Ing.Petr VESELÝ
FVT Brno

METODA "LOGICAL CONSTRUCTION OF PROGRAMS" PRO KONSTRUKCI PROJEKTU / PROGRAMU

0. ÚVOD

Definice : Informatika je věda o systematickém a racionálním zpracování těch informací, v jejich ekonomickém a sociálním kontextu, které zvyšují lidské znalosti a usnadňují komunikaci mezi lidmi. Zpracování se provádí obzvláště pomocí počítačů.

ZJEDNODUŠENÍ : Řešení (zpracování) kteréhokoliv informačního problému v sobě zahrnuje tři základní etapy. Činnosti, které v těchto etapách probíhají se vzájemně prolínají tak, že nelze jednotlivé etapy od sebe přesně oddělit. Etapy si nazvěme takto :

- analýza problému
- vlastní řešení
- tvorba dokumentace

AXIOM : Pro realizaci vlastní řešení mnoha informačních problémů je počítač svojí mocností tím nejvhodnějším prostředkem, pro většinu ne ale nezbytným. Pokud se už počítač pro realizaci vlastního řešení použije, slouží pouze jako technický prostředek.

Je možno říci, že činnosti, které probíhají ve třech etapách (viz ZJEDNODUŠENÍ) je vhodné provádět tak, aby etapy navzájem prolínaly. Při analýze problému a vlastním řešení (vlastní řešení zatím bez ohledu na konkrétní počítač) je možno použít týchž logických a formálních postupů, týchž dokumentač-

ních prostředků. V průběhu takové práce vzniká současně dokumentace srozumitelná uživateli i řešiteli, dokumentace logicky vznikající při řešení problému se snižující se mírou abstrakce při každém kroku. Každá úroveň řešení problému, každý prvek takové úrovně je charakterizován svými vstupními daty a podmínkami, výstupními daty a podmínkami a transformacemi (algoritmy) vstupů na výstupy. Na konci takového řešení (po kroku s nejnižší mírou abstrakce) je k dispozici materiál - dokumentace (schemata, texty v pseudokódu), kterou lze bez větší námahy možno implementovat na některý konkrétní počítač.

V současné době se používá několika metod, které vycházejí z výše uvedených principů. Z těchto metod jsou nejznámější tyto :

a) STRUCTURED DESIGN

Metoda je popsána v literatuře (1), (2), (3) a je o ní stručná zmínka ve článku (4), (8).

b) PRINCIPLES OF PROGRAM DESIGN

Metoda je popsána v literatuře (5) a je v ní stručná zmínka ve článku (6), (8).

c) LOGICAL CONSTRUCTION OF PROGRAMS

Metoda je popsána v literatuře (7) a je o ní stručná zmínka ve článcích (8) a tento článek.

d) META STEPWISE REFINEMENT

Metoda je popsána v literatuře (9) a je o ní stručná zmínka v článku (8).

Metoda buduje na předpokladu, že dělá-li se něco vícekrát, výsledek se zlepšuje. Vychází se z jednoduchého řešení problému, toto řešení se stále rozšiřuje a zjemňuje. Řešení se rozšiřuje vždy jen o jeden detail, který se analyzuje.

Pro analyzovaný detail se vybere nejlepší řešení a v tomto řešení se pokračuje výběrem dalšího detailu. Metodou se produkuje stromově strukturované řešení. Metoda používá Millsova přístupu TOP-DOWN , Wirthovy metody STEP-WISE a DIJKSTROVA principu LEVEL-STRUCTURE.

e) HIGHER ORDER SOFTWARE

Metoda je popsána v článcích (8) a (10)

O metodách a) až d) jsou podrobné informace v citované literatuře. O metodě a) a b) jsou k dispozici i stručné články v češtině - proto o těchto metodách není v tomto článku ani stručná zmínka. O metodě e) jsem kromě článku (8) nestudoval žádný materiál.

Až na metodu, která je předmětem tohoto článku vznikly všechny ostatní metody v USA.

Doporučuji prostudovat článek (8), ve kterém je stručný popis všech pěti metod a navíc pokus o srovnání těchto metod z několika hledisek.

