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, skripty, 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 obsaženy 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 každého 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 prováděny automaticky a funkce je určena pro implementaci analytických funkcí.
- „Scripts“ jsou určené pro jednorázové provedení některých úkonů. Narozdíl od Expert Advisorů nemají skripty přístup k funkcím indikátorů.
- Libraries jsou knihovny, ve kterých 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 (v dřívějších verzích). Ke zlepšení čitelnosti se doporučuje používání symbolů.

Komentáře

Víceřádkové komentáře začínající symboly /* a zakončené symboly */. Tyto komentáře nemohou být vnořené.
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 namel 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) čísla s plovoucí desetinným bodem
- Color (color) - barva
- Datetime (datetime) – datum a čas

Barvu a datové typy potřebujeme pouze k usnadnění vizualizace a ke 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) jsou data přenášena do typu s maximální přesností. Před provedením přiřazovacích 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 v apostrofech nebo hexadecimálních ASCII-kódech znaku jako např. '\x10' je znakem konstanty typu celého čísla. Některé znaky, jako jednoduchý apostrof ('), uvozovky (") otazník (?), obrácené lomítko (\) a řídící znaky, jako jsou kombinace znaků začínajících obráceným lomítkem(\) 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 čísel s plovoucím desetinným bodem

Konstanty desetinných čísel sestávají z části celých čísel, 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 typ „string type“. Každá konstanta – řetězec je uložena v oddělené části paměti. Pokud potřebujete vložit uvozovky (") do řádku, musíte před něj vepsat obrácené lomítko(\). Pokud umístíte nejprve obrácené lomítko (\), můžete za něj vložit zvláštní znak pro konstantu. Délka řetězce sestává z 0 až 255 znaků. 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 jsou ohraničeny apostrofy. Numerické hodnoty barev jsou uvedeny v rozsahu 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, dne, hodiny, minuty a sekundy. Konstanta je ohraničena apostrofem a začíná písmenem D. Konstanta se může pohybovat v rozmezí od 1.ledna 1970 do 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;`
Součin hodnot	`z = 3 * x;`
Podíl hodnot	`i = j / 5;`
Zůstatek po dělení	`minutes = time % 60;`
Přičtení 1 k proměnné	`i++;`
Odečet 1 od proměnné	`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 hodnoty 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 čísly. Vyšší číslice jsou zamítnuty, přenos tak slouží rozsahu 0-31 bitů.
U operace %= výsledné znaménko je roven znaménku děleného čísla.

Porovnávací operace (relace)

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 pokud je a rovno b                         a == b;
True pokud a není rovno b                       a != b;
True pokud je a menší než b                     a < b;
True pokud je a vyšší než b                     a > b;
True pokud je a menší nebo rovno b              a <= b;
True pokud je a vyšší nebo rovno b              a >= b;
```

Dvě desetinná čísla, která nejsou normalizovaná, nemohou být přiřazeny operacemi == nebo != . Pro tento účel je nutné odečíst jedno od druhého a normalizovaný výsledek srovnat s nulou.

Booleanovy operace

Operand 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, pokud „a“ je 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 x je kontrolována první, hodnota y je kontrolována, jen je-li x true. Hodnota výrazu je true(1), pokud x a y hodnoty jsou true

Příklad:

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

Bitové operace

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

b = ~n;

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

Příklad:

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

Reprezentace x pomocí binárních kódů je posunuta doprava o y číslic. Přesunutí vlevo je logický přesun, kdy bity na pravé 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 mají rozdílnou binární hodnotu; hodnota výrazu obsahuje 0 u všech ostatních.

Příklad:

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

Poznámka: Bitové operace jsou prováděny pouze celými čísly.

Ostatní operace

Indexování. Při přiřazování i-tého elementu pole 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 definován výrazem seskupení [0].
```

Vyvolání funkce x1,x2,…,xn argumenty. Výraz akceptuje hodnotu vrácenou funkcí. Pokud je vrácená hodnota typu „void“, nemůžete umístit takové volání funkce právě v přiřazovací operaci. Mějte na paměti, že výrazy x1,x2,…,xn mají 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 výše 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 vyskytovat na jednom řádku.

