METODIKA ZÁPISU ALGORITMŮ VE VYŠŠÍCH JAZYCÍCH

Vladimír P a v l í k

Anotace

Příspěvek se zabývá především otázkou zápisu algoritmů ve vyšších programovacích jazycích s cílem sjednocení a zpřehlednění zápisů algoritmů při zachování zásad strukturovaného programování. Dále se zabývá metodikou tvorby identifikátorů různých objektů v programu, které mají globální charakter, a názvů souborů, obsahujících zdrojové programy.

Celá problematika vychází z bohatých zkušeností při tvorbě programových systémů reálného času týmem programátorů. Je proto orientována na týmovou práci programátorů při budování složitějších programových systémů. Některé poznatky lze uplatnit, kromě systémů reálného času, i v jiných oblastech programování.

Příklady jsou uváděny v novějších programovacích jazycích, jako je PASCAL, C, PL/M, jejichž používání by mělo v současné době převládat (PL/M je míněn převážně pro 8-bitové mikropočítače) na všech typech mikropočítačů, minipočítačů a větších počítačů.

Klíčová slova

STRUKTUROVANÉ PROGRAMOVÁNÍ, PROGRAMOVACÍ JAZYK, VÝVOJOVÉ A CÍLOVÉ PROSTŘEDÍ, DATOVÉ STRUKTURY, GLOBÁLNÍ IDENTIFIKÁTORY, ALGORITMUS PROGRAMU, STRUKTURA PROGRAMU.

1. Úvod

Principy strukturovaného programování byly definovány již počátkem 70. let, dnes jsou tyto zásady uznávány převážnou většinou programátorů i u nás. Výhody strukturovaného programování jsou nesporné ve všech etapách vývoje programů: od návrhu datových struktur a algoritmu, přes realizaci až k úplnému odladění systému. Strukturované programování také přináší řadu vedlejších výhod, jako je srozumitelnost, snadná modifikovatelnost programů a další výhody.

Moderní programovací jazyky, které respektují zásady strukturovaného programování, mají zajištěnou životaschopnost, protože se snadno učí, dobře se s nimi pracuje, a proto jsou rozšířeny na celém světě (především se jedná o jazyky PASCAL, C, PL/I a jiné). Při vývoji složitějších programových systémů zvláště v řešitelských týmech, však i přesto existuje celá řada problémů, které za člověka nevyřeší ani dobrý programovací jazyk, ani skutečnost, že máme v týmech dobré programátory.

Tento krátký příspěvek si neklade za cíl poučovat programátory o tom, jak by měli zapisovat své algoritmy v jazycích PASCAL, C nebo PL/M, ale poukazuje na ty zásady, na první pohled bezvýznamné, které mohou usnadnit práci programátorů v řešitelských kolektivech při vývoji větších programových systémů. U větších programových systémů je obvykle větší množství společných proměnných, datových struktur, souborů, atd., takže zavedením mnemotechniky

při vytváření společně užívaných identifikátorů různých objektů se všech programátorům usnadní orientace v celém systému. Dále je u týmové práce důležitá zásada jednotného zápisu algoritmů všemi programátory a to jak na straně dat, tak i na straně vlastních algoritmů (těla programu). Dodržováním těchto zásad se programy stanou srozumitelnými nejen autorovi po letech, ale i dalšími členy týmu, a také může být sjednocena programová dokumentace.

Zkušenosti shrnuté v příspěvku vycházejí z realizace univerzálního programového systému, který slouží k implementaci řídicích a informačních systémů. V žádném případě nemohu tvrdit, že řešení tohoto systému probíhalo příkladně a bez problémů, ale v každém případě na začátku řešení byly snahy o jednotnost zápisu algoritmů alespoň na úrovni dokumentace a jednotná forma zápisu globálních identifikátorů, ale některé principy nebyly dotaženy až do konce, a tak se někdy dodržováním jistých zásad řešení zkomplikovalo místo aby se řešení zjednodušilo. Právě ony negativní zkušenosti z vývoje velkého programového systému mě vedly k definování metodických zásad týmové práce.

