Petr Podešva
VAKUS Praha

POKUS O STRUKTURALIZACI NORMOVANÉHO PROGRAMU

1. Normované a strukturované programování

Metody normovaného a strukturovaného programování patří mezi nejznámější metody programování. Obě tyto metody si všímají různých aspektů programátorské práce. První si všímá toho, že programy určité, dost široké třídy mají podobnou strukturu - t.j. že se v nich často opakují stejné funkce v určitém pevném pořadí. Normované programování specifikuje obecnou strukturu programu, zpřesněným opakujícím se funkcím vykazuje v programu pevné místo a tím zbavuje programátora řady rutinních činností, neboť mu nabízí standardní postup. Pravidla normovaného programování jsou značně konkrétní a mají povahu normy. Přínos normovaného programování nekončí ovšem usnadněním programátorské práce - minimálně stejně důležitá je jednotnost a přehlednost takto vzniklých programů a tedy i programové dokumentace.

Strukturované programování nemá tuto normativní povahu. Jde spíše o soubor doporučení a zásad, jak psát programy tak, aby byly přehledné a snadno modifikovatelné. Je ponecháváno na programátorovi, jak se bude těmito zásadami řídit v praxi. Také původ strukturovaného programování je jiný - vzniklo v době, kdy se ukázalo, že složitost programu vzrůstá mnohem rychleji než složitost výchozího problému a lidské schopnosti při produkci a analyze složitých programů jsou omezeny. Strukturované programování poskytlo prostředky, jak myšlenkovou náročnost složitých

programů podstatně snížit.

2. Sloučení obou přístupů.

Obě tyto techniky mají na první pohled málo styčných bodů. Každá sama o sobě však představuje velký přínos. Při naší práci na projektu JEP a MZDY pro resort spojů nás napadlo, zda by nebylo možné nalézt vhodný formální aparát, který by dovoloval využít obě tyto techniky s maximálním efektem.

Projekt JEP a MZDY se pro tyto účely dobře hodil. Jde o projekt značně složitý - jeho složitost je do značné míry určena složitostí čs. pracovního práva. Jen program, který řeší měsíční zúčtování mezd má 4500 řádků v COBOLu.

Zárodek vhodného formálního aparátu byl po ruce - nabízí jej normované programování. Tato metoda dělí program na samostatné bloky: A, B, C, D, E atd. a tím poskytuje programátorovi prostředek, jak rozdělit program na samostatné funkce. V blocích A - C a F je vyřešeno s minimálním programátorovým příspěním poskytnutí věty ke zpracování. Její vlastní zpracování se programuje v blocích E a H. V případě složitého programu je tedy blok E a H nejrozsáhlejší - ostatní bloky tvoří nepoměrně menší část programu. Zde přínos normovaného programování končí - je na programátorovi, aby svůj problém vyřešil v bloku E případně H bez pomoci jakéhokoli standardu.

Vzhledem ke složitosti úlohy se nabízí použít metody strukturovaného programování. Jak bylo shora řečeno, jde o soubor různých doporučení a zásad. Jedna z nejdůležitějších zásad je zásada programování shora dolů. Tato zásada člení problém na různé úrovně abstrakce. Každá úroveň představuje virtuální počítač určité mohutnosti. Programátor vyřeší svůj problém na určité úrovni abstrakce a to prostředky, které mu tato úroveň poskytuje. Pokud tyto prostředky jsou pouze prostředky zvoleného programovacího jazyka, je problém vyřešen. Pokud ne, musí progra-

mátor definoval další, nižší úroveň abstrakce, její vlastní prostředky a jejich prostřednictvím vyřešit prostředky předcházející vyšší úrovně abstrakce. Programuje tím další virtuální počítač menší složitosti. Stanovení jednotlivých úrovní (tj. prostředků, kterými lze disponovat) záleží na složitosti problému - strukturované programování se tím hlouběji nezabývá. Prakticky řešení problému na určité úrovni abstrakce je volání podprogramů a tedy prostředky jsou podprogramy. Protože často nastává situace, že určitý prostředek je volán z různých úrovní, dovoluje strukturované programování, aby takový prostředek mohl být uveden pouze na nejnižší úrovni, kde je zapotřebí, a aby nadřazené úrovně k němu dovolovaly přístup. V opačném případě by musel být uveden všude tam, kde je použit. O úrovni, která sama neobsahuje určitý prostředek, ale která dovoluje, aby tento prostředek byl v nějaké jí nadřazené úrovni použit, mluvíme jako o úrovni průhledné (z hlediska zmíněného prostředku).

