D 22941

D 22941

ČESKOSLOVENSKÁ VĚDECKOTECHNICKÁ SPOLEČNOST
NOSITEL ŘÁDU PRÁCE

Knižnice ČSVTS — FEL

101

POPIS JAZYKA FEL - PASCAL - 80 A PŘEKLADAČE PF 80

12,-

Praha 1985

©
ČSVTS — FEL ČVUT

STÁTNÍ TECHNICKÁ KNIHOVNA V PRAZE
1401/86
D 22 941
3.2.86
519.688.3 (035)
800.92 FEL-PASCAL (035)
PV
12,-
3 401 452 --
CS 4405 P S.

Obsah:

Předmluva

1. ÚVOD
   1.1 Souhrnně o Pascalu
   1.2 Souhrnně o implementaci Fel-Pascal-80

2. LEXIKÁLNÍ ELEMENTY
   2.1 Obecně
   2.2 Speciální symboly
   2.3 Vyhrazená slova
   2.4 Identifikátory
   2.5 Čísla
   2.6 Návěští
   2.7 Řetězce znaků
   2.8 Komentáře
   2.9 Operační kódy
   2.10 $-poznámky
   2.11 Lexikální alternativy

3. KOMPILAČNÍ JEDNOTKY
   3.1 Struktura programu
   3.2 Hlavní jednotka programu
      3.2.1 Hlavička programu
      3.2.2 Specifikace
         3.2.2.1 Specifikace port
         3.2.2.2 Specifikace alias
         3.2.2.3 Specifikace common
         3.2.2.4 Specifikace origin
         3.2.2.5 Specifikace global
      3.2.3 Blok
   3.3 Deklarační jednotky

4. DEKLARACE
   4.1 Deklarace návěští
   4.2 Deklarace konstant

4.3 Deklarace typů
4.3.1 Obecně
4.3.2 Jednoduché typy
4.3.2.1 Obecně
4.3.2.2 Standardní jednoduché typy
4.3.2.3 Výčtové typy
4.3.2.4 Typy interval
4.3.3 Strukturované typy
4.3.3.1 Obecně
4.3.3.2 Typy pole
4.3.3.3 Typy záznam
4.3.3.4 Typy množina
4.3.3.5 Typy soubor
4.3.4 Typy ukazatel
4.3.5 Kompatibilita typů
4.4 Deklarace proměnných
4.5 Deklarace procedur a funkcí
4.5.1 Obecně
4.5.2 Deklarace procedury
4.5.3 Deklarace funkce
4.5.4 Formální parametry
4.6 Rozsahy platnosti

5. VÝRAZY
5.1 Obecně
5.2 Proměnné
5.2.1 Obecně
5.2.2 Selektor pole
5.2.3 Selektor záznamu
5.2.4 Dereference ukazatele
5.2.5 Přístupová proměnná souboru
5.2.6 Přetypování proměnné
5.3 Konstruktor množiny
5.4 Operace
5.4.1 Obecně
5.4.2 Aritmetické operace
5.4.3 Boolovské operace
5.4.4 Množinové operace
5.4.5 Relační operace
5.4.6 Operace s ukazateli
5.5 Zápis funkce
5.5.1 Obecně
5.5.2 Standardní funkce
5.5.2.1 Obecně
5.5.2.2 Aritmetické funkce
5.5.2.3 Ostatní funkce

6. PŘÍKAZY
6.1 Obecně
6.2 Jednoduché příkazy
6.2.1 Přiřazovací příkaz
6.2.2 Příkaz procedury
6.2.2.1 Obecně
6.2.2.2 Standardní procedury
6.2.3 Příkaz skoku
6.2.4 Prázdný příkaz
6.3 Strukturované přékazy
6.3.1 Složený příkaz
6.3.2 Příkaz if
6.3.3 Příkaz case
6.3.4 Příkaz repeat
6.3.5 Příkaz while
6.3.6 Příkaz for
6.3.7 Příkaz with

7. SOUBORY
7.1 Logické soubory
7.1.1 Textové soubory
7.1.1.1 Vytváření textových souborů
7.1.1.2 Výstupní konverze
7.1.1.3 Čtení textových souborů
7.1.1.4 Vstupní konverze
7.1.2 Binární soubory

7.1.2.1 Vytváření binárních souborů
7.1.2.2 Čtení binárních souborů
7.2 Fyzické soubory
7.2.1 Pracovní soubory
7.2.2 Permanentní soubory
7.2.3 Fyzické textové soubory
7.2.3.1 Diskové textové soubory
7.2.3.2 Textový soubor na systémovém logickém zařízení
7.2.3.3 Fyzická reprezentace souborů input a output
7.2.4 Fyzické binární soubory
7.2.5 Nestandardní procedury systému ovládání souborů

