PROSTŘEDKY A PROSTŘEDÍ PRO VÝVOJ PROGRAMŮ

Ing. Pavel Ježek

Motto : Jakmile byl tento problém vyřešen,
ukáže se, že byl velmi jednoduchý.

1. Úvod

Vždy, když se objeví nějaký nový programovací jazyk nebo prostředek, můžeme v obci programátorské pozorovat několik charakteristických tendencí. Nadšenci zajásají a s chutí se na novinku vrhnou, fesimisté ohrnou nos a prohlásí např. že to je hračka pro děti. Kdo však uvažuje nezaujatě, snaží se zjistit, jaký je praktický význam dotyčného prostředku a jaké klady, zápory a důsledky by mělo jeho eventuální zavedení. Každý programovací prostředek vytváří prostředí s určitými vlastnostmi, v jehož rámci probíhá tvorba programového vybavení a které má nemalý vliv na myšlení i produktivitu práce programátora. Cílem následujících úvah není podávat podrobné charakteristiky různých programovacích prostředků, ale ukázat vlastnosti jimi vytvořeného prostředí.

2. Klasické programování

Vysvětlovat problematiku programování pomocí klasických jazyků není nutné. Jaké však tyto jazyky vytvářejí prostředí? Programátor, tak jak postupně získává zkušenosti, se často dostane do situace, kdy by starší program nejraději vyhodil a napsal jej úplně znovu a jinak. Postupně si klade otázky :

- jak zpřehlednit zdrojový text, aby se v něm lépe vyznal
- jak jej rozčlenit na logicky související části
- jaký zvolit systém při pojmenovávání proměnných a návěští
- jakými opatřeními zvýšit spolehlivost programu a jak snížit pracnost při jeho údržbě

- jak dospět k univerzálně a opakovaně použitelným modulům
- jak rozdělit programy podle své funkce, aby se z nich dal vytvořit dobrý programový systém
- a podobně.

Nalézáním řešení těchto problémů získává programátor praxi a stále více se přibližuje k zásadám strukturovaného programování. Klasický programovací jazyk, např. FORTRAN, se vyznačuje tím, že k práci potřebujeme v podstatě editor zdrojových textů, překladač, spojovací program a knihovník. Vytváří prostředí, který má určité specifické rysy :

- všechny části jsou vzájemně odděleny (jsou to samostatné programy) a pracuje se s nimi postupně. Napíše se zdrojový text, přeloží se a spojí do spustitelného tvaru. Při nalezení chyby je nutno celý postup opakovat,
- výsledkem práce je jednotlivý program, který musí korespondovat s vytvářeným programovým systémem,
- změna ve struktuře dat nebo funkční změna může mít vliv na všechny programy systému - pracná údržba,
- dílčí výsledky můžeme dostat až tehdy, když zcela naprogramujeme alespoň nejdůležitější programy,
- jeden a tentýž problém lze naprogramovat tak, že se k uživateli bude chovat naprosto rozdílně,
- používají se návěstí a proměnné; abychom mohli naprogramovat určitou činnost, pak nemůžeme myslet jen na řešený problém, ale musíme vzít v úvahu i mnoho dalších odtažitých věcí :
  - jakého typu budou proměnné a jakých budou nabývat hodnot
  - jak velká budou pole, jaké budou mít indexy a zda se nepřekročí jejich meze
  - jakou funkci budou tyto objekty v programu zastávat
  - jaký bude přístup k souborům a jaké budou jejich další atributy při otevření a uzavření
  - jak se ošetří vstupně-výstupní operace (konec dat, chyba)
  - jak se bude pracovat s pamětí (sdílení, přidělování, ...)
  - zda se bude program segmentovat

- jak se naloží s různými zvláštními případy
- a podobně.

Vyjmenované vlastnosti prostředí, vytvořeného klasickým programovacím jazykem, ukazují na to, že :

- tvorba aplikace je poměrně pracná a je nutno ji obvykle složit z několika jednotlivých programů
- jsou potřebné značné znalosti nejen samotného programovacího jazyka, ale i operačního systému a systémových programů
- dojde-li ke změnám, pak údržba programovaného systému je značně náročná a může zasáhnout jeho velkou část
- souhrn těchto vlastností podporuje specializaci a profesní rozdělení "uživatel - analytik - programátor".