Zajímavá je skutečnost, že metody b) a c) vznikly v témže období na dvou různých místech (USA, Francie), vycházejí ale z podobných principů.

I. STRUKTURY DAT A PROGRAMŮ

I.1 Úvodní informace

Metoda vychází z principu, že struktura dat je klíčem k úspěšnému návrhu algoritmů, resp. programů.

Schematicky lze naznačit postup při řešení informačního problému v následujícím 6-ti stupňovém cyklu :

1° Identifikuj všechna vstupní data a zorganizuj je do schematu ve formě hierarchické struktury. Vstupní data se člení na několik úrovní (fields, records,

entries, items). V tomto bodu nejde zatím o přesné formáty dat, ale o vzájemné vztahy různých částí vstupních souborů mezi sebou.

2° Definuj, kolikrát se každý element vstupních dat objeví v nadřízeném elementu. Elementy mnemotechnicky pojmenuj.

3° Akce popsané v bodech 1° a 2° proveď pro požadovaná výstupní data.

4° Získej detaily programů identifikováním typů činností (pseudoinstrukcemi), které se budou provádět. Zápis činností je nutno provádět v tomto pořadí :

- vstupy dat (čtení)
- přípravné akce a provádění větvení
- výpočty
- výstupy dat
- zapojování subrutin

5° Jednotlivé části logické sekvence pseudoinstrukcí, vzniklé v bodě 4° identifikuj mnemotechnickými jmény, např. BEGIN PROCESS, END PROCESS, BRANCH, NESTING.

6° Očísluj elementy logické sekvence pseudoinstrukcí a každou část rozpracuj pomocí instrukcí příslušného programovacího jazyka.

1.2 HIERARCHICKÁ ORGANIZACE

Postulát : Vstupní data; výstupy; algoritmy vyjadřující transformace vstupů na výstupy (tj. i programy) jsou informační soubory.

Pravidlo : Jakýkoliv soubor informací lze členit na skupiny informací, postupujeme-li od nejvyšší úrovně (naj-

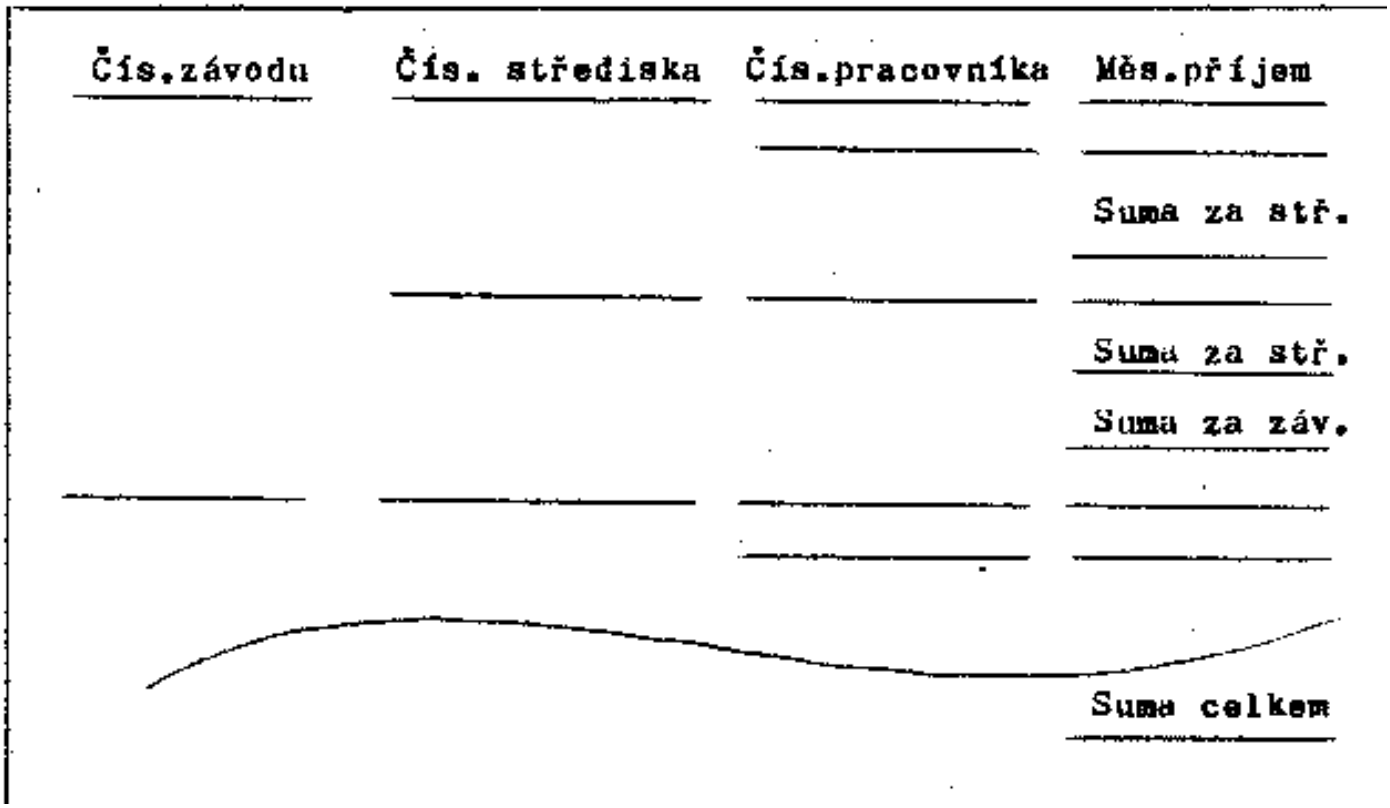
vyšší míra abstrakce) k nižším úrovním s použitím vhodných kriterií pro dělení.