Před zahájením programátorských prací by měly být vedoucím týmu nebo metodikem týmu zodpovězeny následující důležité otázky a definovány následující konvence:

1. Jaký programovací jazyk bude pro řešení použit ?

Volba programovacího jazyka je závislá na tom, jakou problematiku daný tým řeší (např. operační systém nebo jeho nádstavbové systémové programové vybavení - C, aplikační nebo uživatelské programové vybavení - PASCAL, PL/I, FORTRAN), v jakém vývojovém popř. cílovém operačním systému tým pracuje (např. UNIX - C; MS DOS, PP DOS - C, PASCAL; RSX-11M, DOS-RV - PASCAL, FORTRAN; RMX + ISIS, ERČ + DOS-MVS - PL/M; CP/M, MIKROS - PASCAL, PL/I;...) a také je třeba přihlédnout ke zkušenostem řešitelů s určitým programovacím jazykem. Rozhodně však musí být zvolen jediný programovací jazyk pro všechny členy týmu s jistými výjimkami (programování strojově závislých částí nebo kritických větví přímo v assembleru), které povolí vedoucí týmu. Jestliže výběr programovacího jazyku znamená přeškolení některých programátorů na jiný jazyk, pak by to nemělo být překážkou. Zvládnutí jiného programovacího jazyka při znalosti alespoň jednoho jazyka je otázkou max. 14 dnů, zvládnutí celého systému včetně práce s editorem, překladačem, sestavovacím, umisťovacím a ladicím programem by nemělo přesáhnout 1 měsíc. Vzhledem k celkové době řešení (obvykle 2-3 roky) se to vyplatí.

2. Jaké bude použito vývojové prostředí, popř. cílový operační systém ?

V dnešní době je obvykle možnost výběru více operačních systému na jednom počítači. U mikropočítačů se někdy setkáváme s oddělením vývojového prostředí (operační systém, ve kterém vyvíjíme programový systém) od cílového prostředí (operační systém, ve kterém je programový systém provozován). Obvykle však bývá vývojové i cílové prostředí totožné. Před zahájením programátorských prací je nutné zvládnout zvolený operační systém nejen z hlediska způsobu práce s jeho prostředky z konzoly, ale také z hlediska vnitřní struktury a služeb, které poskytuje uživatelským programům. Kromě jazyka C lze jen velmi obtížně převádět programový systém z jednoho prostředí do jiného (z jednoho počítače na jiný), takže správná volba prostředí je nezbytná.

3. Stanovení konvence pro tvorbu globálních identifikátorů.

Určité procento objektů v programech má globální charakter, což znamená, že s nimi pracuje více než jeden program (úloha), obecně může platit, že s globálními objekty přichází do styku více programátorů. U větších systémů je výhodné zavést mnemotechnické označování globálních objektů, aby bylo na první pohled zřejmé, o jaký typ objektu se jedná (úloha, podprogram, funkce, schránka, semafor, proměnná, tabulka,...), a do které části programového systému daný objekt patří.

4. Stanovení konvence pro zápis datových struktur.

Při analytickém návrhu a realizaci programového systému více než jedním pracovníkem je vhodné stanovit formální pravidla pro zápis datových struktur. Pracovně může každý pracovník používat libovolných forem zápisu, ale ve vztahu k týmu (dokumentace, hotové programy,...) by měli všichni programátoři respektovat stanovené formální pravidla. Jednotná forma zápisu většinou závisí na deklaračních příkazech použitého programovacího jazyka, i když není vyloučeno používat nejsrozumitelnější notace jazyka PASCAL alespoň v dokumentaci i při použití jiného programovacího jazyka (např. C).

5. Stanovení konvence pro zápis algoritmů.

Srozumitelný zápis algoritmů je podpořen strukturovanými programovacími jazyky, a proto můžeme hovořit pouze o drobných formálních zásadách, které sjednotí zápis programů od více autorů. Tyto zásady spočívají a aplikování principů strukturovaného programování i na formální zápis algoritmů).

6. Stanovení konvence pro strukturu programu.

Některé programovací jazyky vyžadují přesné pořadí zadávání příkazů (nejprve deklarace, pak teprve algoritmy....). V každém případě je výhodné stanovit i v této problematice přesný řád, který obvykle vychází ze syntaxe programovacího jazyka. Orientace v programech, které jsou vícestránkové, je pak přirozeně snažší.

7. Stanovení konvencí pro speciální operace.

V různých aplikačních oblastech jsou opakovaně využívány např. prostředky operačního systému v oblastech práce s daty, práce v reálném čase,...,
a proto je nutné pro tyto speciální okruhy (jsou často strojově závislé) stanovit formální pravidla zápisu operací (OPEN, CLOSE, READ, WRITE,..., SEND MESSAGE, WAIT MESSAGE, RECEIVE MESSAGE, GET SEMAPHOR, RETURN SEMAPHOR,...) které jsou v různých systémech na různých úrovních.

