Ing.Alexander KIRSCHNER

PVT Bratislava

IMPLEMENTÁCIA DOPLNKOVÝCH RIADIACICH ŠTRUKTÚR V JAZYKU PL/I POMOCOU JEHO PREDPROCESORA

V súčasnosti možno považovať štrukturované programovanie už za všeobecne akceptovanú metodológiu respektíve technológiu programovania. Obídeme problém, čo všetko tvorí štrukturované programovanie a zameriame sa len na problematiku štrukturovaného kódovania programov. Jej jadro spočíva v používaní základných riadiacich štruktúr /sekvencia, selekcia a iterácia, niekedy sa ku nim zaraďuje aj viacvetvová selekcia -[3]/, čo vedie ku tvorbe okrem iného korektných, čitateľných a ľahšie modifikovateľných programov. V novších programovacích jazykoch /PL/I, Pascal, Ada/ existujú pre tieto základné riadiace štruktúry príslušné "štrukturované" príkazy, do starších programovacích jazykov /FORTRAN, COBOL/ sa tieto črty v súčasnosti zväčša dorábajú /napríklad chystaný "štrukturovaný " COBOL 81 [2] ap./.

Tento príspevok sa zaoberá možnosťou implementácie doplnkových riadiacich štruktúr typu REPEAT-UNTIL, LOOP-EXITIF a SELECT-CASE v jazyku PL/I pomocou jeho predprocesora. Budeme hovoriť o jazyku PL/I tak, ako je implementovaný v F-kompilátore operačného systému OS/EC a predpokladáme jeho základnú znalosť.

1. Základné riadiace štruktúry a jazyk PL/I

Spomenuté tri základné riadiace štruktúry sa niekedy zúžene chápu ako jediné riadiace štruktúry štrukturovaného kódovania. Pri projektovaní a programovaní sa vyskytujú rôzne úlohy a v niektorých prípadoch môže byť veľmi účelné a účinné použitie aj iných korektných riadiacich štruktúr, ktoré zjednodušia logiku návrhu programu alebo zvýšia čitateľnosť zdrojového textu. Ak neuvažujeme o rozšírení základného jazyka /ako je to v optimalizačnom alebo v tzv. checkout kompilátore PL/I/, zostávajú v zásade dve možnosti:

- tvoriť doplnkové riadiace štruktúry kombinovaním neskokových príkazov jazyka a príkazov skoku,
- implementovať predprocesorové procedúry, ktoré umožnia používať "ďalšie príkazy jazyka" - predprocesorové príkazy.

Vzhľadom na to, že predprocesorové procedúry jazyka PL/I sa vyvolávajú použitím mena príslušnej predprocesorovej procedúry /s prípadnými argumentami/, budeme pre účely tohoto článku v ďalšom miesto pojmu predprocesorová procedúra používať aj pojem predprocesorový príkaz /pôjde o príkazy UNTIL, REPEAT, LOOP, EXITIF, ENDLOOP, SELECT, CASE, DEFAULT alebo ENDSELECT/ na rozdiel od príkazov predprocesora, ktorých je v jazyku PL/I celkove deväť a začínajú znakom %. /V článku bude z nich zmienka len o príkaze predprocesora %INCLUDE./

2. Predprocesor PL/I

Jazyk PL/I je jediný širšie používaný jazyk vyššej úrovne, ktorý poskytuje možnosť predspracovania zdrojového textu programu tzv. predprocesorom. Predprocesor PL/I sa uvedie do činnosti udaním režimu práce MACRO pri vyvolaní kompilátora.

Uvedieme tu len základné informácie o tomto predprocesore potrebné pre tento článok, a to o predprocesorových procedúrach a o príkaze predprocesora %INCLUDE.

Predprocesorové procedúry sa od iných procedúr PL/I od-

lišujú tým, že sa na ne možno obracať len v predprocesorovom štádiu kompilácie a tým, že sú to vždy len funkcie. Taká funkcia môže mať udané vstupné parametre a môže vracať buď reťazec znakov alebo číslo v pevnej rádovej čiarke na miesto, kde bol jej názov použitý.

