Ing. Jiří Velák
INCOMA Gottwaldov

GENERÁTOR COBOLSKÝCH PROGRAMŮ CMCGEN

1. Úvod

Makrogenerátor CMCGEN vznikl v první polovině roku 1977, když jsme v ústavu připravovali instalaci počítače Siemens 4004/151. Hledali jsme tehdy prostředek racionalizace programování, který by pomohl
- snížit pracnost programování včetně údržby programů
- podpořit moderní programovací metody
- sjednotit programování.

Dodavatelská firma sice nabízela generátory programů podle jednotlivých metod (APG - normalizované programování, ET - rozhodovací tabulky, COLUMBUS - strukturované programování), jenže se jednalo o prostředky uzavřené, navzájem izolované, málo účinné, složité co do používání a drahé co do placení.

Za této situace jsme se pokusili pomoci si sami. Podobný úkol lze splnit v rozumném termínu pouze tehdy, využijeme-li v maximální míře toho, co před námi naprogramovali jiní; v našem případě to byl překladač ASSEMBLER a jeho makrogenerátor.

2. Princip generátoru

Uložme do knihovny makroinstrukcí překladače ASSEMBLER následující makroinstrukci:

```
MACRO
&ID      &PROG,&AUTHOR,&DATE
PUNCH '            ID DIVISION.'
PUNCH '            PROGRAM-ID.&PROG.. '          (1)
PUNCH '            AUTHOR.&AUTHOR.. '
PUNCH '            DATE-WRITTEN.&DATE.. '
PUNCH '            DATE-COMPILED.TODAY. '
PUNCH '            REMARKS. '
MEND
```

Dále poskytněme překladači ASSEMBLER takovýto podivuhodný program

```
&ID      TEST,JUNGWIRT,13.1.79
REPRO
ENVIRONMENT DIVISION.
REPRO                                            (2)
DATA DIVISION.
REPRO
PROCEDURE DIVISION.
REPRO
   STOP RUN.
```

Překladač nám vydĕruje následující "objektový modul":

```
ID DIVISION.
PROGRAM-ID.TEST.
AUTHOR.JUNGWIRT.
DATE-WRITTEN.13.1.79                             (3)
DATE-COMPILED TODAY.
REMARKS.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION.
   STOP RUN.
```

Což je, jak doufám, formálně i logicky správný program v programovacím jazyce COBOL.

Dále přinějeme překladač COBOL, aby takto vzniklý modul zpracoval. V naší instalaci se to děje přes virtuální děrovač a snímač děrných štítků (LSPOOL2). Modul se tedy fyzicky neděruje.

Do generační procedury CMCGEN (příloha I) je zařazen ještě pomocný program EDITOR (příloha I), jehož úkolem je tisk kompilačního výtisku v přehledném formátu (bloková struktura, stránkování, potlačení tisku apod.) a odfiltrování některých "nečistot" z překladu ASSEMBLER (ESD,TXT, END, REND - štítky). U překladače ASSEMBLER je tisk kompilačního výtisku potlačen, tisknou se jen zprávy o chybách.

Vraťme se ještě k programu (2). Štítky REPRO slouží k tomu, aby překladač ASSEMBLER ponechal beze změny původní cobolské výroky. Zřejmě i REPRO - štítky lze generovat, dokážeme-li ve zdrojovém programu rozlišit volání makra od cobolského výrazu. Pak stačí jednoduchý pomocný program (REPROL - příloha I), který provede "předzpracování" zdrojového programu pro překladač ASSEMBLER. Program (2) se nám teď zredukoval takto:

```
ZID     TEST,JUNGWIRT,13.1.79
ENVIRONMENT DIVISION.
DATA DIVISION.                              (4)
PROCEDURE DIVISION.
    STOP RUN.
```

Ve srovnání s programem (3) jsme ušetřili právě 50 % výroků. Jaký tento poměr bude u reálných programů, záleží především na nás.

Popsaným postupem jsme získali nový vyjadřovací prostředek. V programu můžeme standardní části generovat makrotechnikou, zbytek napíšeme v COBOLu. V našem prostředku se sčítají možnosti makrogenerátoru ASSEMBLER a COBOLu.