2. Tvorba globálních identifikátorů objektů

Obecně lze používat identifikátory libovolně dlouhé, ale myslím si, že nejvhodnější je používat identifikátory s průměrným počtem 6 znaků a maximálním počtem znaků okolo 12. Jestliže z identifikátoru vyhradíme 2 rozpoznávací znaky, zůstane poměrně dost znaků pro stanovení významu identifikátoru.

Globální identifikátory mohou mít ve větším systému následující tvar:

XY...

kde X - označení subsystému (část programového systému), do kterého objekt patří (je v něm deklarován).
Y - typ objektu (např. T - úloha
P - podprogram
F - funkce
M - zpráva jiné úloze
S - semafor
V - proměnná
A - pole proměnných ...)
... - libovolné znaky vyjadřující výstižně obsah identifikátoru.

Při zavedení výše uvedené konvence se všichni členové týmu budou rychle orientovat i v relativně složitém programovém systému. Pro tvorbu objektů platí několik zásad:

- počet objektů (lokálních i globálních) by měl být minimální, neboť množství symbolů zhoršuje přehlednost programu
- každý objekt by měl mít svůj specifický význam (např. určitá proměnná by se měla používat jen v účelu, pro který byla zavedena), neboť nelze obecně zavést proměnnou, která by každou chvíli sloužila jinému účelu jako např. registry v assembleru
- i když si předchozí dvě zásady protiřečí, je dobré hledat vhodný kompromis mezi oběma zásadami a např. zavést přirozené identifikátory I, J, K, L, M, N pro řízení cyklů, nebo reálné proměnné X, Y, Z, popř. s indexem pro výpočty matematických vztahů, popř. deklarovat strukturu s ukazatelem a tuto strukturu využívat na různá data, jejichž struktura se s deklarovanou strukturou shoduje, ...
- název programové jednotky (program, podprogram, funkce, úloha) by se měl podle možností shodovat s názvem textového souboru, v němž je uložen zdrojový text dané programové jednotky

3. Strukturovaný zápis dat

Datové struktury můžeme zapisovat různými způsoby. V literatuře se ještě často používá grafické vyobrazení, které u jednodušších datových struktur je postačující a především je velmi přehledné. Každý prvek struktury je v orámovaném poli s uvedením významu, nebo zkratky a jednotlivá pole bývají očíslovaná. Zavedením menších a větších polí rozlišujeme typ proměnné (Integer, Byte,...). Tento způsob se však hodí jen pro velmi jednoduché datové struktury (typu Record). U složitějších datových struktur, ve kterých se vyskytují variabilní záznamy, víceúrovňové záznamy kombinované s poli,apod., se grafické zobrazení nedá použít, nebo je velmi složité.

Obecné datové struktury lze zapsat pomocí prostředků jazyka PASCAL. U jiných jazyků jsou prostředky pro zápis datových struktur omezenější (zvláště PL/M), a tak skutečný zápis dat v programovacím jazyce bývá někdy komplikovaný (zavedením ukazatelů). Pokud to prostředky jazyka dovolují, je dobré mít jediný

zápis dat uváděný v dokumentaci i v deklaračních příkazech programů. Obecně je možné, a někdy je to velmi výhodné, používat prostředků pro zápis dat převzatých z jazyka PASCAL zvláště v programové a uživatelské dokumentaci, zatímco v programech budou uváděny deklarace disponibilními prostředky jazyka.