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

Složený operátor

Složený operátor (blok) 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;
  }
```

Operátor výrazu

Výraz zakončený středníkem (;) je operátor. Zde je několik příkladů operátorů výrazu:

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

Příklad:

```
x=3;
y=x=3; // chyba
```

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

```
fclose(file);
```

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

Operátor „break“

Break; operátor ruší provádění nejbližšího externě vloženého operátoru switch, while nebo for. Řízení je předáno 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 operátoru je ohraničen svorkami. Výraz by neměl obsahovat operátor pro přiřazení.

Příklad:

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

Funkce Void (nevrací hodnotu) jsou nutné k dokončení (prostřednictvím "return;") operátoru bez výrazu.

Příklad:

```
return;
```

Zakončující svorka předpokládá jednoznačné 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“ je adresován na nejbližší předchozí operátor „if/ do bloku, kde se „else“ nevyskytuje.

Příklad:

```
//   Výraz "else" odkazuje na druhý "if" operátor:
if(x>1)
   if(y==2)
      z=5;
else z=6;

//   Výraz "else" odkazuje na první"if" operátor:
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 varianta "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 proměnnými. 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 výrazu false hned na začátku, 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 třeba 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)".
Každý z výrazů 1-3 může obsahovat několik dalších výrazů spojených čárkou ','.

Příklad:

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

Functions

Definice funkcí

Funkce je definována vrácenou hodnotou deklaračního typu, formálními parametry a sloučeným operátorem (block), který popisuje akce provádění funkcí.

Příklad:

```
double                          // typ
linfunc (double a, double b)    // pojmenování funkce a
                                // seznam parametrů
  {                             // vložené operace
   return (a + b);              // vrácená hodnota
  }
```

Operátor "return" může vrátit hodnotu výrazu obsaženou v tomto operátoru. V případě potřeby hodnota výrazu přejímá typ výsledné funkce. Funkce, která hodnotu nevrací, musí být typu "void".

Příklad:

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

Volání funkce

function_name (x1,x2,...,xn)

Argumenty (vlastní parametry) jsou přenášeny podle svých hodnot. Každý výraz x1,…,xn je propočítán a hodnota přenesena do funkce. Posloupnost výpočtů výrazů a nahrávání hodnot je zaručena. Během vykonávání systém kontroluje počet a typ argumentů předaných funkci. Takový způsob adresování se nazývá „value call“ (volání hodnotou). Ještě je zde další způsob – „call by link“ (volání odkazem). Volání funkce je výraz, který přejímá hodnotu vrácenou funkcí. Funkce může být otevřená nebo charakterizovaná v jakékoliv části programu:

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

Zvláštní funkce při spuštění – init/deinit

Každý program začíná pracovat prostřednictvím funkce "init()". "Init()" funkce, spojená s grafem, je spuštěná rovněž po zapnutí klientského terminálu, v případě změny finančního symbolu a/nebo periodicity grafu.

Každý program je zakončen funkcí "deinit()" . Funkce "deinit()" je spuštěna rovněž vypnutím klientského terminálu , uzavřením okna grafu, změnou finančního symbolu a/nebo periodicity grafu.

Když je přijata nová hodnota měnového páru, je provedena funkce "start()" u přiřazených expert advisorů a uživatelských indikátorů. Pokud u nových cenových nabídek funkce "start()" , která byla spuštěna u předchozí cenové nabídky, byla splněna, bude funkce provedena pouze po zásahu operátoru "return()". Veškeré nové cenové nabídky obdržené během provádění programu jsou ignorovány.

Oddělení programu od grafu, změna finančního symbolu a/nebo periodicity grafu, uzavření grafu a také opuštění klientského terminálu přeruší provádění programu.

Provádění skriptů není závislé na příchozích cenových nabídkách.

Variables

Definice

Definice jsou používány k definování proměnných a k deklaraci typů proměnných a funkcí definovaných jinde. Definice není operátor. Proměnné musí být deklarovány před použitím. K inicializaci proměnných mohou být použity pouze konstanty.

Základními typy jsou:

- string – řetězec znaků;
- int – celé číslo;
- double – číslo plovoucího bodu (dvojitá přesnost);
- bool - booleanovo číslo "true" nebo "false".