Príkaz predprocesora %INCLUDE má jedinú funkciu, a to zaradiť na miesto, kde bol použitý, zdrojový text /člen/ z knižnice, ktorá je popísaná cez v príkaze uvedené DD-meno, respektíve ak v príkaze DD-meno nie je použité, cez implicitné DD-meno SYSLIB. Pritom príkaz predprocesora %INCLUDE môže byť hniezdovaný, t. j. môže sa vyskytovať aj vo vkladanom tex'e.

3. Implementácia doplnkových riadiacich štruktúr

Doplnkové riadiace štruktúry pre jazyk PL/I možno implementovať s využitím predprocesora najrôznejším spôsobom. Riešenie, ktoré uviedli C. L. McGowan a J. R. Kelly v[1] sa vyznačuje všeobecnosťou a univerzálnym použitím. V ďalších odstavcoch ho popisujeme v zmodifikovanej podobe, ktorá zjednodušuje a sprehľadňuje jeho použitie.

V odstavcoch 3.1 až 3.3 popisujeme postupne štruktúry REPEAT-UNTIL, LOOP-EXITIF a SELECT-CASE. Pri popise každej z nich uvádzame použitiu príkazov zodpovedajúci vývojový diagram, jej stručný popis a implementačný princíp /porovnaním predprocesorového zdrojového kódu a postprocesorového - vygenerovaného - zdrojového kódu/.

Formálne pravidlá úpravy kódovania sú internou štandardizačnou konvenciou, im je formálne prispôsobený aj vygenerovaný zdrojový kód a sú zrejmé zo zápisu kódov v odstavcoch 3.1 až 3.3.

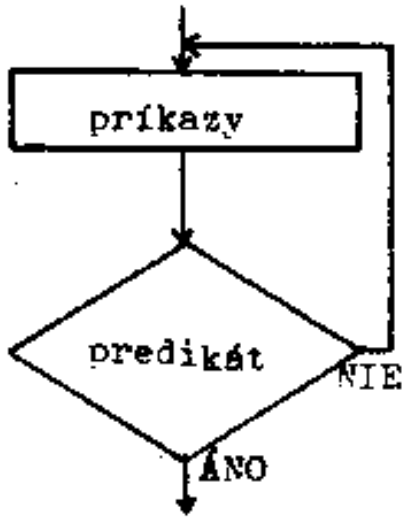
3.1 Štruktúra REPEAT-UNTIL

Dvojica predprocesorových príkazov REPEAT a UNTIL reprezentuje vývojový diagram, ktorý je uvedený na obr. 1. Pri ich použití treba uviesť nasledový predprocesorový kód,

z ktorého sa vygeneruje zdrojový text uvedený vpravo:

```
                              /*** REPEAT - BEGIN ***/
                              R#unik:
REPEAT;                       DO;
   príkazy            ---->     príkazy
  UNTIL(predikát);              IF ¬(predikát) THEN GO TO R#unik;
                                END R#unik
                              /*** REPEAT - END   ***/;
```

V zápise vyššie sú "príkazy" ľubovoľnou sadou príkazov PL/I, prípadne ide o jediný príkaz, ale tiež tam môžu byť ďalšie predprocesorové príkazy uvádzané v tomto článku. "Predikát" je ľubovoľný booleovský výraz PL/I. Tu a aj v odstavcoch 3.2 a 3.3 návestia so syntaxou X#unik sú unikátne v rámci programu, ktorý využíva v článku uvádzané predprocesorové príkazy a líšia sa buď vo štvormiestnom celom čísle "unik" alebo v "predpone" X, ktorou môže byť písmeno R, D, E, L, alebo S. Príklady takýchto návestí sú R#1001, S#1001, S#1002 ap. Generuje ich jedna predprocesorová procedúra.



