Ing. Karel METZL
Závod výpočetní techniky OKD, Ostrava

METODA NORMALIZOVANÉHO PROGRAMOVÁNÍ VE ZPRACOVÁNÍ HROMADNÝCH DAT

1. Úvod.

Výpočetní technika je obor s neobyčejně prudkým tempem rozvoje. Samočinné počítače prošly za velmi krátkou dobu své existence přes několik generací. V současné době dožívá 2. generace, v největším rozmachu jsou stroje 3. generace, mluví se o 3 1/2 generaci a 4. generace již na sebe nedá jistě dlouho čekat.

Přestože pojem generace je dán hardwarem, podle použité součástkové základny, je s každou generací dán obor a rozsah použití počítače spolu s úrovní dostupného software. Počítače, které byly původně používány výhradně pro vědeckotechnické výpočty, přešly z velké části na prozaičtější práci - na zpracování hromadných dat. Software, který původně tvořilo několik málo subrutin, se značně rozrostl. Dnes je samozřejmostí rozsáhlý operační systém vybavený kompilátory problémově orientovaných jazyků, třídícími, spojovacími a různými pomocnými programy.

Pro současný stav software je charakteristické, že výrobce dodává (často prodává) s počítači i aplikační software. Formou t. zv. "pekidžů" se dodává na příklad systém obecné

databanky, systémy pro práci s terminály, různé specializované systémy pro materiálové hospodářství, personalistiku, řízení výroby atd.

Podíl firemního aplikačního software se bude určitě dále zvyšovat. Přesto však bude třeba stále psát i vlastní uživatelské programy a jde o to, aby to byly programy dobré. Pokusme se definovat vlastnosti, které by měl takový program mít.

1. Efektivní práce a úměrné nároky na zdroje.
2. Rychlé napsání.
3. Rychlé a spolehlivé odladění.
4. Odolnost vůči změnám.
5. Přenosnost mezi programátory.

Významnost jednotlivých vlastností je závislé na typu programu, ale přesto se pro většinu programů z oboru zpracování hromadných dat uvádějí v tom pořadí, jak jsem je zapsal. A zde bych chtěl zvolat s autorem odkazu [1] "Nejsou naše priority špatné?".

Podívejme se na vlastnost uvedenou pod bodem 1 a považovanou tudíž za nejvýznamnější. Centrální jednotky dnešních počítačů jsou natolik rychlé a paměti tak rozsáhlé, že už zdaleka nešetříme každou instrukcí či každým bytem paměti. Vždyť kdybychom byli v tomto směru důslední, nemohli bychom používat ani problémově orientované jazyky bg dokonce ani ne samotný operační systém.

Ani s vlastnostmi uvedenými pod body 2 a 3 to při pečlivém zkoumání nedopadne lépe. Naši nadřízení i zákazníci jim sice dávají prioritu číslo 1, ale podle seriozních statistik je tvorbě nových programů věnováno méně než 20 % programátorské práce a hlavní náplní je provádění změn v programech dříve napsaných. A tak nakonec vidíme, že nejdůležitější se ukázaly vlastnosti uvedené v seznamu pod body 4 a 5.

Dobrý program musí být především logicky přehledný. Už při psaní je třeba počítat s jeho dalším několikaletým "ži-

votem" plným změn a nepoužívat tudíž žádné speciální programátorské obraty, které v budoucnu znemožní provést změnu komukoliv jinému kromě tvůrce (a často i jemu). Samozřejmostí jsou i standardizované a dostatečně mnemotechnické identifikátory spolu s četnými komentáři přímo ve zdrojovém programu. Minuty, které se ušetří jejich vynecháním, stojí v budoucnu často hodiny a někdy i dny.

Je snazší určiti si cíl než nalézti k němu cestu. Začínajícímu programátorovi trvá léta než objeví již objevené a vypracuje si svůj programátorský styl. Nemusí to být vždy ještě styl nejlepší, nemluvě již o tom, že tím trpí přenosnost programů mezi programátory.