Při vytváření hierarchické struktury je nutno řídit se následujícími pravidly :

- nižší úrovně struktury se tvoří, dokud lze nalézt elementy, které se v elementech vyšší úrovně vyskytují více než jedenkrát
- v hierarchické struktuře musí být jasně naznačeny vztahy každého elementu ke všem elementům vyšších úrovní
- proces tvorby hierarchické struktury předpokládá vždy precisní definici právě děleného elementu a kriteria pro dělení.

Příklad rozpisu výstupního souboru (tisk. sestavy) do schematu, která se v popisované metodě používají na rozpis struktur.

Zadání má se vytisknout tisková sestava dle schematu



Rozpis odpovídající hierarchické struktury (podřízené elementy s sebou nesou informaci o počtu výskytů v nadřízených elementech) :

1. ÚROVEŇ	2. ÚROVEŇ	3. ÚROVEŇ	4. ÚROVEŇ
TISK. SESTAVA	ZÁVOD (Zx)	ČÍS. ZÁVODU (1x)	ČÍS.STŘEDISKA (1x)	ČÍS.PRACOVNÍKA (1x)
				MĚS.PŘÍJEM (1x)
			PRACOVNÍK (Px)	SUMA ZA STŘEDISKO (1x)
		STŘEDISKO (Sx)	SUMA ZA ZÁVOD (1x)	
	SUMA CELKEM (1x)			

1.3 OPAKUJÍCÍ SE STRUKTURY

V kapitole je uvedeno jak popsat hierarchickou strukturu vstupních dat, aby bylo možno z tohoto popisu zobrazit požadované výstupy a algoritmy potřebné pro transformaci vstupů na výstupy.
Při vytváření hierarchické struktury vstupních dat se řídíme dříve základními pravidly, uvedenými v kapitole 1.2 s tím rozdílem, že některé úrovně struktury (ty které se pro zpracování vstupů na výstupy nepoužijí) se vynechají.

Příklad : Vstupní soubor pro vytištění tiskové sestavy (viz př. v kap. 1.2) obsahuje jednu větu pro každého pracovníka. Věta má tuto strukturu :

DALŠÍ INF.	ČÍS.ZÁVODU	ČÍS.STŘEDISKA	ČÍS.PRACOVN.	PŘÍJEM

Soubor je setříděn vzestupně, kdy nejvyšším klíčem jsou "DALŠÍ INFORMACE", nejnižším "ČÍS.PRACOVNÍKA". Úroveň "DALŠÍ INFORMACE" lze ignorovat (předpokládáme, že čísla závodů jsou jedinečná.)