Uveďme si několik příkladů strukturovaných zápisů datových struktur v jazyce PASCAL s důrazem na formu zápisu. Klíčová slova budou podtržena.

```
type
  Hlav_zpr = record                    (*Hlavička zprávy          *)
    Spojka,                            (*  Spojka zprávy do fronty *)
    Delka  : Integer;                  (*  Celková délka zprávy    *)
    Typ    : Byte;                     (*  Typ zprávy              *)
    Poc_schranka,                      (*  Adresa počáteční schránky *)
    Schranka_odp : Integer;            (*  Adresa schránky odpovědi *)
          end;

var
  Zprava_1,                            (*Příklady zpráv, které obsa- *)
  Zprava_2 : Hlav_zpr;                 (*hují pouze hlavičku        *)
  Zprava_3 : record                    (*Zpráva pro nast./čtení času *)
    Záhlaví : Hlav_zpr;                (*  Záhlaví zprávy          *)
    Status  : Integer;                 (*  Stav po provedení služby *)
    Hodina,                            (*  Hodina                  *)
    Minuta,                            (*  Minuta                  *)
    Sekunda : String[2];               (*  Sekunda                 *)
          end;
  Zprava_4 : record                    (*Komunikační zpráva         *)
    Záhlaví : Hlav_zpr;                (*  Záhlaví zprávy          *)
    Status,                            (*  Stav po provedení komunik.*)
    Kom_zona: Integer;                 (*  Adresa komunikační zóny *)
    Delka_KZ: Byte;                    (*  Délka komunikační zóny  *)
          end;
```

Všechny uvedené příklady zpráv jsou jednoduché záznamy, které je možné zapsat graficky. Využitím uživatelské deklarace typu však opakující se část zpráv (hlavička) může být deklarována pouze jednou, čímž se ušetří zápis. V jazyce PL/M se v takových případech používá tzv. literál, který deklaruje opakující se text, a při použití textu se uvede jen identifikátor literálu:

```
DECLARE HLAV$ZPR LITERALLY           /*Hlavička zprávy          */
  SPOJKA : ADDRESS,                  /*  Spojka zprávy do fronty */
  DELKA  : ADDRESS,                  /*  Celková délka zprávy    */
  TYP : BYTE,                        /*  Typ zprávy              */
  POC$SCHRANKA : ADDRESS,            /*  Adresa počáteční schránky */
  SCHRANKA$ODP : ADDRESS             /*  Adresa schránky odpovědi */
```

Všimněme si některých formálních aspektů zápisů:

- začátky klíčových slov record a end, které vymezují začátek a konec deklarace záznamu, jsou přesně pod sebou (ve stejném sloupci), aby byl záznam opticky vymezen.

- identifikátory jednotlivých prvků deklarací jsou pod sebou, při deklaraci záznamu začínají identifikátory prvků záznamu o 2 mezery vpravo od identifikátoru záznamu (struktura dat odpovídá i formě zápisu)
- začátky a konce komentářů jsou ve shodných sloupcích, což zpřehledňuje program (na první pohled je oddělen překládaný zdrojový text od komentářů) a dále umožňuje snadnou verifikaci (kontrola úplnosti komentáře, t.j. zda ke každému začátku komentáře "(*" existuje i konec komentáře "*)"). Při libovolném umisťování komentářů se obtížně hledají znaky začátku a konce komentářů a chybějící znaky jsou příčinou mnohých chyb.
- je velmi vhodné, když každý deklarovaný prvek (identifikátor) má svůj vlastní komentář. Z tohoto důvodu je při deklaraci každý prvek umisťován na samostatném řádku a vpravo je v rámci komentáře vysvětlen význam deklarovaného prvku. Je-li komentář delší, může pokračovat na novém řádku (na straně komentářů), zatímco deklarační část na levé straně zůstane volná. U deklarace prvků struktury typu záznam mohou být formálně odpovídající komentáře posunuty rovněž o 2 mezery vpravo, jako jsou posunuty identifikátory jednotlivých prvků struktury.

Uveďme si další příklad složitější deklarace:

```
type
  Inf_blok = record                    (*Informace datového bloku     *)
    Typ,                               (*  Typ bloku                  *)
    Perioda,                           (*  Perioda, se kterou bude    *)
                                       (*  blok zpracováván           *)
    Faze,                              (*  Fáze, ve které má být blok *)
                                       (*  zpracován                  *)
    Poc_mtpar : Byte;                  (*  Počet metaparametrů        *)
    Delka_DT,                          (*  Délka datové tabulky       *)
    Adresa_DT,                         (*  Adresa datové tabulky      *)
    Delka_AT,                          (*  Délka adresové tabulky     *)
    Adresa_AT : Integer;               (*  Adresa adresové tabulky    *)
          end;

  Inf_sit : array[1..256] of           (*Informace sítě 256 datových   *)
    Inf_blok;                          (*bloků                         *)

var
  Inf_system : record                  (*Informace o řídicím systému  *)
    Jmeno : String 40 ;                (*  Jméno řídicího systému     *)
    Site : array[1..100] of            (*  Informace systému o 100    *)
      Inf_sit;                         (*  sítích                     *)
            end;
```

