VIZUALIZAČNÍ SYSTÉM PROCOP 2.0

JAZYK BÁRA

PROGRAMÁTORSKÁ PŘÍRUČKA

SPOL. S R. O.
OSTRAVA 1998

Jazyk Bára
Programátorská příručka

Vytištěno dne : 7. listopadu 2001

1 Úvod

Jazyk Bára je rychlý výpočetní jazyk určený pro numerické výpočty v systému ProCop. Používá se pro vyhodnocování numerických výrazů a podmínek dynamizací a zároveň pro tvorbu složitějších výpočetních algoritmů prostřednictvím tzv. Bára Skriptů.

1.1 Použité konvence

Při popisu jazyka Bára jsou z důvodů větší přehlednosti v příručce dodržovány jisté konvence:

- Odkazy na jinou kapitolu této příručky nebo na její část, jsou psány kurzívou, a jsou uzavřeny do uvozovek, např. v kapitole *"Jednoduché příkazy"*
- Poznámky v textu jsou psány skloněným písmem (kurzívou) a jsou u levého okraje označeny symbolem knihy - viz kapitola *„Ikony“*
- Syntaktické diagramy jsou psány jiným typem písma a navíc jsou u levého okraje označeny ikonou počítače viz kapitola *„Ikony“*

V popisu syntaxe jsou dále použity následující konvence

- Popisné texty v diagramech jsou psány obyčejným písmem.
  `tělo procedury`
- Všechna klíčová slova jsou psána tučným písmem.
  `Program Function While For`
- Identifikátory proměnných, typů, funkcí apod. jsou psány kurzívou.
  `identifikátor_proměnné identifikátor_typu`
- Oddělovače a operátory jsou pro lepší přehlednost zvýrazněna stínováním
  `:= , . ;`
- Část diagramu uzavřená ve špičatých závorkách je nepovinná a je možno ji vypustit.
  `If` `podmínka` **`Then`** `příkaz_1 <` **`Else`** `příkaz_2 >`
- Jsou-li před ukončovací špičatou závorkou tři tečky, je možno část diagramu uzavřenou v těchto špičatých závorkách mnohonásobně opakovat.
  `< identifikátor, …>`
- Jsou-li některé položky syntaktického diagramu od sebe odděleny znakem ‘ | ’, je možno použít kteroukoliv z položek.
  `deklarace_typu | identifikátor_typu`

1.2 Ikony

K lepší orientaci v textu nám dále poslouží symboly při levém okraji stránky. Těmito ikonami jsou v příručce zvýrazněny pasáže, které mají speciální charakter:

Ikona	*Význam*	*Ikona*	*Význam*
	*Poznámky v textu*		*Syntaktický diagram*
	*Upozornění na důležité informace*		*Příklad použití*

2 ZÁKLADNÍ RYSY

2.1 Speciální symboly a rezervovaná slova

Definice základních znaků

- Písmena - znaky 'A' - 'Z', 'a' - 'z', velké i malé znaky národních abeced (ěščř…)
- Podtržítko - znak '_'
- Dekadické číslice (někdy jen číslice) - arabské číslice 0..9
- Hexadecimální číslice - arabské číslice 0..9, písmena 'A'-'F' nebo 'a' - 'f'
- Mezerové znaky (bílé znaky) - mezera, tabelátor, konec řádku

Rezervovaná slova

V jazyce Bára mohou být použita tato klíčová slova:

Analog	**For**	**Record**
And	**Function**	**Ref**
Array	**Global**	**Return**
Begin	**Goto**	**Step**
Binary	**If**	**Text**
Counter	**Label**	**Then**
Discrete	**Local**	**To**
Do	**Of**	**True**
Else	**Or**	**Type**
End	**Procedure**	**While**
False	**Program**	

2.2 Poznámky ve zdrojovém textu

Poznámky ve zdrojovém textu mohou být dvojího druhu:

- poznámky vnořené kdekoli ve zdrojovém textu a dlouhé libovolný počet řádků - uzavírají se do složených závorek, musí vždy začínat znakem '{' a končit znakem '}'. Všechen zdrojový text uzavřený závorkami je ignorován a celá poznámka je považována za mezeru
- poznámky na konci řádku - začínají vždy dvěma lomítky '//' a mají platnost pouze do konce řádku. Všechen text na řádku za těmito znaky je zcela ignorován. Tento typ poznámky nemá ukončovací znak - poznámka je automaticky ukončena koncem řádku

```
// zacatek procedury
procedure  Kon{  chybná  poznámka  }trolaMezi(  hodnota,  horni,  dolni:
analog; porucha : binary );
local
  chyba : binary;    {poznámka O.K.}
begin
  chyba := horni < {tato poznámka je O.K.} hodnota OR hodnota < dolni;
  AlarmniPromenna := chyba AND NOT( porucha );      // spusti alarm
end;
// konec procedury
```

2.3 Identifikátory

Identifikátory se mohou skládat z písmen, číslic nebo podtržítka - znaku '_'. Na prvém místě identifikátoru nesmí být nikdy číslice. Identifikátory se nerozlišují podle velikosti písmen - identifikátory 'Ident', 'ident' a 'IDENT' jsou shodné.

```
_Velikost6
Proměnná
&Pomocná              // toto je příklad chybného identifikátoru
2Mezivýsledek         // a toto taky
```

2.4 Číselné konstanty

- Celá čísla dekadická

Skládají se ze sekvence dekadických číslic.

- Čísla hexadecimální

Číslo se skládá ze sekvence hexadecimálních číslic jimž předchází znaky '0x'.

```
0xFFFF1111
0x01234
0xABCDEF
0xFG                  // chybný zápis hexadecimálního čísla
```

- Čísla v exponenciálním tvaru

Číslo v exponenciálním tvaru lze obecně zapsat jako:

```
[ +|- ] XX[.XX][E[ +|- ]XX]
```

kde:
části uzavřené v hranatých závorkách jsou nepovinné a mohou být vypuštěny
pro znaménka oddělená znakem '|' platí, že může být na daném místě vždy jen jedno z nich
znaky 'X' značí libovolné dekadické číslice

```
-1.235E+24
0.01
25986
+2E-02
```

- Znaky

Znaková konstanta je znak uzavřený do apostrofů. Pro speciální znaky je možno použít namísto znaků zpětné lomítko s hexadecimálním vyjádřením znaku v ASCII tabulce.

```
'A'
'%'
'\x00'
'\xE6'
```

- Textové konstanty

Textová konstanta je libovolný text uzavřený do uvozovek. V systému je možno používat textové konstanty maximální délky 255 znaků.

```
"Toto je textová konstanta"
```

2.5 Jednoduché datové typy

Programovací výpočetní jazyk Bára rozeznává pět jednoduchých typů proměnných. Tyto typy jsou v úzké návaznosti na typy používané v databázích monitorovacího systému. Jednotlivé typy jsou mezi sebou vzájemně nekompatibilní a typovou konverzi je nutno uskutečnit pomocí volání interních funkcí jazyka (viz dále).

Deklarace typů proměnných

Jednotlivé typy se deklarují pomocí klíčových slov shodných se jmény typů. Jsou to: Analog, Binary, Counter, Discrete a Text. Tato klíčová slova je nutno použít při deklaraci proměnných nebo v hlavičkách procedur a funkcí (viz další kapitoly).

Typ ANALOG

Tento typ je čtyřbajtové číslo s plovoucí desetinnou čárkou (v jiných jazycích například typ *float* nebo *real*). Rozsah tohoto typu je 3.4 x 10-38 až 3.4 x 10+38 s přesností asi 6 platných číslic .

Protože se jedná o číslo s plovoucí desetinnou čárkou, je vhodné si uvědomit, že doba jeho zpracování je delší než doba zpracování proměnných ostatních typů. Je proto vhodné omezit jeho použití na nejmenší možnou míru. Rychlost zpracování je také silně závislá na tom, zda je k dispozici matematický koprocesor.

Konstanty tohoto typu je možno zapsat jako celé dekadické číslo nebo číslo v exponenciálním tvaru (viz kap. 'Číselné konstanty').

Typ Binary

Typ reprezentuje pouze logické hodnoty 1 nebo 0. Pro zápis konstant tohoto typu je možno použít hodnotu 0 a 1, případně vyhrazená klíčová slova 'True' - nabývá hodnoty log. 1 a 'False' - má hodnotu log. 0.

Typ Counter

Jedná se o celočíselný typ délky 32 bitů se znaménkem s rozsahem od -2,147,483,648 do 2,147,483,647. Konstanty tohoto typu je nutno zapsat jako celé dekadické číslo nebo hexadecimální číslo ve zmíněném rozsahu.

Typ Discrete

Typ je obdobou typu Counter, avšak jeho délka je 8 bitů a je bezznaménkový. Znamená to, že jeho rozsah je pouze 0 až 255.

Typ Text

Typ text je statický ukazatel na text. V současné verzi není možno deklarovat proměnné typu text delší než 255 znaků.

Změna typu proměnných mezi sebou je možná pomocí tzv. konverzních funkcí. Jejich podrobný popis najdete v kapitole 'Interní funkce/Konverzní funkce'.

3 DIREKTIVY PREPROCESORU

Pro přehlednější zdrojový kód je možno použít direktivy preprocesoru #INCLUDE a #DEFINE.

