Ing. Josef Tvrdík

Báňské projekty, Ostrava

POZNÁMKY KE STRUKTUŘE FORTRANSKÝCH PROGRAMŮ

Úvod

Rychlé zvýšení počtu počítačů v několika posledních letech podstatně rozšířilo okruh uživatelů i v oblasti tzv.vědecko-technických výpočtů. Užívání programů v této oblasti definitivně přestalo být otázkou několika lidí majících zpravidla bezprostřední kontakt s autorem programu. Programy jsou nyní provozovány na více počítačích často různých typů v různých výpočetních střediscích, případně i zemích. Uživatel - neprogramátor má k dispozici pouze uživatelský manuál a tato výbava by měla být postačující k tomu, aby pro svoje vstupní data obdržel v co nejkratším čase výsledky úlohy.

Důsledkem této skutečnosti je požadavek vysoké spolehlivosti programů. Lze říci, že nároky na spolehlivost programů v oblasti vědecko-technických výpočtů se během několika let kvalitativně zvýšily. Navíc nároky uživatelů na komplexnější automatizaci duševní činnosti a stále se rozšiřující využití výpočetně i logicky náročných moderních matematických metod vedou k tvorbě velice rozsáhlých programů či programových systémů. Rychlé a efektivní programování takových úloh vyžaduje většinou současnou práci několika programátorů.

Dosud nejrozšířenějším jazykem užívaným k programování těchto úloh je Fortran a zřejmě ještě několik let bude tento jazyk běžně užíván. Programování ve Fortranu je však často silně ovlivněno dříve rozšířeným nesystematickým přístupem k tvorbě programů, neboť pravidla jazyka Fortran k nesystematičnosti téměř vybízejí.

Nové požadavky na kvalitu programů, především zvýšené nároky na spolehlivost, vyžadují zlepšení programovacího stylu a využití takové metody programování, která by umožňovala rychlou a jednoduchou tvorbu spolehlivých programů. Jednou z metod, která dosažení těchto cílů slibuje, je metoda strukturovaného programování.

Zásady strukturovaného programování

Metodou strukturovaného programování se zabývá mnoho autorů, na př. [1, 2, 3, 4, 7] a v těchto pracích je možno se s principy této metody podrobněji seznámit.

Strukturované programování lze stručně charakterizovat jako systematickou metodu řešení problému, jejíž hlavní zásadou je respektování faktu, že schopnost našeho uvažování není nekonečná. Tato zásada se projevuje volbou co nejjednodušších prostředků k řešení úlohy. To znamená, že úloha se řeší postupnou dekompozicí na části směrem shora dolů tak, aby kterákoliv část (podúloha) byla na dané úrovni snadno řešitelná a aby měla jednoznačně definované okolí, t. zn. vstupy a výstupy. Počet úrovní dekompozice a počet podúloh (procedur) na dané úrovni podle povahy problému volíme takový, aby každá podúloha byla dostatečně jednoduchá a snadno programovatelná ve zvoleném programovacím jazyku.

K dekompozici problému až do úrovně programovacího jazyka nepotřebujeme příkaz skoku a postačí velice jednoduché algoritmické konstrukce:

a) sekvence příkazů

b) if (p) then

else +)

c) while (p) do

d) do until (q)

Pokud máme k dispozici programovací jazyk přímo umožňující uvedení algoritmické konstrukce, znamená to, že až do úrovně zdrojového jazyka včetně se obejdeme bez příkazu skoku (GO TO), který je jednou z hlavních příčin nepřehlednosti programu.

Snaha o co největší přehlednost řešení vede i k triviálním formálním doporučením. Zdrojová forma zápisu jednotlivých částí úlohy, t. zn. procedur nebo podprogramů nemá přesáhnout jeden list počítačového papíru, abychom udrželi program co nejpřehlednější. Programátorem volená jména by měla být pokud možno samovysvětlující. Dále, komentář je považován za součást zdrojového textu určenou člověku - čtenáři programu. Proto by měl být volen tak, aby čtenáři programu pomáhal k pochopení zvolené metody řešení. Mimo jiné to zn., že s podstatnou změnou v programu má být změněn i příslušný komentář. V této formě je pak zdrojový text dostatečně podrobnou a srozumitelnou dokumentací programu a není třeba jej doplňovat vývojovými diagramy, které je obtížné aktualizovat a jejichž schopnost popsat algoritmus většinou není lepší než přehledně napsaného zdrojového textu.