Možnost, aby kterákoliv úroveň byla průhledná, dovoluje, aby mezi prostředky každé úrovně patřily všechny prostředky zvoleného programovacího jazyka. Programování na každé úrovni abstrakce se tedy skládá z používání elementárních řídících struktur programovacího jazyka (další zásada strukturovaného programování) a volání podprogramů vyřešených na nižších úrovních. Někdy je z hlediska programování vhodnější mluvit o jednotlivých úrovních abstrakce jako o vrstvách. Podprogramy v každé vrstvě mohou být uspořádány v pořadí, ve kterém jsou vyvolávány (pokud je to ovšem možné).

Je samozřejmý požadavek, aby program v každé vrstvě byl fyzicky souvislý. Každou vrstvu je vhodné označit - může tvořit blok ve smyslu normovaného programování. Nejvyšší vrstva bude tvořit blok pro zpracování věty, tedy blok E. Další vrstva může být blok H - to ještě odpovídá

duchu normovaného programování. Ostatním vrstvám je možné přiřadit další písmena abecedy, tj. I, J, K atd. Pokud normované programování některých těchto písmen používá, např. písmene I, je jej v programu možné zaměnit jiným písmenem. Protože jednotlivé vrstvy jsou jistým způsobem uspořádány: - uspořádání je definováno vztahem 'je podřízena' - je vhodné zachovat toto uspořádání i v programu, tj. abecední uspořádání jednotlivých bloků. Potom vrstva K (tj. blok K) je podřízena vrstvě J a nadřízena vrstvě L.

Program se tedy dělí na jednotlivé bloky. Pokud je blokem vrstva z hlediska strukturovaného programování, rozpadá se tato vrstva na jednotlivé podprogramy. Výjimku tvoří nejvyšší a nejnižší vrstva. Nejvyšší vrstva - vrstva E - obsahuje pouze volání podprogramů z ostatních vrstev. V nejnižší vrstvě už musí být všechny podprogramy vyřešeny.

3. Moduly.

Dělení bloku (vrstvy) na podprogramy je dělení syntaktické. Z funkčního hlediska je zřejmé, že některé podprogramy budou spolu do určité míry souviset - např. mají společná data, vyvolávají se v určité časové posloupnosti, při jisté příležitosti, stejným podprogramem atd. Takovou skupinu podprogramů, které mají více společného mezi sebou než se zbytkem programu nazveme modul (jsme si ovšem vědomi toho, že se tento termín používá i v jiném smyslu). Tím se program rozpadá na řadu modulů. Podle toho, jak zřetelné jsou dělící čáry mezi jednotlivými moduly můžeme chápat každý modul jako samostatný program.

Dělení programu na jednotlivé moduly je velmi užitečné. Konkrétně v programu pro měsíční zúčtování mezd je takové dělení nasnadě - neboť zde lze poměrně lehce jednotlivé funkce programu od sebe odlišit. Tak např.

modul DOVOLENÁ realizuje všechny činnosti, které mají vliv na čerpání dovolené. Jde o tyto funkce: stanovení národku na dovolenou při navázání pracovního poměru, krácení dovolené pro nepřítomnost v práci, náhrada mzdy za čerpanou dovolenou, kontrola splnění nároku na dovolenou k 31. 12., proplacení nevybrané dovolené k 30. 4. atd. Všechny tyto funkce, z nichž každá se realizuje jedním, nebo několika programy, mají společná některá data, i když se volají při různých příležitostech: jednou při nástupu pracovníka, podruhé při rozvázání pracovního poměru, jindy při zpracování odchylky - čerpání dovolené. Podprogramy tohoto modulu se objevují v mnoha vrstvách a v některých vrstvách jich může být více.

Protože jednotlivé podprogramy konkrétního modulu jsou rozmístěny v různých vrstvách programu a tedy spolu fyzicky nesouvisí, je vhodné je nějakým shodným způsobem označit. To je docíleno tím, že identifikátor podprogramu obsahuje jak označení bloku (vrstvy), tak i označení modulu. Toto označení modulu (např. modul DOVOLENÁ - písmeno DOV, modul NEMOCENSKÉ - písmeno N, modul DAŇ ZE MZDY - DAN, modul SRÁŽKY - SRAZKY, modul PRŮMĚRNÉ VÝDĚLKY - PRUM atd.) je použito též k označení pracovních dat modulu ve WORKINGSTORAGE SECTION.

