příkazu .
Příklad:

```
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);
```

Operace "comma" je prováděna zleva doprava. Pár výrazů, oddělený čárkou, je propočítán zleva doprava s následujícím odstraněním levé hodnoty výrazu. Veškeré postranní účinky výpočtu levé strany se mohou projevit před kalkulací pravého výrazu. Výsledný typ a hodnota se shodují s typem a hodnotou pravého výrazu.

Prioritní pravidla

Každá skupina operací v tabulce má stejnou prioritu. Čím vyšší je priorita, tím výče je umístěna skupina v tabulce. Posloupnost provádění příkazů určuje uskupení operací a operandů.

```
()      Vyvolání funkce                 Zleva doprava
[]      Volba elementu pole
!       Negace                      Zleva doprava
~       Bitové negace
-       Znak změny operace
*       Násobení                Zleva doprava
/       Dělení
%       Dělení modulu
+       Součet                      Zleva doprava
-       Odečet
<<      Přesun vlevo                    Zleva doprava
>>      Přesun vpravo
 <      Méně než                     Zleva doprava
<=      Méně než nebo rovno
 >      Více než
>=      Více než nebo rovno
==      Rovná se                        Zleva doprava
!=      Nerovná se
&       Bitová operace AND          Zleva doprava
^       Bitová operace EXCLUSIVE OR           Zleva doprava
|       Bitová operace OR             Zleva doprava
&&      Logická AND                    Zleva doprava
||      Logická OR                     Zleva doprava
=       Přiřazení                     Zleva doprava
+=      Přiřazení součtu
-=      Přiřazení odečtu
*=      Přiřazení násobení
/=      Přiřazení dělení
%=      Přiřazení modulu
>>=     Přiřazení přesunu vpravo
<<=     Přiřazení přesunu vlevo
&=      Přiřazení bitové operace AND
|=      Přiřazení bitové operace OR
^=      Přiřazení bitové operace EXCLUSIVE OR
,       Comma (čárka)                           Zleva doprava
```

Ke změně pořadí používejte závorky.

Operators

Formát a vkládání

Formát. Jeden operátor může obsadit jeden nebo několik řádků. Dva nebo více operátorů se mohou vyskytovt na jednom řádku.

Vkládání. Provedení řídícího příkazu operací (if, if-else, switch, while a for) může být vkládáno jedno do druhého.

Mísící operátor

Mísící operátor (block) sestává z jednoho nebo více operátorů jakéhokoliv typu s ohraničením svorkami {}. Uzavírací svorka by neměla být následována středníkem (;).
Příklad:

```
if(x==0)
```

```
{
 x=1; y=2; z=3;
}
```

Vyjádření operátoru

Vyjádření zakončené středníkem (;) je operátor. Zde je několik příkladů vyjádření operátorů:

Operátor přiřazení.
Identifier=expression;
Ve výrazu můžete použít přiřazovací operátor jen jednou.
Příklad:

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

Operátor pro vyvolání funkce
Function_name (argument1,…, argumentN);
Příklad:

```
fclose(file);
```

Nulový operátor
Skládá se pouze ze středníku (;) only. Používá se k označení neplatného řídícího operátoru.

Operátor „break“

Break; operátor ruší provádění nejbližšího externě vloženého aktivátoru operace (while nebo for). Funkce je předána operátoru následujícím po tom zrušeném. Jedním z účelů používání tohoto operátoru je dokončení procesu „smyčky“ po přiřazení určité hodnoty k proměnné.
Příklad:

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

Operátor „Continue“

Continue; přesouvá operaci na začátek nejbližšího externího operátoru cyklu (while nebo for), čímž vyvolá proces dalšího opakování. Účel je opačný než u operátoru „break“.

Příklad:

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

Operátor „Return“

Return; operátor rušící aktuální funkci a vracející řízení vyvolanému programu.
Return(výraz/vyjádření); operátor rušící aktuální funkci a vracející řízení vyvolanému programu spolu s hodnotou výrazu. Výraz operátora je ohraničen svorkami. Výraz by neměl obsahovat operátor pro přiřazení.