Příklad:

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

Dodatečné typy jsou:

- datatime – datum a čas, celé číslo bez znaménka, obsahující sekundy od 0 hodin, 1.ledna, 1970.
- color – celé číslo vyjadřující soubor tří barevných komponentů.

Dodatečné typy mají smysl pouze při vyjadřování vstupních dat, pro jejich pohodlnější prezentaci v listu vlastností.

Příklad:

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

Pole

Pole je indexovaná sekvence identického typu dat.

```
int a [50];          //Jednorozměrné pole s 50 celými čísly.
double m[7][50];     //Dvojrozměrné pole sedmi dalších polí,
                     //každé z nich sestává z 50 celých čísel.
```

Pouze celé číslo může být indexem pole. Vyjádřeny mohou být max. čtyřrozměrná pole.

Definice lokálních proměnných

Proměnná vyjádřená vevnitř jakékoliv funkce se nazývá lokální. Zaměření lokálních proměnných je vymezeno na vnitřní limity vyjádřené funkce. Lokální proměnná může být inicializována na konci jakéhokoliv výrazu. Každým vyvoláním funkce se provede inicializace lokálních proměnných. Lokální proměnné jsou ukládány v paměti vymezené pro odpovídající funkci.

Formální parametry
Příklady:

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

Formální parametry jsou lokálního typu. Jejich zaměření spočívá v zablokování funkce. Pojmenování formálních parametrů musí být jiné než u externích proměnných a lokálních proměnných definovaných v rámci jedné funkce. V bloku funkce směrem k formálnímu parametru jsou přiřazeny některé hodnoty. Formální parametry mohou být inicializovány konstantou. V tom případě je považována inicializační hodnota za hodnotu výchozí (default). I parametry následující po inicializačních parametrech by měly být inicializovány.

Voláním této funkce nemůže dojít k vynechání inicializačních parametrů, na jejich místo bude vložena výchozí hodnota „default".

Příklad:

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

Parametry jsou vloženy prostřednictvím hodnoty. To znamená, že modifikace odpovídajících lokálních proměnných uvnitř vyvolané funkce nebude žádným způsobem promítnuta ve volání funkce. Je možné vložit pole jako parametr. U každého pole vloženého jako parametr však není možné změnit element daného pole.

Existuje zde možnost změny parametru prostřednictvím odkazu. V tom případě bude modifikace takovéhoto parametru promítnuta do proměnné ve volané funkci.
K dokumentování, že je parametr zaměněn odkazem, je nutné po vepsání dat vložit znak &.

Příklad:

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

Pole mohou být rovněž změněny odkazem, veškeré úpravy pak budou vedeny jako odkazy promítnuté v inicializačním poli. Parametry změněné prostřednictvím odkazu nemohou být inicializovány výchozími hodnotami (default).

Statické proměnné

Paměťová třída "static" definuje statické proměnné. Specifikátor "static" je vyjádřen před datovým typem.

Příklad:

```
   {
    static int flag
   }
```

Statické proměnné jsou vlastně konstanty, jelikož po opuštění funkce nejsou jejich hodnoty ztraceny. Jakékoliv proměnné v bloku, s výjimkou formálních parametrů ve funkci, mohou být definovány jako statické. Statické parametry mohou být inicializovány odpovídajícím typem konstanty, na rozdíl od jednoduché lokální proměnné, která může být inicializována jakýmkoliv výrazem. Pokud nedojde k přímé inicializaci, statická inicializace je inicializována prostřednictvím nuly. Statické proměnné jsou inicializovány jednoduchou hodnotou před vyvoláním funkce "init()". To znamená, že při opuštění funkce, v níž byla statická proměnná vyjádřena, není hodnota proměnné ztracena.

Definování globálních proměnných

Definovány jsou na stejné úrovni jako funkce, tzn. v žádném bloku nejsou na lokální úrovni.

