Ing. Josef B r z i c k ý
Ústav výpočetní techniky SPK, Praha

REALIZACE SCHÉMAT STRUKTUROVANÉHO PROGRAMOVÁNÍ V PSEUDOKÓDU A V JAZYCE PL/I

Ke zlepšování techniky programování, ke zvyšování efektivnosti programátorské práce existuje řada možných přístupů. Jsou doporučovány normované programování, generátory programů, modulární programování a jeho varianta strukturované programování a další. Je zřejmé, že záleží na specifice řešených úloh, který z přístupů případně která jejich kombinace budou na daném pracovišti zvoleny.

V tomto příspěvku se zabývám otázkami strukturovaného programování. Jde o přístup, který je obecný, nepředpokládá určitou vyhraněnou specializaci řešených úloh. Je součástí komplexní koncepce, která je propagována např. firmou IBM pod pojmem IPT - Improved Programming Technologies /Zdokonalené programovací techniky/. Východiskem této koncepce je strukturovaný návrh s případným použitím systému HIPO - Hierarchy plus Input - Process - Output /Hierarchie plus vstup - zpracování - výstup/. Ze strukturovaného návrhu vychází vlastní strukturované programování v pseudokódu a ve vyšších programovacích jazycích. Tato linie je doplněna týmovou organizací programátorské práce - Chief programmer team /tým vedoucího programátora/, pracovními diskusemi o vyvíjených programových systémech /Structured walk-throughs/ a využitím knihoven zachycujících stav vývoje programového systému DSL - Development Support Libraries.

Z této širší problematiky zdokonalených programovacích technik je tento příspěvek věnován pouze výseku: Stručnému přehledu obecných schemat strukturovaného programování /dále SP/, vysvětlení pojmu pseudokódu a realizaci schémat SP v pseudokódu a v jazyce PL/I.

Podle teorie SP lze celý program sestavit z několika základních schémat /struktur/. Ta jsou již poměrně známa, proto uvádím pouze stručný přehled a grafické znázornění:

1: Sekvence dvou funkcí.

Je to vlastně formalizace téměř samozřejmé myšlenky, že se příkazy provádí - pokud není uvedeno jinak - v pořadí, v jakém se objevují v programu.

2: Výběr ze dvou nebo více možností na základě testování podmínky /podmínek/:

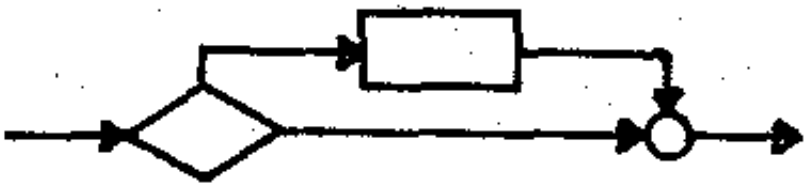
a) IF THEN
   Funkce je provedena jen při splnění určité podmínky.
b) IF THEN ELSE
   Při splnění podmínky je provedena jedna funkce, v opačném případě druhá.
c) CASE
   Užívá se pro vícenásobné větvení, které by jinak vyžadovalo příliš mnoho IF.

3: Opakování funkce /nebo několika funkcí/ při splnění podmínky /podmínek/.

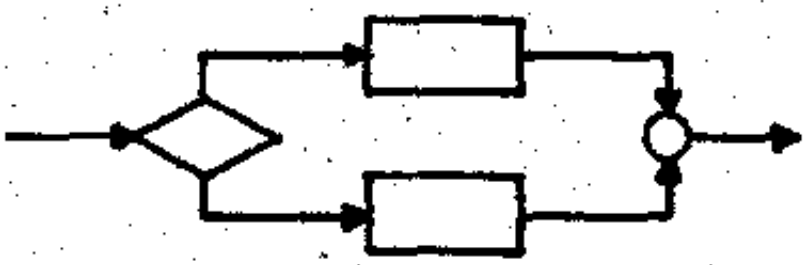
a) DO WHILE
   Podmínka je testována na začátku smyčky.
b) DO UNTIL
   Podmínka je testována na konci smyčky /zaručen jeden průchod/.