Na následující úrovni (ČÍS. ZÁVODU) je nutno začít s dělením (tvořením) hierarchické struktury. V tomto okamžiku se objevují "OPAKUJÍCÍ SE STRUKTURY" - pro každou (mimo nejnižší) úroveň se ve vstupním souboru vyskytuje věc elementů nižší úrovně téže struktury. Rozpis hierarchické struktury vstupního souboru bude pak vypadat takto :

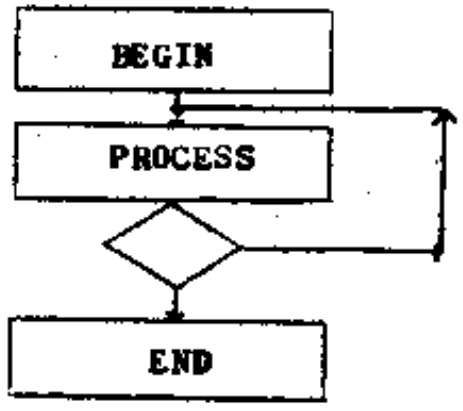
1. ÚROVEŇ	2. ÚROVEŇ	3. ÚROVEŇ	4. ÚROVEŇ
Vst. soubor { ZÁVOD (Zx)	{ STŘEDISKO (Sx)	{ PRACOVNÍK (Px)	{ ČÍS.ZÁVODU (1x) ČÍS.STŘEDISKA (1x) ČÍS.PRACOVNÍKA (1x) PŘÍJEM (1x)

Hierarchická struktura programu se dedukuje ze struktury vstupních dat. Element vstupních dat, který je "OPAKUJÍCÍ SE STRUKTURA" odpovídá element programu, který se skládá ze tří částí :

- část "BEGIN", provádí se 1x
- část "PROCESS", která se provádí opakovaně a jejíž poslední instrukcí je podmíněný skok
- část "END", provádí se 1x

Schematicky :

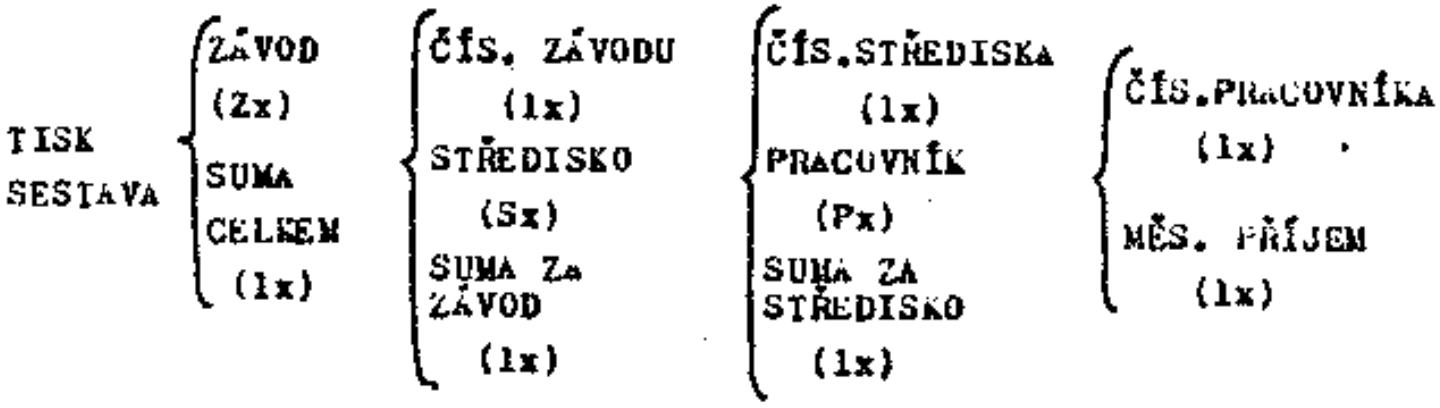
PROGRAM { BEGIN (1x)
PROCESS (Nx)
END (1x)