3.1 Vložené soubory

Jazyk Bára umožňuje vkládání další souborů pomocí direktiv preprocesoru

```
#INCLUDE <jmeno_souboru>
```

případně

```
#INCLUDE "jmeno_souboru"
```

Direktiva #INCLUDE provede vložení zdrojového kódu z požadovaného souboru na dané místo.

Je-li jméno souboru uzavřeno ve špičatých závorkách <…>, předpokládá se, že se jedná o soubor v adresáři se systémovými Bára Skripty monitorovacího systému ProCop.

Je-li jméno souboru uzavřeno v uvozovkách "…", předpokládá se, že se jedná o soubor v aktuálním adresáři (případně v adresáři s projektem).

Pokus o dvojí vložení téhož souboru je automaticky rozpoznán a soubor je vložen pouze jednou.

3.2 Makra

Jazyk Bára umožňuje definovat jednoduché makra pomocí příkazu

```
#DEFINE identifikátor_makra tělo_makra
```

Pokud je v dalším zdrojovém textu nalezen identifikátor makra, pak se makro rozvine vložením těla makra na místo identifikátoru. Tímto způsobem jsou například definovány konstanty pro nejrůznější souborové operace, operace s alarmy, displeji apod.

4 VÝRAZY

Výrazy se skládají z operandů a operátorů. Části výrazů s předností ve vyhodnocování je možno uzavřít do kulatých závorek.

4.1 Operátory

Implementovány jsou operátory binární - mají tedy vždy dva operandy: jeden před operátorem a jeden za operátorem. Oba operandy musí být téhož typu. Výsledný typ operátoru spolu s dovolenými typy operandů jsou uspořádány do tabulky.

Operátor	*Úroveň operandů*	*Typy operandů*	*Typ výsledku*
***, /**	*1*	*A, C, D*	*stejné jako operand*
%	*1*	*D, C*	*stejné jako operand*
+, -	*2*	*A, C, D*	*stejné jako operand*
AND, OR	*3*	*B, C*	*stejné jako operand*
>, <, =, <>, <=, >=	*4*	*A, B, C, D*	*B*

V případě binárního operandu fungují operátory AND a OR jako logický součet a součin. Pokud je operand typu Counter, fungují tyto operátory jako bitový součet a součin.

Co se týče úrovní operátorů, platí tato základní pravidla:

- operand mezi dvěma operátory různých úrovní je vyhodnocen operátorem s vyšší úrovní
- operand mezi operátory stejných úrovní je předán k vyhodnocení levému operátoru
- část výrazu uzavřena do závorek je vyhodnocena jako oddělený výraz s nejvyšší úrovní

Poznámka : Unární logický operátor Not není implementován, je však k dispozici funkce Not s jedním parametrem.

4.2 Operandy

- konstanty - jedná se o jednoduché číselné konstanty jak byly popsány v kap. Číselné konst.
- proměnné - zde je možno použít identifikátory už deklarovaných proměnných
- volání funkce - zastoupen identifikátorem funkce s případnými parametry - bude podrobněji popsáno v kap. Příkazy jednoduché/Příkaz volání funkcí
- výraz uzavřený v závorce

```
konstanta | proměnná | volání funkce | ( výraz )
```

Pro přístup k položkám proměnných typu záznam se používají znaménka '.' a pro typ pole index uzavřený mezi hranaté závorky '[]' viz kapitola 5.2 „Deklarace složených typů“.

4.3 Výraz

Výraz je spojením operátorů a operandů, vyhodnocuje se postupně podle úrovní jednotlivých operátorů a jeho výsledkem je číselná hodnota určitého typu. Syntakticky jej lze znázornit:

```
operand < operátor operand … >
```

Není možno použít dva operátory za sebou, ale je nutno použít závorek.

```
    3 * -1      //chybný výraz
    3 * (-1)
```

Příklady výrazů – priorita operátorů:

```
global
  X : counter = 1;
  Y : counter = 2;
  Z : counter = 3;
  A  : analog;
  B : binary;
  C : counter;


begin
  A := 3.14;
     // výsledek 3.14

  C :=  Analog2Counter( A ) + X + Y * Z;
     // vyhodnocuje se nejprve Y * Z a k výsledku
     // se přičte X a A, výsledek 10

  B := 2 < X + ( Y AND  Z * 2 );
     // nejprve Z * 2, poté bitové AND s Y,
     // přičte se X a porovná s 2, výsledek TRUE
end.
```

5 STRUKTURA PROGRAMU

Struktura programu

```
Program <název_programu> ;
      deklarace návěští
      deklarace složených typů
      deklarace globálních proměnných
      deklarace procedur
      deklarace funkcí
      tělo programu
      .
```

Jednotlivé deklarace je možno mezi sebou libovolně opakovat. Žádná z deklarací není povinná, a je možno ji vynechat.

V dalších kapitolách budou postupně popsány všechny části programu.

5.1 Deklarace návěští

Návěští jsou místa programu, na něž je možno v případě potřeby 'skočit' pomocí příkazu skoku. Skok - tedy změna posloupnosti vykonávání příkazů - je možný pouze v rámci jednoho podprogramu.

Každé návěští musí být nejprve deklarováno (v místě hlavičky podprogramu, popřípadě mezi globálními deklaracemi hlavního programu) a poté definováno v místě, kam je třeba skočit. Deklarace začíná klíčovým slovem Label, poté následují jména návěští oddělená čárkou a seznam je ukončen středníkem. Seznam jmen návěští se může skládat z několika částí oddělených středníkem.

```
Label < identifikátor_návěští <,identifikátor_návěští…>  ; …>
```

Příklad deklarace návěští :

```
Label
  l1,l2;
  l3;
```

Definice návěští se provádí v libovolném místě složeného příkazu. Stačí napsat pouze identifikátor návěští a za ním dvojtečku. Syntaktický diagram vypadá takto:

```
identifikátor_návěští :
```

Příklad:

```
begin
l1:
  if( cnt < 100 ) then
    begin
      cnt := cnt + 1;
      glob := SQR( cnt );
      goto l1;
    end;
end.
```

5.2 Deklarace složených typů

Mezi složené typy patří typy pole a záznam. Deklarace typů je uvozena klíčovým slovem Type. Jednotlivé deklarace typů jsou ukončeny středníkem.

```
Type identifikátor:
     deklarace_typu;
```

Deklarace typu pole

Je možno deklarovat jednorozměrná pole jednoduchých typů i pole typů složených. Délka pole může být maximálně 255 položek. Prvky pole se indexují od 0 do (počet prvků-1). Při překročení indexu pole je generována chyba při běhu programu. Je-li potřeba deklarovat vícerozměrné pole, je třeba deklarovat pole typu pole. Maximálně je možno vytvořit třírozměrná pole.

```
Array [celočíselná_konstanta] Of
     deklarace_typu | identifikátor_typu
```

Deklarace typu pole začíná klíčovým slovem Array. Následuje rozměr pole vyjádřený kladnou celočíselnou konstantou uzavřený do hranatých závorek. Typ prvků pole je uveden za klíčovým slovem Of a může být vyjádřen identifikátorem již existujícího typu nebo deklarací nového typu.

```
type
  TMyLinearArray : array[ 32 ] of analog;
  TMy3DArray : array[ 32 ] of array[ 10 ] of array[ 5 ] of analog;
  TMyRecordArray : array[ 32 ] of record
    a : analog;
    b : binary;
  end;
```

Deklarace typu záznam

Záznam (struktura) je složena z položek různých typů. Každá položka záznamu má svůj identifikátor unikátní v rámci tohoto záznamu. Položkou záznamu může být jednoduchý nebo složený typ.

```
Record
   identifikátor položky : deklarace_typu | identifikátor_typu ;
   <
   identifikátor položky : deklarace_typu | identifikátor_typu ;
   …>
End
```

Deklarace typu záznam je uvozena klíčovým slovem Record. Následuje seznam položek oddělený středníky. Konec záznamu je označen klíčovým slovem End. Jednotlivé položky se skládají z identifikátoru položky následované identifikátorem typu položky nebo deklarací typu položky.

```
type
  TMyRecord : record
     Value  : Analog;
     Name   : text;
     Status : discrete;
     Vector   : TMyLinearArray;
  end;
```

Prvky složených typů nesmí být externí typy (kanály monitorovacího modulu).

Přístup k položkám složených typů

Pro přístup k položkám typu pole se používá index uložený v hranatých závorkách. Indexem musí být výraz typu Counter. Obdobně pro přístup k položkám typu záznam se používá znaménko '.' oddělující identifikátor proměnné od identifikátoru položky záznamu. V případě vzájemně vnořených typů se používají kombinace indexace a přístupu pomocí znaménka '.'.

Pro ilustraci jsou použity typy deklarované v předchozích dvou příkladech.

```
global
  arr : TMyLinearArray;
  arr3d : TMy3DArray;
  rec : TMyRecord;
begin
  arr[ 4 ] := arr3d[ 2*1 + 3  ][ 3 ][ 0 ];
  rec.Value := rec.Vector[ 5 ];
```

5.3 Deklarace proměnných

Proměnné se rozlišují podle místa deklarace, způsobu uchování hodnoty a použití do tří skupin: proměnných globálních, lokálních a externích.