8. PROMĚNNÉ IDENTIFIKOVANÉ UKAZATELEM
8.1 Ukazatele dynamických proměnných
8.2 Ukazatele statických proměnných

9. STRUKTURA GENEROVANÉHO KÓDU
9.1 Znakový assembler
9.2 Struktura procedur a funkcí
9.2.1 Statické procedury a funkce
9.2.2 Dynamické procedury a funkce
9.2.3 Systémové procedury a funkce
9.3 Vnitřní reprezentace proměnných

10. SESTAVOVÁNÍ PŘELOŽENÉHO PROGRAMU
10.1 Nesegmentované programy
10.2 Segmentované programy
10.3 Knihovny
10.3.1 Obecně
10.3.2 Knihovna PFRLIB
10.3.2 Knihovna PFCLIB
10.3.3 Knihovna PFLIB
10.4 Programy pracující bez operačního systému (aplikační programy)

11. OPTIMALIZACE

12. VOLÁNÍ PŘEKLADAČE A ŘÍZENÍ PRŮBĚHU PŘEKLADU
12.1 Volání překladače pod operačním systémem CP/M
12.2 Použití souboru maker
12.3 Řízení průběhu překladu
12.3.1 Přepínače
12.3.2 $-poznámky

Přílohy:

Přehled syntaxe

PŘEDMLUVA

Programovací jazyk Pascal byl navržen počátkem sedmdesátých let profesorem N.Wirthem na Eidgenossiche Technische Hochschule v Zürichu. Autor sledoval tímto návrhem dva hlavní cíle:

- vytvořit jazyk vhodný pro systematickou výuku programování a založený na jasných, jednoduchých a srozumitelných konstrukcích,
- umožnit realizaci překladačů tohoto jazyka, které by na současných počítačích pracovaly spolehlivě a efektivně.

Dokladem splnění těchto cílů je stále rostoucí popularita Pascalu mezi pedagogy i mezi programátory a v současné době již značné množství jeho implementací na nejrůznějších typech počítačů včetně mikropočítačových systémů. Jazyk Pascal prokázal své dobré vlastnosti nejen při výuce programování (je považován za nejvhodnější programovací jazyk v této oblasti), ale i při praktickém programování celé řady úloh, pro něž je příznačná potřeba práce se složitějšími a převážně nečíselnými datovými strukturami. Efektivita některých implementací umožňuje jeho použití (a tím i zjednodušení programování) i v takových oblastech, kde obvyklým programovacím jazykem je strojově orientovaný jazyk.

Jednou z implemantací jazyka Pascal, které si kladou za cíl nabídnout uživateli spojení efektivity psaní programu s efektivitou výpočtu, je překladač PF 80 na µpočítače řady 8080. Tato implementace vychází z dialektu Pascalu, který byl nazván Fel-Pascal (neboť vznikl na katedře počítačů FEL-ČVUT) a který se vyznačuje některými rozšířeními. Standardní rysy jazyka jsou však implementovány tak, aby vyhovovaly ISO normě jazyka Pascal.

Tato příručka obsahuje popis jazyka Fel-Pascal-80, stručný popis struktury cílového programu a informace potřebné k používání kompilátoru. Od čtenáře očekává základní

znalosti o mikropočítačích řady 8080 a systému CP/M. Popis jazyka zde uvedený nemá formu učebnice, ale příručky. Čtenáři, který nezná jazyk Pascal a nemá dostatek zkušeností z jiných programovacích jazyků, aby mohl nový jazyk zvládnout z příručky, doporučujeme prostudovat nejprve některý učební text určený začátečníkům.

Kompilátor je koncipován tak, aby umožňoval psát efektivní aplikační programy pro mikropočítače spolupracující s nějakým technickým zařízením. Uživatel může psát programy, které spolupracují s libovolným monitorem a nebo pracují vůbec bez monitoru. Systém je vybaven prostředky pro snímání a zpracování dat z AD převodníků a pro výstup na DA převodníky. Je tedy možné přímo z pascalského programu ovládat hardware. Kompilátor je vybaven rozsáhlými optimalizacemi, které zkracují vygenerovaný kód, a takovou koncepcí podpůrných knihoven, která minimalizuje délku připojovaných knihovních podprogramů.

Kompilátor PF 80 byl vytvořen na elektrotechnické fakultě ČVUT metodou autoimplementace pomocí křížového překladu na počítači TESLA 200 a později na počítači SM4-20. Na tvorbě systému se podíleli tito pracovníci:

Ing. Jiří Daněček - 1.,2. a 3. průchod překladače, návod
Ing. Zdeněk Muzikář - 4. průchod překladače, přenos pod DOS RV
Ing. Jan Zajíc - 5.,6. a 7. průchod překladače, knihovna
Ing. Josef Vogel - 8. průchod překladače, návod, přenos pod DOS RV
Ing. František Drdák - knihovna, přenos pod CP/M

Na tomto místě chceme poděkovat všem, kteří se na projektu jakýmkoliv způsobem podíleli a umožnili jeho dokončení.