4: Vyvolání funkce /segmentu/ na nižší úrovni - CALL

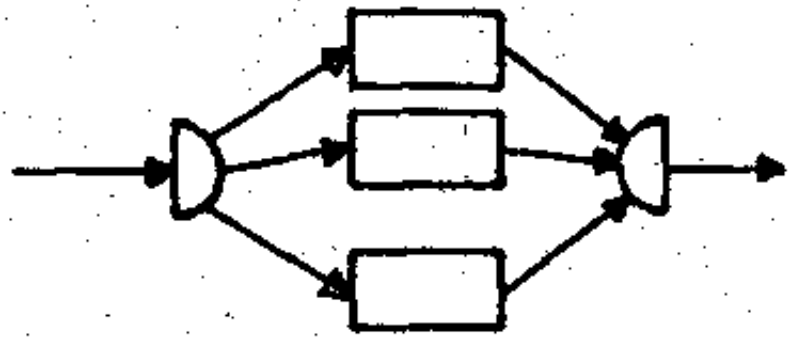
SEKVENCE



IF THEN

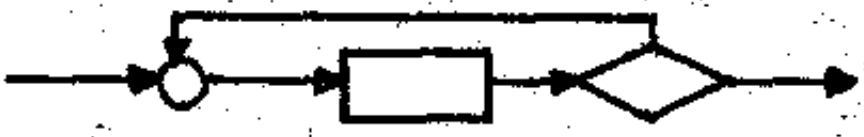


IF THEN ELSE

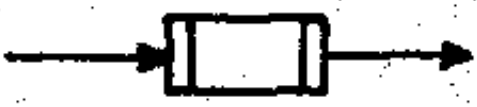


CASE

DO WHILE



DO UNTIL



CALL

Základní myšlenka spočívá v tom, že každý funkční blok graficky ve schematech znázorněný jako obdélník je možno nahradit libovolným základním schematem. Každá funkce i celé základní schema má pouze jeden vstupní bod na začátku a jeden výstupní bod na konci.

Za předpokladu, že existuje strukturovaný návrh programového celku obsahující především hierarchii segmentů a jejich vztahy, máme před vlastním zápisem programu - kódováním možnost volby. Buď provést pro každý segment přípravnou operaci v podobě vytvoření postupového diagramu nebo v podobě zápisu programu v pseudokódu, případně žádnou přípravnou operaci neprovádět. Způsob tvorby postupových diagramů využívající grafické symboly stejné nebo obdobné těm, které byly užity o několik řádek výše, je notoricky znám. Současný trend však směřuje k druhé, případně třetí možnosti přípravy na vlastní kódování. Proto několik dalších odstavců je věnováno pseudokódu.

Pseudokód je slovně vyjádřená obdoba dosud u nás často používaných postupových diagramů. Je proti nim bližší vlastnímu zakódovanému programovému textu. Kromě několika vžitých anglických klíčových slov se výhradně užívá českých slov. Klíčová slova se obvykle píší velkými písmeny, zatímco český text pro odlišení malými. Každá funkce /obdélníkový blok v postupovém diagramu/ je vyjádřena buď českým názvem funkce nebo jednoduchou rozkazovací větou nebo - pokud to zjednoduší zápis a neuškodí jeho čitelnosti - příkazem ve vyšším programovacím jazyce. Na př.:

```
Tisk hlavičky
Čti záznam souboru surovin
CENA = JEDN_CENA * MNOZSTVI
```

Jak se v pseudokódu promítají výše popsaná schemata SP:

1. Sekvence dvou funkcí je realizována jednoduše tím, že funkce jsou zapisovány na dvě po sobě jdoucí řádky, přičemž začátky obou funkcí jsou pod sebou:

2: Pro výběr jsou užívána vybraná klíčová anglická slova IF, THEN, ELSE, ENDIF, CASE, OF, ENDCASE

a)
```
IF p
THEN f1
ENDIF
```

b)
```
IF p
THEN f1
ELSE f2
ENDIF
```

c)
```
CASE v OF
   h1: f1
   h2: h3: f2
   h4: f3
     .
     .
     .
   hm: fm
ENDCASE
```

Kde znamená	p	podmínku,
	f1,f2,...,fm	funkce,
	v	proměnnou nabývající řady hodnot,
	h1,h2,...,hm	hodnoty proměnné.