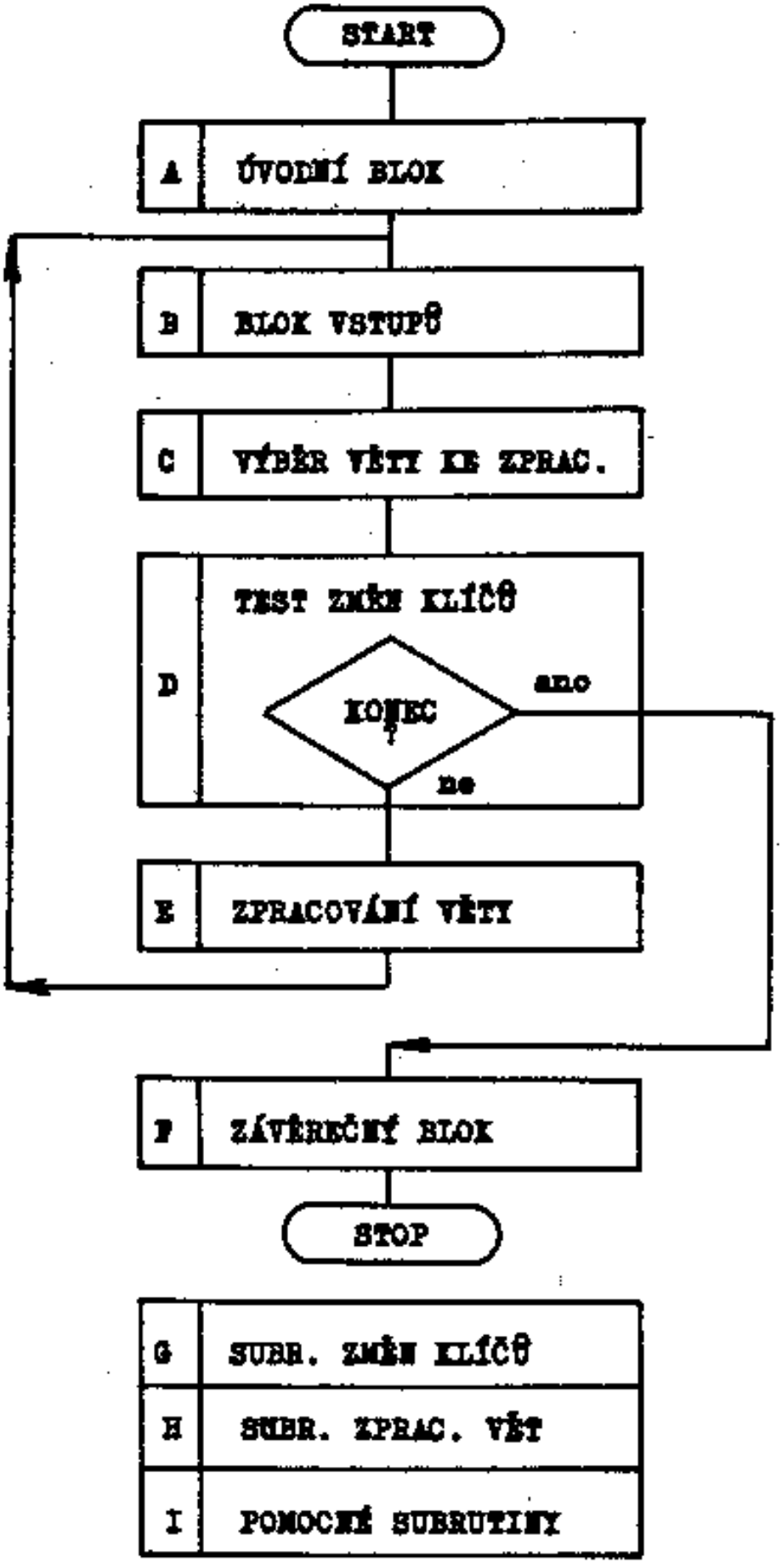
V posledních letech se objevila řada metod, které se snaží vnést do uvedené situace pořádek a pokouší se o sjednocení programátorského stylu. Neměl jsem sice možnost seznámit se se všemi podrobně, ale zdá se, že pro oblast zpracování hromadných dat vyhovuje nejlépe metoda normalizovaného programování popsaná v odkazech [3] a [4]. Dovolte mi nyní, abych stručně vysvětlil základní principy této metody.

2. Základní principy metody normalizovaného programování.

Normalizované programování tvoří ucelený systém jednotné logické výstavby programu. Řízení postupu programu je standardizované a zřetelně oddělené od vlastního zpracování, které se děje ve vyvolávaných subrutinách. Metoda je zcela obecná, nezávislá na typu počítače a programovacím jazyku.

2.1. Bloková struktura programu.

Základní myšlenka normalizovaného programování vychází ze skutečnosti, že většina programů z oblasti hromadného zpracování dat má v podstatě stejný průběh. Každý program lze tudíž rozdělit do bloků A až I s pevně vymezenou funkcí a seřazených v programu ve stanoveném pořádku (viz obr. 1.). Funkční náplň jednotlivých bloků i jejich pořadí je pro všechny programy závazná. Naproti tomu je samozřejmé, že



Obr. 1. Obecný vývojový diagram normalizovaného programu

rozsahy jednotlivých bloků jsou závislé na typu konkrétního programu, že dokonce některé bloky nemusí být v každém programu obsaženy. Tak na příklad programy, do kterých vstupuje pouze jeden setříděný vstupní soubor, nebudou obsahovat blok C - Výběr věty ke zpracování.

2.2. Řídící oblasti souborů.

Každý vstupní soubor se kterým se v programu pracuje má v pracovní paměti vymezenou určitou řídící oblast o pevně stanovené struktuře. Navíc jsou v programu dvě další klíčové oblasti (shodné struktury) pro uložení řídících oblastí současně zpracovávaného souboru a posledně zpracovávaného souboru (obr. 2.). Řídící oblast obsahuje tři části (obr. 3a.).