Strukturované programování, jako ostatně každá systematická metoda, přispívá nejen k usnadnění řešení problému, ale především usnadňuje odstraňování chyb při ladění programů a zvyšuje spolehlivost hotových programů.

Aplikace zásad strukturovaného programování ve Fortranu

Fortran je nestrukturovaný jazyk a proto je stěží možné programovat v tomto jazyku, aniž bychom užili příkazu

- - -

+) Konstrukce s else podle některých autorů [5,6] způsobuje nepřehlednost dekompozice a tím i programu a tudíž nedoporučují její užívání.

skoku[+]. Avšak domníváme se, že metoda strukturovaného programování rozhodně znamená tvrdý požadavek zápisu programu bez příkazu skoku, a tudíž otázka využití zásad této metody při programování ve Fortranu není nesmyslná.

Analyzujeme-li texty fortranských programů psaných tradičním způsobem, zjišťujeme, že přehlednost a čitelnost programů snižují především tyto faktory:

- přílišná délka programových jednotek
- zbytečné užívání příkazu GO TO způsobené příliš komplikovaným uvažováním autora
- užívání osobitých triků
- zbytečně vysoký počet návěští v programu a jejich nesystematické označení
- užívání aritmetického IF
- opticky příliš zhuštěný zápis programového textu bez mezer a odstavcování
- nevýstižné nebo žádné komentáře v textu programu
- špatná nebo žádná mnemotechnika jmen
- užívání COMMONu místo parametrů

Použitím zásad strukturovaného programování lze tyto faktory eliminovat nebo omezit na přijatelnou míru.

U části uvedených faktorů pomůže dokonce už i využití triviálních principů strukturovaného programování. Mnemotechnická jména je možno užívat až do délky 6 znaků. Text jednotlivých příkazů lze zpřehlednit vhodným zařazením mezer a logicky související sekvence příkazů (odstavce) je možno odsadit vpravo, neboť kompilátor tyto mezery v pro -

---

+) Ve většině úloh by snad bylo možné se obejít bez příkazu skoku tak, že by každá sekvence dvou a více příkazů prováděných podmíněně byla koncipována jako samostatná programová jednotka - SOUBROUTINE. Avšak takovýto způsob zápisu by přehlednost programu nezvýšil, spíše naopak. Kromě toho by měl i další nepříznivé efekty - prodloužení času potřebného k linkování programu a k výpočtu.

gramovaném textu při překladu ignoruje. Do textu programu lze zařadit dostatečně výstižné komentáře a je možné tyto komentáře aktualizovat, pokud při ladění programu dojde k závažnému zásahu. Je sice pravda, že komentáře ve fortranských programech jsou na samostatných štítcích a programátor tedy není při změně příkazu ke změně komentáře donucen, ale objektivní překážky ke změně komentáře nemá. Zhoršení čitelnosti programu způsobené užitím aritmetického podmíněného příkazu se zabrání jednoduše tím, že se tento příkaz nebude užívat a nahradí se logickým IF (pokud ovšem programátor není nucen programovat ve Fortranu nižší úrovně neobsahujícím logické IF). Podobně zlepšení čitelnosti programu lze dosáhnout preferováním parametrů před COMMONem.

Přílišné délce programovaných jednotek je možno zabránit včasnou dekompozicí úlohy tak, aby zdrojový text jednotlivých programových jednotek měl přijatelnou délku (za přijatelnou maximální délku je možno považovat 1-2 stránky). Rozhodnutí o tom, jak má být úloha dekomponována, (t.zn. počet úrovní a počet programovaných jednotek na jednotlivých úrovních), závisí na povaze úlohy a na úvahách řešitele.

Je tedy zřejmé, že většinu příčin nepřehlednosti a špatné čitelnosti fortranských programů lze odstranit využitím triviálních principů strukturovaného programování. Z faktorů uvedených výše nebyly dosud diskutovány pouze tři a to užívání osobitých triků, zbytečné užívání GO TO a s tím související zbytečné množství návěští.