4. Konvence zápisu podprogramů.

Ve strukturovaném programování je zásada jednoho vstupu do podprogramu a jednoho výstupu. Tuto zásadu lze v COBOLU velmi snadno dodržet. Vstup do podprogramu je přes jeho fyzický začátek a výstup přes jeho fyzický konec. Z hlediska COBOLU tvoří podprogram vždy jednu sekci. To má několik výhod. Volání podprogramu v tomto případě obsahuje jenom jedno jméno procedury v příkazu PERFORM. Dále tím, že COBOL dovoluje členit sekci na paragrafy, je možné členit podprogram na jednodušší části.

Tolik diskutovaný příkaz GO TO lze sice i v COBOLU postrádat, ale někdy jen za cenu značných komplikací. Horké hlavy ortodoxních odpůrců příkazu GO TO již vychladly do té míry, že je možné jej v určité míře - jen v rámci podprogramu - používat. I v tomto případě jen ve smyslu skoku vpřed, ale to bohatě dostačuje. Při tomto omezeném použití neporuší čitelnost programu.

Program tedy tvoří sekci obsahující jeden nebo několik paragrafů. Jména jednotlivých paragrafů jsou odvozena od jména sekce, které jsou součástí. První sekce obsahuje alespoň jeden paragraf, potom poslední paragraf sekce obsahuje pouze příkaz EXIT.

Jméno podprogramu se skládá ze tří částí:

1) označení bloku, do kterého patří
2) označení modulu, kterého je součástí
3) slovního popisu jeho funkce

Další paragrafy sekce obsahují navíc další symbol, kterým se mezi sebou liší. Symbolem je buď mnemotechnická zkratka nebo pořadové číslo. Poslední paragraf obsahuje znak E (EXIT). Příklad:

```
I-DAN-ZE-MZDY  SECTION.
      ......
      ......
      ......
I-DAN-ZE-MZDY-1.
      ......
      ......
      ......
I-DAN-ZE-MZDY-E.
      EXIT.
```

Každý podprogram je relativně samostatný celek. Je určen svou funkcí, svými vstupy a výstupy. To je jediné, co je třeba znát na každé vrstvě, kde se používá - jen to a nic víc. To je opět jeden z projevů principu abstrakce: uživatele podprogramu zajímá pouze to, co program dělá, ale ne to, jak to dělá.

Protože COBOL nedovoluje předávání aktuálních parametrů podprogramu, které značně zlepšuje jeho srozumitelnost, ale vyžaduje předání parametrů na pevné místo v paměti a podobným způsobem volaný podprogram předává volajícímu výsledky, stává se program méně přehledný. Přehlednost lze zvýšit umístěním vstupních a výstupních údajů do deklarace společných dat modulu viz příklad popisu dat modulu DAŇ ZE MZDY, kde pole W-D-KATEG o 10 prvcích je vstup do podprogramu. Dále je možné za jménem podprogramu (tj. sekce) uvést formou komentáře funkci podprogramu, jeho vstupy a výstupy. Příklad:

```
J-DAN-DANOVE-SKUPINY SECTION.
*    VYPOCET ZAKLADNI A NAHRADNI DANOVE SKUPINY PRO
     DANY MESIC
*    VSTUP: E-VYZ-OSOBY, E-VYZ-OSOBY-INV, G-ZM-VYZ-OSOB
*    VYSTUP: W-D-DAN-SKUP-ZAKL,W-D-DAN-SKUP-NAHR
     .....
     .....
     .....
     .....
```

Popis funkce podprogramu je u většiny podprogramů zbytečný - vyplývá z názvu podprogramu, pokud je tento vhodně zvolen. Je zapotřebí pouze u složitějších podprogramů.

Rozsah podprogramu je omezen tak, aby byl snadno přehledný a srozumitelný. Pokud tyto podmínky nesplňuje, rozpadne se ve smyslu strukturovaného programování do více podprogramů. Z tohoto výkladu je zřejmé, že podprogram takto pojímaný se blíží pomnu modul ve smyslu modulárního programování.