Autoři

1. ÚVOD

1.1 Souhrnně o Pascalu

Na tomto místě se stručně zmíníme o základních pojmech a vlastnostech standardního Pascalu. Rozšíření Fel-Pascalu-80 jsou vyjmenována v následujícím čl.1.2.

Program v jazyku Pascal obsahuje popisy akcí, které mají být provedeny, a popisy dat, s nimiž akce operují. Popisy akcí se nazývají příkazy, popisy dat jsou obsahem deklarací.

Data jsou reprezentována konstantami a proměnnými. Každá konstanta nebo proměnná musí být před svým použitím deklarována. Deklarací konstanty se zavádí identifikátor označující neměnnou hodnotu nějakého typu. Deklarací proměnné se zavádí identifikátor označující datový objekt s proměnnou hodnotou. Množina přípustných hodnot proměnné je dána typem proměnné, který je stanoven při deklaraci proměnné.

Typ specifikuje nejen přípustnou množinu hodnot, ale též množinu operací, které jsou pro datové objekty daného typu definovány. Typ může být standardní, v tom případě je definován jazykem, nebo může být definován v programu. Pro označení typu slouží identifikátory typů, které se zavádějí deklaracemi typů. Typy se dělí na jednoduché, strukturované a typy ukazatel.

Množiny hodnot jednoduchých typů jsou uspořádané. Zvláštní skupinu jednoduchých typů tvoří ordinální typy, jejichž hodnoty jsou uspořádány podle ordinálních čísel, které jsou těmto hodnotám přiřazeny. Standardními ordinálními typy jsou typy integer, boolean a char, standardním jednoduchým typem je dále typ real. Nové ordinální typy se definují výčtem identifikátorů označujících hodnoty nového typu. Z ordinálních typů lze vytvářet intervaly, které specifikují souvislé podmnožiny hodnot ordinálních typů.

Datové objekty strukturovaných typů se skládají ze složek. Jsou zavedeny čtyři třídy strukturovaných typů (v jejichž rámci se definují konkrétní strukturované typy): typ pole, typ záznam, typ množina a typ soubor.

Objekty typu pole se skládají z pevného počtu složek stejného typu. Složky pole se rozlišují pomocí indexů, což jsou hodnoty ordinálních typů. Typ složek pole a typy indexů jsou určeny popisem typu pole (tj. konstrukcí, která konkrétní typ pole zavádí).

Objekty typu záznam se skládají z pevného počtu složek, které nemusí být stejného typu a které se rozlišují pomocí identifikátorů. Počet složek záznamu, jejich identifikátory a typy jsou určeny popisem typu záznam. Konkrétní typ záznam může mít několik variant. To umožňuje, aby různé proměnné byly formálně téhož typu, přestože mají různou strukturu.

Hodnotou objektu typu množina může být libovolná podmnožina množiny hodnot bázového typu. Bázový typ je určen popisem typu množina a může to být pouze ordinální typ.

Objekt typu soubor je strukturován jako libovolně dlouhá posloupnost složek stejného typu. Tyto objekty reprezentují v pascalském programu obvyklé soubory dat uložené ve vnějších pamětech. Jsou to objekty s postupným výběrem složek, tzn. v každém okamžiku je přímo přístupná nanejvýš jedna složka souboru.

Kromě proměnných zavedených deklaracemi se mohou používat také dynamické proměnné vytvořené při výpočtu. Dynamické proměnné jsou identifikovány pomocí hodnot typu ukazatel. Každý typ ukazatel má přitom určen doménový typ, tj. typ proměnných, které jsou hodnotami daného typu ukazatel identifikovány.

Příkazy se dělí na jednoduché a strukturované. Jednoduchými příkazy jsou přiřazovací příkaz, příkaz procedury, příkaz skoku a prázdný příkaz. Strukturovanými příkazy se předepisuje postupné, podmíněné nebo opakované provádění

dílčích příkazů. Strukturovanými příkazy jsou složený příkaz, podmíněné příkazy, příkazy cyklu a příkaz with.

Přiřazovacím příkazem se předepisuje výpočet hodnoty a jejího přiřazení proměnné nebo složce proměnné. Operace přiřazení hodnoty je definovaná pro všechny typy s výjimkou typu soubor, každé proměnné však lze přiřadit pouze takovou hodnotu, kterou připouští její typ. Předpis pro výpočet hodnoty má formu výrazu, který se skládá z operandů a operátorů. Operandem může být proměnná nebo její složka, konstanta, množina nebo zápis funkce. Operátory se dělí na aritmetické, boolovské, množinové a relační. Pro každý operátor jsou stanoveny přípustné typy operandů a typ výsledku.

