POUŽITIE ŠTRUKTÚRY it ti PRI NÁVRHU PROGRAMOV

Ing. Ivan Schnapp, Výpočtové laboratórium SIPK

1. Úvod

Ako poznamenal už v r. 1968 Dijkstra /1/, ľudský intelekt je prispôsobený skôr na zvládnutie statických vzťahov a človek horšie dokáže predstaviť si procesy rozvíjajúce sa v čase. Ak pre porozumenie programu treba ho krok za krokom vykonávať, taký program je ťažšie zrozumiteľný než iný program, ktorého funkcia je zrejmá už zo statického popisu - zdrojového textu. Preto sa musíme snažiť, aby súvis medzi zdrojovým textom programu a dynamickým procesom - jeho vykonávaním - bol čo najzrejmejší.

Prvým krokom v tomto smere boli zásady štrukturovaného programovania: obmedzenie používania go to na najnižšiu možnú mieru, dôsledné používanie riadiacich štruktúr typu **if** then else, do while, do until, ktoré nahradili predošlé obmedzené štruktúry. Ďalším zovšeobecnením riadiacich štruktúr sú Dijkstrove štruktúry cô dc a iF fi /2/. Nedávno zaviedol Harlan novú štruktúru it ti, ktorá je ešte všeobecnejšia a nenahrádza ale doplňuje štruktúry do od a if fi /3/.

Chceme na príkladoch ukázať, že programy s touto štruktúrou sú jasnejšie a ich správnosť je očividnejšia než u ekvivalentných programov používajúcich do while a if then else, a to najmä pre programátorov s malými skúsenosťami. To je veľmi dôležité, pretože zavedením štruktúry it ti do návrhu programov by sme mohli skrátiť čas výuky nových programátorov.

Programovací jazyk so štruktúrou it ti a zodpovedajúci kompilátor zatiaľ neexistuje. Je tu však možnosť na báze nejakého rozvinutého jazyka, napr. PL/I, vytvoriť hybridný jazyk nahradením riadiacich štruktúr štruktúrou it ti. Programy v hybridnom jazyku môžeme ručne alebo predprocesorom konvertovať do konvenčného jazyka. Výsledné programy sú síce do značnej miery úsporné na pamäť porovnateľné s programami vytvorenými podľa zásad štrukturovaného programovania.

2. Popis štruktúry it ti

Matematicky presná definícia it ti sa nachádza v /2/. Tu uvádzame len neformálny popis štruktúry a jej funkcie. Vzhľadom na súvis medzi rozhodovacími tabuľkami a štruktúrou it ti budeme používať terminológiu z oblasti RT.

Štruktúra it ti je logický súčet pravidiel uzavretý znakmi it a ti. Každé pravidlo sa skladá z dvoch častí: jednej alebo viacerých podmienok /ich logického súčinu/ a sekvencie činností. Tieto dve časti sú oddelené od seba šipkou smerujúcou doprava. Činnosti v sekvencii sú oddelené bodkočiarkami. Každá sekvencia končí šipkou orientovanou nahor alebo nadol.

Príklad:

it

$p_1 \wedge p_2 \rightarrow č_1; č_2 \downarrow$

$\vee\ p_1 \wedge \sim p_2 \rightarrow č_1; č_4 \downarrow$

$\vee \sim p_1 \rightarrow č_2; č_3; č_4 \uparrow$

ti

Štruktúra it ti sa vykonáva tak, že spomedzi pravidiel, ktorých všetky podmienky sú splnené, vyberie sa náhodne jedno a vykoná sa jemu príslušná sekvencia činností. Ak končí šipkou nahor, celý postup sa opakuje. Táto iterácia končí vtedy, keď na konci sekvencie je šipka orientovaná nadol.

Programy so štruktúrou it ti sú nedeterminované v tom istom zmysle ako programy obsahujúce štruktúry do od a if fi Táto vlastnosť uľahčuje dokazovanie správnosti programu, resp. odvodzovanie správneho programu podľa Dijkstrovej metódy /2/,/4/,/5/.

Z nedeterminovaného programu možno odvodiť determinovaný, ktorý býva efektívnejší, ale obvykle sa pritom zhoršuje zrozumiteľnosť. Preto je výhodné najprv navrhnúť nedeterminovaný program, o ktorého správnosti sme presvedčaní, a potom ho transformovať na determinovaný.