3. Další možnosti

Makrogenerátor překladače ASSEMBLER moderního počítače má celou řadu možností, které by bylo škoda nevyužít. Jsou to např. lokální a globální proměnné, indexované proměnné, možnost větvení a smyček (AIF), poziční a klíčové parametry, možnost volání maker na více úrovních, práce se znakovými řetězci.

Příklad použití smyčky pro generaci části popisky věty:*

```
       MACRO
       &MAC    &LEVEL,&NAME,&LIMIT,&PIC
       LCLA    &A                                    (5)
.BACK  ANOP
&A     SET     &A+1
       PUNCH '       &LEVEL &NAME.&A  PIC &PIC..'
       AIF     (&A LT &LIMIT).BACK
       MEND
```

Při vyvolání makra

```
       &MAC    05,POLE,50,X(6)
```

se generuje:

```
       05 POLE1   PIC x(6).
       05 POLE2   PIC x(6).                          (6)
          .
          .
          .
       05 POLE50  PIC x(6).
```

Výhodné použití globální proměnné ukážeme na příkladě dogenerování chybějících částí programu:

```
       MACRO
       &DD
```

* Syntaxe makrojazyka je převzata z [1].

```
         GBLC   &DIVI
         PUNCH '        DATA DIVISION. '          (7)
&DIVI    SET    'D'
         MEND
         MACRO
         $FS
         GBLC   &DIVI
         AIF    ('&DIVI'  EQ  'D').OK
         $DD
.OK      PUNCH '        FILE SECTION. '
         MEND
```

Ve zdrojovém programu pak píšeme

```
         $DD
         $FS
```

nebo jen

```
         $FS
```

V obou případech se generuje

```
         DATA DIVISION.
         FILE SECTION.
```

Podle tohoto principu lze propojit makra přes celý program. Ve zdrojovém programu pak píšeme jen to, co opravdu potřebujeme, ovšem ve správném pořadí, jak to vyžaduje syntax cílového jazyka.

4. Píšeme generátor zdrojových programů

Nejprve si vyjasníme funkce, které bude generátor pokrývat. Čím lépe budeme znát konečnou podobu generátoru, tím lépe a radostněji se nám bude tentýž vytvářet. Vyjdeme ze syntaxe cílového jazyka a z osvědčených programovacích metod. Budeme postupovat velmi pragmaticky: z těchto metod vyloupíme to, co je v nich dobré a pro nás výhodné. Tyto

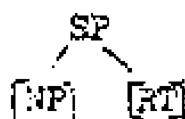
metody si nejprve uvedeme do správného vzájemného vztahu, abychom je mohli podle potřeby kombinovat. Tato otázka má klíčový význam koncepční i metodický. Pokusíme se o charakteristiku tří programovacích metod – strukturovaného programování, normalizovaného programování a rozhodovacích tabulek.

Strukturované programování je obecná programovací metoda. Definuje konečný počet struktur, jejichž kombinací lze realizovat libovolný algoritmus.

Normalizované programování je metoda, umožňující pohodlné programování určité třídy úloh, pracujících na sekvenčních souborech. Normalizované programování úspěšně vyřešilo problematiku řízení sekvenčního vstupu.

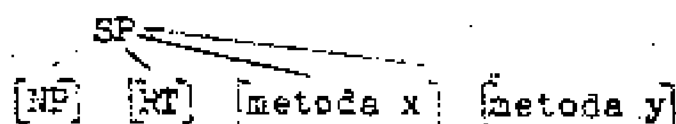
Rozhodovací tabulka pomáhá řešit složité rozhodovací situace. Podobně jako normalizované programování je vhodná pro realizaci určité třídy algoritmů. Nelze ji použít tam, kde se o ničem nerozhoduje.

Z vyjmenovaných metod pouze strukturované programování má charakter obecné metody. Proto jej budeme chápat jako prostředek hlavní, ostatní metody jako pomocné:



Kterýkoliv z pomocných prostředků může chybět. Zcela primitivní programy vystačí s jediným strukturním blokem (příloha III a IV). Blok sekvenčního vstupu a rozhodovací tabulka budou pro nás představovat jednoduché bloky typu BEGIN-END (příloha V.).