Příkazem procedury se vyvolá procedura. Procedura je buď standardní nebo je deklarovaná v programu. Procedury mohou mít parametry čtyř druhů: parametry volané hodnotou, parametry volané odkazem, procedurální parametry a funkcionální parametry. Totéž platí pro funkce, které se vyvolávají zápisem funkce.

Příkaz skoku slouží pro předání řízení příkazu označenému návěštím. V pascalském programu se používá jen zřídka, převážně jako prostředek pro předčasné ukončení strukturovaného příkazu.

Složeným příkazem se předepisuje postupné provedení dílčích příkazů v něm uvedených. Používá se především jako příkazové závorky, tzn. tehdy, když posloupnost příkazů má být chápána jako jediný příkaz.

Podmíněnými příkazy jsou příkaz if a příkaz case. Příkazem if se předepisuje větvení programu do dvou alternativ (z nichž jedna může být prázdná) na základě hodnoty výrazu typu boolean. Příkazem case se předepisuje větvení programu do více variant na základě hodnoty výrazu ordinálního typu.

Příkazy cyklu jsou příkazy while, repeat a for. Příkaz while opakuje provádění dílčího příkazu, pokud je spl-

něna nějaká podmínka. Příkaz repeat opakuje provádění dílčích příkazů až do splnění nějaké podmínky. Příkazem for se předepisuje opakované provedení dílčího příkazu pro nějakou posloupnost hodnot ordinálního typu.

Zvláštním strukturovaným příkazem je příkaz with, pomocí něhož lze zkráceně vyjádřit několikanásobný přístup ke složkám proměnné typu záznam.

Pascal je jazyk s blokovou strukturou. Blok je konstrukce, která obsahuje deklarace a příkazy a tvoří rámec platnosti deklarací v ní uvedených. Ve formě deklarací procedur a funkcí mohou být bloky do sebe vnořovány. S vnořováním bloků souvisí pravidla o lokalitě objektů a viditelnosti nelokálních objektů.

Standardní vlastnosti jazyka Pascal jsou definovány normou ISO 7185, která rozlišuje dvě úrovně jazyka. Výše uvedená charakteristika odpovídá úrovni 0, úroveň 1 se liší tím, že zavádí schema konformního pole. Tento druh parametrů umožňuje používat procedury a funkce, jejichž parametrem je pole nespecifikované délky (volané hodnotou nebo odkazem).

1.2 Souhrnně o implementaci Fel-Pascal-80

Jazyk Fel-Pascal-80 vychází z úrovně 1 ISO normy Pascalu, poskytuje však mnohé další prostředky. Parametrem překladu lze určit, že překladač bude akceptovat jen programy ve standardním Pascalu, implicitně však systém poskytuje tato rozšíření:

1. Příkaz case je rozšířen o část otherwise, která se provede v případě, že index větvení má hodnotu různou od všech uvedených konstant.
2. Operace s ukazateli jsou rozšířeny tak, aby bylo možné ukazateli přiřadit i adresu deklarované proměnné a hodnotu ukazatele "posunout" o délku vnitřní representace. Kompatibilní jsou i ukazatelé se stejným doménovým typem.

3. Celá čísla je možno zapisovat i v oktalovém nebo hexadecimálním tvaru.
4. Je možné provádět konverzi celočíselných typů na výčtové typy.
5. Pro celočíselné typy jsou předdefinovány standardní funkce low a high, které reprezentují nižší a vyšší byte slova, ve kterém je hodnota uložena.
6. Je definován příkaz for bez řídící proměnné, který pouze počítá průchody cyklem.
7. Speciální konstrukcí je možno měnit typ proměnné.
8. Procedury pack a unpack slouží k přesunu a nezáleží na tom, zda jsou parametry zhuštěná či nezhuštěná pole. Prvním parametrem procedury unpack může být i konstanta typu řetězec znaků.
9. Byl doplněn další druh parametrů procedur a funkcí - konstantní parametry (i pro konformní schéma pole).
10. V příkazu procedury nebo při zápisu funkce je možno zapsat bezvýznamný parametr jako prázdný.
11. Příkazem write je možno provádět při ladění výpisy úseků paměti.
12. Kompatibilita vzhledem k přiřazení je rozšířena tak, že "krátké" konstanty typu řetězec znaků jsou doplněny zprava mezerami.
13. Úseky deklarací návěstí, konstant, typů a proměnných v bloku se mohou opakovat a na jejich pořadí nezáleží.
14. Zdrojový program je možno rozdělit do několika samostatně překládaných kompilačních jednotek. Sestavením modulů, které kompilátor překladem kompilačních jednotek vytvořil a připojením potřebných knihovních modulů vznikne cílový program, který může být i segmentován. Cílový program může obsahovat i moduly vzniklé překladem z jazyka PL/M nebo assembleru. Komunikace mezi kompilačními jednotkami je založena na principu globálních a

externích proměnných, procedur a funkcí; je možná i vazba prostřednictvím fortranských bloků společných proměnných (COMMON), a to jak pojmenovaných, tak i bezejmenných.