Příklad:

```
int Global_flag;

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

Rozsah globálních proměnných se týká celého programu. Globální proměnné jsou přístupné ze všech funkcí určených v programu. Inicializovány jsou nulou, pokud není žádná jiná inicializační hodnota přímo definována. Globální proměnná může být inicializována pouze odpovídajícím typem konstanty. Inicializace globálních proměnných je prováděna jednoduchou hodnotou před provedením funkce "init()".

Poznámka: Není nutné mísit proměnné vyjádřené na globální úrovni s globálními hodnotami klientského terminálu, k nímž je umožněn přístup prostřednictvím funkce GlobalVariable...().

Definování externích proměnných

Paměťová třída "extern" definuje externí proměnné. Specifikátor "extern" je vyjádřen před datovým typem.
Příklad:

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

Externí proměnné definují vstupní data v programu, jsou dostupné z listu programových vlastností. Nemá význam definovat externí proměnné ve skriptech. Pole nemohou samy sebe prezentovat jako externí proměnné.

Inicializování proměnných

Všechny proměnné mohou být inicializovány během svého definování. Veškeré trvale umístěné proměnné jsou inicializovány prostřednictvím nuly (0), pokud není přímo určena jejich inicializační hodnota. Globální a statické proměnné mohou být inicializovány pouze konstantou odpovídajícího typu. Lokální proměnné mohou být inicializovány jakýmkoliv výrazem a ne pouze konstantou. Inicializace globálních a statických proměnných je

provedeny pouze jednou. Inicializace lokálních proměnných je provedena pokaždé voláním odpovídající funkce.

Základní typy
Příklady:

```
int mt = 1;             // inicializace celého čísla
// inicializace čísla plovoucího bodu (zdvojená přesnost)
double p = MarketInfo(Symbol(),MODE_POINT);
// inicializace řetězce
string s = "hello";
```

Pole
Příklad:

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

Seznam elementů pole musí být ohraničen svorkami. Pokud je určen rozměr pole, hodnoty, které nejsou přesně určeny, jsou rovné nule.

Definice externích funkcí

Typ externích funkcí určený jinou komponentou programu musí být jasně definován. Absence takovéto definice může vést k chybám během kompilace, sestavování nebo provádění všeho programu. Během popisování externího objektu používejte klíčové slovo #import s odkazem modulu.
Příklady:

```
#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
```

Preprocessor

Pokud je první znak v programovém řádku #, znamená to, že je tento řádek kompilačním příkazem. Takovýto kompilační příkaz je zakončen znakem konce řádku.

Vyjádření konstanty

#define identifier_value
Identifikátor konstanty se řídí stejnými pravidly jako u pojmenování proměnné. Hodnota může být jakéhokoliv typu. Příklad:

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

Kompilátor nahradí každý výskyt identifikátoru ve vašem zdrojovém kódu odpovídající hodnotou.

Řídící kompilace

#property identifier_value
Seznam předdefinovaných identifikátorů konstant: Příklad:

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

Konstanta	**Typ**	**Popis**
link	string	Odkaz ne webové stránky společnosti
copyright	string	Jméno společnosti
stacksize	int	Velikost svazku
indicator_chart_window	void	Vyobrazení indikátoru v oknu tabulky
indicator_separate_window	void	Vyobrazení indikátoru ve zvláštním okně
indicator_buffers	int	Počet pamětí pro výpočet, až 8
indicator_minimum	int	Spodní ohraničení tabulky
indicator_maximum	int	Vrchní ohraničení tabulky
indicator_color X	color	Barva vyobrazení řádku X, kde hodnota X je 1 až 8

Kompilátor zapíše vyjádřené hodnoty do nastavení vykonatelného modulu.

Soubory „include“

Poznámka: Příkazový řádek #include může být umístěn kdekoliv do programu, obvykle však jsou veškeré podmnožiny umisťovány na začátek zdrojového kódu.
#include <file_name>
Příklad:

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

Preprocesor nahradí tento řádek obsahem souboru win32.h. Lomené závorky znamenají, že soubor win32.h bude převzat z výchozího (default) adresáře (v terminálu obvykle terminal_directory\experts\include). Aktuální adresář není vyhledáván.
#include "file_name"
Příklad:

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