3. Použitie štruktúry it ti

Použitie it ti ukážeme na dvoch typických príkladoch z oblasti hromadného spracovania dát.

Príklad 1: Závod má niekoľko dielní. Za každú dielňu a za celý závod treba zrátať počty pracovníkov a množstvo odpracovaných hodín. Pre každého pracovníka závodu existuje pracovný výkaz obsahujúci číslo dielne a počet odpracovaných hodín. Výkazy sú usporiadané v súbore VYKAZY vzostupne podľa čísla dielne.

Riešenie príkladu v hybridnom jazyku na báze PL/I je takéto:

```
PRACOVNICI_ZAVODU,HODINY_ZAVODU=0;
PRACOVNICI_DIELNE,HODINY_DIELNE=0;
READ FILE(VYKAZY) INTO(VYKAZ);
REFERENCNE_CISLO=CISLO_DIELNE;
it
 EOF(VYKAZY)→                                          | koniec hlavn.
  PUT LIST(REFERENCNE_CISLO,PRACOVNICI_DIELNE,HODINY_DIELNE);
  PRACOVNICI_ZAVODU=PRACOVNICI_ZAVODU+PRACOVNICI_DIELNE;
  HODINY_ZAVODU=HODINY_ZAVODU+HODINY_DIELNE;
  PUT DATA(PRACOVNICI_ZAVODU,HODINY_ZAVODU);
  ↓
EOF(VYKAZY)&CISLO_DIELNE¬=REFERENCNE_CISLO →          | nie koniec
  PUT LIST(REFERENCNE_CISLO,PRACOVNICI_DIELNE,HODINY_DIELNE);   | zmena dielny
  PRACOVNICI_ZAVODU=PRACOVNICI_ZAVODU+PRACOVNICI_DIELNE;
  HODINY_ZAVODU=HODINY_ZAVODU+HODINY_DIELNE;
  PRACOVNICI_DIELNE=1;
  HODINY_DIELNE=HODINY;
  REFERENCNE_CISLO=CISLO_DIELNE;
  READ FILE(VYKAZY) INTO(VYKAZ);                      | nie koniec
  ↑                                                   | nie zmena dielny
EOF(VYKAZY)&CISLO_DIELNE=REFERENCNE_CISLO →
  PRACOVNICI_DIELNE=PRACOVNICI_DIELNE+1;
  HODINY_DIELNE=HODINY_DIELNE+HODINY;
  READ FILE(VYKAZY) INTO(VYKAZ);
  ↑
ti
```

? predchádzajúci súbor – je klúčenie neobvyklé

Funkcia tohto programu je jasná už zo statického popisu - zdrojového textu. Je zrejmé, že pri čítaní novej vety zo súboru VYKAZY môžu nastať len **tri** prípady:

- koniec súboru, t.zn. aj koniec poslednej dielne,
- zmena čísla dielne, čiže koniec dielne,
- ani jeden z týchto prípadov, čiže výkaz je pre tú istú dielňu.

Činnosti pre všetky tri prípady sú zrejmé. Pri zmene dielne môžeme premenné PRACOVNICI_DIELNE nastaviť na 1 a HODINY_DIELNE na údaj HODINY z **prvého** výkazu novej dielne aj bez predchádzajúceho nulovania. Zrejmé je aj v ktorých prípadoch pokračuje iterácia a v ktorých nie. Na pochopenie tohto programu a presvedčenie sa o jeho správnosti netreba vykonávať žiadnu sekvenciu príkazov.

Iná je situácia s riešením podľa zásad štrukturovaného programovania:

```
ON ENDFILE(VYKAZY) KONIEC='1'B;
KONIEC='0'B;
PRACOVNICI_ZAVODU,HODINY_ZAVODU=0;
READ FILE(VYKAZY) INTO(VYKAZ);
DO WHILE(¬KONIEC);
 PRACOVNICI_DIELNE,HODINY_DIELNE=0;
 REFERENCNE_CISLO=CISLO_DIELNE;
 DO WHILE(¬KONIEC&CISLO_DIELNE=REFERENCNE_CISLO);
  PRACOVNICI_DIELNE=PRACOVNICI_DIELNE+1;
  HODINY_DIELNE=HODINY_DIELNE+HODINY;
  READ FILE(VYKAZY) INTO(VYKAZ);
 END;
 PUT LIST(REFERENCNE_CISLO,PRACOVNICI_DIELNE,HODINY_DIELNE);
 PRACOVNICI_ZAVODU=PRACOVNICI_ZAVODU+PRACOVNICI_DIELNE;
 HODINY_ZAVODU=HODINY_ZAVODU+HODINY_DIELNE;
END;
PUT DATA(PRACOVNICI_ZAVODU,HODINY_ZAVODU);
```