3. Zlepšené klasické programování

Výše zmíněné nedostatky se snaží výrobci softwaru odstranit nebo zmírnit. Zajímavou novinkou bylo zavedení interpretačních jazyků, jakým je BASIC. Ten se snažil zjednodušit příkazy jazyka a přinesl i jiná zkvalitnění práce, např. výpis obsahu proměnných a jeho změna při běhu programu, ladění programu po částech apod. Dalším pokrokem byla integrace jazyků do vývojového programovacího systému, jako je Turbo-Pascal. Zde se podařilo zvýšit komfort při práci tím, že všechny části, tj. editor, překladač i linker tvoří jedno prostředí, v němž se lze plynule pohybovat.

Při nalezení chyby ji editor ukazuje ve zdrojovém textu. Jedinou nevýhodou je, že takto vytvořené pascalské programy nelze spojovat s programy, napsanými v jiných jazycích. Ukažme si, že vylepšené prostředí klasických jazyků nemusí být pouze předmětem zájmu softwarových firem, ale že se o ně lze pokusit i v našich běžných podmínkách. Při práci s FORTRANem bylo stále více jasné, že je třeba něco udělat pro zvýšení produktivity práce a zavedení standardizace. Tak vznikly prostředky sice povětšinou známé, ale v souhrnu tvořící rovněž prostředí zlepšeného klasického programování. Lze je rozdělit do

tří částí : knihovna podpůrných podprogramů (řeší dílčí podproblémy), obslužné programy (standardizují práci v prostředí) a předprocesor jazyka (zvyšuje úroveň programování). Tento systém může uživatel dále rozšiřovat.

Knihovna podpůrných podprogramů má za cíl usnadnit a sjednotit práci s dílčími podproblémy, které se při tvorbě programů často vyskytují. Obsahuje podprogramy pro :

- práci se soubory (otevření, čtení/zápis vět proměnné délky)
- práci se znakovými řetězci (poli znaků)
- pomocné činnosti (konverze a převody, setřídění apod.)
- práci s datumem.

Obslužné programy - protože operační systém obvykle obsahuje hodně systémových programů, lze k jednomu výsledku dospět různými cestami. Kvůli zastupitelnosti programátorů a jednotné dokumentaci to přirozeně není žádoucí. Naopak je třeba stejné činnosti dělat stejným způsobem pomocí vybraného programového vybavení.

Pro sjednocení těchto činností vznikly programy :

XFTR - slouží k formátování FORTRANského zdrojového textu. Ten se píše editorem jako polotovar od 1. sloupce obrazovky a obsahuje jednoduché direktivy pro formátování. Výsledkem je normální zdrojový text, jenž obsahuje standardizované komentáře a který může mít hlavičku pro doplnění základních údajů o vyvíjeném programu (dokumentační část textu)

XVYP - standardní výpis FORTRANských zdrojových textů

XPOR - provádí porovnání dvou zdrojových textů a slouží hlavně k rozpoznání jednotlivých verzí téhož programu

XVBR - slouží k výběru částí zdrojových textů z různých hotových programů a k jejich spojení do jednoho souboru. Tím odpadá psaní vícenásobně použitelných částí

XZMN - může provádět hromadné změny ve zdrojových nebo jiných textech (např. změna názvů proměnných)

XTXT - slouží k tisku dokumentace, psané analytiky a programátory. Text obsahuje jednoduché direktivy pro řízení tisku. Výhodou je, že při změnách lze z celé dokumentace vytisknout třeba jen jednu stránku

Předprocesor jazyka FORTRAN XPPF značně zlepšuje práci s jazykem FORTRAN-IV nebo FORTRAN-77. Vznikl proto, aby bylo možno využít nových poznatků ze strukturovaného programování. Hostitelský jazyk byl rozšířen o pseudopříkazy, které umožňují programovat pomocí tří základních metod : metoda shora-dolů, Jacksonova metoda, metoda struktur IT-TI a rozhodovacích tabulek. Program vygenerovaný pomocí XPPF lze také napsat "ručně", avšak pracněji. Místo toho se programátor může lépe soustředit na vlastní řešený problém, i když i zde musí myslet na různé "rozptylující" okolnosti, popsané u klasického programování. Při práci s XPPF je třeba dodržovat určité zásady. Programátor však není omezován a všechny dobré vlastnosti FORTRANu zůstávají zachovány. Je třeba dodržet tyto rozsahy návěští :