```
Kompilátor nahradí tento řádek obsahem souboru mylib.h. Jelikož je toto
jméno ohraničeno uvozovkami, je provedeno vyhledání aktuálního adresáře
(kde je umístěn hlavní soubor se zdrojovým kódem). Pokud není soubor
v aktuálním adresáři nalezen, jsou vyhledány adresáře definované
v nastavení kompilátoru. Pokud není soubor nalezen ani zde, je vyhledán
výchozí adresář (default).
```

Funkce pro import a ostatní moduly

#import "file_name"
func1();

func2();
#import

Příklad:

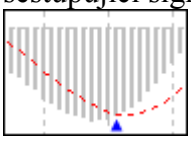
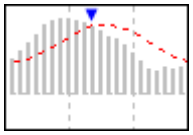
```
#import "user32.dll"
   int MessageBoxA(int hWnd,string lpText,string lpCaption,
                   int uType);
   int MessageBoxExA(int hWnd,string lpText,string lpCaption,
                     int uType,int wLanguageId);
#import "melib.ex4"
#import "gdi32.dll"
   int      GetDC(int hWnd);
   int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

Funkce jsou importovány z kompilovaných modulů MQL4 (*.ex4 files) a z modulů operačního systému (*.dll files). Později jsou deklarovány i importované funkce. Nový příkaz #import (může postrádat parametry) dokončí popis importované funkce.

Expert Advisor Sample

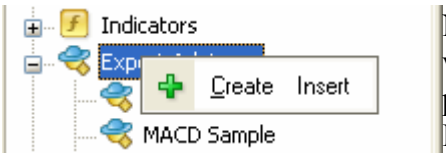
Principy vývoje programů MQL4 jsou vyobrazeny na příkladu vytvářejícím jednoduchý „Expert Advisor“ založený na systému standardního indikátoru MACD. V tomto Expert Advisoru rovněž uvidíme příklady implementace vlastností, jako jsou nastavení úrovní funkce take profit s podporou funkce trailing stop či jiných prostředků, zajišťujících bezpečnou práci. V našem případě je obchodování prováděno prostřednictvím otevírání a řízení jednoduché pozice.

Obchodní principy:

- **Vstup Long (NÁKUP)** – MACD indikátor je pod úrovní nula, stoupá a přetíná sestupující signální linii.

- **Vstup Short (PRODEJ)** – MACD indikátor je nad úrovní nula, klesá a přetíná stoupající signální linii.

- **Long exit** – při aktivaci limitu pro „take profit“, aktivaci funkce „trailing stop“ nebo při protnutí MACD se signální linií (MACD indikátor je nad úrovní nula, klesá a přetíná stoupající signální linii).
- **Short exit** – při aktivaci limitu pro „take profit“, aktivaci funkce „trailing stop“ nebo při protnutí MACD se signální linií (MACD indikátor je pod úrovní nula, stoupá a přetíná sestupující signální linii.
).

Důležité upozornění: k vyloučení nevýznamných změn MACD indikátoru (drobné 'rezonance' v tabulce) z naší analýzy zavedeme další měření, kontrolující rozsah 'rezonancí', a to tímto způsobem: velikost indikátoru by měla být alespoň 5 jednotek ve vztahu k minimální ceně (5*Point, což činí pro USD/CHF = 0.0005 a pro USD/JPY = 0.05).

Krok 1 – Zapsání popisu Expert Advisoru



Najeďte kurzorem myši do sekce „Expert Advisors“ v navigačním okně, klikněte na pravé tlačítko myši a zvolte příkaz "Create a new Expert" v menu, které se objeví. Inicializační „průvodce“ funkce „Expert Advisor“ se vás dotáže na vložení určitých dat. V okně, které se objeví, zapište jméno (Name) Expert Advisoru - MACD vzorek, autor (Author) – indikuje vaše jméno, odkaz (Link) – odkaz na vaše webové stránky a poznámky (Notes) – příklad testu Expert Advisoru na základě MACD.