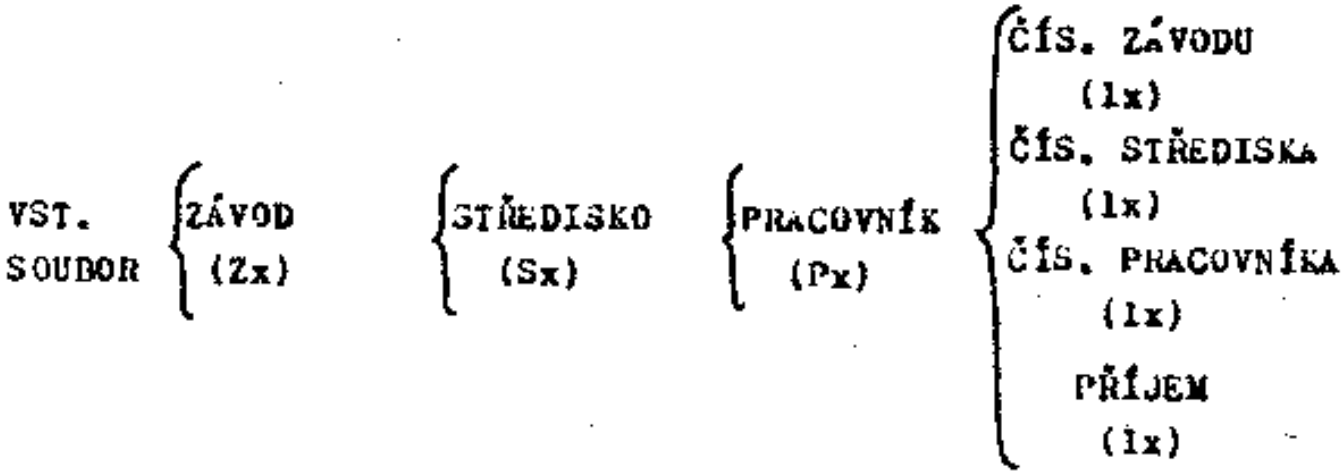


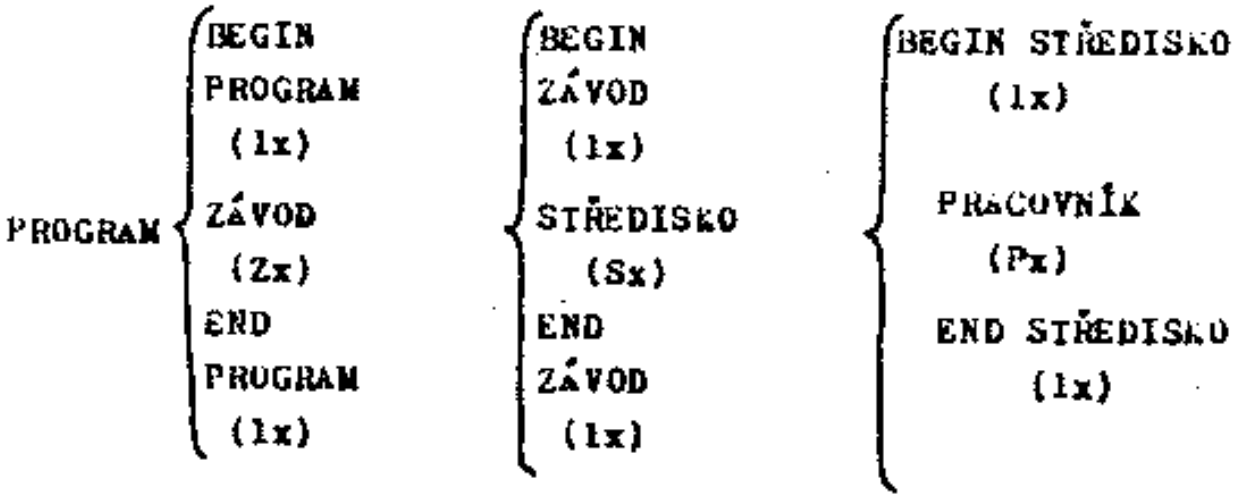
Organizace programu se provádí ve 2 etapách :

- organizace do logické sekvence (viz př. dále)
- detailní organizace (do instrukcí)

Příklad tvorby hierarchické struktury programu a následně logické sekvence ze struktur vstupů a výstupů.