Deklarace globálních proměnných

Globální proměnné jsou společné pro všechny procedury a funkce programu. Jsou 'viditelné' a použitelné z kteréhokoli místa programu za vlastní deklarací proměnné. Deklarace začíná vždy klíčovým slovem Global. Poté následují řádky deklarací proměnných jednoho typu oddělené středníkem.

Každý řádek se skládá z unikátního jména nové proměnné (je-li proměnných několik, pak oddělených čárkami), poté následuje dvojtečka a identifikátor typu. Je-li požadováno, aby proměnná při prvním cyklu běhu programu měla definovanou hodnotu (tedy aby byla inicializována), je třeba doplnit deklaraci znakem '=' spolu s konstantou příslušného typu.

```
Global < identifikátor < ,identifikátor …> :
                          identifikátor_typu < = konstanta > ; …>
```

```
global
  a1, a2, a3 : analog = 3.6E-1;
  c : counter;
  d, pom : discrete = 0xF;
  _bin : binary = FALSE;
```

Deklarace lokálních proměnných

Deklarace lokálních proměnných je syntakticky obdobná deklaraci proměnných globálních. Deklaraci uvozuje klíčové slova 'Local' namísto 'Global'.

Další rozdíly jsou pouze ve vlastnostech a místě deklarace lokálních proměnných. Zatímco globální proměnné jsou přístupné k použití ze všech míst programu (za místem deklarace), lokální proměnné jsou vytvářeny jen dočasně a jsou použitelné pouze zevnitř procedury nebo funkce, v níž jsou deklarovány.

Zde je už vidět druhý rozdíl, a to v místě deklarace. Lokální proměnné se definují vždy mezi hlavičkou a tělem procedury (funkce). Přesná struktura deklarace procedur je popsána ve zvláštní kapitole.

Třetí rozdíl je v inicializaci lokálních proměnných. Lokální proměnná (je-li to požadováno) je inicializována vždy při vstupu do procedury. Je třeba mít na zřeteli, že inicializaci provádějí při každém vstupu do procedury interní funkce (shodné jako při zápisu proměnná = konstanta) a tedy zpomalují běh programu.

Externí proměnné

Externí proměnné zasluhují zvláštní pozornost . Jsou jediným místem, kde se program napojuje na proces. Externí proměnné jsou ve skutečnosti proměnné uložené v databázích V/V modulů a jejich modifikací je možno upravovat potřebné hodnoty a parametry procesu. Vzhledem k tomu, že procesní hodnoty jsou distribuovány do podřízených řídících vrstev, je nutno mít na paměti dopravní zpoždění v systému. Tím se stane, že po zápisu hodnoty trvá určitý čas, než dojde k fyzickému nastavení hodnoty. Tento čas může být až řádu jednotek minut (pro telefonní modemy).

V současné verzi není třeba externí proměnné deklarovat, je možno používat skutečné názvy řídicích proměnných. Externí proměnné jsou deklarované jako typ pole rozšířený o strukturu příznaků. Pseudodeklarace těchto typů najdete v kapitole „Externí typy“.

5.4 Deklarace procedur

Procedura musí být před prvním použitím (voláním) nejprve deklarována. Deklarace procedury se skládá z hlavičky se seznamem parametrů procedury, z nepovinné deklarace lokálních proměnných, návěští a vlastního těla procedury.

```
hlavička procedury
  <
    <deklarace lokálních proměnných>
    <deklarace návěští>
  …>

  tělo procedury ;
```

Hlavička procedury

Hlavička procedury začíná klíčovým slovem Procedure. Poté následuje identifikátor procedury. V případě, že je třeba deklarovat parametry procedury, je tato deklarace uzavřena do kulatých závorek. Deklarace hlavičky je ukončena středníkem.

```
Procedure identifikátor_procedury < (deklarace parametrů) > ;
```

Deklarace parametrů procedury

Má-li procedura nějaké parametry, je nutno je deklarovat v hlavičce procedury. Deklarace parametrů je uzavřena kulatými závorkami.

Parametry mohou být libovolných typů, mohou být seřazeny v libovolném pořadí a každý z nich má svůj identifikátor. Deklarace parametrů jednoho typu začíná vždy seznamem identifikátorů parametrů oddělených čárkami. Následuje dvojtečka a identifikátor typu parametru. Je-li třeba deklarovat parametry dalšího typu, následuje středník a poté se deklarace opět opakuje. Je-li potřeba deklarovat parametry předávané odkazem, použijeme před seznam jmen parametrů klíčové slovo Ref.

```
< < Ref > identifikátor_parametru < ,identifikátor_parametru >
        : identifikátor_typu  …>
```

Parametry procedur i funkcí jsou volány hodnotou nebo odkazem. Volání hodnotou znamená, že při volání procedur s parametry se nejprve vyhodnotí výraz, který je na místě parametru, a jeho výsledná hodnota se předá proceduře. Procedura může libovolně pracovat s parametry bez nebezpečí modifikace originálních hodnot. Při předávání parametrů odkazem se namísto hodnoty proměnné předává odkaz (adresa) na tuto proměnnou a tato proměnná může být při práci s parametrem v proceduře modifikována.

Poznámka : Do procedury je možno předávat složené typy pouze odkazem, nikoliv hodnotou.

```
procedure Mocnina( ref vysl : analog; a : analog );
begin
  vysl := a * a;
end;
```

Deklarace lokálních proměnných

Deklarace proměnných byla podrobně popsána v kapitole 5.3. Proto zde nebude popisována.

Deklarace návěští

Deklarace návěští je popsána v kapitole 5.1 a nebude zde znovu popsána.

Tělo procedury

Tělo procedury je vlastně složený příkaz (tedy začíná klíčovým slovem Begin a končí klíčovým slovem End). Poté následuje středník. Popis syntaxe složeného příkazu najdete v kapitole „Příkaz složený“.

Jediným rozšířením je použití příkazu návratu 'Return'. Vykonáním příkazu návratu je běh procedury ihned bezpodmínečně ukončen a řízení se předá volajícímu podprogramu. Popis syntaxe příkazu návratu najdete v kapitole „Příkaz návratu“.

```
//Proměnné AlarmniPromenna, Teplota1, Teplota2, Stav jsou proměnné z
//databáze proměnných monitorovacího projektu.

procedure KontrolaMezi( hodnota, horni, dolni:analog; porucha:binary );
local
  chyba : binary;
begin
  chyba := ( horni < hodnota ) OR ( hodnota < dolni );
  AlarmniPromenna := chyba AND NOT( porucha );   // spusti alarm
  return;           // navrat z procedury
  chyba := TRUE;   // toto se nikdy neprovede
end;

procedure Kontroly;
begin
  KontrolaMezi( Teplota1, 50, 20, Stav );
  KontrolaMezi( Teplota2, 80, -10, Stav );
end;

begin
  Kontroly();      // telo programu
end.
```

5.5 Deklarace funkcí

Deklarace funkcí je obdobou deklarace procedur s několika odlišnostmi. Změněna je syntaxe hlavičky funkce oproti proceduře; každá funkce musí být ukončena klíčovým slovem Return následovaným návratovou hodnotou (výrazem).

V této kapitole budou popsány pouze odlišnosti oproti deklaraci procedur.

Hlavička funkce

Hlavička funkce začíná klíčovým slovem Function. Poté následuje identifikátor funkce. Je-li třeba deklarovat parametry funkce, je tato deklarace uzavřena do kulatých závorek. Následuje klíčové slovo Of uvozující identifikátor návratového typu procedury. Deklarace hlavičky je ukončena středníkem.

```
Function identifikátor_funkce < (deklarace parametrů) >
                                          Of  identifikátor_typu ;
```

Návratová hodnota funkce

Každá funkce musí být ukončena příkazem návratu začínající klíčovým slovem Return a následovaným výrazem stejného typu jako je návratová hodnota funkce (viz. hlavička - identifikátor typu za Of). Výraz se nejprve vyhodnotí a výsledek je předán do místa volání jako hodnota funkce (v příkladech např. hodnota druhé mocniny čísel nebo obsah kruhu).

Je-li ve funkci několik příkazů návratu, je vyhodnocen ten, ke kterému interpret jazyka dojde dříve (viz příklad funkce Obsah).

```
function MySQR( a : analog ) of analog;
begin
  return a * a;
end;

function Obsah( r : analog ) of analog;
begin
  if r < 0 then return 0;
  return Pi() * MySQR( r );
end;
```

5.6 Tělo programu

Tělo programu je vlastně složený příkaz ukončený tečkou. Části zdrojového textu za tečkou se nekompilují.