Nyní můžeme bez obtíží vytvořit jakoukoliv programovou strukturu a nebýt při tom ochuzeni o výhody specializovaných metod, tedy i takových, které ještě dnes neznáme:



Budeme tedy vytvářet trvale otevřený systém. Zvolený prostředek, tj. makrogenerátor překladače ASSEMBLER je pro to velmi vhodným pomocníkem.

4.1 Definice funkcí generátoru

Pokusíme se nyní definovat funkce našeho generátoru a v dalším textu si ukážeme, jak se tyto funkce dají realizovat:

- generování syntakticky povinných částí programu (názvy divizí, sekcí ...)
- generování struktur podle Nassi-Shneidermannových diagramů
- řízení sekvenčního vstupu (normalizované programování)
- rozhodovací tabulky
- generování popisek souborů
- ostatní uživatelská makra

4.2 Syntakticky povinné části programu:

Postup byl již naznačen v kapitole 2. Pro názvy maker zvolíme, pokud možno, počáteční písmena názvu divize, sekce, popisu (ZWSS = WORKING STORAGE SECTION, ZPD = PROCEDURA DIVISION atd.). Makra šikovně zřetězíme, aby v cílovém programu nic nechybělo (kap. 3.). Vhodným parametrem však zajistíme možnost vypnutí resp. zapnutí této "autogenerační" funkce v libovolném místě programu.

U některých maker naprogramujeme speciální funkce. Např. u makra ZFD (File Description) budeme požadovat dodatečné vlastnosti:

- objeví-li se volání makra ZFD ve WORKING-STORAGE SECITON, vynecháme popis FD
- je-li první písmeno názvu souboru "S", generujeme popis SD

Tato vlastnost je výhodná při vytváření knihovních popisek souborů (viz dále). Dosáhneme tím toho, že jedné a téže popisky souboru použijeme pro deklaraci FD, SD nebo jen holé popisky věty ve WORKING-STORAGE SECTION (viz příloha 2.), aniž bychom se o něco museli starat.

Použité globální proměnné si zapíšeme do zvláštního seznamu. Budeme jich potřebovat více a bylo by smutné, kdyby se nám křížily.

4.3 Struktury podle Nassi-Shneidermannova diagramu

Prostudujeme základní strukturní bloky, definované ve strukturním programování (sekvenční blok, smyčka, rozhodování, "CASE"). Napíšeme a uložíme do knihovny příslušná makra; doporučené názvy:

ZBEGI - začátek bloku
ZBEND - konec bloku
ZCASE - rozhodování s více výstupy
ZOF - výstup z ZCASE
ZWHIL - smyčka s testem na začátku
ZWHEN - výskok ze smyčky (varianta s testem kdekoliv)
ZCYCL - začátek smyčky s testem kdekoliv
ZIF - podmínka

Uvedená makra budou generovat počáteční resp. závěrečné výroky v bloku, reference, testy apod. Uvnitř maker musíme pomocí vhodných globálních proměnných sledovat, v kolikáté úrovni struktury se nalézáme a o jaký blok se jedná. Na začátku a konci bloků můžeme generovat příslušné řídící znaky (štítky) pro program EDITOR, který nám zlepší čitelnost programu.

Eventuelní zprávy o chybách tiskneme pomocí MNOTE, nebo ještě lépe jako poznámku do cílového jazyka. Časem totiž zjistíme, že protokol z překladu ASSEMBLER je nadbytečný a že pro programátora je důležitý jediný kompilač-

ní výtisk se všemi hláškami. Je to přehlednější a stručnější.

4.4 Řízení sekvenčního vstupu

Tuto část převezmeme z normalizovaného programování. Půjde nám pouze o bloky B, C, D (vstup. výběr věty, testování klíčů). V podstatě vystačíme se dvěma makry:

ŞKEYS - definice řídících oblastí ve WORKING-STORACE
- otevření sekvenčního souboru při prvním průchodu

ŞIPT - čtení sekvenčního souboru
- přesun klíčů do řídících oblastí
- výběr věty pro zpracování
- testování změn klíčů
- test na konec posledního souboru
  uzavření vstup. souboru

Skupina maker ŞIPT pak bude tvořit běžný blok ve struktuře programy (BEGIN-END) - viz příloha V.