Škodlivost užívání triků ukážeme na malém příkladu [6]:

```
   DO 10   I = 1, N
   DO 10   J = 1, N
10 X(I,J) = (I/J) * (J/I)
```

Pozorný čtenář jistě poznal, že výsledkem tohoto postupu je jednotková matice X typu N*N, ale stálo ho to zbytečnou námahu, která mohla být ušetřena, pokud by tato operace byla zapsána průhledněji, na př. takto:

```
   DO  20  I = 1, N
     DO  10  J = 1, N
       IF (I .EQ. J)   X(I,J)  = 1.
       IF (I .NE. J)   X(I,J)  = 0.
10   CONTINUE
20 CONTINUE
```

nebo takto:

```
   DO  10  I = 1, N
     DO  10  J = 1, N
       X(I,J)  = 0.
       IF (I .EQ. J)  X(I,J)   = 1.
10 CONTINUE
```

Pokud bychom chtěli ušetřit nějaké mikrosekundy strojového času při výpočtu, i když se domnívám, že úspora mikrosekund již není hlavním kriteriem v posuzování kvality programu, je možné užít zápis, který je rozhodně přehlednější (a podstatně rychlejší) než "trikový ":

```
   DO  20  I = 1, N
     DO  10  J = 1, N
10     X(I,J)   = 0.
20   X(I,I)   = 1.
```

Užívání triků značně ztěžuje odstraňování chyb při ladění programu a téměř znemožňuje zásah do programu nejen jiným programátorům, ale i autorovi programu po jistém časovém období, během kterého na použitý trik zapomene. Jediný způsob, jak zabránit nepříznivým důsledkům triků, je neužívat je vůbec.

Zbytečné užívání příkazu skoku a tudíž i zbytečná návěští příkazů lze omezit především zjednodušením myšlenkových postupů při programování a vhodnou volbou vyjadřovacích možností, které Fortran poskytuje [6]. Zbytečně složité uvažování může vést dokonce až k programům tohoto stylu:

```
   IF  (X1 .LT. X2)   GO TO  30
   IF  (X2 .LT. X3)   GO TO  50
```

```
   XMIN = X3
   GO TO 70
30 IF (X1 .LT. X3) GO TO 60
   XMIN = X3
   GO TO 70
50 XMIN = X2
   GO TO 70
60 XMIN = X1
70 ....
```

Daleko jednodušší logický postup této úlohy a tím i průhlednější zápis je tento:

```
   XMIN = X1
   IF (X2 .LT. XMIN)  XMIN = X2
   IF (X3 .LT. XMIN)  XMIN = X3
```

K nejjednodušššímu a nejčitelnějšímu zápisu této úlohy dojdeme, vzpomeneme-li si včas na vhodnou standardní funkci, to znamená volíme vhodný vyjadřovací prostředek:

```
   XMIN = AMIN0 (X1, X2, X3)
```

Další možnost jak omezit počet příkazů skoku v programu a předem vyloučit především příkazy skoku směrem nahoru v textu programu, je užit při řešení úlohy nejdříve nějakého třeba fiktivního strukturovaného jazyka. Na př. v tomto jazyku je úloha zapsána takto:

```
if (p) then
       příkazy 1
       if (q) then
              příkazy 2
              end
       příkazy 3
       end
```

kde p, q jsou logické výrazy.

Takto zapsaný program zakódujeme v jazyku Fortran (logické výrazy a sekvence příkazů jsou zapsány pouze symbolicky):

```
   IF ( .NOT. (p) )     GO TO 20
```

```
      příkazy 1
      IF  ( .NOT. (q) ) . GO TO 10
        příkazy 2
10    příkazy 3
20    CONTINUE
```

Tutéž úlohu můžeme zapsat ve strukturovaném jazyku trochu jinou formou, vycházející ze zásad metody funkcionálního programování [5], v níž se neužívá else a potlačují se složité víceúrovňové struktury:

```
if  (p)     then
            příkazy 1
            end
if (p   q)  then
            příkazy 2
            end
if (p)      then
            příkazy 3
            end
```

Pak zápis ve Fortranu (opět se symbolickým vyjádřením logických výrazů a sekvencí příkazů) je tento:

```
   IF  ( .NOT.  (p))          GO TO 15
     příkazy 1
15 IF  ( .NOT.  (p .AND. q))  GO TO 25
     příkazy 2
25 IF  ( .NOT.  (p))          GO TO 35
     příkazy 3
35 ...
```