5. Příklad členění podprogramů modulu DAŇ ZE MZDY

Tento modul se vyvolává z modulu SRÁŽKY - konkrétně z podprogramu H-SRAZKY (daň ze mzdy je první zákonná srážka). Obsahuje 20 podprogramů. Podstatná část je zachycena na následujícím obrázku. Je zde ukázáno rozložení podprogramů do jednotlivých vrstev. Obrázek ovšem nepopisuje vazby mezi jednotlivými podprogramy, tj. které podprogramy

jsou z konkrétního podprogramu volány atd. Tato informace by byla nejzajímavější, ale je nesnadné ji na tomto formátu zachytit. Šipka určuje pouze pořadí jednotlivých vrstev.

```
        ↓
    H-SRAZKY
        ↓
    J-DAN-DANOVE-SKUPINY
    J-DAN-HRUBY-PRIJEM
    J-DAN-ZDANENY-CASTI-MEZD
        ↓
    K-DAN-DANOVA-SKUPINA
    K-DAN-ODMENY-ZA-HVT
    K-DAN-ODMENY-ZA-DOB-SMENY
    K-DAN-ODMENY-2000
    K-DAN-PODILY
    K-DAN-Z-JEDNORAZ-MZDY
        ↓
    L-DAN-PODLE-RUC-TAB
    L-DAN-Z-JEDNORAZ-MZDY
    L-DAN-GENERACE-MS
        ↓
    M-DAN-PODLE-MES-TAB
```

6. Popis dat.

Myšlenku strukturalizace lze též využít při popisu dat. Tak např. data modulu DAŇ ZE MZDY mohou být popsána takto (jen část):

```
01   W-DAN.
   02  W-D-KATEGORIE-PRIJMU-VSTUP
     03  W-D-KATEG                    OCCURS 10 ...
   02  W-D-HLAVNI-VSTUPY.
     03  W-D-HRUBY-PRIJEM
     03  W-D-DAN-CELK
     03  W-D-DAN-RUCNI
   02  W-D-POM-VSTUPY.
     03  W-D-DRUHY-PRIJMU
         04  W-D-MZDA-ZDAN-MES
         04  W-D-MZDA-ZDAN-JINAK
         04  W-D-MEZDAN-PRIJMY
```

Všechny pracovní údaje modulu DAŇ ZE MZDY jsou tedy identifikovány stejným prefixem W-D- (W jako WORKING STORAGE). Podobně data modulu NEMOCENSKÉ mají prefix W-N- atd. I pracovní data jsou strukturována - viz např. údaje W-D-HLAVNI-VSTUPY a W-D-POM-VSTUPY. Jejich deklarace nemá v programu jiný smysl, než klasifikaci podřízených položek - v programu nejsou citovány v žádném příkazu. Tato strukturalizace je užitečná, neboť i na úrovni dat dovoluje postižení základních funkcí modulu a má tedy určitou dokumentární hodnotu.

Všechna data používaná modulem nelze ovšem tak jednoduše shrnout do jedné struktury (zde rubriky skupinového údaje na úrovni 01). Řada údajů, které modul používá jsou použity i v jiných modulech a není vždy snadné rozhodnout, ve kterém modulu mají větší důležitost. Je rovněž třeba si uvědomit, že COBOL, i když poskytuje programátorovi ve způsobu členění a representace dat značně mocný prostředek, neposkytuje mu to, co jej při vzrůstající složitosti programů začíná dím dál tím víc zajímat - různé stupně lokality dat (např. viditelnosti) a ochranu dat. Určitého stupně lokálnosti dat lze v COBOLU dosáhnout rozdělením programu na několik jednodušších, komplikovaných odděleně.

7. Závěr.

Sloučení obou přístupů se ukázalo jako velmi výhodné. Přispělo jak k zjednodušení programování, tak i k zvýšení přehlednosti programu a dokumentace. Tím, že poskytuje formální aparát, nutí programátora k dodržování určité disciplíny mnohem více, než kdyby strukturalizace programu byla ponechána jen na něm formou obecných zásad a doporučení.

Toto sloučení rovněž usnadnilo přechod od technického projektu k prováděcímu, neboť strukturované programování přímo navazuje na metodu HIPO použitou při tvorbě

technického projektu. Jde vlastně o použití stejných zásad jak pro programování, tak pro dokumentaci. V tomto případě odpovídá každému bodu HIPO jeden podprogram tak úzce, že slovní popis v HIPO lze použít jako komentář k podprogramu.