Prinajmenšom noví programátori, ale aj tí, ktorým nie sú známe tieto programové schémy, v mysli prejdú programom, aby zistili ako sa chová v prípade prvej a poslednej vety súboru, ako aj pri zmene dielne. Samozrejme, programátori

používajúci tutoschému takéto ťažkosti nemajú vďaka tomu, že o jej správnosti sa presvedčili už skôr a vnímajú ju ako celok s určitými zaručenými vlastnosťami.

Príklad 2: Treba aktualizovať matričný súbor pomocou zmenového súboru a vytvoriť nový matričný súbor takto: Ak pre vetu starého matričného súboru existuje v zmenovom súbore veta s rovnakým kľúčom, do nového matričného súboru sa zapíše veta zmenového súboru. Ostatné vety zmenového aj matričného súboru sa opíšu do nového súboru. Všetky tri súbory sú vzostupne usporiadané podľa kľúčov.

Program v vyvidnom jazyku s it ti vyzerá takto:

```
READ FILE(M) INTO(MATR);
READ FILE(Z) INTO(ZMENA);
it
 EOF(M)&EOF(Z) → ↓                                  | koniec čiarkového s
 EOF(M)&¬EOF(Z) → it                                | koniec súb. M
                   EOF(Z) → ↓
                  ¬EOF(Z) → WRITE FILE(N) FROM(ZMENA);
                            READ FILE(Z) INTO(ZMENA);
                   ti ↑
¬EOF(M)&EOF(Z) → it                                 | koniec súb. Z
                   EOF(M) → ↓
                  ¬EOF(M) → WRITE FILE(N) FROM(MATR);
                            READ FILE(M) INTO(MATR);
                   ti ↑                             | ani koniec čiarky
¬EOF(M)&¬EOF(Z) → it
                   MKLUC<ZKLUC → WRITE FILE(N) FROM(MATR);
                                 READ FILE(M) INTO(MATR); ↓
                   MKLUC>ZKLUC → WRITE FILE(N) FROM(ZMENA);
                                 READ FILE(Z) INTO(ZMENA); ↓
                   MKLUC=ZKLUC → WRITE FILE(N) FROM(ZMENA);
                                 READ FILE(Z) INTO(ZMENA);
                                 READ FILE(M) INTO(MATR); ↓
                   ti ↑
ii
```

Nasleduje jeden z možných štrukturovaných programov:

```
POSLEDNY=HIGH(n);
ON ENDFILE(M)  MKLUC=POSLEDNY;
ON ENDFILE(Z)  ZKLUC=POSLEDNY;
READ FILE(M) INTO(MATR);
READ FILE(Z) INTO(ZMENA);
IF MKLUC<ZKLUC THEN
 REFERENCNY_KLUC=MKLUC;
ELSE
 REFERENCNY_KLUC=ZKLUC;
DO WHILE(REFERENCNY_KLUC<POSLEDNY);
 IF MKLUC<ZKLUC THEN
  DO;
   WRITE FILE(N) FROM(MATR);
   READ FILE(M) INTO(MATR);
  END;
 ELSE IF ZKLUC<MKLUC THEN
  DO;
   WRITE FILE(N) FROM(ZMENA);
   READ FILE(Z) INTO(ZMENA);
  END;
 ELSE
  DO;
   WRITE FILE(N) FROM(ZMENA);
   READ FILE(Z) INTO(ZMENA);
   READ FILE(M) INTO(MATR);
  END;
 IF MKLUC<ZKLUC THEN
  REFERENCNY_KLUC=MKLUC;
 ELSE
  REFERENCNY_KLUC=ZKLUC;
END;
```

Ozrozumiteľnosti týchto dvoch programov platí v podstate to, čo o dvojici programov v prvom príklade. Všimnime si, že v programe s it ti se zvlášť ošetrujú prípady, keď nastane EOF len pre jeden subor. Väčšina programátorov, ktorí nepoznajú horeuvedenú štrukturovanú programovú schému, používa tento prístup; svedčí to o tom,že programovanie

s it ti je bližšie prirodzenému mysleniu ľudí. Pravda, a nicn tento prístup spravidla neprehľadní program. Vsimnime si ešte do seba vložené štruktúry it ti.