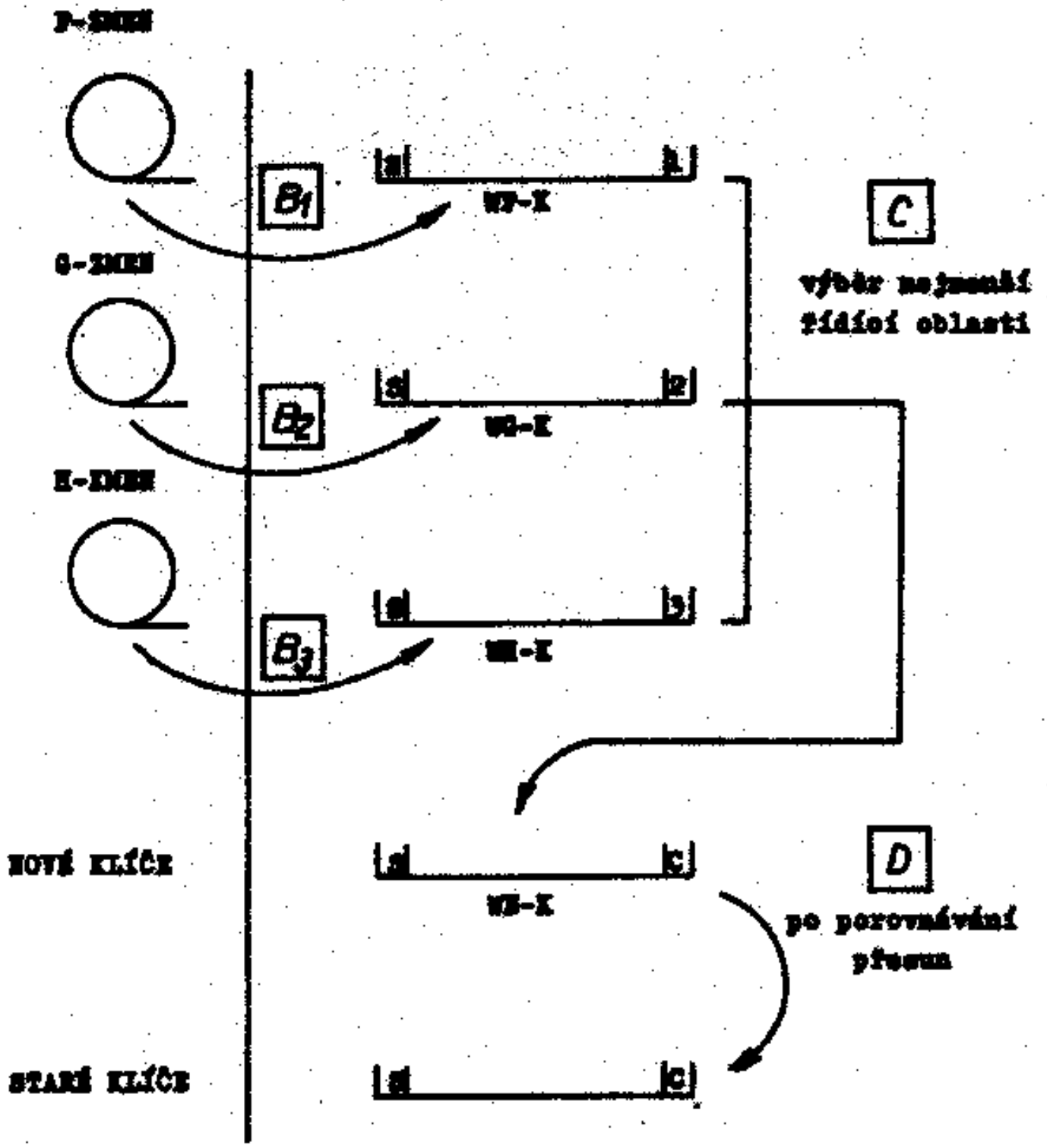
a. Ukazatel stavu souboru S (1 znak).
   0 ..... čti
   1 ..... nečti (stav po provedeném čtení)
   2 ..... soubor byl dočten do konce
   3 ..... soubor není zpracováván
   (v daném konkrétním běhu se zpravidla rozhodne podle sejmutých parametrů)

b. Třídící klíče
   seřazené podle priority od nejvyšší do nejnižší.
   Na příklad podnik, závod, hospodářské středisko.

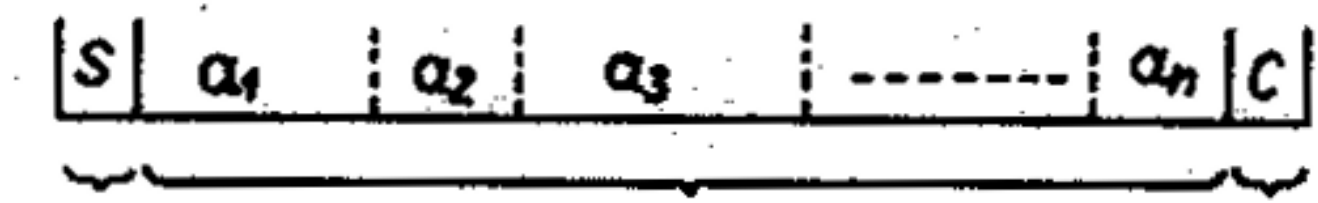
c. Číslo souboru C (1 znak).
   Pro každý soubor je stanoveno pořadové číslo. Jeho stanovení není nahodilé a volba bude vysvětlena v popisu bloku C.

Tak na příklad řídící oblast 14538203102 může znamenat:

1 ..... stav po provedeném čtení
45 ..... číslo podniku
38 ..... číslo závodu
20 ..... číslo hospodářského střediska
310 ..... číslo činnosti
2 ..... soubor číslo 2



Obr. 2. Princip řídících oblastí

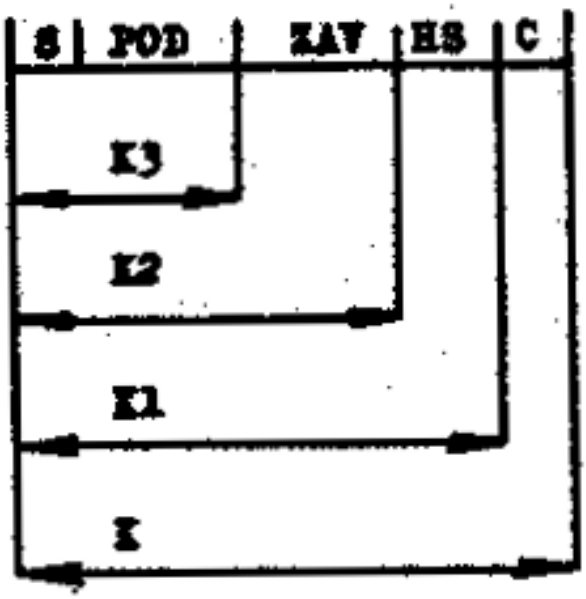


ukazatel stavu souboru

třídicí klíče podle priorit

číslo souboru

Obr. 3a. Struktura třídicí oblasti




```
1 WK-K
  5 WK-K1
    10 WK-K2
       15 WK-K3
          20 WK-S
          20 WK-POD
       15 WK-ZAV
    10 WK-HS
  5 WK-C
```

Obr. 3b. Příklad struktury třídicí oblasti a popisu

V problémově orientovaných jazycích se doporučuje popsat řídící oblast jako strukturu s podřízenými úrovněmi. Usnadňuje to pozdější testování v bloku D (obr. 3b.).