Příklad:

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

Void funkce (neplatné) jsou nutné k dokončení (prostřednictvím "return;") operátoru bez výrazu.
Příklad:

```
return;
```

Zakončující svorka předpokládá implicitní provedení funkce „return“ pro operátora bez výrazu.

Podmiňujjící operátor if

if (výraz)
operator;

Vyjádření if je true – operátor je aktivován. Vyjádření If je false – řízení je předáno následujícímu operátoru.
Příklad:

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

Podmiňující operátor if-else

if (výraz)
operator1
else
operator2

Pokud je výraz If true, operátor1 je aktivován a proces je předán operátoru následujícímu po operátoru 2 (operátor 2 není aktivován). Pokud je výraz If false, je aktivován operátor 2.

"else" část operátoru "if" může být vynechána. U vloženého operátoru „if“ s vynechaným výrazem „else“ se může objevit dvojznačnost. Pokud k tomu dojde, „else“ adresuje nejbližší předchozí operátor „if/ do bloku, kde se „else“ nevyskytuje.

Příklad:

```
//   Výraz "else" referuje ke druhému "if" operátoru:
if(x>1)
   if(y==2)
      z=5;
else z=6;

//   Výraz "else" referuje k prvnímu"if" operátoru:
if(x>1)
  {
   if(y==2)
      z=5;
  }
```

```
else z=6;

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

Operátor „Switch“

switch (výraz)
{
case constant: operators
case constant: operators
...
default: operators
}

Srovnává hodnotu výrazu s konstantami všech proměnných typu "case" a předává kontrolu operátoru, který se blíží hodnotě výrazu. Každá variante "case" může být označena celým číslem nebo konstantním znakem či výrazem. Konstantní výraz nesmí obsahovat proměnné a funkční požadavky.

Příklad:

```
case 3+4: //platný
case X+Y: //neplatný
```

Operátory spojené s označením "default" jsou aktivovány v případě, že žádná z konstant v operátorech !“case“ není rovna hodnotě výrazu. "Default" varianta nemusí být vždy konečná. Pokud se žádná z konstant nepřibližuje hodnotě výrazu a “default“ varianta se zde nevyskytuje, není provedena žádná akce. Klíčový pojem"case" a konstanta jsou jen označením a pokud jsou operátory aktivovány pro některou z variant "case", program bude dál aktivovat operátory následujících variant až po dosažení operátoru break. To umožňuje napojení jedné série operátorů s různými variantami. Konstantní výraz je vypočítán během kompilace. Žádna ze dvou konstant v operátoru „switch“ nemůže mít stejné hodnoty.

Příklad:

```
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;
   }
```

Cyklický operator „ while“

while (výraz) operátor

Pokud je výraz true, operátor je opakován až do dosažení hodnoty výrazu false. Pokud je výraz false, kontrola je předána dalšímu operátoru.

Poznámka: Hodnota výrazu je definována před aktivací operátoru. Proto, pokud je hodnota false hned zpočátku, operátor se neaktivuje vůbec.

Příklad:

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

Cyklický operátor „ for“

for (výraz1; výraz2; výraz3)
operátor;

Výraz1 popisuje inicializaci cyklu. Výraz2 je kontrola zrušení cyklu. Pokud je hodnota true, je provedena smyčka, výraz3 je aktivován. Cyklus je opakován až do momentu dosažení hodnoty výrazu2 false.
Pokud je hodnota false, cyklus je zrušen a kontrola předána dalšímu operátoru. Výraz3 je propočítán po každém opakování. Operátor for je ekvivalentní pro následující sérii operátorů.
:

výraz1;
while (výraz2)
{
operátor;
výraz3;
};

Příklad:

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

Kterýkoliv z těchto tří výrazů nebo všechny mohou chybět u operátoru "for", je však teřba nezapomínat na středníky (;), které je oddělují.

Pokud je vynechán výraz2, jeho hodnota je považována za trvalé true. Operátor "for(;;)" je nepřetržitý cyklus ekvivalentní operátoru "while(l)".