6 PŘÍKAZY JEDNODUCHÉ

6.1 Příkaz skoku

Pomocí příkazu skoku je možno změnit posloupnost vykonávání programu. Příkaz skoku se skládá z klíčového slova Goto za nímž následuje identifikátor návěští, kam má být přeneseno řízení.

```
Goto identifikátor_návěští
```

Vykonáním příkazu skoku program provede příkaz ležící bezprostředně za místem definice návěští (viz kapitola „Deklarace návěští“).

```
label l;
global
  a:counter;
begin
  a:=1;
l:
  a:=a+1;
  if a<10 then goto l;
end.
```

6.2 Přiřazovací příkaz

Přiřazovací příkaz slouží k přiřazení hodnoty výrazu proměnné. Je možno přiřazovat hodnotu libovolnému typu proměnné, avšak typ proměnné musí být shodný s výsledným typem použitého výrazu.

```
identifikátor_proměnné := výraz
```

Syntakticky je přiřazovací příkaz velmi jednoduchý: skládá se z identifikátoru proměnné, jíž má být hodnota přiřazena, znaku přiřazení ':=' a výrazu.

```
  a:=1
  a:=a+1
  a:=300
  b:=2*(1+SQRT(a__+2))
```

6.3 Příkaz volání procedur a funkcí

Příkaz volání procedur a funkcí provádí příkazy dané procedury nebo funkce. Volaná procedura nebo funkce musí být deklarována před příkazem volání. Jedná-li se o volání funkce, lze její návratovou hodnotu použít jako operand v přiřazovacím příkazu, v podmínce atp.

Příkaz začíná identifikátorem volané funkce nebo procedury následovaný parametry oddělenými čárkami a uzavřenými v kulatých závorkách. Kulaté závorky jsou povinné i v případě, že se volané funkci nebo proceduře žádné parametry nepředávají.

```
identifikátor_procedury ( < parametr, …> )
```

nebo

```
identifikátor_funkce ( < parametr, …> )
```

Typ použitých parametrů musí odpovídat parametrům v deklaraci funkce. Pro případnou typovou konverzi je možno použít interní konverzní funkce jazyka Bára viz kapitola „Konverzní funkce“.

```
//deklarace funkce Minimum vracející minimální hodnotu z a,b,c
function Minimum ( a, b, c : analog ) of analog
  begin
  if ( Min (a,b) = a ) then
    return Min (a,c)
    else
    return Min (b,c);
  end;


//Použití volání funkce v přiřazovacím příkazu
....
minHodnota := Minimum (x, y, z);
....

//Použití volání funkce v podmínce
....
if ( Minimum (hodnota_1,hodnota_2, Counter2Analog(hodnota_3) ) > 0)
   then .....
```

6.4 Příkaz návratu

Příkaz návratu slouží k okamžitému ukončení procedury nebo funkce a návratu do volajícího podprogramu. Příkaz návratu začíná klíčovým slovem Return. V případě, že se příkaz nachází v proceduře, následuje dále pouze středník. V případě, že se jedná o funkci, musí za klíčovým slovem Return následovat výraz, jehož typ je shodný s typem funkce. Výraz se před odchodem z funkce vyhodnotí a výsledek se předá jako výsledná hodnota funkce volajícímu podprogramu.

```
Return < výraz {jen pro funkce} >
```

Každá funkce musí obsahovat alespoň jeden příkaz návratu, který zabezpečí předání výsledku funkce volajícímu podprogramu. V opačném případě není návratová hodnota funkce definována.

```
function SQR( a : analog ) : analog;
begin
  return a * a;   // navratova hodnota funkce
end;


procedure Compute( param : analog );
label l;
local
  r : analog = 2;
begin
l:
  r := SQR( r );
  if( r > param ) return;       // navrat
  goto l;
end;
```

7 PŘÍKAZY STRUKTUROVANÉ

7.1 Příkaz složený

Příkaz složený se skládá ze sekvence jednoduchých nebo složených příkazů oddělených od sebe středníkem. Celý příkaz je uvozen klíčovým slovem Begin a ukončen klíčovým slovem End. Všechny příkazy uvedené v těle složeného příkazu se provádějí sekvenčně v pořadí, v jakém jsou zapsány. Jedinou výjimku tvoří příkaz skoku a příkaz návratu, které mohou například přeskočit několik jiných příkazů.

```
Begin
  <
     příkaz skoku ;
     příkaz návratu ;
     přiřazovací příkaz ;
     podmíněný příkaz ;
     volání procedury ;
     složený příkaz
  …>
End ;
```

7.2 Podmíněný příkaz

Podmíněný příkaz úplný a neúplný slouží k testování hodnot proměnných a k podmíněnému vykonávání potřebných akcí v závislosti na výsledku testu. Podmíněný příkaz úplný je složen z následující konstrukce

```
If podmínka Then příkaz_1 < Else příkaz_2 >
```

kde podmínka musí být výraz typu Binary, příkaz_1 je libovolný příkaz, který se provede v případě výsledku podmínky True a příkaz_2 se provede v případě výsledku podmínky False. Neúplný podmíněný příkaz je ukončen za příkazem_1 - postrádá tedy větev vykonávanou v případě, že výsledek podmínky je False.

```
Function Odmocnina( hodnota : Analog; chyba : Binary) Of Analog;
begin
  chyba := false;
  if ( hodnota < 0 ) then
    begin
    chyba := true;
    return 0;
    end
  else
    return Sqrt(hodnota);
end;
```

7.3 Programové smyčky

Programové smyčky slouží pro opakované vykonávání bloku příkazů. Jsou definovány smyčky typu For To Step, While Do a Do While.

Smyčka typu FOR - TO - STEP

Smyčka typu For provede přiřazení počáteční hodnoty do řídicí proměnné a provede porovnání řídicí proměnné s výrazem uvedeným za To. Vyhovuje-li řídicí proměnná omezující podmínce:

```
proměnná <= výraz za TO   // pro krok > 0
proměnná >= výraz za TO   // pro krok < 0
```

pak se provede tělo cyklu. Při dalších průchodech se přičte k řídicí proměnné krok (implicitně 1) a znovu se vyhodnocuje omezující podmínka. Krok je možno explicitně nastavit použitím klíčového slova Step s následující číselnou konstantou (krok je možno nastavit i záporně).

```
For  přiřazovací_příkaz To výraz < Step konstanta > příkaz
```

Například :

```
for j:=0 to SampleCount
  begin
  //tělo cyklu

  end;
```

nebo

```
for i:=2*Pi() to 0 step (-0.01)
  begin
  //tělo cyklu

  end;
```

Smyčka typu WHILE - DO

Smyčka typu While - Do (smyčka s podmínkou na počátku) opakovaně provádí tělo cyklu, dokud platí omezující binární podmínka na počátku smyčky. Není-li podmínka nikdy splněna, tělo cyklu se vůbec neprovede.

```
While podmínka Do příkaz
```

```
function GetLastValidSample ( ref t : THTrendChannelAnalog) of analog;

local
  c, tm : counter;

begin
  tm := t.LastSampleTime;           // čas posledního vzorku
  c  := t. SampleCount;             // celkový počet vzorků

  while ( Not( t.Valid[ tm ] ) ) do //dokud není vzorek validní
    begin
    tm := tm - t.Period;

    //ošetření případu, kdy v trendu není žádný validní vzorek
    c  := c - 1;
    if ( c < 0 ) then
      return 0;

    end;

  return  t [ tm ];
end;
```

Smyčka typu DO - WHILE

Smyčka typu Do - While (smyčka s podmínkou na konci) provede tělo cyklu a otestuje omezující binární podmínka na konci smyčky. Je-li podmínka splněna, je tělo smyčky provedeno znovu atd. Není-li podmínka nikdy splněna, tělo cyklu se provede právě jednou.

Aby bylo možno určit, není-li ukončovací klíčové slovo While vnořenou smyčkou typu While - Do, je nutno tělo smyčky psát jako složený příkaz.

Ve všech typech smyček může být tělem smyčky jednoduchý příkaz nebo složený příkaz (uvozen Begin a End).

```
Do příkaz While podmínka
```

Například:

```
i := 10;
do
  begin
  //tělo cyklu

  i := i - 1;

  end;
while ( i >= 0 )
```

8 INTERNÍ FUNKCE

8.1 Konverzní funkce

Typ	*Analog*	*Binary*	*Counter*	*Discrete*
A		*Analog2Binary*	*Analog2Counter*	*Analog2Discrete*
B	*Binary2Analog*		*Binary2Counter*	*Binary2Discrete*
C	*Counter2Analog*	*Counter2Binary*		*Counter2Discrete*
D	*Discrete2Analog*	*Discrete2Binary*	*Discrete2Counter*	
T	*Text2Analog*	*Text2Binary*	*Text2Counter*	*Text2Discrete*

- function Analog2Binary(a : analog) of binary;
  Funkce vrací hodnotu False pro hodnotu 0, jinak True.
- function Analog2Counter(a : analog) of counter;
- function Analog2Discrete(a : analog) of discrete;
  Obě funkce konvertují vstupní proměnnou a do příslušných typů beze změny hodnoty, je-li její rozsah v rozmezí požadovaného typu. Jinak je výsledek funkce nedefinován.
- function Binary2Analog(b : binary) of analog;
- function Binary2Counter(b : binary) of counter;
- function Binary2Discrete(b : binary) of discrete;
  Všechny funkce vrací hodnotu 1 pro True, v případě hodnoty parametru False vrací hod. 0.
- function Counter2Analog(c : counter) of analog;
  Konverze je úspěšná vždy, pouze může dojít ke ztrátě přesnosti čísla.
- function Counter2Binary(c : counter) of binary;
  Vrací hodnotu False pro hodnotu parametru c 0, jinak vrací True.
- function Counter2Discrete(c : counter) of discrete;
  Funkce konvertuje proměnnou c do typu discrete beze změn v případě, je-li její hodnota v požadovaných mezích, jinak není výsledek funkce definován.
- function Discrete2Analog(d : discrete) of analog;
  Konverze je úspěšná vždy beze změny hodnoty.
- function Discrete2Binary(d : discrete) of binary;
  Vrací hodnotu False pro hodnotu parametru d 0, jinak vrací True.
- function Discrete2Counter(d : discrete) of counter;
  Konverze je úspěšná vždy beze změny hodnoty.
- Function Text2Analog(t: text) of analog;
- Function Text2Binary(t: text) of binary;
- Function Text2Counter(t: text) of counter;
- Function Text2Discrete(t: text) of discrete;
  Všechny funkce konvertují textový řetězec na daný typ. Konverze je úspěšná vždy. Pokud řetězec obsahuje nevalidní znaky, je konverze ukončena za posledním platným znakem (bráno zleva).

```
Analog2Counter( 2 * Pi() ) - Time > 3
```