Toto schema zobrazuje hierarchickou strukturu programu, jehož logika je zcela determinována. Schema je možno převést do jiného zobrazení (např. blokového schematu).

1. 4. DETAILNÍ ORGANIZACE PROGRAMU

Pro detailní organizaci programu je nutno vyjít z očíslované logické sekvence programu, která vznikla ze zobrazení hierarchické struktury programu.

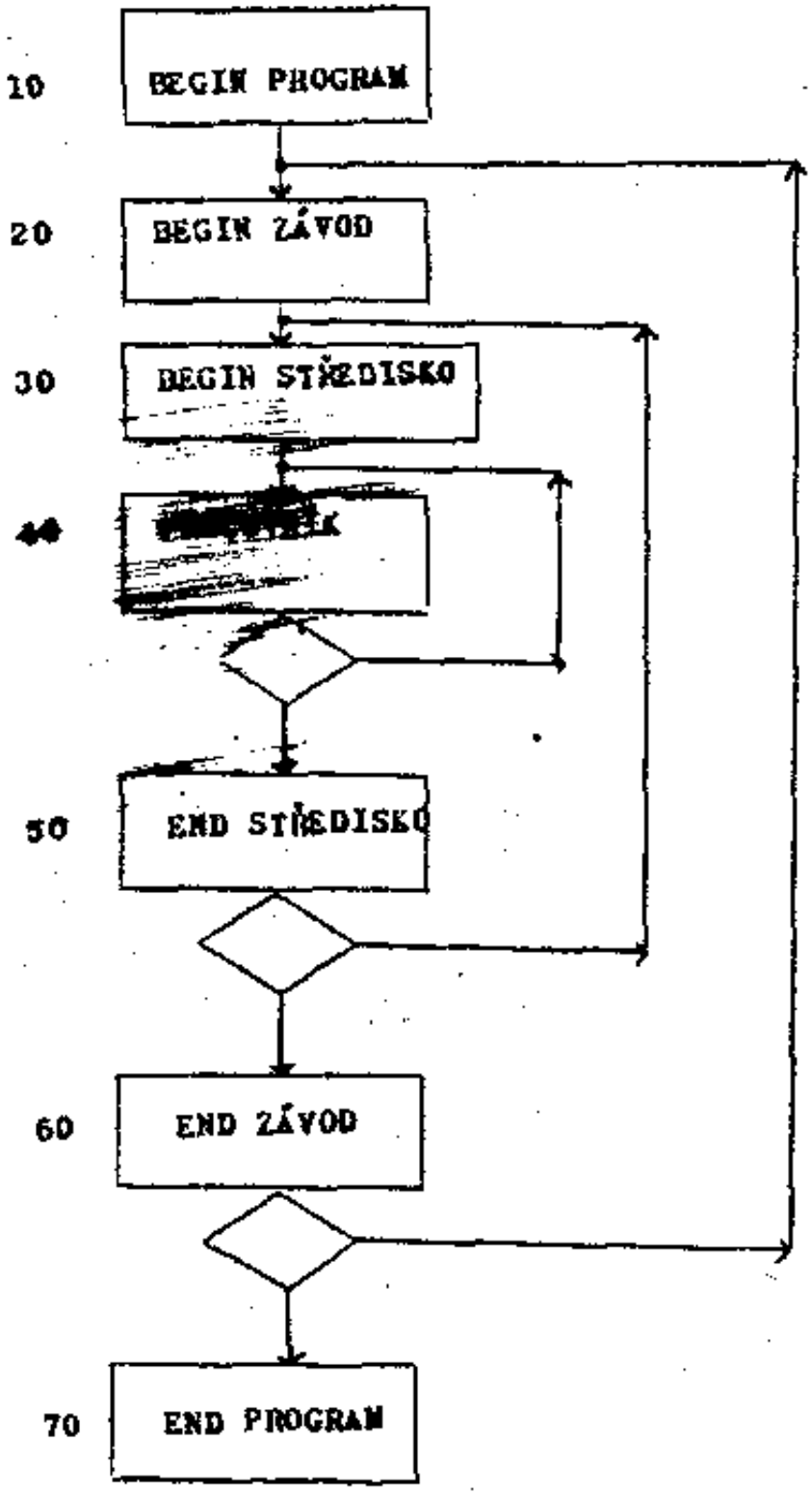
Detailně organizovaný program se skládá z pseudoinstrukcí (instrukcí) těchto typů :