- návěští volená programátorem pro vlastní potřebu pro vlastní potřebu :	1- 999
- návěští volená programátorem pro označení bloků :	1000-20999
- návěští volená programátorem pro FORMATy a další :	21000-79999
- návěští generovaná předprocesorem :	80000-99999

Při programování jsou k dispozici tyto pseudopříkazy :

∂BLOCK	1 2 3	x = před příkazem může být použito návěští
x ∂BACK	1 2 3	
x ∂PERFORM	1 2 3	1 = použití pro metodu shora-dolů
x ∂PERFORM-WHILE	1 2 3	
x ∂PERFORM-UNTIL	1 2 3	2 = použití pro Jacksonovu metodu
x ∂REPEAT-UNTIL	1 2 3	

* ∂SWITCH	1 2 3	3 = použití pro metodu struktur IT-TI
∂THEN	1	
∂ELSE	1	
∂END	1 2	
∂SELECT	2	Navzájem nelze kombinovat :
∂OR	2	
∂DO-WHILE	2 1 3	
∂DO-UNTIL	2 1 3	- složené podmínky a selekce
* ∂DO-END	2 1 3	- složené podmínky a IT-TI
* ∂EXIT	2 1 3	- selekce a struktury IT-TI
* ∂EXIT-ALL	2 1 3	- struktura IT-TI nesmí být vložena do cyklu " DO-..."
∂IT	3	
∂PART-TI	3	
∂PART-IT	3	PERFORMy a cykly lze neproti tomu použít všude
∂TI	3	

Omezení předprocesoru jsou :

- max. délka věty vstupních souboru	80 znaků
- max. počet zpracovatelných vět vstupních souborů	32767
- max. počet úrovní hierarchického rozkladu	20
- max. počet vnoření řídících struktur	40

Ze vstupního "prazdrojového" textu vygeneruje XPPP zdrojový text, který je překládán kompilátorem FORTRANu. Je-li ve vstupním textu nalezena chyba, pak výstup není způsobilý k překladu. Je třeba chybu odstranit a generovat znovu. Uveďme příklad jednoduchého programu, řešeného pomocí XPPP :

```
C *** PREVOD CASOVYCH VELICIN ***
C
      PROGRAM CAS
      IMPLICIT INTEGER (A-Y), REAL (Z)
      DATA     H,M,KON /'H ', 'M ', 'K '/
C
30000 FORMAT  ('1',20X,'*** PREVOD CASOVYCH VELICIN ***'/1X)
30001 FORMAT  ('0',PREVOD HOD.MIN --  MIN ... H'/
```

```
                1X,'PREVOD MIN --  HOD.MIN ... M'/
                1X, K O N E C ..................... K ......'/
30002   FORMAT (A1)
30011   FORMAT (1X, 'ZADEJTE HODINY,MINUTY : ')
30012   FORMAT (F7.2)
30013   FORMAT (1X, 'ZADEJTE MINUTY : ')
30014   FORMAT (F7.0)
30015   FORMAT (25X,F7.2,' HOD.MIN = 'F8.0 'MIN ')
30016   FORMAT (18X,F7.0,' MIN = ',F8.2,'HOD.MIN')
30020   FORMAT (1X, '**** KONEC PROGRAMU CAS ****/1X)
C       WRITE (1,30000)
        PERFORM 1000          * VOLBA *
C
        IT
        PART-TI
        IF (VAR.EQ.EON)
           WRITE (1,30020)
        PART-IT
        IF (VAR.EQ.H)
            PERFORM 1001    * HOD.MIN --  MIN *
            PERFORM 1000    * VOLBA *
        PART-IT
        IF (VAR.EQ.M)
            PERFORM 1002    * MIN --  HOD.MIN *
            PERFORM 1000    * VOLBA *
        EI
        STOP
C
C ****  VOLBA ************************************************************
C
         BLOCK 1000
      1 WRITE (1,30001) ; READ (1,30002) VAR ;
        IF (VAR.NE.H.AND.VAR.NE.M.AND.VAR.NE.KON) GO TO 1
         BACK
C
C ****  HOD.MIN --  MIN  *************************************************
C
```

```
      BLOCK 1001
    2 WRITE (1,30011) ; READ (1,30012) ZCASHM ;
      IF (ZCASHM.LT.0) GO TO 2
      ZCASM = ZCASHM*100. - AINT(ZCASHM*40.
      WRITE (1,30015) ZCASHM, ZCASM
      BACK
C
C *** MIN --   HOD.MIN ***********************************
C
      BLOCK 1002
    3 WRITE (1,30013) ; READ (1,30014) ZCASHM  ;
      IF (ZCASM.LT.0) GO TO 3
      ZCASHM = ZCASM/100. + AINT (ZCASM/60.)*0.4
      WRITE (1,30016) ZCASM, ZCASHM
      BACK
C
C *** KONEC PROGRAMOVE JEDNOTKY ***
C
      END
```