Krok 2 – Vytvoření primárních struktur programu

Zdrojový kód testu Expert Advisoru bude zabírat pouze několik stránek, ale takovýto rozsah je často těžké pochopit, zejména, pokud nejste profesionálním programátorem – jinak bychom tento návod nepotřebovali vůbec, že ? :)

Abychom získali základní náhled na strukturu standardního Expert Advisoru, podívejme se na níže uvedený popis:

1. Inicializace proměnných
2. kontrola inicializačních dat
    - kontrola grafu, počtu svíček v grafu
    - kontrola hodnot externích proměnných: Počet lotů, S/L, T/P, T/S
3. Nastavení interních proměnných pro rychlý přístup dat
4. Kontrola obchodního terminálu – je prázdný? Pokud ano, pak:
    - zkontroluje: dostupnost financí na účtu atd..
    - zda je možné zaujmout „long position“ (NÁKUP)?
        - otevření „long position“ a opuštění
    - zda je možné zaujmout „short position“ (PRODEJ)?
        - otevření „short position“ a opuštění

    opuštění Expert Advisoru...

5. Kontrola dříve otevřených pozic v cyklu
    - Pokud se jedná o „long position“
        - Měla by být uzavřena?
        - Měl by být posunut trailing stop?
    - Pokud se jedná o „short position“
        - Měla by být uzavřena?
        - Měla by být posunut trailing stop?

Zdá se to být docela jednoduché, vyskytují se zde jen 4 hlavní bloky.

Nyní vyzkoušejme generování kódu krok po kroku pro každou sekci strukturálního schématu:

1. **Inicializace proměnných**
   Veškeré proměnné používané v tomto programu musí být nejprve definované v souladu se syntaxí programu MetaQuotes Language 4 . Proto vkládáme blok pro inicializační hodnoty na začátku programu

   ```
   extern double TakeProfit = 50;
   extern double Lots = 0.1;
   extern double TrailingStop = 30;
   extern double MACDOpenLevel=3;
   extern double MACDCloseLevel=2;
   extern double MATrendPeriod=26;
   ```

   MetaQuotes Language 4 je doplněn pojmem "external variables". Externí hodnoty mohou být nastaveny zvenčí bez modifikace zdrojového kódu programu expert. Tím je zajištěna dodatečná flexibilita. V našem programu je proměnná MATrendPeriod definována externí proměnnou. Definici této proměnné vložíme na začátek programu.

   ```
   extern double MATrendPeriod=26;
   ```

2. **Kontrola inicializačních dat**
   Tato část kódu se obvykle používá ve všech expert programech s drobnými úpravami, jedná se totiž o standardní kontrolní blok:

   ```
   // kontrola inicializačních dat
   // je důležité se ujistit, že expert pracuje s normálním
   // grafem, a že uživatel nechyboval v nastavování externích
   ```

```
// proměnných (Lots, StopLoss, TakeProfit,
// TrailingStop) v našem případě kontrolujeme funkci „TakeProfit"
// v tabulce s méně než 100 svíčkami
   if(Bars<100)
     {
      Print("bars less than 100");
      return(0);
     }
   if(TakeProfit<10)
     {
      Print("TakeProfit less than 10");
      return(0);  // kontrola funkce TakeProfit
     }
```

3. **Nastavení interních proměnných pro rychlý přístup k datům**
   Ve zdrojovém kódu je velmi často nutný přístup k hodnotám indikátoru nebo práce s vypočtenými hodnotami. Ke zjednodušení kódování a zrychlení přístupu jsou data vložena do interních proměnných.

```
int start()
  {
   double MacdCurrent, MacdPrevious, SignalCurrent;
   double SignalPrevious, MaCurrent, MaPrevious;
   int cnt, ticket, total;
// ke zjednodušení kódování a zrchlení přístupu
// jsou data vložena do interních proměnných
   MacdCurrent=iMACD(NULL,0,12,26,9,PRICE_CLOSE,MODE_MAIN,0);
   MacdPrevious=iMACD(NULL,0,12,26,9,PRICE_CLOSE,MODE_MAIN,1);
   SignalCurrent=iMACD(NULL,0,12,26,9,PRICE_CLOSE,MODE_SIGNAL,0);
   SignalPrevious=iMACD(NULL,0,12,26,9,PRICE_CLOSE,MODE_SIGNAL,1);
   MaCurrent=iMA(NULL,0,MATrendPeriod,0,MODE_EMA,PRICE_CLOSE,0);
   MaPrevious=iMA(NULL,0,MATrendPeriod,0,MODE_EMA,PRICE_CLOSE,1);
```