- čtení vstupu
- skoky a přípravy skoků
- výpočty a přípravy výpočtů
- výstupy a přípravy výstupů
- volání subrontin

Sled detailních instrukcí odpovídá určitému elementu logické sekvence programu, pokud je tento sled prováděn na témže místě programu tolikrát, kolikrát se má provést odpovídající element z logické sekvence.

Pokud odpovídá týž sled instrukcí věci elementům logické sekvence, musí se sled naprogramovat vícekrát, resp. je možno použít subrontiny.
Detailní organizace je ukončena spojením sledů instrukcí dle typů do jednoho sledu instrukcí, který již může být prakticky proveden.

Příklad : Přepišme zobrazení hierarchické struktury programu (viz př. v kap. 1.2) do blokového schematu, které bude zobrazovat logickou sekvenci programu :



Analýza použití instrukcí vstupu

- čteme jen 1 vst. soubor, který je organizován sekvenčně
- na otázku kolikrát - se má číst

vst. soubor si odpovíme : tolikrát, kolik je v souboru vět + 1 (tj. P + 1)

- na otázku, kdy se má provádět čtení : P x v místě zpracování věty pracovníka (pseudoinstrukce 40) a 1 x na začátku programu (pseudoinstrukce 10)
- v programu stačí použít 2x čtení vstupního souboru, 1x na místě pseudoinstrukce 10, 1 x na místě pseudoinstrukce 40 (Px)

Analýza použití skoků

pseudoinstrukce	instrukce	následující pseudoinstrukce
40	pokud přečtené středisko = právě zpracovávané středisko	40
50	pokud přečtený závod = právě zpracovávaný závod	30
60	pokud není EOF	20

Analýza přípravy skoků

Aby bylo možné provést skokovou instrukci, musí být určeno, zda vstupující data patří do zpracování, odpovídající příslušným pseudoinstrukcím logického schematu. Toto rozhodnutí závisí na výsledku srovnání 2 údajů

- údaj o právě zpracovávaných datech (REF)
- údaj o právě přečtených datech (IDENT)

v našem příkladě je nutno provést tyto přípravné instrukce pro skoky :

Pseudoinstrukce	Instrukce
20	Přesun "ČÍS.ZÁVODU" do REF pro "ČÍS.ZÁVODU"
30	Přesun "ČÍS.STŘEDISKA do REF pro ČÍS. STŘEDISKA

Analýza výpočtů

Pseudoinstrukce	Instrukce
10	Nulování SUMA CELKEM
20	Nulování SUMA ZA ZÁVOD
30	Nulování SUMA ZA STŘEDISKO
40	Při čtení PŘÍJEM do SUMA ZA STŘEDISKO
50	Při čtení SUMA ZA STŘEDISKO do SUMA ZA ZÁVOD
60	Při čtení SUMA ZA ZÁVOD do SUMA CELKEM

Analýza použití instrukcí výstupu

Pseudoinstrukce	Instrukce
20	Edice ČÍS. ZÁVODU
30	Edice ČÍS. STŘEDISKA
40	Edice ČÍS. PRACOVNÍKA
40	Tisk a obnovení řádku
50	Edice SUMA ZA STŘEDISKO
50	Tisk a obnova řádku
60	Edice SUMA ZA ZÁVOD
60	Tisk a obnova řádku
70	Edice SUMA CELKEM
70	Tisk a obnova řádku