Jak je patrné, oba uvedené způsoby vedou k přehlednému zápisu úlohy ve Fortranu, k textu čitelnému shora dolů. Druhý způsob se nám jeví přehlednější, i když obsahuje oproti prvnímu způsobu o jeden příkaz GO TO a jedno návěští navíc, neboť má všechny IF příkazy na jedné úrovni a tudíž je velice snadné rozeznat v textu programu, při kterých podmínkách se jednotlivé sekvence příkazů provádějí. Logické výrazy užívající operátoru .NOT. je možno pro zrychlení vý -

počtu při kódování úlohy zjednodušit. Toto zjednodušení však většinou zhorší čitelnost programu a navíc je v něm skryto nebezpečí chyby.

Důležitým prvkem strukturovaného programovacího jazyka je příkaz while do. Fortran tento příkaz neobsahuje a tudíž programátoři, zvláště ti, pro které byl Fortran prvním programovacím jazykem, jemuž se naučili, užívají konstrukce while do většinou nedůsledně. Užitečnost a účinnost tohoto příkazu ukazují následující příklady:
Tradiční způsob uvažování ve smyslu fortranského příkazu cyklu vede k tomuto zápisu programu (součet řady) - podobný příklad je uveden v [6] :

```
      DOUBLE PRECISION FUNCTION SIN (X, E)
C     SIN (X), REAL*8, PRESNOST E
      DOUBLE PRECISION E, TERM, SUM
      TERM = X
      DO 10 K = 2, 100, 2
      TERM = - TERM * X**2 / (K * (K + 1))
      IF (DABS (TERM) .LT. E) GO TO 30
   10 SUM = SUM + TERM
   30 SIN = SUM
      RETURN
      END
```

Je zřejmé, že konvergenční test je špatně umístěn, neboť poslední počítaný TERM není zahrnut v SUM. Této chybě lze předejít naprogramováním úlohy nejdříve ve fiktivním strukturovaném jazyku obsahujícím příkaz while do. Obecný zápis úlohy je tento:

```
inicializuj
while    (důvod pro smyčku)  do
      příkazy smyčky
      end do
```

Uvedenou úlohu tedy můžeme napsat:

```
SIN  = X
TERM = X
```

```
K = 2
while (K<100 & abs(TERM) > E) do
     TERM = - TERM*X**2 /(K*(K + 1 ))
     SIN  = SIN + TERM
     K = K + 2
     end do
```

a pak tento zápis přeložit do FORTRANU:

```
     ...
     SIN = X
     TERM = X
     DO 20 K = 2, 100, 2
     IF (DABS (TERM) .LT. E)    GO TO 30
       TERM = - TERM*X**2 / FLOAT (K*(K + 1))
       SIN  = SIN + TERM
  20 CONTINUE
  30 RETURN
     END
```

Příkaz while do je do Fortranu převeden jako příkaz IF následující za příkazem DO, ve kterém je obsažena i část inicializačních příkazů a řízení smyčky. I fortranský zá - pis umožňuje zřetelně oddělit inicializační a řídící příkazy od příkazů výpočtových. Umístění příkazu IF (důvod pro smyčku) bezprostředně za příkazem DO zabrání bezdůvodnému provádění příkazů cyklu.

Druhý příklad, na kterém je ilustrována užitečnost po- užití while do, je úsek z hlavního programu (výpočtu potrubní sítě), ve kterém je volán opakovaně větší počet podprogramů. Tradiční (a nyní fungující) forma zápisu této části programu je tato:

```
     ...
     DO 400 I = 1, NZ
400  GMN(I) = GMNS(I)
     NKOR = 0
500  CALL ZTRATY
     CALL PRUTOK
```

```
    M = 0
640 IF (JZ .GT. 0)          CALL CROSS
    CALL TLAKY
    IF (JZ .EQ. 0)          GO TO 660
    M = M + 1
    CALL KONVRG (KON)
    IF (KON .EQ. 0 .AND. M .LT. MAXIT)  GO TO 640
660 CALL KRIKOR (KT)
    IF (KT .EQ. 2)          NKOR = NKOR + 1
    IF (KT .NE. 2)          GO TO 710
    DO 700 I = 1, NZ
700 GMN (I) = GMNS (I)
    IF (KT .EQ. 2  .AND. NKOR .LT. NKRMAX)  GO TO 500
710 ...
```

NZ, MAXIT a NKRMAX jsou dříve inicializované celočíselné proměnné, KON a KT jsou parametry podprogramů KONVRG a KRIKOR, podle jejichž hodnoty se rozhoduje v programu o dalším pokračování výpočtu.