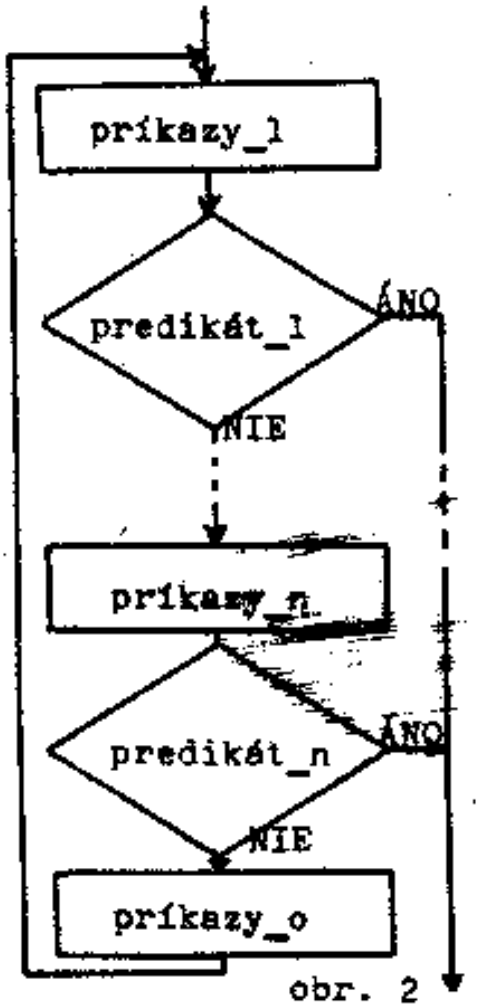
obr. 1

3.2 Štruktúra LOOP-EXITIF

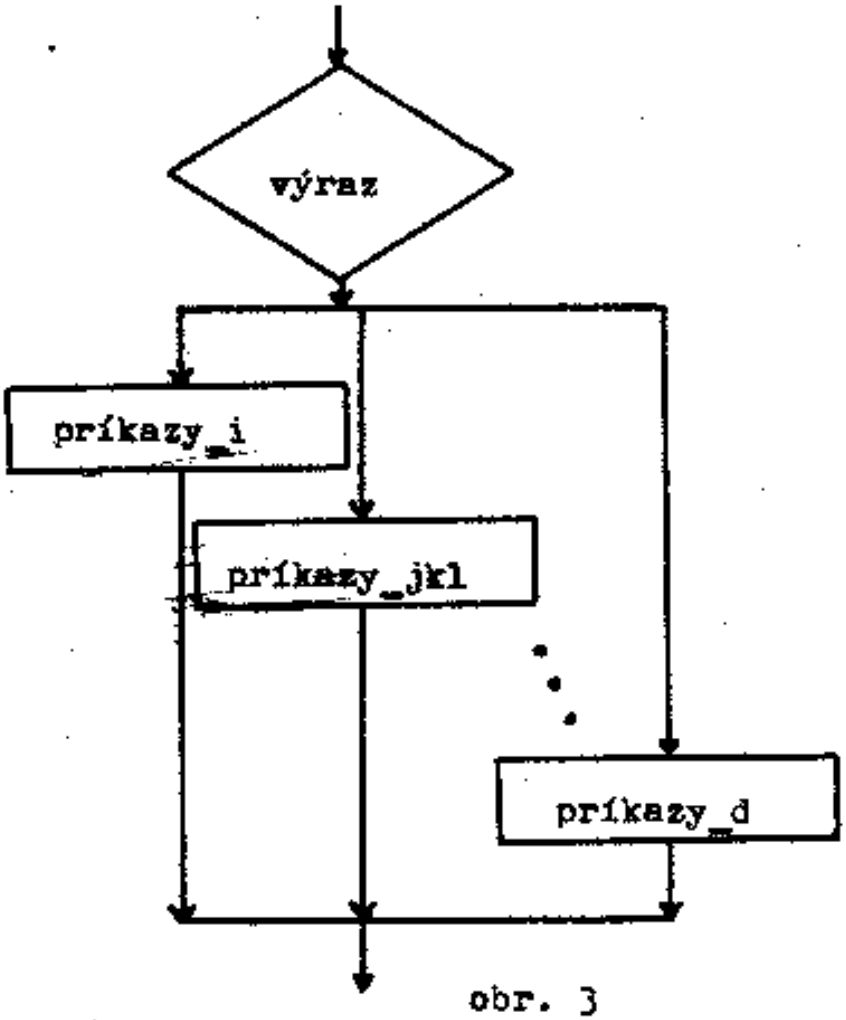
Adekvátne použitá trojica predprocesorových príkazov LOOP, EXITIF a ENDLOOP zodpovedá vývojovému diagramu, ktorý je uvedený na obr. 2. Je to už zložitejšia riadiaca štruktúra, pre ktorú existuje viac možných účinných použití. Najčastejším je to, kedy "príkazy_1" tvorí príkaz čítania sekvenčného súboru, "predikát-1" testuje koniec súboru a zostávajúce sady príkazov spracúvajú prečítanú vetu. Predprocesorový zdrojový kód, ktorý je potrebné uviesť a vygenerované príkazy jazyka PL/I, ktoré sa