4. Prevod hybridných programov do PL/I

Teraz uvedieme pravidlá, podľa ktorých možno mechanicky konvertovať hybridne programy do PL/I:

I. Treba transformovať podmienky typu ECF(meno suboru). Pre každý súbor s takouto podmienkou generujeme deklaráciu

DCL KONIEC_meno suboru BIT(1);

ďalej CK príkaz

ON ENDFILE(meno suboru)KONIEC_meno suboru='1'B;

a ďalej za deklarácie zaradíme príkaz

KONIEC_meno suboru='0'B;

Výraz ECF(meno suboru) v podmienkach pravidiel nahradíme výrazom KONIEC_meno suboru.

II. Štruktúra it ti v hybridnom programe predstavuje vlastne rozhodovaciu tabuľku. Jednotlivé pravidlá sú spredu ohraničené znakmi it,↑ alebo↓, zozadu znakmi ↑,↓ alebo ti. V pravidle sú podmienky od činností oddelené šipkou →.

Najjednoduchší spôsob prevodu takejto tabuľky je tento: Jedno za druhým transformujeme zvlášť každé pravidlo. Pred pravidlo postavíme slovo IF, šipku → nahradíme slovom THEN a sekvenciu činností vrátane šipky↑ alebo↓ uzavrieme do slov DO; a END;/ak obsahuje aspoň jeden príkaz/. Nedostatkom tohto spôsobu je nevelmi efektívny výsledný program. Zložitejšie postupy dávajúce menší alebo rýchlejší program možno nájsť napr. v /6/ alebo /7/.

III. Nakoniec nahradíme znak it náveštím ITi a znak ti náveštím TIi, kde i je číslo slúžiace na rozlíšenie náveští: Ako transformujeme štruktúry it ti, postupne im priraďujeme čísla i=1,2,3,.. Vnútri i-tej štruktúry nahradíme šipky↑ príkazom GO TO ITi, šipky↓ nahradíme príkazom GO TO TIi.

Podľa týchto pravidiel možno konvertovať programy manuálne alebo možno zostaviť predprocesor; vtedy je vhodné nahradiť znaky it, ti,→,↑,↓ rezervovanými slovami napr. IT, TI, THEN, UP, DOWN. Do seba vložené štruktúry it ti pri manuálnom prevode je najlepšie konvertovať vo viacerých prechodoch, najprv vonkajšie, potom vnútorné štruktúry.