4.5 Rozhodovací tabulky

Naprogramovat makra pro rozhodovací tabulky je nejsložitější. Dá to asi tolik práce, co všechny ostatní části generátoru dohromady. Neobejdeme se bez indexových globálních proměnných a musíme počítat s tím, že generace rozhodovací tabulky spolkne znatelně více strojového času, než jiné programy. Z těchto důvodů je vhodné udělat rozhodovací tabulky až nakonec a to za předpokladu, že máme dostatečné publikum programátorů, kteří je budou používat.

V naší verzi jsou k dispozici makra:

ŞCOND - definice podmínek
ŞACT - definice akcí (podprogramů)
ŞRULE - definice pravidel
ŞELSE - pravidlo pro nedefinované kombinace, generace celého algoritmu

Příklad použití RT - příloha V.

4.6 Definice součtové sestavy po sloupcích

Tato skupina maker je vázána na používání části REPORT-WRITER. Je vhodná pro sestavy s více úrovní součtů. Umožňuje stručný a pohodlný popis textové sestavy a snadné provádění oprav ("roztažení", "sražení" sestavy, vložení dalšího sloupce kamkoliv, vložení další úrovně součtů). Pozice sloupce se udává vždy relativně k pozici sloupce předchozího, resp. k levému okraji sestavy.

Makra, používaná v naší verzi:

ZVERT - popis jednoho sloupce tiskové sestavy včetně nadpisů sloupce

ZRG - signál k vygenerování příslušné tiskové skupiny (řádku)

ZRPAR - zadání parametru pro opravu distance mezi sloupci, horizontální posun celé sestavy

Příklad použití - příloha IV.

4.7 Generování popisek souborů

Popisky souborů si budou psát a ukládat do knihovny programátoři sami. My jim k tomu poskytneme jen základní aparát (makro ZFD, viz dříve) a rady do života:

a) Vysvětlíme jim syntaktická pravidla knihovní makroinstrukce, tj. přednesame jim vybrané kapitoly z makrojazyka ASSEMBLER, popíšeme instrukci PUNCH. Znalost ASSEMBLERu jako takového není nutná.

b) Najdeme vhodnou systematiku označování maker, souborů, datových vět a položek a budeme dbát, aby ji programátoři ve vlastním zájmu dodržovali. Zde by měl stačit odkaz na příslušné metodické pokyny pro programování.

4.8 Ostatní uživatelská makra

To, co bylo řešeno v předchozím odstavci, platí přirozeně obecně. Kterýkoliv programátor má možnost kdykoliv cokoliv řešit vlastním makrem (např. situace toho typu, jak je popsána na začátku kap. 3). Popsaný případ se vyplatí už při prvním použití - makro je velmi jednoduché. Pro tento účel stačí definovat makro pouze uvnitř zdrojového programu - přesně tak, jak co děláte v assembleru. Obecnější případy, přesahující rámce jediného programu ukládáme do knihovny k dispozici ostatním. Celá věc má jistě své organizační pozadí, ale to je jiná věc. Podstatné je toto: jakákoliv opakovaná činnost (podobná definice dat, programátorský obrat atd.) budiž nám signálem k tomu, abychom se pokusili ji zobecnit a zakódovat ve formě nové makroinstrukce - vyplatí se to. To by měla být součást výchovy programátorů.

5. Některé vlastnosti generátoru

Nyní vystoupíme do vyšší letové hladiny, jak říkají aviatici. Naučíme se generátoru používat a přitom budeme studovat jeho vlastnosti.