Důsledkem nedisciplinovaného používání příkazu GO TO, především směrem vzhůru v textu programu, a zbytečně komplikovaně zařazeným příkazům, ve kterých jsou počítány a testovány počty průchodů úseky programu, je tato část programu velice špatně čitelná. Dvojí inicializace vektoru GMN v textu programu a zařazení některých logicky podmíněných příkazů ukazuje, že program byl opravován během ladění, t. zn., že zápis programu byl příliš uspěchaný bez důkladné předcházející analýzy problému. Jak ukazuje zápis ve strukturovaném jazyku (bez použití else), není tato úloha tak složitá:

```
KT   =  2
NKOR =  1
while   (KT = 2 & NKOR ≤ NKRMAX)
         do
         inicializuj  GMN
         call  ZTRATY
```

```
          call  PRUTOK
          if    (JZ > 0)  then
                          KON  =  0
                          M    =  1
                          while  (KON = 0 & M ⩽ MAXIT)
                                do
                                call  CROSS
                                call  TLAKY
                                call  KONVRG (KON)
                                M = M + 1
                                end do
                          end
          if  (JZ ⩽ 0)    then call TLAKY end
          call  KRIKOR  (KT)
          NKOR  =  NKOR + 1
          end do
```

Po překladu do Fortranu dostaneme

```
      ...
      KT  =  2
      DO  40  NKOR  =  1,   NKRMAX
        IF  (KT .NE. 2)     GO  TO  50
          DO  10  I  =  1,  NZ
10          GMN(I)   =  GMNS(I)
          CALL  ZTRATY
          CALL  PRUTOK
          IF  (JZ .LE. 0)   GO  TO  30
            KON  =  0
            DO  20  M  =  1, MAXIT
              IF  (KON  .NE.  0)   GO  TO  40
              CALL  CROSS
              CALL  TLAKY
20            CALL  KONVRG  (KON)
30        IF  (JZ  .LE.  0)  CALL TLAKY
40        CALL  KRIKOR  (KT)
50        ...
```

Tato forma zápisu je nepochybně přehlednější než původní, neboť neobsahuje skoky směrem nahoru a jednotlivé části programu jsou odděleny vhodným užitím příkazu cyklu. Navíc tento zápis umožňuje jednoduché rozšíření programu o diagnostiku průběhu výpočtu zařazením potřebných příkazů za konec DO cyklů, t. zn. za návěští 20, resp. 40.

Závěr

Mnohé z uvedených postupů využití zásad strukturovaného programování při programování v jazyku Fortran byly užívány dříve, než pro ně byl tento název zaveden a než tato metoda dosáhla současné šíře použití. Snad právě tento fakt je důkazem užitečnosti metody strukturovaného programování.

Pokud tento referát přispěje alespoň inspirativně k dalšímu zjednodušení zápisu fortranských programů a tím ke zvýšení jejich spolehlivosti, k úspoře času a námahy programátorů při ladění a dokumentaci programů, pak splnil svůj účel.

Literatura:

1. Dahl O.J., Dijkstra E.W., Hoare C.A.R., Structured Programming, Academic Press, London, New York 1972
2. Kořejš J., Principy strukturovaného programování, Informační systémy 4, č. 2 a 3 /1975/
3. Knuth D.E., Structured Programming with go to Statements, ACM Comp. Surveys 6, 261 /1974/
4. Wirth N., On the Composition of Well - Structured Programs, SCM Comp. Surveys 6, 247 /1974/
5. Bloom A.M., The "else" must go, too, DATAMATION č. 5 /květen 1975/
6. Kernighan B.W., Plauger P.J., Programming Style: Examples and Counterexamples, ACM Comp. Surveys 6, 303 /1974/
7. Brzický J., Strukturované programování a prvé zkušenosti s jeho použitím v jazyce PL 1, Seminář "Metody programování počítačů III: generace", Havířov 1975