2.3. Systém výhybek.

Pružnost programu je zajištěna systémem výhybek, které lze v jednotlivých blocích nastavovat, v jiných testovat případně nulovat.

Typickým příkladem je výhybka nastavovaná v bloku E po zpracování věty. Před jejím nastavením (to znamená při prvním průchodu) nejsou v subrutinách změny klíčů tištěny mezisoučty.

3. Popis funkce jednotlivých bloků programu.

3.1. A - Úvodní blok.

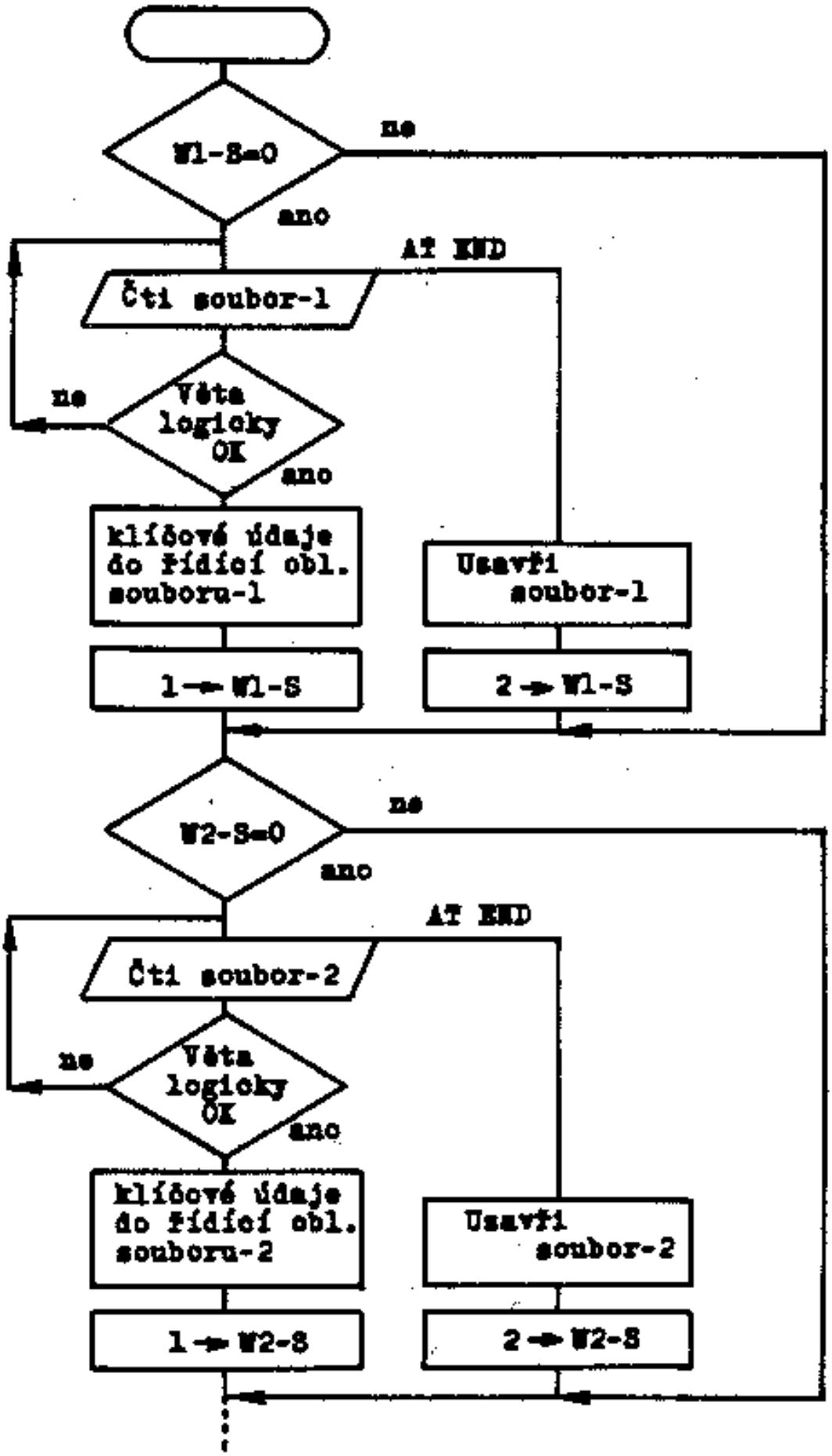
V úvodním bloku se do programu především vloží všechny informace potřebné k určení varianty programu. Zpravidla se to děje sejmutím parametrů. Parametry se po zkontrolování na logickou správnost interpretují. To znamená nastavují se výhybky, hodnoty některých proměnných nebo ukazatelé stavu v řídících oblastech těch souborů, které se nemají zpracovávat.

Úvodní blok může dále obsahovat sejmutí tabulek nebo jiných údajů potřebných pro výpočet. U nejjednodušších programů se úvodní blok omezí na otevření všech souborů.