15. Globální a externí objekty je možno přejmenovat tak, že lze pracovat i se systémovými identifikátory (např. $$ALFA,FLNG).
16. Externím objektům je možno přidělit absolutní adresy.
17. Externí proměnné je možno pomocí specifikace port ztotožnit s porty a tak provádět efektivně vstupní a výstupní operace.
18. Pomocí přepínačů překladu a tzv. $-poznámek je možné řídit způsob překladu a tvar protokolu o překladu a tak vytvářet co nejefektivnější překlad kompilační jednotky.
19. Základním produktem kompilátoru je tzv. relativní modul. Parametr překladu však může určit, že výstupem bude modul v assembleru, který lze případně podrobit ručním zásahům pomocí editoru.
20. Pomocí standardní procedury inline je možno přímo do Pascalského programu vkládat kusy kódu napsané v assembleru.
21. Knihovna systému obsahuje řadu nestandardních procedur, které jsou uživateli přístupné a rozšiřují tak možnosti systému.

Kompilátor PF 80 je rozsáhlý segmentovaný program. Vstupující zdrojový text je zpracován v osmi průchodech:

1. Lexikální analýza
2. Syntaktická analýza
3. Zpracování sémantiky
4. Optimalizace vnitřní formy programu
5. Generování vnitřního assembleru
6. - " -
7. Generování relativního modulu nebo makroassembleru
8. Výpis protokolu o překladu

Kompilátor PF 80 je pokračováním řady kompilátorů jazyka Fel-Pascal, které jsou vytvořeny pro systémy Tesla 200, ADT a SMEP. S výjimkou systémově závislých konstrukcí (např. portů) je jazyk Fel-Pascal-80 shodný s jazykem Fel-Pascal-SMEP a je rozšířením jazyka Fel-Pascal-ADT. Kromě základní verze kompilátoru pod systém CP/M, je k dispozici také křížová verze na počítačích řady SMEP pod systémem DOS RV.

2. LEXIKÁLNÍ ELEMENTY

2.1 Obecně

Lexikální elementy jsou

- operační kódy,
- speciální symboly,
- vyhrazená slova,
- identifikátory,
- čísla,
- návěští,
- řetězce znaků.

Vytvářejí se ze znaků abecedy ASCII. V identifikátorech, ve vyhrazených slovech a v číslech nerozlišuje překladač malá a velká písmena.

Za oddělovače lexikálních elementů se považují

- mezery,
- oddělovače řádků,
- tabulátory (09H) a znaky FF (0CH),
- komentáře,
- $-poznámky.

Mezi dvěma po sobě jdoucími lexikálními elementy nebo před prvním lexikálním elementem kompilační jednotky nemusí být žádný nebo může být libovolný počet oddělovačů lexikálních elementů. Za identifikátorem, vyhrazeným slovem, číslem bez znaménka nebo návěštím však musí být alespoň jeden oddělovač lexikálních elementů, jestliže následující lexikální element je opět identifikátor, vyhrazené slovo, číslo bez znaménka nebo návěští. Uvnitř lexikálního elementu nesmí být žádný oddělovač lexikálních elementů.

Příklad:

<u>for</u>I:=1<u>to</u>10<u>do</u>	nesprávný zápis
<u>for</u> I:=1 <u>to</u> 10 <u>do</u>	nutné oddělení lexikálních elementů

Oddělovače řádků se musí používat tak, aby žádný řádek zdrojového textu kompilační jednotky neobsahoval více než 80 znaků.

Aby bylo možno bez problémů vypisovat protokol o překladu na terminálu, který má 80 znaků na řádku, je třeba, aby délka vstupního řádku nepřesáhla 64 znaků.

2.2 Speciální symboly

Speciální symboly jsou lexikální elementy, které mají speciální význam; používají se jako operátory a omezovače syntaktických konstrukcí. Speciálními symboly jsou následující znaky, resp. dvojznaky:

```
+   -   *   /   =   <   >   [   ]   .   ,
:   ;   ^   (   )   <>  <=  >=  :=  ..
```

2.3 Vyhrazená slova

Vyhrazená slova jsou posloupnosti písmen, které mají speciální význam. Používají se jako operátory a omezovače syntaktických konstrukcí. V této příručce jsou vyhrazená slova zvýrazněna podtržením, ve zdrojovém textu kompilační jednotky se píší bez podtržení. Vyhrazené slovo nesmí být použito jako identififikátor.