Příklady:

```
IF    změna podniku nebo závodu
THEN  tisk závěru tabulky
      tisk hlavičky
      naplnění porovnávacích proměnných
ENDIF

IF    výroba je větší než 1 000
THEN  tisk varovné zprávy
ELSE  výpočet ceny
ENDIF
```

```
CASE   číslo opravy OP
   1:  oprav číslo materiálu
   2:  oprav cenu
     .
     .
     .
   8: 9: oprav spotřebované množství
ENDCASE
```

3: Pro opakování jsou užívána klíčová anglická slova DOWHILE, DOUNTIL, ENDDO

```
a) DOWHILE  p
      f1
   ENDDO

b) DOUNTIL  p
     f2
   ENDDO
```

Příklady:

```
Čti 1: záznam souboru surovin
DOWHILE  ještě data
    spracuj předchozí záznam
    čti záznam souboru surovin
ENDDO

DOUNTIL  koeficient < 0.8
     propočet vzorce efektivnosti
ENDDO
```

4: Pro volání subfunkce je možno použít buď pouze souhrnného názvu nebo formulace

```
CALL f1
```

Ve všech případech se doporučuje dodržovat odstavcování podle uvedených schemat. Pochopitelně popsaný způsob výstavby pseudokódu není dogma, může být modifikován podle potřeb daného pracoviště nebo i jednotlivých řešitelů. Na příklad při použití jazyka PL/I je lépe omezit užití schemat 2c) a 3b), pro která zatím jazyk nemá jednoduché vyjádření.

Převod připraveného programu v pseudokódu do vyššího programovacího jazyka by neměl být problémem. Většina programovacích obtíží je zpravidla vyřešena už při zápisu pseudokódu. V dalším textu je ukázána realizace základních schemat SP v jazyce PL/I:

1. Sekvence příkazů je realizována prostým zápisem příkazů jazyka za sebou /s výjimkou užití skokových příkazů/.

2. Jednoduše jsou řešitelná první čtyři schemata výběru

a) IF podmínka
THEN příkaz;

b) IF podmínka
THEN příkaz-1;
ELSE příkaz-2;

Jednotlivé příkazy mohou být nahrazeny skupinou příkazů:

```
IF    podmínka
THEN DO;
          příkaz-1;
          příkaz-2;
             .
             .
             .
     END;
ELSE DO;
          příkaz-n;
             .
             .
             .
     END;
```

c) Vícenásobné větvení odpovídající schematu CASE nemá zatím v jazyce PL/I vlastní specializovaný příkaz */, lze jej opsat pomocí vkládaných příkazů IF, např:

---

*/ V připravovaných kompilátorech firmy IBM je pro tyto účely vyhrazen nový příkaz SELECT.

```
IF KOD = 'A'
THEN CALL PRIPAD_A;
ELSE IF KOD = 'B'
     THEN CALL PRIPAD_B;
     ELSE IF KOD = '8'
          THEN CALL PRIPAD_8;
          ELSE IF KOD = '9'
               THEN   atd.
```

Pro přehlednost programu je tento způsob řešení přijatelný jen do rozumného počtu vložených příkazů IF. Jiný způsob simulace CASE ukazuje následující příklad:

```
DCL PRIPAD (4) LABEL INIT (PRIPAD_1,PRIPAD_2,PRIPAD_2,
  PRIPAD_4);
INDEX = <celočíselný výraz> ;
GOTO PRIPAD (INDEX);
 PRIPAD_1:
   DO;
       příkaz-1
         .
         .
         .
       GOTO KONEC_P;
   END;
 PRIPAD_2:
   DO;
       příkaz-n
         .
         .
         .
       GOTO KONEC_P;
   END;
 PRIPAD_3;
    .
    .
    .
 PRIPAD_4
    .
    .
 END;
KONEC_P:
```

V příkladu se předpokládá, že již dříve bylo ověřeno, že hodnota indexu nemůže ležet mimo meze 1 až 4. V této simulaci je také výjimečně použit příkaz GOTO pro zajištění návratu všech větví do jediného výstupního bodu.