3.2. B - Blok vstupů. (obr. 4.)

Pro každý vstupní soubor je vytvořen v bloku B příslušný podblok. Jednotlivé podbloky jsou seřazeny za sebou a při průchodu blokem B se přechází z jednoho na druhý.

Na začátku podbloku je vždy test ukazatele stavu řídící oblasti příslušného souboru. Je-li ukazatel různý od nuly, přechází se na další podblok. Po přečtení věty může následovat test na věcnou správnost (na příklad zpracovávají-li se



Obr. 4. Vývojový diagram bloku vstupů

z daného souboru pouze věty určitého typu). Klíčové údaje z přečtené věty se pak přesouvají do řídící oblasti daného souboru a nakonec se změní ukazatel souboru na 1. Součástí čtení je i test konce souboru. Je-li soubor dočten do konce, uzavře se a ukazatel stavu se nastaví na hodnotu 2.

Při prvním průchodu blokem B jsou ukazatelé stavu všech přítomných souborů nastaveny na počáteční hodnotu 0 a čtou se tudíž postupně věty ze všech souborů. Při dalších průchodech se čte už jen věta jediného souboru (viz blok C).

3.3. C - Výběr věty ke zpracování.

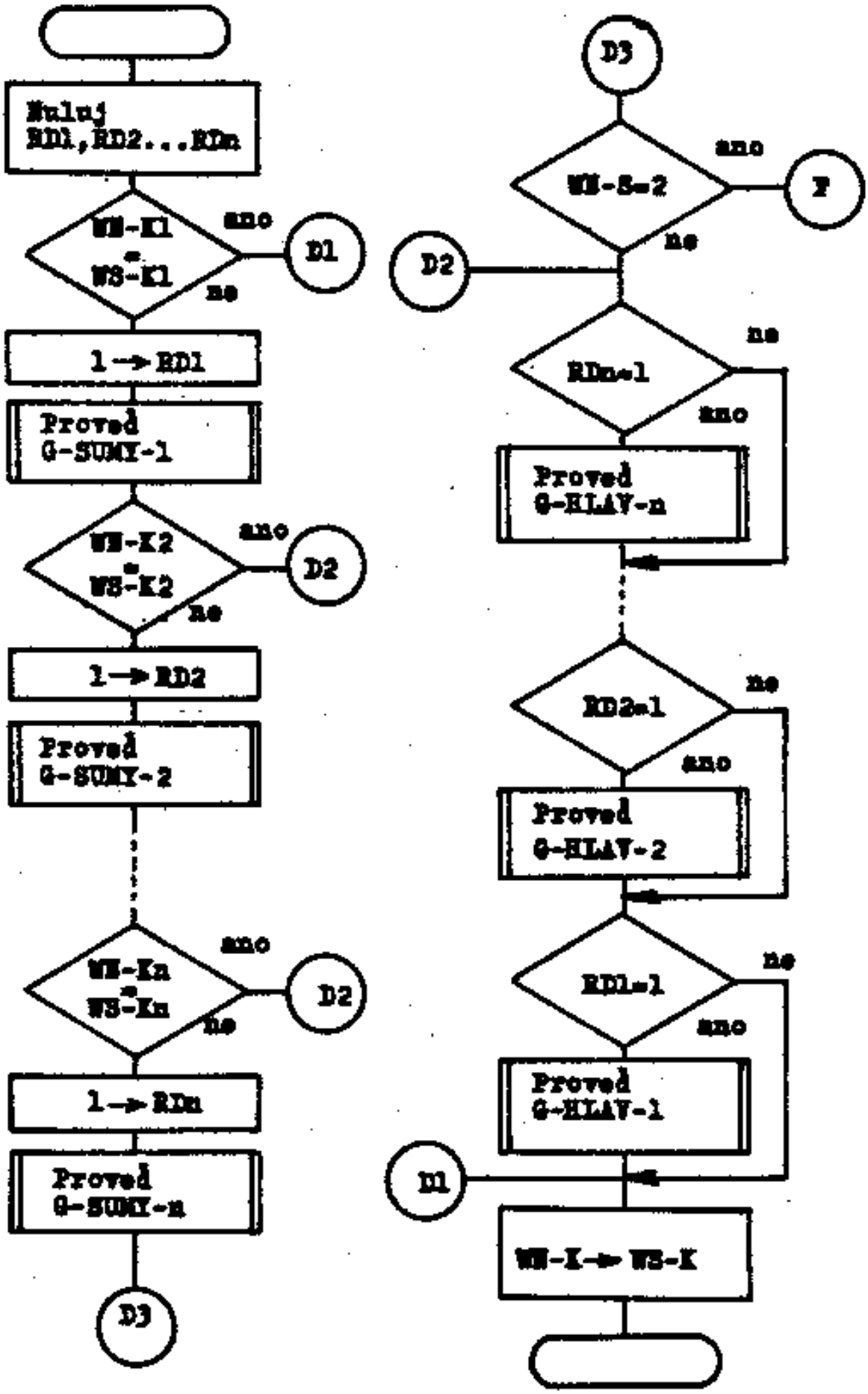
V bloku C se hledá soubor jehož řídící oblast je nejmenší a tato řídící oblast se přesouvá do řídící oblasti současně zpracovávaného souboru (v obr. 2. označena NOVÉ KLÍČE). Současně se nastaví ukazatel stavu souboru s nejmenší řídící oblastí na hodnotu 0. To znamená, že při následujícím průchodu blokem B bude čtena pouze věta tohoto souboru.