Všechny komponenty výše popsané FORTRANské podpory byly programovány pouze ve FORTRANu a jsou k dispozici ve zdrojových tvarech. Fungují pod operačním systémem CP/M, ale bylo by možno je snadno převést i pod jiné systémy, v nichž je FORTRAN implementován.

Shrneme-li vlastnosti, které poskytuje prostředí zlepšeného klasického programování, pak musíme konstatovat, že :

- komfort při práci je pomocí různých prostředků zvýšen. Vývoj směřuje jednak k integraci jednotlivých částí do ucelené ho systému pro vývoj programů a jednak ke standardizaci činností,
- přes podporu moderních metod programování zůstávají základní rysy klasických jazyků zachovány. Z toho plyne, že aplikační programátor musí stále znát specielní informace, které bezprostředně k řešení problému potřebuje.

4. Programování pomocí relační databáze

S rozšířením mikropočítačové techniky je spojen další rozvoj software. Dospělo se k tabulkovým a databázovým procesorům a k integrovaným systémům. I tyto nové prostředky mohou obsahovat značné množství příkazů. Celkově však vytvářejí zcela nové prostředí pro programování. Zkusme je charakterizovat na příkladu relační databáze PAND (československý produkt pro zpracování dat na 8-mi a 16-ti bitových mikropočítačích) :

- relační databáze je opravdu mohutný prostředek. Produktivita práce může být výrazně vyšší. Za čas, potřebný k realizaci klasického programu, lze vyřešit celou úlohu, ekvivalentní soustavě několika programů,
- práce má vysoký komfort (nabídky, okénka, detekce chyb ..),
- to spolu s charakterem jazyka umožňuje velmi omezit specielní znalosti o výpočetním systému a místo toho provést kvalitní analýzu řešeného problému. Profesní rozdělení "uživatel - analytik - programátor" se nepodporuje, ale naopak se stírá,
- je kladen důraz na dobrý návrh datové základny. Změna ve struktuře dat však většinou nemá tak tvrdé důsledky,
- dílčí výsledky lze získat, i když úloha není zdaleka dokončená,
- velmi dobře se realizuje prototypový přístup k řešení. Jsme schopni uživateli rychle nabídnout první prototyp úlohy a pak jej dále ve vzájemné spolupráci rozvíjet,
- sympatickou vlastností je také to, že prostředí databáze se chová jednotně nejen k programátorům navzájem, ale i k uživatelům. To má nedocenitelný význam jak při údržbě úloh, tak i při jejich obsluze (zastupitelnost),
- samotné programování má oproti klasickým jazykům nové rysy :
  - jedná se spíše o popisování nebo deklarování
  - těžiště práce nespočívá v proměnných, polích a návěštích, proměnnou (aniž si to uvědomíme) je např. název údaje, zadaný v popisu struktury věty
  - odpadá otevírání a zavírání souborů, vstupně-výstupní pří-

kazy - to si ošetřuje systém sám
- objevují se zcela nové příkazy, např. k vytvoření nabídky,
- automaticky se ošetřují zvláštní případy a dosazují se implicitní hodnoty
- je zabudován generátor sestav, sort, textový a datový editor apod.