8.2 Matematické funkce

Jsou implementovány základní matematické a goniometrické funkce pracující pouze s datovým typem Analog:

Funkce	Matematicky	Popis
Pi	3.1415	Ludolfovo číslo (bez parametru)
Sqr	x^2	druhá mocnina
Sqrt	$x^{1/2}$	druhá odmocnina
Exp	e^x	exponenciála, x-tá mocnina čísla e
Pow	y^x	x-tá mocnina y (dva parametry)
Pow10	10^x	x-tá mocnina čísla 10
Log	$log_{10}(x)$	dekadický logaritmus
Ln	ln x, $log_e(x)$	přirozený logaritmus
Rad2Deg		převod radiánů na stupně
Deg2Rad		převod stupňů na radiány
Sin	sin(x)	sinus x (parametr v rad)
Cos	cos(x)	cosinus x (parametr v rad)
Tan	tg(x)	tangens x (parametr v rad)
Asin	arcsin(x)	arcussinus x (výsledek v rad)
Acos	arccos(x)	arcuscosinus x (výsledek v rad)
Atan	arctg(x)	arcustangens x (výsledek v rad)
Abs	|x|	absolutní hodnota x
Sign	sign(x)	signum (1 pro kladné číslo, -1 pro záporné číslo)
Round		zaokrouhlení
RoundDown		zaokrouhlení vždy dolů
RoundUp		zaokrouhlení vždy nahoru
Min	mininum(x, y)	minimum ze zadaných argumentů
Max	maximum(x, y)	maximum ze zadaných argumentů
Select	select(p, x, y)	je-li výraz p hodnoty TRUE, vrací x, jinak y
Random	random(x)	vrací pseudonáhodné číslo v intervalu 0..(x-1)

8.3 Funkce pro práci s datem a časem

Další skupinou funkcí jsou funkce pro práci s datem a časem. Jazyk Bára používá sekundový formát data. Určuje počet sekund od 1.1.1980. Jeho výhodou je možnost sčítání a odčítání data a času. Všechny funkce vracejí typ Counter.

Funkce	Parametr	Popis
Year		vrací aktuální rok
Month		vrací aktuální měsíc
Day		vrací aktuální den
Hour		vrací aktuální hodinu
Minute		vrací aktuální minutu
Second		vrací aktuální sekundu
Time		vrací aktuální denní čas v sekundách od půlnoci
Date		vrací aktuální datum ve dnech od 1.1.1980
DateTime		vrací aktuální datum a čas v sekundách od 1.1.1980
GetSecCount		vrací počet sekund od startu Windows
GetTickCount		vrací počet milisekund od startu Windows
GetYear	datum	vrací rok z data zadaného parametrem
GetMonth	datum	vrací měsíc z data zadaného parametrem
GetDay	datum	vrací den z data zadaného parametrem
GetHour	čas	vrací hodinu z času zadaného parametrem
GetMin	čas	vrací minutu z času zadaného parametrem
GetSec	čas	vrací sekundu z času zadaného parametrem
GetDayOnWeek	GetDayOnWeek(x)	vrací den v týdnu (0-pondělí,6-neděle)
CreateDateTime	CreateDateTime(y,m,d,h,m,s)	vytvoří ze zadaného data sekundový čas

Je-li udán parametr datum nebo čas, je možno použít i funkcí vracející datum i čas společně (např. DateTime). Platí totiž rovnost:

```
DateTime() = (Date() * 86400) + Time()
```

V uvedeném případě je potřeba vynásobit hodnotu vrácenou funkcí Date (počet dní od 1.1.1980) počtem sekund za jeden den tj.

```
60 (sec/min) * 60(min/hod) * 24(hod/den) = 86400(sec/den)
```

Chci získat číslo zítřejšího dne. Volám:

```
GetDay( (Date() + 1) * 86400 )
```

Chci získat čas o hodinu vyšší, než je okamžitý:

```
Time() + 3600
```

8.4 Funkce pro práci s řetězci

Pro práci s řetězci jsou v systémovém hlavičkovém souboru <string.bah> definovány konstanty EOS (konec souboru – hodnota 0) a STRING_MAX_LEN (maximální možní délka řetězce – hodnota 255), a dále je pro přehlednější práci s řetězci definován typ char jako proměnná typu discrete.

Funkce	***Popis***
Stralloc( ) of text	*Funkce alokuje paměťové místo pro řetězec dlouhý max. 255 znakù. Operace je nutná pro ostatní funkce modifikující obsah řetězce. Je nutno zapnout podporu pro řetězcové funkce !*
Strfree(t: text)	*Funkce uvolní paměť řetězce alokovaného funkcí stralloc.*
Strcpy(dest, src: text) of text	*Funkce kopíruje obsah řetězce src do řetìzce dest. Řetězec dest musí být alokován pomocí funkce stralloc.*
Strcat(dest, src: text) of text	*Funkce připojí obsah řetìzce dest k řetìzci src. Řetězec dest musí být alokován pomocí funkce stralloc.*
Strlen(src: text) of counter	*Funkce vrací aktuální délku řetězce.*
Strfindch(src: text; c: counter) of counter	*Funkce vyhledá první výskyt znaku c v řetězci a vrátí jeho polohu. V případě nenalezení vrací hodnotu -1.*
Strgetch(src: text; inx: counter) of char	*Funkce vrací znak z řetězce uložený na zadané pozici.*
Strputch(dest: text; inx: counter; c: char) of text	*Funkce zapíše znak do řetězce na zadanou pozici. Řetězec dest musí být alokován pomocí funkce stralloc.*
Strinsch(dest: text; inx: counter; c: char) of text	*Funkce vloží znak do řetězce na zadanou pozici. Znaky za touto pozicí odsune. Řetězec dest musí být alokován pomocí funkce stralloc.*
Strdelch(dest: text; inx: counter) of text)	*Funkce vyjme znak z řetězce ze zadané pozice. Znaky za touto pozicí odsune. Řetězec dest musí být alokován pomocí funkce stralloc.*
Strrdvld(src: text) of binary	*Funkce vrací TRUE, je-li řetězec v poøádku a je určen pro čtení.*
Strwrvld(src: text) of binary	*Funkce vrací TRUE, je-li řetězec v pořádku a je určen pro čtení nebo zápis.*
IsAlpha(c: char) of binary	*Funkce vrací TRUE, je-li znak písmenem (pracuje včetně písmen s diakritikou podle zadané kódové stránky Windows).*
IsAlphaNumeric(c: char) of binary	*Funkce vrací hodnotu výrazu (isalpha() or isnumeric()).*
IsNumeric(c: char) of binary	*Funkce vrací TRUE, je-li znak číslicí 0 až 9.*
IsUpper(c:char) of binary	*Funkce vrací TRUE, je-li znak velkým písmenem (včetně diakritiky podle nastavené kódové stránky Windows).*
IsLower(c:char) of binary	*Funkce vrací TRUE, je-li znak malým písmenem (včetně diakritiky podle nastavené kódové stránky Windows).*
ToUpper(c:char) of char	*Funkce převede znak na velké písmeno (včetně diakritiky prodle nastavené kódové stránky Windows).*
ToLower(c:char) of char)	*Funkce převede znak na malé písmeno (včetně diakritiky prodle nastavené kódové stránky Windows).*

Pro práci s řetězci je dále určen systémový Bára Skript <string.bal>, ve kterém jsou definovány další čtyři funkce pro práci s řetězci.

Funkce	Popis
Strclr(s: text) of text	*Funkce vymaže řetězec. Proměnná s obsahuje prázdný řetězec.*
Straddch(dest: text; c: char) of text	*Funkce přidá znak c na konec řetězce dest.*
Strupper(s: text) of text	*Funkce převede řetězec s na velká písmena*
Strlower(s: text) of text	*Funkce převede řetězec s na malá písmena*

8.5 Funkce pro bitovou aritmetiku

Jazyk Bára umožňuje provádět bitový součin a součet pomocí operátorů AND a OR a dále je implementována funkce pro bitovou negaci INV. Oba bitové operátory a funkce INV pracují pouze s číslem typu Counter. Návratová hodnota je rovněž typu Counter.