Protože částí řídící oblasti je i číslo souboru, nemůže při porovnávání dojít k rovnosti a jedna řídící oblast je vždy nejmenší.

Zde je třeba poznamenat, že při rovnosti všech třídících klíčů je jako poslední uvolněna věta souboru s nejvyšším číslem a podle toho je nutno volit očíslování souborů. Na příklad při aktualizaci kmenového souboru větami několika souborů, se musí nejvyšší číslo přidělit aktualizovanému kmenovému souboru. Pro začátek se doporučuje provést si pro ověření správnosti volby ladění "na sucho".

3.4. D - Test změny klíčů. (obr. 5.)

V bloku D se porovnává řídící oblast současně zpracovávaného souboru s řídící oblastí posledně zpracovávaného souboru (v obr. 2 NOVÉ KLÍČE a STARÉ KLÍČE). Testy se provádějí postupně po jednotlivých úrovních (viz obr. 3b.) a při změnách na určité úrovni se nastavují výhybky případně vyvolá-



Obr. 5. Vývojový diagram bloku D

vají příslušné subrutiny z bloku G.

Blok D je proti původnímu vývojovému diagramu uveřejněnému v [3] poněkud rozšířen. V naší verzi je umožněno vyvolávat kromě součtových subrutin (od nejnižší úrovně až po změnu) i hlavičkové rutiny (od změny až po nejnižší úroveň).

Součástí bloku D je test na konec programu. Jsou-li dočteny všechny vstupní soubory, předává se řízení na závěrečný blok F.

Po provedených testech je nahrazena řídící oblast posledně zpracovávaného souboru (STARÉ KLÍČE) řídící oblastí současně zpracovávaného souboru (NOVÉ KLÍČE).

3.5. E - Zpracování věty. (obr. 6.)

V bloku E se provádí zpracování jednotlivých vět. Pro každý vstupní soubor je, podobně jako v bloku B, dán jeden podblok, který může být dále ještě členěn podle typu věty. Podbloky jsou řazeny za sebou a během průchodu blokem E je zpracován pouze jeden z nich. O tom, který bude zpracován, rozhoduje číslo souboru v řídící oblasti současně zpracovávaného souboru. Vlastní zpracování se pak provádí ve vyvolávaných subrutinách bloku H.

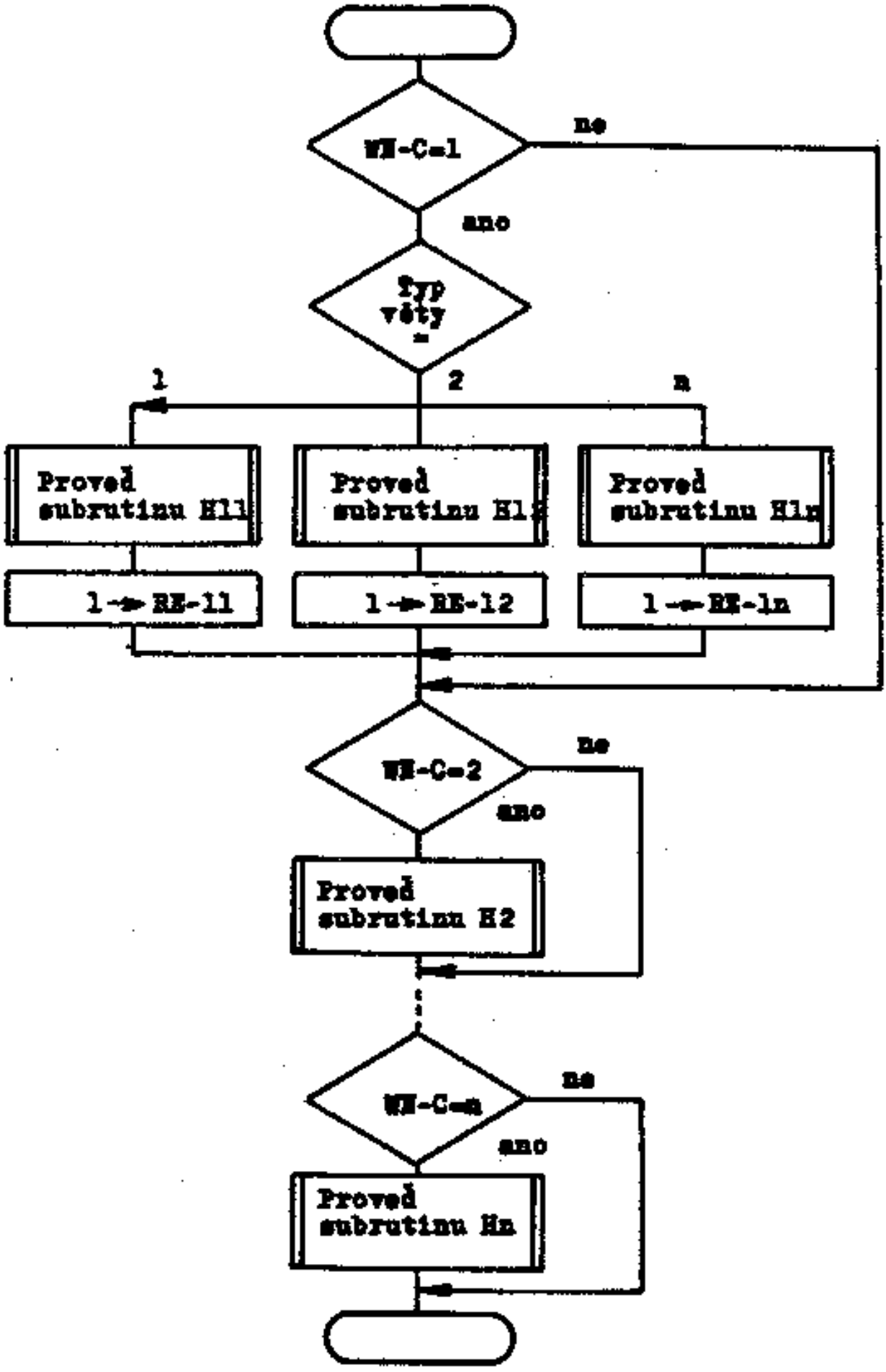
Blok E uzavírá hlavní smyčku a po jeho průchodu se předává řízení zpět na blok vstupů B.