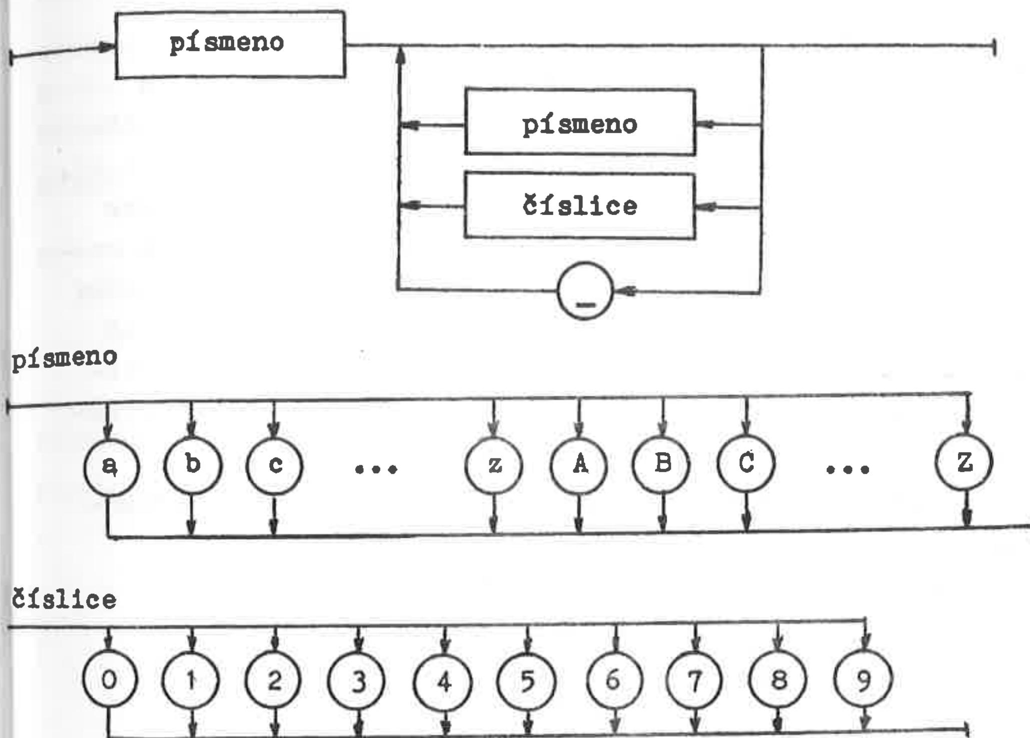
Vyhrazenými slovy jsou tato slova:

alias	and	array	begin	case
common	const	div	do	downto
dynamic	else	end	external	file
for	fortran	forward	function	global
goto	if	in	label	mod
modul	nil	not	of	or
origin	otherwise	packed	procedure	program
record	repeat	set	static	system
then	to	type	until	var
while	with	port		

2.4 Identifikátory

Identifikátory se používají pro označování konstant, typů, proměnných, procedur a funkcí. Jsou to posloupnosti písmen, číslic a znaků podtržení "_" začínající písmenem. Identifikátory mohou mít libovolnou délku. Velikost písmen není důležitá. Identifikátor se nesmí shodovat se žádným vyhrazeným slovem. Identifikátory se rozlišují podle všech znaků.

identifikátor



Příklady:

X CAS CTIINTEGER Cti_Integer A1

Význam identifikátoru je dán buď jazykem (předdefinované identifikátory) nebo deklarací v programu. Výskyt identifikátoru v konstrukci, která definuje jeho význam, se nazývá definičním výskytem nebo též definičním místem identifikátoru. Definiční výskyty jsou v syntaktických diagramech vyjádřeny značkou

Každému definičnímu výskytu odpovídá rozsah platnosti, v němž lze identifikátor odpovídajícím způsobem použít. Použití definovaného identifikátoru je v syntaktických diagramech vyjádřeno značkou, ve které je ke slovu "identifikátor" připojen přívlastek udávající požadovanou vlastnost identifikátoru, například

Každý identifikátor musí být nejprve definován a teprve potom může být použit (jedinou výjimkou je použití identifikátoru typu v popisu typu ukazatel, viz 4.3.4). V každém rozsahu platnosti může mít identifikátor pouze jeden definiční výskyt. Konstrukce, které tvoří rámec rozsahu platnosti, se však mohou do sebe vnořovat. Definiční výskyt ve vnořené konstrukci přitom zastiňuje nelokální význam identifikátoru. Podrobná pravidla vymezující rozsahy platnosti, definiční výskyty a používání identifikátorů jsou uvedena v čl.4.6.