Funkce	Popis
INV(value: counter)	*Funkce vrací bitovou negaci čísla typu Counter.*

8.6 Funkce pro řízení běhu programu

Funkce	Popis
Sleep(delay: counter)	*Funkce pozastaví běh programu na danou dobu (v milisekundách). Po uplynutí této doby program pokračuje v místě volání funkce Sleep.*
Suspend	*pozastaví běh programu do dalšího spuštění dynamizací. Po novém spuštění program pokračuje v místě volání funkce Suspend.*
Exit	*Funkce Exit ukončí bezpodmínečně běh programu. Při novém volání programu program začíná od začátku.*

8.7 Funkce pro práci se soubory

Pro práci se soubory je potřeba vložit systémový hlavičkový soubor <files.bah>, ve kterém jsou definovány jednak konstanty pro volání funkcí FileOpen a FileSeek a jednak se zavádí typ HFILE (identifikátor souboru) jako proměnná typu counter.

Funkce	Popis
IniReadString(file, section, entry: text; dest :text) of binary	*Funkce přečte řetězec znaků ze souboru typu INI.*
IniWriteString(file, section, entry: text; dest :text) of binary	*Funkce zapíše řetězec znakù do souboru typu INI.*
FileExist(file: text) of binary	*Funkce vrátí TRUE v případě, jestliže zadaný soubor existuje.*
FileDateTime(file: text) of counter	*Funkce vrací datum a čas poslední modifikace souboru v sekundách od 1.1.1980.*
FileRename(newfile, oldfile: text) of binary	*Funkce přejmenuje soubor. Nový soubor musí být na stejném disku.*
FileCopy(destfile, srcfile: text) of binary	*Funkce zkopíruje soubor.*
FileDelete(file: text) of binary	*Funkce smaže soubor zadaného jména.*
FileOpen(file: text; openMode: discrete) of HFILE	*Funkce otevře soubor a vrátí jeho identifikátor. Identifikátor souboru je třeba používat pro následné souborové funkce. Při chybě vrací hodnotu –1.*
FileClose(hfile: HFILE) of binary	*Funkce zavře soubor.*
FileEnd(hfile: HFILE) of binary	*Funkce vrací TRUE, jestliže se ukazatel v souboru dostane na jeho konec.*
FileSeek(hfile:HFILE; offset:counter; from:discrete) of counter	*Funkce změní polohu ukazatele v souboru.*
FileSize(hfile: HFILE) of counter	*Funkce vrátí délku otevřeného souboru.*
FileReadString(hfile: HFILE; dest: text) of binary	*Funkce přečte řádek z textového souboru (ukončený znaky CR-LF).*
FileReadBytes(hfile: HFILE; dest: text; cnt: counter) of binary	*Funkce přečte zadaný počet bytù z datového souboru.*
FileWriteString(hfile: HFILE; dest: text) of binary	*Funkce zapíše řádek do textového souboru a ukončí jej znaky CR-LF."*
FileWriteBytes(hfile: HFILE; dest: text; cnt: counter) of binary	*Funkce zapíše zadaný poèet bytù do datového souboru.*

Pro úplnost jsou zde uvedeny i možné hodnoty parametru „openMode“ pro funkci „FileOpen“ a parametru „from“ funkce „FileSeek“. Parametr „OpenMode“ je orientován příznakově, takže lze použít kombinaci několika příznaků oddělených operátorem OR.

Typ přístupu	*Popis*
OPEN_READ	*Otevře soubor pro čtení.*
OPEN_WRITE	*Otevře soubor pro zápis.*
OPEN_RDWR	*Otevře soubor pro čtení i zápis.*

Typ sdílení	*Popis*
SHARE_NONE	*Soubor nelze sdílet.*
SHARE_READ	*Soubor lze sdílet pro čtení.*
SHARE_WRITE	*Soubor lze sdílet pro zápis.*

Další příznaky	*Popis*
CREATE_NEW	*Vytvoří nový soubor. Vrátí chybu, jestliže soubor již existuje.*
CREATE_OVERWRITE	*Vytvoří nový soubor. Přepíše soubor, jestliže již existuje.*
CREATE_EXISTING	*Otevře soubor. Vrátí chybu, jestliže soubor neexistuje.*
CREATE_ALWAYS	*Otevře soubor vždy. Neexistoval-li, vytvoří nový.*
CREATE_TRUNCATE	*Otevře soubor a zkrátí jej. Vráti chybu, jestliže soubor neexistuje.*

Parametr „from“ funkce „FileSeek“ udává, vůči čemu se má posun v souboru provést.

Funkce	*Popis*
SEEK_BEGIN	*Posun se provede relativně vůči začátku souboru*
SEEK_CURRENT	*Posun se provede relativně vůči aktuální pozici v souboru*
SEEK_END	*Posun se provede relativně vůči konci souboru*

8.8 Funkce pro práci s alarmy a událostmi

Pro práci s alarmy, událostmi a se systémovým zápisníkem je potřeba vložit systémový hlavičkový soubor <alarms.bah>. Obsahuje definice konstant příznaků a priorit alarmů. Pro vyvolání alarmu slouží následující funkce:

Funkce	*Popis*
SendAlarm (txt: text; flags: counter; priority: discrete)	*Zapíše text do alarmů*
SendEvent (txt: text; flags: counter; priority: discrete)	*Zapíše text do událostí*
SendLogbook(txt: text; flags: counter; priority: discrete)	*Zapíše text do syst. zápisníku*

Příznaky (flags)	*Popis*
ALF_DISPLAY	*Zobrazit alarm mezi nekvitovanými alarmy*
ALF_PRINT	*Tisknout alarm na tiskárně*
ALF_ARCHIVE	*Archivovat alarm v archivu alarmů*
ALF_SIREN	*Spustit sirénu*
ALF_ACCEPT	*Zapsat do alarmu datum a čas kvitace*
ALF_STANDARDMSG	*Text alarmu se nemění(jen pro interní použití)*
ALF_HASPARAM	*Alarm má další parametry(jen pro interní použití)*

Při vzniku a zániku alarmů je pak vhodné použít předdefinované kombinace příznaku nazvané:

ALF_SIGNAL
Zobrazí *alarm mezi nekvitovanými alarmy, vytiskne na tiskárně, archivuje v archivu alarmů, spustí sirénu a zapíše čas kvitace alarmu* (or ALF_STANDARDMSG)

ALF_NORMAL
Zobrazí *alarm mezi nekvitovanými alarmy, vytiskne na tiskárně, archivuje v archivu alarmů a zapíše čas kvitace alarmu*

Priorita	*Popis*
ALP_LOW	*Alarm s nízkou prioritou (priorita=0)*
ALP_MEDIUM	*Alarm se střední prioritou (priorita=128)*
ALP_HIGH	*Alarm s vysokou prioritou (priorita=255)*

8.9 Funkce pro práci s technologickými displeji

Pro práci s technologickými displeji je potřeba vložit systémový hlavičkový soubor <display.bah>, obsahuje definice potřebných konstant.

Funkce	*Popis*
AccessDisplay(display, param: text; action: discrete)	*Provede s diplejem požadovanou akci*

Příznaky (flags)	*Popis*
ACD_OPEN	*Znovu otevře daný displej bez ohledu na to, je-li již otevřen.*
ACD_FOCUS	*Zobrazí displej (pokud není otevřen, tak jej otevře)*
ACD_CLOSE	*Zavře požadovaný displej*
ACD_MINIMIZE	*Zmenší požadovaný displej do ikony*
ACD_MAXIMIZE	*Zvětší displej na celou plochu aplikace Process Monitor*
ACD_RESTORE	*Obnoví původní rozměry displeje*
ACD_PRINT	*Vytiskne displej na tiskárně (je li nakonfigurována)*
ACD_UPDATE	*Překreslí požadovaný displej.*

8.10 Funkce pro práci se zvukem

Pro práci se zvukem slouží funkce PlaySound a StopSound. Jejich deklarace naleznete v hlavičkovém souboru <sound.bah>.

Funkce	*Popis*
PlaySound(soundName: text; autoRepeat: binary)	*Přehraje požadovaný zvuk. Má-li parametr autoRepeat hodnotu true, bude daný zvuk přehrávat stále dokola až do ukončení přehrávání příkazem StopSound.*
StopSound(allSounds: binary)	*Ukončí přehrávání zvuku. Pokud má parametr allSounds hodnotu true, vyprázdní současně frontu zvuků pro přehrání.*

9 SEZNAM CHYBOVÝCH HLÁŠENÍ

9.1 Obecné chyby

- **"Očekává se ')' !"**

Počet pravých a levých závorek ve výrazu nebo při volání funkce neodpovídá. Zkontrolujte správnost zadání výrazu.

- **"Očekává se 'znak' !"**
- **"Chyba syntaxe !"**
- **"Neznámý identifikátor !"**

Tato skupina chybových hlášení je nejčastěji zapříčiněna syntaktickou chybou při zápisu zdrojového textu. Zkontrolujte proto správnost zápisu kódu. Chyba může být také indikována v případě chybného ukončení předchozích deklarací (např. po deklaraci proměnné začneme deklarovat funkci, avšak v klíčovém slově 'FUNCTION' se vytratí některé písmenko - kompilátor proto tuto zkomoleninu považuje za jméno další proměnné).

- **"Chyba v parametrech !"**
- **"Chyba v typu parametru !"**
- **"Funkce nemá žádné parametry !"**

V tomto případě nastala chyba při volání procedury nebo funkce. Je chybný počet nebo typ parametrů (může být i chybně zapsán výraz na místě parametru).