3.6. F - Závěrečný blok.

Závěrečný blok obsahuje činnosti, které je nutno provést před ukončením programu a vlastní ukončení. Tiskne se zde výsledné tabulky, různé rekapitulace a závěrečné zprávy. Uzavírají se výstupní nebo UPDATE soubory.

3.7. G - Subrutiny pro změny klíčů.

Blok G obsahuje subrutiny, které jsou vyvolávány z bloku D při změnách třídících klíčů.


WH-C=1
ne
ano
typ
věty
=
1
2
n
Proved
subrutinu H11
Proved
subrutinu H1
Proved
subrutinu H1n
1 → HE-11
1 → HE-12
1 → HE-1n
WH-C=2
ne
ano
Proved
subrutinu H2
WH-C=n
ne
ano
Proved
subrutinu Hn

3.8. H - Subrutiny pro zpracování vět.

Blok H obsahuje subrutiny vyvolávané z bloku zpracování věty E.

3.9. I - Pomocné subrutiny.

Blok I obsahuje pomocné subrutiny vyvolávané zpravidla z bloků G a H.

4. Standardizace a formální úprava zdrojového programu.

Standardizace a formální úprava zdrojového programu zvyšují jeho přehlednost a jsou důležitým činitelem přispívajícím k přenosnosti programu mezi programátory. Na rozdíl od zásad normalizovaného programování, je standardizace závislá na programovacím jazyku i typu stroje a zmíníme se zde pouze o některých oblastech jimiž by se měla zabývat.

4.1. Standardizace návěští.

Z návěští musí být na první pohled patrno ke kterému bloku normalizovaného programování náleží a co se pod ním bude řešit. Pro snazší orientaci ve zdrojovém programu a vazbu na blokové schema by mělo obsahovat i pořadové číslo.

Na příklad: A-NULOV-TAB1-20
G-SUMA-ZAV-130

4.2. Standardizace jmen dat.

Jméno datové položky má kromě mnemotechnického jména obsahovat i označení ke které oblasti dat položka patří a tudíž kde se k ní najde její deklarace.

Na příklad:		
	E-KMEN	soubor s kmenovými údaji
	E2-POD	číslo podniku z věty 2 souboru E
	W8-POD	číslo pod. v prac. pam.
	T5-POD	čís. pod. v tisk. řádku

Pro některé obecně používané pojmy se doporučuje sestavit podnikový seznam závazných mnemotechnických zkratek.

4.3. Standardizace grafické úpravy programu.

Grafická úprava zvyšuje přehlednost programu a usnadňuje tak provádění změn. Předepsat lze na příklad sloupce od kterých se závazně píší které výroky, nebo stanovit, které výroky či jejich části musí začínat na novém řádku atd.

4.4. Standardizace dokumentace programu.

Standard závazně předepisuje obsah složky dokumentace k programu. Hlavní a nejdůležitější část tvoří výpis poslední kompilace, protože vždy zachycuje současný stav programu. Zdrojový program má být samodokumentující s dostatkem vysvětlivek a komentářů. Typ některých komentářů lze přímo standardem předepsat. Na příklad každý program by měl obsahovat jméno autora, datum napsání, stručný popis činnosti, závažné změny, kdy a kdo je provedl atd.

5. Výhody a nevýhody normalizovaného programování.

Podívejme se znovu na vlastnosti dobrého programu tak jak byly popsány v úvodu a porovnejme jak dalece k nim přispívá normalizované programování.

Jako nejdůležitější vlastnost dobrého programu z oblasti zpracování hromadných dat jsme si označili odolnost vůči změnám. Podstatným rysem metody normalizovaného programování je, že soustřeďuje části programu, které k sobě patří do jediného místa a jediného bloku. Běžná změna není tedy roztříštěna po celém programu, ale provede se do přesně určeného místa. Nejdůležitější je však, že po provedené změně si program stále zachovává svou podobu a s přibývajícími změnami se neznehodnocuje.

Zatím jsme mluvili pouze o běžných malých změnách nedotýkajících se struktury, v řeči normalizovaného programování

o změnách na úrovni bloků G, H, I. Výhoda normalizovaného programování vynikne tím spíše u změn větších, u těch které se struktury programu dotýkají. Každý, kdo na příklad programoval aktualizační program nenormalizovaným způsobem, si jistě dovede představit, do jakých nesnází by ho přivedl požadavek přidat ještě další soubor změn. Znamenalo by to přebudování celého programu spojené pochopitelně s dalším laděním. V normalizovaném programu stačí přidat podbloky do bloků B, E, provést drobnou změnu v bloku C a program pracuje bez nejmenších potíží. A co je hlavní - tuto dříve vysoce kvalifikovanou opravu dovede provést kterýkoliv programátor obeznámený s metodou normalizovaného programování.