budú ďalej kompilovať, uvádzame nižšie. "Príkazy_x" a "predikát_x" tu majú rovnaký význam, ako v odstavci 3.1.

```
LOOP;                          /*** LOOP - BEGIN ***/
   príkazy_1                   L#unik:
  EXITIF (predikát_1);         DO;
     :                           príkazy_1
   príkazy_n                     IF predikát_1 THEN GO TO E#unik;
  EXITIF (predikát-n);    ⟶       :
   príkazy_o                     príkazy_n
  ENDLOOP;                       IF predikát_n then GO TO E#unik;
                                 príkazy_o
                                 GO TO L#unik;
                                 E#unik:
                                 END L#unik
                                 /*** LOOP - END   ***/;
```



obr. 2



obr. 3

3.3 Štruktúra SELECT-CASE

Štruktúru viacnásobného vetvenia zobrazenú na obr. 3 je možné na úrovni predprocesorového zdrojového textu realizovať vhodným použitím predprocesorových príkazov SELECT, CASE, ENDSELECT a prípadne DEFAULT. Spôsob zápisu týchto príkazov a vygenerovaný zdrojový text v jazyku PL/I je uvedený ďalej.

```
                    /*** SELECT - BEGIN ***/
                    S#unik:
                    DO;
                      I#unik = výraz;
                      IF I#unik<=0 | I#unik>M#unik THEN GO TO
                                                      D#unik;
SELECT(výraz);        ELSE GO TO C#unik(I#unik);
  CASE(i):            C#unik(i):
    príkazy_i           príkazy_i
  CASE ((j,k,l)):       GO TO E#unik;
    príkazy_jkl       C#unik(j): C#unik(k): C#unik(l):
    :          →        príkazy_jkl
  DEFAULT:              GO TO E#unik;
    príkazy_d           :
  ENDSELECT;          D#unik: C#unik(neobsadené): ...:
                        príkazy_d
                      E#unik: END S#unik;
                      DCL (I#unik, M#unik STATIC INIT(max))
                           FIXED BIN, C#unik(max) LABEL
                    /*** SELECT -   END ***/;
```

V tomto zápise "príkazy_..." majú rovnaký význam, ako v odstavci 3.1. "Výraz" je celočíselný. I#unik a M#unik sú pomocné číselné premenné a C#unik je pole návestí s počtom prvkov "max", čo je najvyššia celočíselná hodnota, ktorá sa vyskytla v špecifikácii príkazov CASE a zisťuje sa automaticky. Ako je vidieť zo vzoru zápisu, v príkaze CASE sa môže vyskytovať jedna alebo viac celočíselných hodnôt, na ktorých poradí nezáleží, ak ich je však viac, musia byť v zátvorke.