- **"Neočekávaný konec souboru !"**

Chyba je indikována v případě neočekávaného konce souboru. Program pravděpodobně není regulérně ukončen příkazem 'END.' (tečka je povinná). Další častou příčinou je některá neukončená poznámka (ve složených závorkách).

- **"Nekompatibilita typů !"**

Chyba nastane při kompilaci přiřazovacího příkazu v případě, kdy nesouhlasí typy proměnné a přiřazovaného výrazu. Ujistěte se o správnosti zápisu a případně použijte konverzní funkce.

- **"Chyba syntaxe v deklaraci návěští !"**

Zkontrolujte správnost deklarace návěští podle příručky.

- **"Chyba syntaxe v deklaraci proměnných !"**

Vyskytla se chyba v sekci deklarace proměnných. Překontrolujte správnost deklarace podle příručky.

- **"Duplicitní definice návěští !"**

Chyba je ohlášena, jestliže jsou nalezena dvě místa v jedné proceduře s definicí návěští shodného jména (syntax: 'návěští:').

- **"Výsledek výrazu v podmínce musí být typu Binary !"**

Výsledek výrazu v podmínce musí být vždy typu BINARY. Zkontrolujte správnost zápisu výrazu případně použijte konverzní funkce.

- **"V podmínce se očekává 'THEN' !"**

Nastává často v případě syntaktické chyby v podmíněném příkaze, kdy kompilátor nemůže najít klíčové slovo 'THEN'.

- **"Výsledek návratového výrazu musí být stejného typu jako funkce !"**

Zde nesouhlasí typ návratového výrazu za klíčovým slovem 'RETURN' s typem funkce. Překontrolujte správnost zápisu výrazu a použitých typů případně použijte konverzní funkce.

- **"Chyba v hlavičce procedury nebo funkce !"**

Chyba v deklaraci hlavičky nebo funkce. Zkontrolujte posloupnost klíčových slov, případně závorky a středníky.

- **"Chybné jméno procedury nebo funkce !"**

Jméno procedury se shoduje s rezervovaným slovem, je nutné je změnit.

- **"Chybný identifikátor !"**

Použitý identifikátor není správně zapsán (chybné znaky nebo jejich posloupnost) - je třeba upravit podle dovolených pravidel.

- **"Duplicitní identifikátor !"**

Identifikátor jména použitého v deklaraci již byl deklarován v některé předchozí části programu.

- **"Chybný identifikátor typu proměnné !"**

Při zapisování identifikátoru jména proměnné došlo k chybě. Jediné dovolené identifikátory typu jsou "ANALOG", "BINARY",... , identifikátory složených typů nebo typů externích procesních proměnných.

- **"Příliš mnoho parametrů !"**

Procedura nebo funkce může mít maximálně 16 parametrů.

- **"Chyba syntaxe v deklaraci parametrů !"**

V deklaraci parametru je syntaktická chyba. Zkontrolujte oddělovače parametrů, závorky, posloupnost jmen proměnných a jejich typů.

- **"Chyba v inicializační konstantě !"**

Inicializační konstanta byla chybně zapsána. Podívejte se na správný zápis konstanty do kapitoly Číselné konstanty.

- **"Deklarované návěští není definováno !"**

Návěští bylo deklarováno v hlavičce funkce, avšak nebylo určeno místo skoku.

- **"Chybný identifikátor typu funkce !"**

Identifikátor typu funkce byl chybně zapsán. Platí obdobné jako při chybě "Chybný identifikátor typu proměnné !".

9.2 Chyby v deklaraci programových smyček

- **"Řídicí proměnná cyklu FOR musí být typu Analog,Counter nebo Discrete!"**

Řídicí proměnná cyklu typu FOR musí být jednoho z vyjmenovaných algebraických typů.

- **"Výraz za TO nelze porovnávat s řídicí proměnnou !"**

Typ omezujícího výrazu (za TO) ve smyčce typu FOR musí být shodný s řídicí proměnnou.

- **"Chyba syntaxe, očekává se TO ve smyčce typu FOR !"**

Chybná syntaxe zápisu smyčky typu FOR, nebyla nalezena povinná část zápisu smyčky - klíčové slovo TO.

- **"Krok smyčky musí být stejného typu jako řídicí proměnná !"**

Konstanta určující krok smyčky (nepovinné) musí být stejného typu jako řídicí proměnná smyčky.

- **"Chyba syntaxe, očekává se DO ve smyčce typu WHILE !"**

Chybná syntaxe zápisu smyčky typu WHILE DO, nebyla nalezena povinná část zápisu smyčky - klíčové slovo DO.

- **"Chyba syntaxe, očekává se WHILE ve smyčce typu DO !"**

Chybná syntaxe zápisu smyčky typu DO WHILE, nebyla nalezena povinná část zápisu smyčky - klíčové slovo WHILE.

9.3 Chyby při práci se složenými typy

- **"Typ nebyl nalezen !"**

Identifikátor typu nebyl nalezen, zkontrolujte jeho jméno a syntaxi.

- **"Chyba syntaxe v deklaraci typů !"**
- **"Očekává se OF v deklaraci typu pole !"**

Deklarace typu neodpovídá pravidlům pro deklaraci typů popsaným v této příručce.

- **"Duplicitní identifikátor typu !"**

Identifikátor typu již byl použit, je třeba zvolit jiný název pro typ.

- **"Příliš mnoho položek v deklaraci strukturovaného typu !"**

Strukturovaný typ může mít v současné verzi max. 16 položek. V případě potřeby většího počtu položek, deklarujte část položek jako zvláštní typ a tento typ vnořte.

- **"Externí typ nesmí být položkou složeného typu !"**

Položkami složeného typu smí být pouze jednoduché typy nebo uživatelem definované typy.

- **"Typ indexu pole musí být typu Counter !"**
- **"Délka pole musí být v rozsahu 1 až 255 položek !"**
- **"Index do pole je větší než délka tohoto pole !"**

Index do proměnné typu pole musí být výraz typu Counter a musí být v rozsahu 0 <= index < rozsah, přičemž rozsah pole musí být v intervalu <1;255>.

- **"Očekává se ']' !"**
- **"Očekává se '[' !"**

Chybí otevírací nebo uzavírací závorka indexu pole. Doplňte ji nebo změňte zadání.

- **"Pole má příliš mnoho rozměrů !"**

V současné verzi je možno vnořit pole max. do třetí úrovně - je tedy možno vytvořit maximálně třírozměrná pole.

9.4 Interní chyby

- **"Interní neidentifikovaná chyba !"**
- **"Interní chyba při zápisu kódu !"**

K těmto chybám by nemělo při používání nikdy dojít a jejích případný výskyt je nutno hlásit dodavateli SW.

9.5 Chyby za běhu programu

- **"Stack overflow !"**
  Chyba přetečení zásobníku.

- **"Division by zero !"**
  V programu se vyskytlo dělení nulou.

- **"Invalid parameter !"**
  Volání funkce Sleep se záporným parametrem a matematické funkce s chybným parametrem (např. odmocnina ze záporného čísla apod.)

- **"Bad array index !"**
  Odkaz na index pole větší než deklarovaná délka pole

- **"Timeout occured !"**
  Doba běhu programu překročila definovaný maximální možný čas běhu. Do doby běhu programu se počítá pouze čas běhu programu bez přerušení funkcí Sleep nebo Suspend.

- **"Internal error !"**
  Interní chyba v programu. Tato chyba by neměla nikdy nastat.

- **"String using error !"**
  Chyba při použití stringových operací. Například použití nealokovaného řetězce, zápis za max. délku řetězce (255 znaků) apod.

- **"User exit !"**
  Ukončení programu funkcí Exit.

- **"Using File Error !"**
  Volání souborové funkce s nevalidním identifikátorem souboru.

- **"Functions are not enabled in HW key !"**
  Volání funkce, která není povolena v HW klíči. Jedná se o souborové a řetězcové funkce, které je potřeba mít povoleny v HW klíči.

- **"External functions not installed !"**
  Externí funkce není instalována. Za externí funkci je považováno například funkce pro přepínání displejů, funkce pro práci s alarmy a zvukem apod. Tato chyba by neměla nikdy nastat.

10 Dodatky

10.1 Externí typy

Typ TIOChannel

Vstupně/výstupní kanály byly deklarovány jako pole hodnot některého z jednoduchých datových typů. Později byly rozšířeny o příznaky popisující jejich stav. Příznaky jsou deklarovány jako struktura základních typů a polí základních typů.

Kanál je tedy typu Record a je definován následovně:

```
Type
  TIOChannelXXXXX : Record
    Value     : Array [YY] Of XXXXX;
    Status    : Discrete;  // Input
    Valid     : Binary;    // Input
    Accept    : Binary;    // InOut
    AccessLog : Binary;    // InOut
    Name      : Text;      // Input
    Descr     : Text;      // Input
    TypeName  : Text;      // Input
  End;
```

kde XXXXX může být některý z typů Analog, Binary, Counter, Discrete, Text. V/V kanál typu TIOChannelAnalog tedy obsahuje hodnotu (Value) typu Analog.