Složitější je otázka rychlosti napsání normalizovaného programu a jeho odladění. Na programátorovi, který byl dříve zvyklý sednout a začít ihned psát program a problémy, které postupně vyvstávaly, řešil za pochodu, se nyní žádá, aby si program dříve promyslel. Musí si určit očíslování souborů, předem uvážit, které činnosti bude provádět v kterém podbloku E, případně si na sucho vyzkoušet, zda jsou jeho úvahy správné. A to vše jsou pro něj nepříjemné ztrátové časy. Neuvědomuje si, že délka uvedených časů je způsobena neznalostí nové metody, že se bude postupně zkracovat, že to co ztratí v této přípravné plánovací fázi se mu vrátí později při vlastním kódování a ladění programu.

Nejvážnější námitkou proti normalizovanému programování je otázka jeho efektivnosti. Normalizované programování je generalizovaný systém a jako takový musí být mutně postižen ztrátou účinnosti, musí být povelově náročnější než řešení přesně stavěné na jediný konkrétní problém. Vždyť každý na první pohled vidí ty "zbytečné" přesuny řídících oblastí atd. Abychom ověřili platnost těchto námitek, prováděli jsme na našem pracovišti srovnávací měření. Sledovali jsme jednak délky programů a jednak nároky na čas centrální jednotky. Měření jsme prováděli na počítači ICL 1905 a na programech napsaných v jazyce COBOL. Došli jsme při tom k zajímavým závěrům. Při porovnání normalizovaného programu s normálním

programem, napsaným zkušeným programátorem, jsme nezjistili žádné významné změny ani v délce programu ani v nárocích na čas centrální jednotky. Naproti tomu při porovnání s programem začátečníka byl normalizovaný program po všech stránkách lepší.

Bylo by nespravedlivé nezmínit se zde o tom, že normalizované programování jako obecná metoda se nehodí pro každý program. Je celá řada programů na př. různé kontrolní a obecné programy ve kterých by použití normalizovaného programování bylo neúčelné. V podstatě lze říci, že normalizované programování je vhodné pouze pro programy do kterých vstupuje alespoň jeden setříděný datový soubor. Takových programů je však při dnešním pojetí zpracování hromadných dat většina.

6. Zkušenosti se zaváděním normalizovaného programování.

Normalizované programování má zřejmě tak mnoho výhod, že teoreticky by jeho zavedení nemělo nic stát v cestě. Skutečnost je však jiná. Velice zasvěceně a vtipně je rozebrána v odkazu [2]. K tomu bych chtěl pouze připojit některé poznatky se zaváděním metody v našem závodě.

Se zaváděním normalizovaného programování jsme začali v době, kdy byl u nás již vytvořen poměrně stabilní tým schopných programátorů. Každý z nich měl vypracován svůj styl práce a neviděl nutnost se ho vzdát kvůli nějakému novému pochybnému systému. Metodu brutálního násilí [2] jsme pochopitelně nemohli ani nechtěli použít, a proto jsme začali pracovat metodou infiltrace [2]. Programátoři, kteří doplňovali stav, dostali důkladné školení o normalizovaném programování a byli vedeni k důslednému dodržování jeho zásad při psaní svých prvních programů. Působili jsme osvětově i u starších programátorů a postupně byla všeobecně uznána metoda normalizovaného programování jako vhodná pro aktualizační programy. Další infiltrací a osvětou jsme se během 3 let dostali až na dnešní stav, kdy je u nás tato metoda stanovena jako závazná pro všechny programy, do kterých vstupuje minimálně

jeden setříděný soubor. Současně byly vydány standardy pro psaní a formální úpravu zdrojových programů.

Za zmínku stojí i zkušenosti se způsobem výuky. Pořádali jsme školení s praktickými ukázkami, vydali písemnou zprávu, ale přesto to stále nebylo to pravé. Snad zde nepříznivě působila i skutečnost, že normalizované programování se sice hodí na většinu programů, ale pouze na malou část z nich se hodí ve své původní podobě. Na ostatní se musí, při dodržování základních zásad, poněkud přizpůsobit. Navíc programátoři byli ve vleku svých vžitých zvyklostí a vnášeli je nevhodně i do normalizovaného programu. Jako nejlepší se ukázal způsob přímého působení, kdy návrhy prvních normalizovaných programů dělal programátor se svým školitelem společně. Tento způsob byl sice náročný pro školitele, ale nejúčinnější.

Zajímavý způsob zavádění normalizovaného programování zvolil bratislavský ÚSIP. Zakoupil od firmy ADV/ORGA generátor normalizovaného programování [5] a jeho použití spolu s metodou chce dále rozšiřovat ve vlastním podniku, ale i mimo něj. Zakoupená verze je vhodná pro ANS COBOL pracující pod operačním systémem DOS.

7. Závěr.

K popularizaci normalizovaného programování by měly přispět i školy jejichž absolventi do programování v budoucnu nastoupí. Strategie programování zahrnující metodu normalizovaného programování by se měla stát nedílnou součástí výuky.

Normalizované programování je propracovaná metoda, která vede v oblasti zpracování hromadných dat ke zkvalitnění uživatelských programů a jako taková si zaslouží co největšího rozšíření.

Literatura.

[1] Are our priorities wrong?="Data Processing",16,1974, č.5,s.284

[2] Normované programování aneb jak každý programátor podle svého blaženosti dojde.="Mechanizace a automatizace administrativy",11,1971,č.2,s.61-63

[3] Normované programování.="Mechanizace a automatizace administrativy",11,1971,č.6,s.180-184,č.7-8, s.215-220

[4] PRŮŠEK,K:Lze programování racionalizovat?="Mechanizace a automatizace administrativy",12,1972,č.1,s.14-18

[5] ADV/ORGA Generator für Normierte Programmierung