Jazykem jsou předdefinovány následující identifikátory:

abs	arctan	boolean	char	chr
cos	dispose	eof	eoln	exp
false	get	high	input	integer
ln	low	maxint	new	odd
ord	output	pack	page	pred
put	read	readln	real	ref
reset	rewrite	round	sin	size
sqr	sqrt	succ	text	true
trunc	unpack	write	writeln	inline

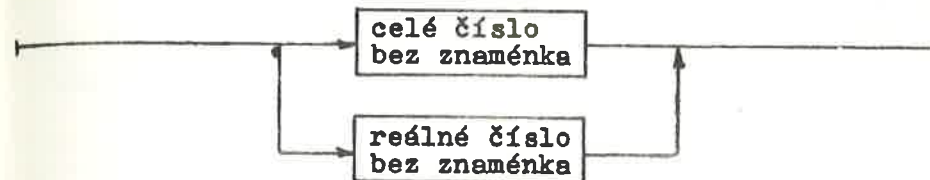
Předdefinované identifikátory označují standardní konstanty, typy, proměnné, procedury a funkce. Význam předdefinovaného identifikátoru však může být změněn definicí v programu.

Kromě standardních procedur a funkcí obsahuje knihovna kompilátoru ještě řadu dalších procedur a funkcí, které však nejsou označeny předdefinovanými identifikátory. Při jejich použití proto musí být odpovídající identifikátory definovány v programu jako identifikátory externích procedur a funkcí.

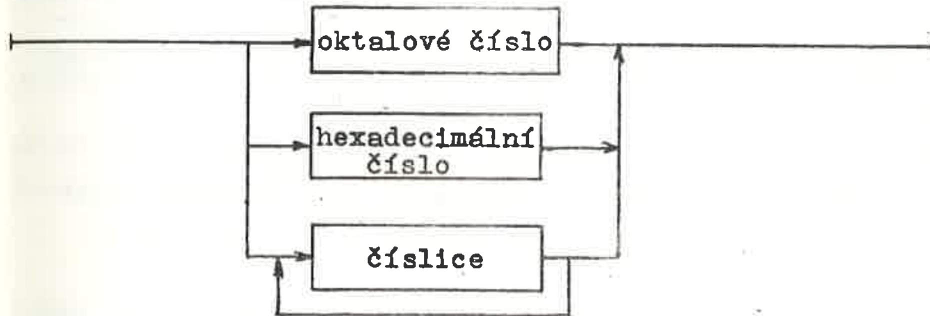
2.5 Čísla

Čísla jsou literály označující číselné hodnoty. Ve zdrojovém programu je lexikálním elementem číslo bez znaménka, případně předcházející znaménko se chápe jako operátor.