2.a) Z opakovacích schemat je možno snadno realizovat schema DO WHILE:

```
DO WHILE /podmínka/;
   příkaz
     :
END;
```

V jazyce PL/I má toto schema další varianty spojené s indexováním:

```
DO proměnná = výraz-1 TO výraz-2 BY výraz-3
      [, výraz-4 TO výraz-5 BY výraz-6]  ...
      [ WHILE /výraz-n/]
```

Pokud není volba WHILE užita je podmínkou opakování pouze nepřekročení horní meze /výraz po TO/.

b) Schema DO UNTIL nemá zatím v PL/I přímý nástroj */, a proto se mu buď vyhýbáme nebo použijeme simulace zřejmé z příkladu:

```
POKRACOVAT = '1'B;
DO WHILE (POKRACOVAT);
   příkaz-1
     :
   IF podmínka
   THEN POKRACOVAT = '0'B;
END;
```

---

*/ V připravovaných kompilátorech firmy IBM je pro tyto účely vyhrazena nová varianta příkazu DO s volbou UNTIL.

4: Pro definici a vyvolání určité subfunkce neboli segmentu máme v PL/I několik možností: Segmentem může být prostý sled příkazů, blok typu BEGIN, interní nebo externí procedura. Některé organizační přístupy ke strukturovanému programování vyžadují, aby každý segment byl uložen samostatně v knihovně. Pro prostý sled příkazů, BEGIN - bloky a interní procedury to znamená uložení do knihovny zdrojových příkazů a zařazení z ní pomocí preprocessorového příkazu %INCLUDE, pro externí procedury zařazení do knihovny load modulů. V praxi jsou někdy jednotlivé segmenty zapisovány po skupinách v jednom programovém celku tvořeném externí procedurou bez prostřednictví knihovny zdrojových příkazů a navzájem výrazně oddělovány /např. užitím preprocessorových příkazů %PAGE, %SKIP/:

Vstup do segmentu se realizuje pomocí příkazu CALL nebo funkčním voláním /u procedur/, případně sekvenčním přechodem z předchozího segmentu /u prostého sledu příkazu a BEGIN-bloků/. Automaticky operačním systémem je vyvolán hlavní segment - PROCEDURE OPTIONS (MAIN).

Nevýhodou segmentů typu prostého sledu příkazů je, že jejich lokální proměnné jsou přístupné zvenčí a mohou být nežádoucně ovlivněny jinými segmenty. Výhodou je úspora paměti a času vzhledem k tomu, že neobsahuje prolog a epilog jako další typy segmentů. V segmentech procedurálního typu by mělo být snahou předávat data pokud možno prostřednictvím seznamů parametrů a argumentů, ne prostřednictvím globálních proměnných, tak aby segmenty byly co nejméně závislé.

Kromě zásady dodržování pouze základních schemat SP popsaných výše je možno doporučit v intencích SP další pravidla s tím, že konkrétní instalační norma pro unifikaci zápisu programového textu se může od těchto pravidel odlišovat:

a) Omezení délky segmentu na 1 - 2 strany výpisu programového textu při překladu.

b) Použití uživatelských jmen, která jasně vyjadřují náplň odpovídajících programových prvků. Přitom je třeba

všechna jména explicitně deklarovat: Všechny soubory musí být explicitně otevřeny a zavřeny.

c) Užití poznámek v rozumné míře tak, aby nerušily čitelnost programu: Poznámek je možno mimo jiné využít k tomu, aby na začátku každého samostatně překládaného programového textu skládajícího se z několika segmentů byl uveden stručný přehled názvů všech segmentů a jejich funkce; Právě tak na začátku každého segmentu lze v poznámce vyjádřit jeho funkci.

d) Vhodné odstavcování zápisu programu přispívající k čitelnosti textu. Náměty na řešení tohoto bodu jsou specifikovány podrobněji:

- Začátek každého segmentu je na nové stránce /užití %PAGE/; Každý segment je označen jménem /návěštím/ zapisovaným od 2. sloupce. U bloků je na stejném řádku příkaz BEGIN nebo PROCEDURE s příslušným seznamem parametrů a volbami: Následující příkazy segmentu se uvádí odsazeny od 4. sloupce. Všechny deklarace, pokud nejsou vyčleněny ve zvláštním segmentu, se uvádí na začátku segmentu;

- Jakýkoliv příkaz, který podle dále uvedených pravidel nemá být odsazen, začíná na nové řádce ve stejném sloupci jako předchozí příkaz. Nevejde-li se na řádku, je jeho pokračování odsazeno o 6 sloupců doprava;

- Návěští - pokud jsou vůbec užívána - začínají vždy ve sloupci 2 a jsou na samostatné řádce.