Výsledkom manuálnej konverzie hybridných programov z Príkladu 1 a 2 sú tieto programy:

```
DCL KONIEC_VYKAZY BIT(1);
ON ENDFILE(VYKAZY) KONIEC_VYKAZY='1'B;
KONIEC_VYKAZY='0'B;
PRACOVNICI_ZAVODU,HODINY_ZAVODU=0;
PRACOVNICI_DIELNE,HODINY_DIELNE=0;
READ FILE(VYKAZY) INTO(VYKAZ);
REFERENCNE_CISLO=CISLO_DIELNE;
IT1:
IF KONIEC_VYKAZY THEN
 DO;
  PUT LIST(REFERENCNE_CISLO,PRACOVNICI_DIELNE,HODINY_DIELNE);
  PRACOVNICI_ZAVODU=PRACOVNICI_ZAVODU+PRACOVNICI_DIELNE;
  HODINY_ZAVODU=HODINY_ZAVODU+HODINY_DIELNE;
  PUT DATA(PRACOVNICI_ZAVODU,HODINY_ZAVODU);
  GOTO TI1;
 END;
ELSE IF CISLO_DIELNE=REFERENCNE_CISLO THEN
 DO;
  PUT LIST(REFERENCNE_CISLO,PRACOVNICI_DIELNE,HODINY_DIELNE);
  PRACOVNICI_ZAVODU=PRACOVNICI_ZAVODU+PRACOVNICI_DIELNE;
  HODINY_ZAVODU=HODINY_ZAVODU+HODINY_DIELNE;
  PRACOVNICI_DIELNE=1;
  HODINY_DIELNE=HODINY;
  REFERENCNE_CISLO=CISLO_DIELNE;
  READ FILE(VYKAZY) INTO(VYKAZ);
  GOTO IT1;
 END;
ELSE
 DO;
  PRACOVNICI_DIELNE=PRACOVNICI_DIELNE+1;
  HODINY_DIELNE=HODINY_DIELNE+HODINY;
  READ FILE(VYKAZY) INTO(VYKAZ);
  GOTO IT1;
 END;
TI1:
```

```
DCL KONIEC_1 BIT(1);
DCL KONIEC_2 BIT(1);
ON ENDFILE(1) KONIEC_1='1'B;
ON ENDFILE(2) KONIEC_2='1'B;
KONIEC_1='0'B;
KONIEC_2='0'B;
III:
IF KONIEC_1 THEN
 IF KONIEC_2 THEN
  GOTO KKK;
 ELSE
  DO;
   III:
   IF KONIEC_2 THEN
    GOTO III;
   ELSE
    DO;
     WRITE FILE(3) FROM (REC2);
     READ FILE(2) INTO (REC2);
     GOTO III;
    END;
  END;
  GOTO III;
 END;
ELSE
 IF KONIEC_2 THEN
  DO;
   III:
   IF KONIEC_1 THEN
    GOTO III;
   ELSE
    DO;
     WRITE FILE(3) FROM (REC1);
     READ FILE(1) INTO(REC1);
     GOTO III;
    END;
   KKK:
   GOTO KKK;
  END;                    /* pokračovanie na ďalšej strane */
```

```
ELSE          /* pokračovanie z predchádzajúcej strany */
  DO;
   IF MKLUC<ZKLUC THEN
    DO;
     WRITE FILE(M)FROM(MATR);
     READ FILE(M)INTO(MATR);
    END;
   ELSE IF ZKLUC<MKLUC THEN
    DO;
     WRITE FILE(M)FROM(ZMENA);
     READ FILE(Z)INTO(ZMENA);
    END;
   ELSE
    DO;
     WRITE FILE(M)FROM(ZMENA);
     READ FILE(Z)INTO(ZMENA);
     READ FILE(M)INTO(MATR);
    END;
   GOTO IT1;
  END;
IT1:
```

Je zrejmé, že druhý program by mohol byť ďalej optimalizovaný: skoky na TI2 a TI3 môžu byť nahradené skokmi rovno na IT1 a príkazy s návesťami TI2 a TI3 môžu byť vynechané. Takáto optimalizácia, ktorá sa robí po generovaní programu z RT, je popísaná napr. v /7/.

Ďalšia optimalizácia /použitá v programe/ spočíva vo vypustení návestí ITi a TIi a príkazov GOTO TIi ak v i-tej štruktúre všetky pravidlá končia šípkou nadol.

Nakoniec, druhý program mohol byť optimalizovaný už pri generovaní testov podmienok /príkazov IF/: vo väčšine prípadov skončí súbor zmien skôr než matričný súbor a tak podmienka KONIEC_Z mala byť testovaná pred KONIEC_M. Optimalizácia tohto druhu je uvedená napr. v /6/.

Literatúra

1/ Dijkstra, E.W.: Go To Statement Considered Harmful
CACM 11, 3 /March 1968/

2/ Dijkstra, E.W.: A Discipline of Programming. Prentice-Hall
Englewood Cliffs, N.J., 1976

3/ Parnas, D.L.: A Generalized Control Structure and Its Formal
Definition. CACM 26,8 /August 1983/

4/ Gries, D.: The Science of Programming. Springer-Verlag,
New York, N.Y., 1981

5/ Honzík, J.: Dokazování programu. V Programování '83,
Dum techniky Ostrava, 1983

6/ Chvalovský, V.: Rozhodovací tabulky. SNTL, Praha 1974
str. 92-110

7/ Myers, H.J.: Compiling Optimized Code from Desicion Tables.
IBM Journal Res. Develop. 16,5 /September 1972/