Příklad na několika řádcích jednoznačně a přehledně definuje strukturu dat řídicího systému realizovaného pomocí blokového jazyka. V tomto jazyce blok vykonává určitou elementární funkci, síť je sestavená z bloků pomocí řídicích vazeb (pořadí vykonávání bloků) a datových vazeb (který datový výstup určitého bloku je datovým vstupem jiného bloku) a vykonává určitou komplexní funkci v řídicím systému, který je pak dán množinou sítí (jednotlivé sítě mají své atributy, jako je priorita, perioda zpracování,...). Na rozdíl

od předchozích příkladů by se tato struktura velmi obtížně graficky zakreslovala. U deklarace dat v jazyce PASCAL je využito víceúrovňových uživatelských deklarací typu, takže je jasné, jaké informace jsou uchovávány pro jeden blok a dále struktura, v jaké jsou vytvářeny sítě a ze sítí systém.

V jazyce PL/M by asi bylo nejvhodnější deklarovat strukturu Inf_blok jako základní proměnnou a pak se pohybovat po datech řídicího systému pomocí směrníku. Uveďme si ještě zápis stejného příkladu v jazyce C:

```
struct inf_blok                          /*Informace datového bloku    */
{
  char typ,                              /*  Typ bloku                 */
       perioda,                          /*  Perioda bloku             */
       faze,                             /*  Fáze bloku                */
       poc_mtpar;                        /*  Počet metaparametrů       */
  unsigned delka_dt,                     /*  Délka datové tabulky      */
           adresa_dt,                    /*  Adresa datové tabulky     */
           delka_at,                     /*  Délka adresové tabulky    */
           adresa_at;                    /*  Adresa adresové tabulky   */
} inf_system [100][256];                 /*Informace o řídicím systému */
```

První index v deklaraci dvourozměrného pole struktur inf_system určuje číslo sítě, druhý index určuje číslo bloku v dané síti.

Popis datových struktur, např. v dokumentaci, se může realizačně lišit od skutečných deklarací v programu, nebo způsobu práce s daty (některá data nemusí být v operační paměti v rámci programu, ale mohou se po částech přenášet z vnější paměti a zpracovávat). Z hlediska popisného a vysvětlujícího musí být zachována struktura dat (význam a pořadí prvků) a organizace složitějších datových struktur. Při realizaci může být použito jiného programovacího jazyka (např. dokumentace v jazyce PASCAL, realizace v jazyce C), který má jiné možnosti deklaračních příkazů, je využíváno základních proměnných řízených směrníky (ukazately),...

4. Strukturovaný zápis algoritmů

Pro zápis algoritmů se často používají vývojové diagramy (většinou pracovně, někdy ! v dokumentaci). Zápis programu ve vyšším programovacím jazyce je naprosto adekvátní vývojovému diagramu, ale má značné výhody:

- dokumentace může být zpracovávána na počítači
- zápis v programovacím jazyce je exaktnější (přesnější)
- zápis algoritmu v programovacím jazyce je přímo vstupem do počítače, přičemž může být současně realizační dokumentací

Příznivci vývojových diagramů však často obhajují diagramy tvrzením, že jsou mnohem srozumitelnější a přehlednější, než suchý zápis programu v programovacím jazyce. Stanovením formálních pravidel (mnohé se již ujaly) pro zápis algoritmů v programovacím jazyce může tuto nevýhodu částečně kompenzovat.

Uveďme si základní konstrukty některých jazyků s přihlédnutím k formální stránce zápisu:

1. Větvení

PASCAL

```
if podmínka 1
then
  příkaz1
else begin
  if podmínka 2
  then begin
    příkaz 2;
    příkaz 3;
    příkaz 4;
  end
  else
    příkaz 5;
end;
```

PL/M

```
IF podmínka 1
THEN
  příkaz 1;
ELSE DO;
  IF podmínka 2
  THEN DO;
    příkaz 2;
    příkaz 3;
    příkaz 4;
  END;
  ELSE
    příkaz 5;
END;
```

C

```
if (podmínka 1)
  příkaz 1;
else
{
  if (podmínka 2)
  {
    příkaz 2;
    příkaz 3;
    příkaz 4;
  }
  else
    příkaz 5;
}
```