- V popisech je každý deklarovaný programový prvek uváděn na nový řádek. Ve strukturách je každá vyšší úroveň /podle stoupajícího čísla úrovně/ odsazena o 2 sloupce. Přívlastky je výhodné zarovnat pod sebe; Společné uvádění přívlastků je pro více proměnných třeba provádět omezeně;

Příklad:

```
DCL 1 VYROBEK,
      2 CISLO_VYR CHAR(12),
      2 NAZEV     CHAR(30),
      2 DILY,
        3 CISLO_DILU CHAR(12),
        3 POCET_DILU FIXED(3);
```

- U všech typů DO - skupin se příkaz následující po příkazu DO začíná na nový řádek s odsazením o 3 sloupce. Konec DO - skupiny, t.j. příkaz END je zarovnán s příkazem DO.

Příklad:

```
DO I=2 TO 8;
  SUMA(I) = SUMA(I) + POLOZKA;
END;
```

- U příkazu IF je na prvém řádku samostatně IF a podmínka, na druhém na stejné úrovni jako IF klíčové slovo THEN a za ním příkaz. Jde-li o příkaz DO, jsou další řádky odstavcovány podle pravidel uvedených u DO - skupin. Větev ELSE začíná na nové řádce na stejné úrovni jako IF a THEN.

Příklad:

```
IF MZDA > 6000
THEN DO;
        ODVOD=1000 * KOEF1;
        CALL UPRAVA(MZDA,ODVOD);
     END;
ELSE CALL UPRAVA2 (MZDA);
```

- V příkazech pro otevření a zavření souborů je každý soubor uváděn na zvláštní řádce.

Příklad:

```
OPEN FILE (OBORY);
     FILE(STAVBY);
```

- V příkazech GET a PUT se uvádí seznam dat na nový (druhý) řádek odsazen o 4 sloupce a případný seznam formátových prvků na nový řádek s tím, že se snažíme umístit formátové prvky pod odpovídající datové prvky.

Příklad:

```
GET EDIT
    (ROK,CASTKA,BODY)
    (A(4),F(5), F(3));
```

- Běžné poznámky by měly začínat ve sloupci 36 a končit ve sloupci 71, pokud se nevejdou na jeden řádek, pokračovat ve sloupci 39 následujícího řádku. Tato pravidla se nevztahují na poznámky na začátku programového textu (přehled segmentů) a na začátku segmentů (jméno a funkce segmentu),

které se zapisují od 2. sloupce a jsou odděleny od vlastního programového textu několika prázdnými řádkami (%SKIP).

Po námětech pro sestavení instalační normy SP závěrem příspěvku několik slov o postupu výstavby strukturovaných programů. Strukturovaný návrh umožňuje přistupovat k výstavbě programových celků odshora dolů. Znamená to, že jako prvý se programuje hlavní segment, který obsahuje už všechna volání modulů na nižší úrovni s úplnými seznamy argumentů. Tyto nižší moduly jsou připravovány teprve postupně v průběhu vývoje programového celku a jsou z počátku nahrazeny pseudoprocedurami. Tyto prázdné pseudoprocedury obsahují jen příkaz PROCEDURE, seznam parametrů, příkaz END a případně několik příkazů pro přípravu fiktivních výstupů z modulu. Ve většině případů postačí tisková zpráva o tom, že došlo k vyvolání daného modulu. Po odladění hlavního segmentu se stejným způsobem stupňovitě odlaďují segmenty na nižší úrovni. Tento postup má tu výhodu, že pro ladění nejsou zapotřebí žádné speciální testovací programy a odpadá závěrečný integrační test.

Použitá literatura:

Firemní manuály IBM,

P. Van Leer: Top-down development using a program design language, IBM Systems Journal Vol. 15, Number 2, 1976