Z rozvoja predprocesorového textu do zdrojového kódu v PL/I je vidieť, že sa najprv vyhodnotí celočíselný výraz a podľa toho, akú má tento výraz hodnotu, sa vykoná príslušná sada príkazov zodpovedajúca jednotlivým predprocesorovým príkazov CASE. Ak "výraz" nadobudne celočíselnú hodnotu, ktorá nie je uvedená v žiadnom príkaze CASE alebo hodnotu mimo intervalu hodnôt špecifikovaných v príkazoch CASE, vykoná sa implicitne sada príkazov "príkazy_d". Ak príkaz DEFAULT nie je uvedený /je nepovinný/, vykoná sa to isté, akoby "príkazy zy_d" pozostávali z jediného prázdneho príkazu.

Vo všeobecnosti možno argumenty príkazov CASE opäť generovať predprocesorom.

4. Použitie

Aby bolo možné používať všetky predprocesorové príkazy opísané v časti 3, musí mať používateľ k dispozícii celkove jedenásť predprocesorových procedúr a niekoľko ďalších predprocesorových deklarácií. Všetok potrebný predprocesorový text je zoskupený do štyroch prvkov /členov/ knižnice predprocesorových textov. Ďalej uvádzame mená týchto štyroch členov a vedľa nich v riadku mená príslušných predprocesorových procedúr, ktoré ten - ktorý prvok obsahuje mimo istých potrebných deklarácií:

STRUCGEN: %UNIQUE
STRUCREP: %REPEAT, %UNTIL
STRUCLOP: %LOOP,%EXITIF, %ENDLOOP
STRUCSEL: %SELECT, %CASE, %DEFAULT, %DEFAULT1, %ENDSELECT

Z názvu jednotlivých prvkov a aj z ich naznačeného obsahu je zrejmé, ktoré predprocesorové príkazy zabezpečujú druhý, tretí a štvrtý prvok. Člen STRUCGEN obsahuje mimo deklarácií procedúru %UNIQUE, ktorá generuje v programe unikátne návestia typu X*unik. Obsah tohoto člena je potrebný pri použití ľubovoľného v článku opísaného predprocesorového príkazu.

V prípade, že chceme využívať v programe príkazy pred-

procesora REPEAT a UNTIL, treba v zdrojovom programe uviesť príkazy predprocesora %INCLUDE STRUCGEN a %INCLUDE STRUCREP, podobne preto, aby bolo možné použiť príkazy LOOP, EXITIF a ENDLOOP, je potrebné uviesť príkazy %INCLUDE STRUCGEN a %INCLUDE STRUCLOP, respektíve pre príkazy SELECT, CASE, DEFAULT a ENDSELECT %INCLUDE STRUCGEN a %INCLUDE STRUCSEL.

Ak chceme využívať dve z troch popísaných doplnkových štruktúr, musí zdrojový kód obsahovať príkazy %INCLUDE STRUCGEN a dva príkazy %INCLUDE pre dva členy, ktoré použitie žiadaných štruktúr zabezpečia. Pre prípad, že chceme v programe využívať všetky tri riadiace štruktúry, použijeme príkaz %INCLUDE STRUCMAS, čo je prvok, ktorý zabezpečí "dotiahnutie" všetkých štyroch doteraz opísaných členov.

Spolu tvoria všetky štyri členy knižnice predprocesorových textov asi 240 riadkov. V prípade, že chceme tieto členy použiť, musí byť knižnica predprocesorových textov popísaná v kroku kompilácie cez DD-meno SYSLIB.

Generovanie unikátnych návestí typu X&uník v procedúre %UNIQUE umožňuje ľubovoľné hniezdovanie predprocesorových príkazov. Ako sme videli v popise použitia, stačí, aby sme do zdrojového textu programu naviac uviedli jeden, dva alebo tri príkazy %INCLUDE a ľubovoľne už potom možno používať tu opísané predprocesorové príkazy. Vzhľadom na to, že predprocesorom vygenerovaný text treba ladiť formálne aj logicky, je vhodné, ak sa pri kódovaní dodržujú v časti 3 naznačené pravidlá formálnej úpravy zápisu predprocesorových príkazov. Vygenerovaný text a aj zdrojový text predprocesora sú potom dobre čitateľné, čomu napomáhajú aj komentáre ohraničujúce začiatok a koniec použitia príslušnej štruktúry.