Nejdůležitějším formálním pravidlem je členění příkazů při větvení programu podle hloubky vnoření (toto pravidlo platí i u dalších konstruktů). Při vnořování příkazů posunujeme příkazy o určitou konstantní jednotku doprava, při ukončení vnoření posuneme příkazy o stejnou konstantní jednotku doleva. Přitom musíme dbát na to, aby klíčová slova (if, then, else, end, { , }) byly přesně pod sebou (ve stejném sloupci). Toto formální pravidlo je velmi výhodné při formální kontrole programu před překladem, kdy kontrolujeme úplnost jednotlivých programových konstruktů (někdy se zapomínají klíčová slova then, end,...). Jestliže překladač vyhlásí chybu o tom, že v programu je porušena bloková struktura (chybí nebo přebývá end, } ,apod.), pak pomocí přiloženého pravítka, nebo listu papíru můžeme snadno odhalit chybu.

Umístění if a then pod sebou nebývá obvyklé, ale je praktické z toho důvodu, že u složitějších podmínek (často víceřádkových) se then psané na jednom řádku s podmínkou může v textu programu ztratit. U složitějších podmínek psaných na více řádcích klíčové slovo then vlastně vyjadřuje konec podmínky. Např.:

```
if ((podmínka 1 and podmínka 2) or
   (výraz A  výraz B)) and
   podmínka 3
then
  příkaz;
```

2. Iterační cyklus

PASCAL

```
for index 1 = výraz 1 to  výraz 2 do
begin
  for index 2 = výraz 3 to výraz 4 do
  begin
    příkaz 1;
    příkaz 2;
    příkaz 3;
  end;
  příkaz 4;
  příkaz 5;
end;
```

PL/M

```
DO index 1 = výraz 1 TO výraz 2;
  DO index 2 = výraz 3 TO výraz 4;
    příkaz 1;
    příkaz 2;
    příkaz 3;
  END;
  příkaz 4;
  příkaz 5;
END;
```

C

```
for (výraz 1; výraz 2; výraz 3)
{
  for (výraz 4; výraz 5; výraz 6)
  {
    příkaz 1;
    příkaz 2;
    příkaz 3;
  }
  příkaz 4;
  příkaz 5;
}
```

3. Obecný cyklus

PASCAL

```
while podmínka 1 do
begin
  while podmínka 2 do
    příkaz 1;
  příkaz 2;
  příkaz 3;
end;
```

PL/M

```
DO WHILE podmínka 1;
  DO WHILE podmínka 2;
    příkaz 1;
  příkaz 2;
  příkaz 3;
END;
```

C

```
while (podmínka 1)
{
  while (podmínka 2)
    příkaz 1;
  příkaz 2;
  příkaz 3;
}
```

3. Přepínač

PASCAL

```
case selektor of
  hodnota 1 : příkaz 1;
  hodnota 2 : příkaz 2;
  .
  .
  .
  hodnota n : příkaz n;
end;
```

PL/M

```
DO CASE selektor;
  příkaz 1;
  příkaz 2;
  .
  .
  .
  příkaz n;
END;
```

C

```
switch (selektor)
{
  case hodnota 1 : příkaz 1;
                   break;
  case hodnota 2 : příkaz 2;
                   break;
  .
  .
  .
  case hodnota n : příkaz n;
                   break;
  default: příkaz n+1;
           break;
}
```

Uvedené příklady formálního zápisu algoritmů nemohou sloužit jako absolutně dokonalý vzor pro každého programátora. Důležité je, aby se v řešitelských týmech řešitelé s touto problematikou zabývali, stanovili si své zásady a pak je důsledně dodržovali. Maximální zvýšení srozumitelnosti programu je u týmové práce nezbytné, neboť nelze spoléhat na dokonalý kolektiv a bezproblémové řešení. Pracovníci odcházejí z řešitelských kolektivů, dochází k dlouhodobým absencím, a také je nutno počítat s tím, že řešení určitého složitého problému bude probíhat v iteracích, kdy např. noví pracovníci (absolventi) mohou po předložení předchozí iterace, kterou dělal někdo jiný, vypracovat podle určitých pokynů novou variantu řešení.