V tomto okamžiku máme rozepsanou logickou sekvenci programu do výkonných instrukcí (obvykle použijeme psedokódu). Další postup je :

- soupis všech instrukcí programu do seznamu - rekapitulace instrukcí
- setřídění rekapitulace instrukcí vzestupně podle čísel pseudoinstrukcí. V rámci každé pseudoinstrukce se jednotlivé instrukce zapisují v tomto pořadí :
  - příprava skoků
  - příprava výpočtů a výpočty
  - příprava výstupů a výstupy
  - vstupy
  - skoky

Program z probraného příkladu by detailně organizován vypadal takto :

```
10   Nulování SUMA CELKEM
     Čtení I. věty
20   Přesun ČÍS. ZÁVODU do REF pro ČÍS. ZÁVODU
     Nulování SUMA ZA ZÁVOD
     Edice ČÍS. ZÁVODU
30   Přesun ČÍS. STŘEDISKA do REF pro ČÍS. STŘEDISKA
     Nulování SUMA ZA STŘEDISKO
40   Při čtení PŘÍJEM do SUMA ZA STŘEDISKO
     Edice ČÍS. PRACOVNÍKA
     Tisk a obnovení řádku
     Čtení další věty
     Test IDENT : REF (pro ČÍS. STŘEDISKA)          40
50   Při čtení SUMA za STŘEDISKO do SUMA ZA ZÁVOD
     Edice SUMA ZA STŘEDISKO
     Tisk a obnovení řádku
     Test IDENT : REF (pro ČÍS. ZÁVODU)             30
60   Při čtení SUMA ZA ZÁVOD do SUMA CELKEM
     Edice SUMA CELKEM
     Tisk a obnovení řádku
     Test na EOF                                    20
```

70 Edice SUMA CELKEM
Tisk a obnovení řádku

Takto v pseudokódu zapsaný program je možno už snadno přepsat v libovolném jazyku na libovolný počítač. Před tímto zápisem je ještě nutno provést kontrolu vytvořeného programu na požadovaný výstup.

1.5 SHRNUTÍ

V kapitole 1 jsou jen velmi schematicky a to pouze na těch nejjednodušších případech, znázorněny principy, které WARNIER ve své učebnici uvádí.

Shrňme si postup:

- definuj hierarchickou strukturu výstupů
- definuj hierarchickou strukturu vstupů
- organizuj program ze vstupů, výsledek kontroluj na výstupy

Program se organizuje ve 2 stupních:

- konstrukce programu v hierarchicky organizovaných „sub-setech", výsledkem je uspořádaný sled „pseudoinstukcí" - logická sekvence
- detailní organizace programu

2. ZÁVĚR

Vzhledem k rozsahu příspěvku nebylo možno uvést některé další zajímavé a důležité poznatky jako komplexní a alternativní stuktury, optimalizace programů a zpracování. Účelem příspěvku bylo naznačit, že předpokládaná metoda může při správném použití usnadnit práci projektantů a programátorů.

LITERATURA

(1) FIREMNÍ LITERATURA IBM
(2) YOURDON, E.; CONSTANTINE, L. - STRUCTURED DESIGN
(3) MYERS G.J. - COMPOSITE / STRUCTURED DESIGN
(4) METZL, TVRDÍK - SBORNÍK „HAVÍŘOV 1979"
(5) JACKSON, M.A. THE PRINCIPLES OF PROGRAM DESIGN
(6) ORBAL - SBORNÍK „COBOL 1979"
(7) WARNIER, J-D - THE LOGICAL CONSTRUCTION OF PROGRAMS
(8) PETERS, L.J.; TRIPP L.L. - COMPARE SOFTWARE DESIGN METHODOLOGY (DATAMATION ZØ 11)
(9) LEDGARD, H. - META STEPWISE REFINEMENT