- **Value**
  Příznak obsahuje hodnotu dané proměnné. Typ hodnoty je stejný jako typ kanálu. Na hodnoty proměnné se můžete odkazovat také přímo jako na kanál s vynecháním příznaku Value.
  Hodnota kanálu je deklarována jako pole i v případě, že se nejedná o pole. Rozměr pole je v tomto případě 1. Na proměnnou se tedy lze i v tomto případě odkazovat také pomocí indexu pole jako na Proměnná[0].
- **Status**
  Příznak Status může obsahovat dodatečné informace o proměnné, jako například překročení mezí, chybový stav čidla apod.
- **Valid**
  Hodnota True označuje validitu dané proměnné. Hodnota False znamená, že daná proměnná není validní. Pokud se jedná o pole proměnných ,váže se validita k tomuto poli jako celku.
- **Accept**
  Pokud je na základě daného V/V kanálu vygenerován alarm, je při kvitaci tohoto alarmu do příznaku Accept zapsána hodnota True. (V současné verzi není tento příznak využíván)

- **AccessLog**
  Pokud je hodnota tohoto příznaku True a do hlavní hodnoty kanálu je proveden zápis nové hodnoty, je vygenerována událost informující o zápisu do dané proměnné.
- **Name**
  Příznak Name obsahuje název proměnné z Visual Designeru včetně prefixů cesty, modulu , případně skupin(y).
- **Descr**
  Příznak Descr obsahuje popis dané proměnné z Visual Designeru.

Příklady použití:

```
// x1, x2, x3 jsou globální nebo lokální proměnné odpovídajících typů
  x1 := c;
  x2 := c[ 5 ];
  x3 := c.Status;
```

Typ THTrendChannel

Historické trendy mají všechny základní příznaky jako V/V kanály rozšířené o některé další položky. Pouze příznak Valid je deklarován jako pole pro každou hodnotu trendu zvlášť.

```
Type
  THTrendChannelXXXXX : Record
    ...
    Valid           : Array [YY] Of Binary  // Input
    ...

    Present         : Array [YY] of Binary  // Input
    Accept          : Binary                // InOut
    AccessLog       : Binary;               // InOut
    Name            : Text;                 // Input
    Descr           : Text;                 // Input
    TypeName        : Text;                 // Input

    TrendName       : Text;                 // Input
    FileName        : Text;                 // Input
    Period          : Counter               // Input
    SampleCount     : Counter               // Input
    LastSampleTime  : Counter               // Input
    Lock            : Binary                // InOut
  End;
```

kde XXXXX může být některý z typů Analog, Binary, Counter, Discrete. Trend typu THTrendChannelAnalog tedy obsahuje hodnoty (Value) typu Analog.

Jednotlivé příznaky mají stejný význam jako v případě V/V kanálu. Navíc obsahuje typ THTrendChannelXXXXX tyto příznaky:

- **Present**
  Hodnota True znamená, že daný vzorek existuje. Hodnota False označuje chybějící vzorek (monitorovací systém například nebyl spuštěn apod.).
- **TrendName**
  Název daného trendu z Visual Designeru.
- **FileName**
  Název souboru s trendem.
- **Period**
  Perioda vzorkování v sekundách.
- **SampleCount**
  Maximální možný počet uchovávaných vzorků trendu.
- **LastSampleTime**
  Datum a čas posledního vzorku..
- **Lock**
  Drží mapu trendového souboru v paměti a soubor s trendy nechává otevřený, čímž výrazně zrychluje přístup k trendům.

Trendy se indexují časem stejně jako validita(Valid) a existence(Present) každého vzorku. Je možno použít 2 způsoby indexace:

- přímo sekundovým časem
- indexy označujícími pořadí v trendovém souboru:
  0: teď (poslední vzorek)
  1: teď – perioda (předposlední vzorek)
  2: teď - 2 * perioda atd..

Příklady použití:

```
x1 := t[DateTime()-3600]; //hodnota před hodinou
x2 := (t[0]+t[1])/2; //průměr z posledních dvou vzorků
x3 := t.Period; //perioda trendu
x4 := t.Valid[ 5 ]; //validita šestého vzorku od konce
```

Typ TLastReceivedSMS

Typ TLastReceivedSMS je použit v modulu pro vysílání a příjem krátkých textových zpráv (IOSMS.IOM). Systémová proměnná LastReceivedSMS díky tomu obsahuje nejen text přijaté SMS ale taktéž informace o času odeslání a tel. číslo odkud byla SMS odeslána.

Typ TLastReceivedSMS má všechny základní příznaky jako V/V kanály rozšířené o některé další položky. Hodnota Value je deklarována jako pole typu Text o deseti prvcích.

```
Type
  TLastReceivedSMS : Record
    ...
    Value     : Array [10] Of Text;     // InOut
    ...

    Source    : Array [10] Of Text;     // Input
    DateTime  : Array [10] Of Counter; // Input
  End;
```

Kromě standardních příznaků přibyly navíc tyto příznaky:

- **Source**
  Telefonní číslo, odkud byla daná krátká textová zpráva (SMS) odeslána.
- **DateTime**
  Datum a čas odeslání SMS.

Oba příznaky je možno indexovat v rozsahu 0-9. Výsledkem je tedy datum a čas + zdrojové telefonní číslo posledních deseti přijatých SMS.

TNitelChannel

Typ TNitelChannel je použit pro všechny proměnné V/V modulu Nitel. Typ TNitelChannel má všechny základní příznaky jako běžné V/V kanály, pouze je rozšířen o příznak Manual.

```
Type
  TNitelChannelXXXXX : Record
    ...

    Manual    : Binary;                //InOut
  End;
```

kde XXXXX může být některý z typů Analog, Binary, Counter, Discrete. Kanál typu TNitelChannelAnalog tedy obsahuje hodnoty (Value) typu Analog.

Příznak Manual znamená:

- hodnota True – proměnná je v manuálním režimu ovládání
- hodnota False – proměnná je automatickém režimu ovládání

Typ PRUTSP

Typ PRUTSP je určen pro práci s časovými katalogy ve stanici PRU, RWP.

Jelikož tento typ je značně složitý a pro běžného uživatele nemá znalost jednotlivých položek tohoto typu žádný význam, nebude dále tento typ popisován.

Pro práci s časovými katalogy je určena speciální entita TSP Control (v programu Visual Designer). Pomocí této entity je možno časové katalogy prohlížet i nastavovat a to vysoce komfortním způsobem. Více informací o entitě TSP Control naleznete v dokumentaci „Visual Designer“.

10.2 Příklady uživatelských programů

Výpočet průměrné hodnoty z trendu

Následující program slouží k výpočtu průměrné hodnoty z posledních N vzorků trendu. Vzorek trendu se zahrnuje do průměru jen v případě, že vzorek je validní.

Výpočet provádí funkce ComputeAverage. Jako parametry se této funkci předávají reference na trendový kanál a počet vzorků pro výpočet průměru.

```
program Average;

function ComputeAverage( ref t : THTrendChannelAnalog; cnt : counter )
  of analog;

local
  i, tm : counter;
  sum, c  : analog;
  period : counter;
begin
  sum := 0;
  c := 0;
  period := t.Period;                  // perioda trendu
  tm := t.LastSampleTime;              // čas posledního vzorku


  for i:= 0 to cnt begin               // z kolika vzorků počítám ??
    if( t.Valid[ tm - i * period ] ) then    // je hodnota validní ?
      begin
         sum := sum + t[ tm - i * period ];  // tak přičti hodnotu
         c := c + 1;                   // a zvyš počitadlo vzorků
      end;
  end;

  if ( c = 0 ) then
    return 0        // v posledních n vzorcích není žádný validní
    else
    return sum / c;  // součet poděl počtem vzorků
end;


begin
  DBS_TMP_Average := ComputeAverage( TRND_Manual, 10 );
end.
```

Pravidelné ukládání hodnot proměnných

Program provádí vždy v danou hodinu uložení technologických proměnných PIT_* do proměnných databázového modulu DB_PIT_*. V našem případě program hodnoty ukládá vždy v 5 hodin ráno. V první podmínce se testuje zda v daný den již nebyly uloženy hodnoty, a pokud ano program se ukončí. Ve druhé podmínce se testuje, zda je aktuální denní čas >5:00 hod, a pokud je podmínka splněna, jsou do proměnných DB_PIT_* přiřazeny hodnoty DB_*. Na závěr je zapsán do proměnné databázového modulu DB_LASTSAVE aktuální čas. Dále je zajištěno ukládání hodnot databázového modulu zapsáním hodnoty true do systémové proměnné DB_SaveData.

Pro správnou funkci programu v Process Monitoru je potřeba zajistit spouštění tohoto programu (jako Bára Script v globálních dynamizacích) několikrát za hodinu. Bude–li například perioda spouštění programu 3 minuty, budou hodnoty uloženy nejpozději v 5:03.

```
program SaveValues;
begin

//pokud se uz dnes ulozilo
if ( GetDay(DB_LastSave) = Day() ) then
  return;

//je-li po 5. hodine
//tady program nedojde, pokud uz ulozil - viz predchozi podminka
if ( Hour()>=5 ) then
  begin
  DB_Pit_Q:=Pit_Q;
  DB_Pit_F:=Pit_F;
  DB_Pit_P:=Pit_P;
  DB_Pit_Cl:=Pit_Cl;
  DB_Pit_PH:=Pit_PH;

  DB_LastSave:=DateTime();
  DB_SaveData:=true;
  end;

end.
```