5. Struktura programu

Poslední důležitou konvencí, o které bych se rád zmínil, je struktura programu. Programovací jazyky většinou vyžadují, aby jednotlivé části programu (deklarace proměnných, podprogramů,...,vlastní tělo programu,...) byly uvedeny v určitém pořadí. Je dobré si stanovit podle určitého programovacího jazyka osnovu, která usnadní orientaci v programu, jestliže se všichni budou této osnovy držet.

Uveďme přirozenou a nejčastější strukturu programu, která vychází z

programovacího jazyka PASCAL, a která může být modifikována podle použitého programovacího jazyka, popř. podle speciálních potřeb řešitelů:

1. Hlavička programu. Kromě povinných částí programu, které vyžaduje jazyk, je vhodné uvést v rámci komentáře následující informace:
   - -název programu
   - -typ (podprogram, funkce, program, úloha,...)
   - -datum poslední verze
   - - zkratka autora

2. Popis funkce programu. Na začátku by měl být uveden v rámci komentáře slovní popis funkce programu. Tato důležitá část programu uvádí cizího programátora do řešené problematiky, popř. slouží i autorovi po letech.

3. Include soubory. Některé jazyky dovolují zařadit do programu i jiné soubory, jako jsou především globální konstanty, popř. tyto soubory vytváření vazbu na cílové operační prostředí.

4. Deklarace externích objektů.

5. Deklarace návěští, konstant, typů a proměnných.

6. Deklarace podprogramů a funkcí. Při deklaraci podprogramů a funkcí se musí uvádět opět deklarace platné pro podprogram (funkci) a dále tělo podprogramu (funkce), tedy vše, co je uvedeno v bodech 2, 5, 6 a 7.

7. Tělo programu. Obsahuje vlastní algoritmus.

6. Závěr

Dnes, kdy výkonnost počítačů stále stoupá, klesá nutnost používat skokové příkazy GOTO a jiné nešvary nestrukturovaného programování. Mou zkušeností mohu potvrdit, že běžné úlohy v reálném čase původně naprogramované v assembleru lze naprogramovat ve vyšším programovacím jazyce v přísně strukturovaném řádu při zachování celkové délky kódu programu, a někdy i dat. Dokonce došlo v některých systémech ke zkrácení kódu i dat! Při zapsání programu ve vyšším jazyce bylo lépe "vidět" do programu, a tak mohl být takový program ještě navíc funkčně zdokonalován. Zásahy do nestrukturovaných programů bývají velmi nebezpečné.

Ještě jednou zdůrazňuji, že tento příspěvek se snaží prezentovat zkušenosti určitého realizačního týmu v určité oblasti programování. I když některé zásady, především ty, které byly prezentovány, mají obecnější platnost, vlastní konkrétní zásady zápisu programů si musí stanovit každý tým sám. Jsem toho názoru, aby se uvedená problematika normalizovala tak, aby zápisy algoritmů v programech byly každému srozumitelné, jako např. je tomu dnes v matematice při zápisech rovnic.

NA ÚROVNI

Ing. Richard Rébr

Příspěvek hledá odpověď na otázku, jaká je úroveň našich programů a programátorů ve srovnání se světovými trendy. Přitom rozebírá skutečnosti, které úroveň ovlivňují.

"Optimista nemůže být nikdy příjemně překvapen".

1. Úvod

1.1 Podstata problému:

Nejprve se dohodněme, že pod pojmem "programátor" budeme rozumět profesi "programátor - analytik" a slovem "naše" označíme vše československé. Jako téma příspěvku si položíme otázku:

"Jsou naše programy a naši programátoři na světové úrovni?"

Zavrhneme diletantské extremistické přístupy typu "všechno je u nás světové" nebo "nic není dobře" a pokusíme se o seriózní hodnocení.

1.2 Náhodný průzkum:

Autor položil výše formulovaný dotaz při náhodných setkáních několika expertům se žádostí o okamžitou odpověď bez přípravy. Ukázalo se, že dotazovaní byli otázkou většinou zaskočeni a uvedeni ve zmatek. Odpovědi - často vynucené drastickými prostředky - se shodovaly v těchto bodech:

- naše průměrné programy nedosahují úrovně světového průměru
- o programátorech se nelze vyjádřit, ale "špičkové" pracovníky asi máme
- bylo by zajímavé a patrně i užitečné podrobit problém úrovně hlubšímu rozboru.

Z rozhovorů s experty též vyplynuly požadavky

- stanovit kriteria, podle kterých úroveň hodnotíme
- hodnotit zvlášť průměr a "špičky".