Jestliže dříve platilo, že vedle klasického jazyka je dobré znát assembler, pak zde platí, že pro speciální případy je dobré znát vyšší jazyk. Data se skutečně stávají materiálem, který je zpracováván programovým nástrojem. Zvětšuje se prostor pro analýzu na úkor programování. V návaznosti na ni je třeba promyslet následující okruhy :

A) SOUBORY DAT

a) Architektura souborů
- skladba souborů, jejich obsah a funkce
- struktury vět souborů
- vztahy mezi soubory (hierarchie)
- časové vztahy mezi soubory (perioda jejich zpracování)

b) Převody dat mezi soubory
- operace s daty (pořízení, převod, kumulace, nulování ...) z hlediska věcného (jaká data) a časového
- (kdy) v návaznosti na a)

B) FUNKCE ÚLOHY

a) Vlastní algoritmus úlohy
- zpracování základních dat v návaznosti na A)
- způsob výpočtů (vzorce)
- vstupy (pořízení dat a jejich kontroly)
- výstupy (informace na obrazovku, tiskárnu, soubor)
- dialog s uživatelem (nabídky, zprávy, hlášení chyb). Vše z hlediska věcného (co) a časového (kdy)

b) Algoritmus zajištění úlohy proti výjimečným stavům
- opravy chybových stavů v souborech
- bezpečnostní kopie souborů
- obnova souborů z kopií. Vše z hlediska věcného (co) a časového (kdy)

C) ORGANIZACE ZAJIŠTĚNÍ ÚLOHY

- odpovědnost za zpracování, vstupy a výstupy z hlediska věcného (kdo, co) a časového (kdy)
- dokumentace úlohy pro uživatele (popis obsluhy) /nejlépe samodokumentavatelnost/
- organizační směrnice pro provoz úlohy

Na ukázku uvedeme jednoduchou úlohu v jazyce FAND :

```
{* * * * * * *  JEDNODUCHE ZPRACOVANI TEXTU  * * * * * * *}
$FILE-TEXT                              {Soubor pro ulozeni textu}
   Radek : A,70
$RPRT_TEXT . pagelimit :=60             {Sestava pro tisk textu}
#I_TEXT
#PH page;

                        - ___ -
#DE Radek;
$PROC_MAIN                              {Hlavni procedura}
   """Vitam Vas pri zpracovani textu !"
     "-------------------------------"""
   %delay(3)  p ZACATEK
$PROC_ZACATEK                           {Hlavni nabidka ulohy}
   %Menuloop ("ZPRACOVANI TEXTU"
              "================"
              @ "
              'x Editace textu ': P EDIT   %,
              'x Tisk    textu ': P TISK   %,
              'x Zruseni textu ': P ZRUS   %,
              'x KONEC ulohy   ': P KONEC  %exit %)
$PROC_EDIT
   "EDITACE TEXTU"""
   "Pripojeni novych radku: F2. Vlozeni radku: INS."
   "Zruseni radku: CTRL-Y. Oprava radku: prepsat nebo F5-U."""
   "Piste nebo opravte text, konec=<ESC>"  E TEXT C@
$PROC_TISK
   "TISK TEXTU"""
   "Pozastaveni a spusteni tisku: CE. Predcasne ukonceni: ESC."""
   %IF (empty(TEXT):"Prazdny soubor !" %delay (3) %,
      %IF ( 'Vystup na obrazovku': R TEXT O %)
      %IF ( 'Vystup na tiskarnu ': R TEXT T ? %) %)
```

```
$PROC_ZRUS
   %IF ( 'Zrusit text':
      %IF ('Opravdu': """Text byl zrusen" D A TEXT^[ %,
                      """Priste nelsete !" %delay(3) %) %)
$ProC_KONEC
   "Konec spracovani textu - diky za spolupraci !"
   " ------------------------------------------"""
```

5. Závěr

Snažili jsme se poukázat na problematiku programovacích prostředků z hlediska prostředí, které vytvářejí vzhledem k programátorovi a k uživateli. Konstatujeme, že těchto prostředků existuje mnoho. Proto máme-li pro daný účel přistoupit k výběru některého z nich, měli bychom se nejprve ptát, zda :

- vytváří výkonné a produktivní prostředí pro programátora
- vytváří jednotné a příjemné prostředí pro uživatele
- se ho lze snadno naučit a ovládat
- je výhodný z hlediska zastupitelnosti

  a podobně.

Na tato kritéria by se také měla obracet pozornost při výuce studentů na školách, neboť tam se získávají první trvalejší znalosti z oblasti programování.

Literatura :

/1/ P. Ježek : UFLIB - příručka programátora
/2/ P. Ježek : XPROG - příručka programátora
/3/ P. Ježek : XPPF - příručka programátora
/4/ B. Tichá a kol.: FYND - základní popis

Ing. Pavel Ježek
Státní statek
Karviná