Použitie predprocesorovej implementácie doplnkových riadiacich štruktúr automaticky zabezpečuje riadené používanie príkazu skoku a eliminuje prípadné problémy z jeho nesprávneho použitia.

```
REPEAT;
   B(J) = A(J);
   J = J + 1;
  UNTIL(A(J) = 0                    );
LOOP;
   READ FILE(INFILE) INTO(V_INFILE);
  EXITIF( EOF_INFILE = '1'B          );
   /*** PROCESS RECORD             ***/
  ENDLOOP;
SELECT( I                            );
   CASE(2);
      /*** PROCESS CASE 2          ***/
   CASE((1,3,5 ));
      /*** PROCESS CASES 1,3,5     ***/
   DEFAULT;
      /*** PROCESS DEFAULT         ***/
   ENDSELECT;
```

obr. 4

```
/*** REPEAT - BEGIN ***/
R#1001:
DO ;
  B(J) = A(J);
  J = J + 1;
  IF ^(A(J) = 0                    ) THEN GOTO R#1001;
  END R#1001
/*** REPEAT - END   ***/ ;
/***  LOOP - BEGIN  ***/
L#1002:
DO ;
  READ FILE(INFILE) INTO(V_INFILE);
  IF  EOF_INFILE = '1'B          THEN GOTO E#1002 ;
  /*** PROCESS RECORD           ***/
  GO TO L#1002;
  E#1002:
  END L#1002
/***  LOOP - END    ***/ ;
/*** SELECT - BEGIN ***/
S#1003:
DO;
   I#1003 =  I                          ;
   IF I#1003<=0 | I#1003>M#1003 THEN GO TO D#1003;
   ELSE GO TO C#1003(I#1003) ;
   C#1003(2): 
     /*** PROCESS CASE 2        ***/
     GO TO E#1003;
   C#1003(1): C#1003(3): C#1003(5):
     /*** PROCESS CASES 1,3,5 ***/
     GO TO E#1003;
   D#1003: C#1003(        4) :
     /*** PROCESS DEFAULT       ***/
   E#1003: END S#1003;
   DCL (I#1003, M#1003 STATIC INIT(        5)) FIXED BIN,
       C#1003(        5) LABEL
/*** SELECT - END    ***/ ;
```

obr. 5

Všeobecný zápis zdrojového kódu v časti 3 sme potrebovali na to, aby sme uviedli princíp jeho rozvoja. Vzhľadom na malý priestor pri všeobecnom rozpise uvádzame na obr. 4 vzorové veľmi krátke výpisy reálneho použitia jednotlivých štruktúr. Dôsledkom použitia týchto štruktúr je kód na obrázku 5.

Použitie štruktúr LOOP-EXITIF a SELECT-CASE môže byť veľmi elegantné. Treba však upozorniť na skutočnosť, ktorá je istým obmedzením použiteľnosti štruktúry SELECT-CASE, a to na už uvedenú požiadavku, že výraz v príkaze SELECT musí nadobúdať celočíselné hodnoty. Je evidentné, že širšie možnosti by poskytoval príkaz SELECT vtedy, keby bolo možné vyhodnocovať jednotlivé prípady aj pre napríklad znakové premenné.

Autor môže prípadným záujemcom poskytnúť na ich magnetickej páske predprocesorový kód potrebný pre použitie popísaných doplnkových riadiacich štruktúr.

Literatúra

[1] McGowan C. L., Kelly J. R.: Top-down structured programming techniques. Petrocelli/Charter, New York 1975.

[2] Fried R., McKenzie R.: The next COBOL standard. Datamation 25/1979/, No. 9, pp. 175 - 180.

[3] Dahl O.-J., Dijkstra E. W., Hoare C. A. R.: Structured programming. Academic press, London - New York 1972.

[4] PL/I. Opisanije jazyka /C51.804.002 D 45/. Moskva 1976.