První, co nás udeří do očí je fakt, že naše programy budou mít velmi členitou hierarchickou strukturu - čistě formálně vzato. Je to tím, že jsme šli na věc od lesa, tj. od nejnižších struktur. Tak jsme získali základní prefabrikáty pro konstrukci prefabrikátů vyšších úrovní. To je podstata metody. Pro ilustraci jsem vybral poněkud extrémní příklad (příloha III): Program vybírá ze vstupního souboru věty určitého typu a třídí je podle zadaných klíčů. Zdrojový program sestává z jediného štítku, zadává se název programu, typ věty a názvy třídících klíčů. Všimněte si ještě konstrukce popisky věty (příloha II) a opakovaného použití této popisky v programu (příloha III). Gene-

ruje se vždy jen požadovaná struktura.

Druhý příklad (příloha IV) je tiskový program, zpracovávající data z předchozího programu. Je to prostá součtová sestava, definovaná po sloupcích (kap. 4). Tento příklad jsem zařadil pro ilustraci následujícího tvrzení:

Jakmile se nám podaří přimět programátora, aby přesunul těžiště programu do deklarační části,* můžeme mu poskytnout mnohem vydatnější pomoc, neboť

a) deklarace se velmi snadno generují
b) formální správnost deklarací je většinou též zárukou jejich správné logiky.

Zbývá nám ještě provést úvahy o procedurní části programu. Všimněme si nejprve jedné společné vlastnosti metod, ze kterých jsme vycházeli (SP, NP, RT): všechny tyto metody (a jistě i další, na které teprve čekáme) představují ústup do procedurálního vyjadřování (jak) k funkčnímu (co). Výhody tohoto trendu jsou zřejmé z předchozího odstavce, který zobecníme: <u>Důsledné uplatnění funkčního přístupu vede k tomu, že formálně správná procedura bude většinou i logicky správná.</u>

6. <u>Za vším hledej člověka!</u>

Ještě si řekneme něco o vztahu programátora k celé záležitosti.

a) Programátor nesmí mít pocit, že je na něm pácháno násilí. Pakliže v něm řečený pocit vznikne, staví se nám na zadní. V našem ústavu není a zřejmě ani nebude používání generátoru povinné.** Přesto (nebo právě proto) se jeho používání vžilo a dnes se jím generují prakticky všechny uživatelské programy.

* deklarační části zde rozuměj všechno mezi ID DIVISION a PR[illegible]URE DIVISION.

** [illegible]vinné je ovšem dodržování určitých formálních zásad, da[illegible]ných Metodikou prováděcích projektů.

b) Programátor by měl podobný prostředek dostat včas, tj. dříve, než si vyšlape vlastní cestičky. V našem případě byl generátor dán k dispozici 3 týdny po oživení počítače, v době kdy se programátoři učili COBOL.

c) Programátor potřebuje udici a ne rybu. Měl by mít trvalý pocit spoluautorství generátoru. V tomto směru se nekladou meze (kap. 4), resp. se to nepřímo vyžaduje (popisky, uživatelská makra).

7. Závěr

Vytváření generátoru jde velmi rychle od ruky. Chyby v makrech se snadno hledají pouhým okem, případně pomocí MTRAC. Pracujeme totiž na sémantické úrovni.

Míra nezávislosti maker je vysoká. Opravy a dodatky lze při troše pečlivosti provádět přímo do rutinní makroknihovny.

V dokumentační části je třeba položit důraz na dostatek příkladů použití generátoru pro všechny základní typy programů. Programátoři si pak nebudou vymýšlet vlastní bizarní struktury, což je u strukturovaného programování určité riziko.

Generátor je snadno přenositelný na jiný typ počítače (IBM, JSEP, M-4030). Některé odlišnosti v syntaxi cílového jazyka je možno překlenout právě na úrovni makrojazyka.

8. Literatura

1. Betriebsystem BS-1000 P-Assembler, Siemens AG, červen 1975
2. COLUMBUS (COBOL) System Unterstützung der Strukturierten Programmierung (BS-1000, BS-2000), Siemens AG, květen 1976
3. COHTET, Cobol - orientierter Entscheidngstabellen-Vorübersetzer (BS-1000, BS-2000), Siemens AG, únor 1976
4. APG - Allgemeiner Programm-Generator, Siemens AG, květen 1976
5. Generátor zdrojových programů ZGENSRC, Informace č. 93, ZVT OKD Ostrava