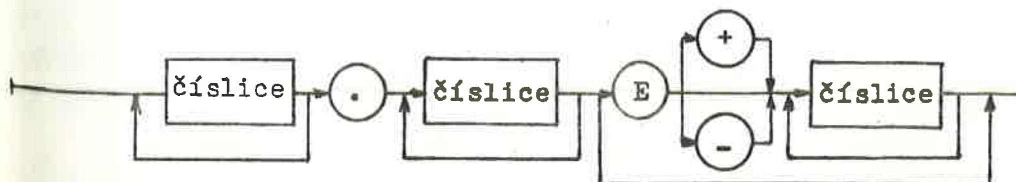
číslo bez znaménka



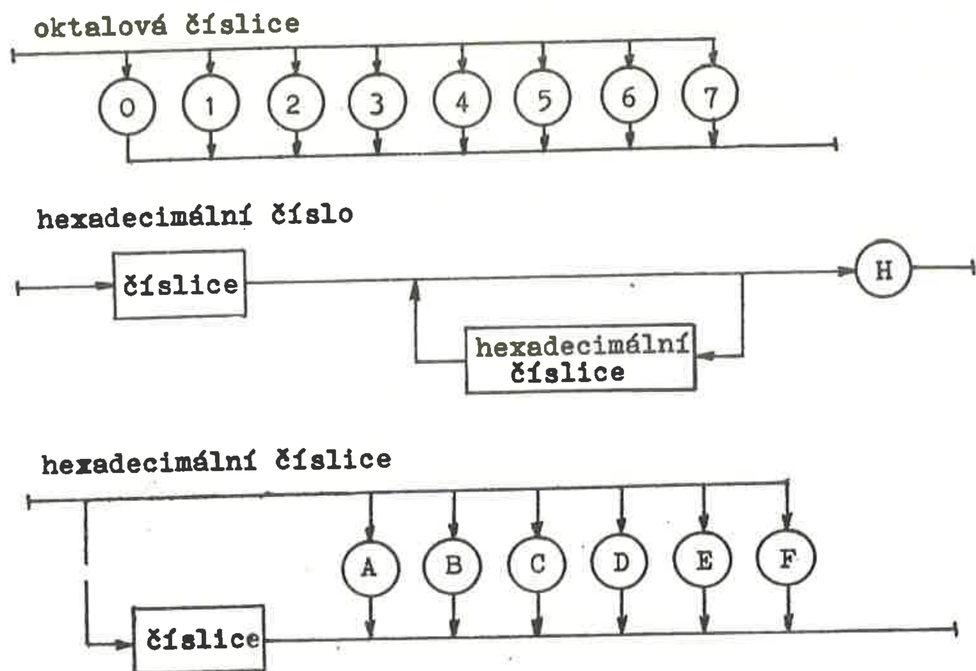
celé číslo bez znaménka



reálné číslo bez znaménka



oktalové číslo



Celé číslo bez znaménka je zápis hodnoty typu integer (viz 4.3.2.2). Oktalové číslo je oktalový zápis hodnoty typu integer. Hexadecimální číslo je hexadecimální zápis hodnoty typu integer. Hexadecimální číslo musí začínat dekadickou číslicí (je-li to třeba i nulou).

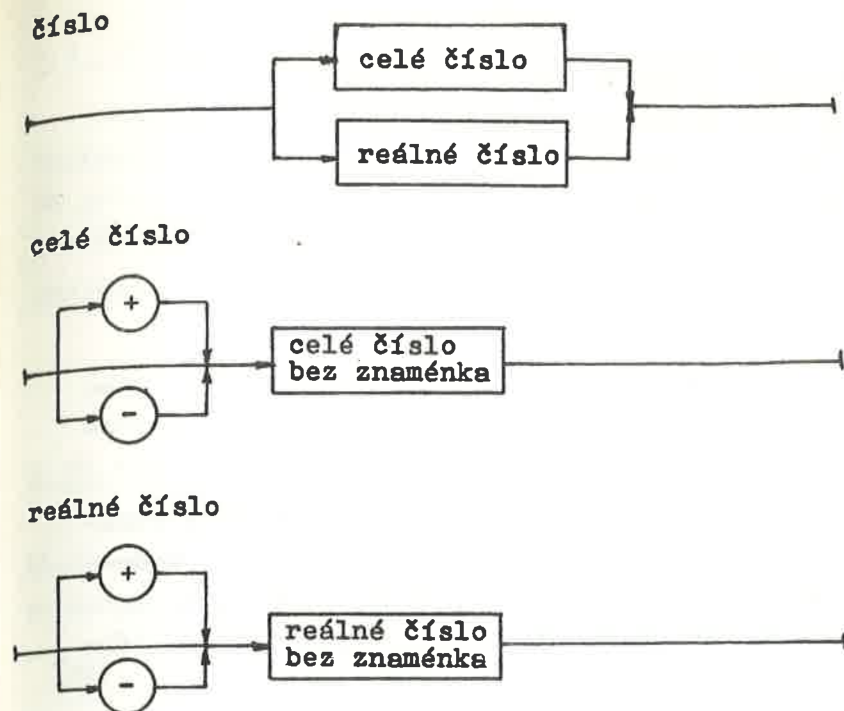
Je-li hodnota celého čísla bez znaménka větší než maxint (maxint je standardní konstanta typu integer s hodnotou 32767), je hlášena varovná zpráva a nad hodnotou je provedena operace mod 2^{16}.

Reálné číslo bez znaménka je dekadický zápis hodnoty typu real (viz 4.3.2.2). Písmeno "E" znamená "krát deset na".

Příklady:

1E10 1 120 5E-3 87.35E+8 47B 100001B
0A7H 96FFH

Zápisy čísel ve vstupním textovém souboru, který je zpracován standardní procedurou read resp. readln (viz 7.1.1.3) mají následující syntaxi:



2.6 Návěští

Návěští slouží pro označení příkazu, na který směřuje příkaz skoku (viz 6.2.3).

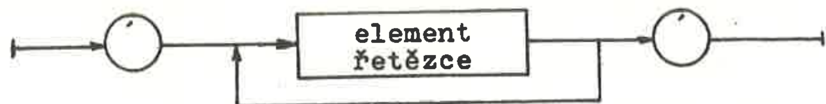
návěští

Návěští se rozlišují svou numerickou hodnotou. Je-li hodnota návěští mimo interval 0..9999 je hlášena varovná zpráva. Stejně jako u identifikátorů i u návěští se rozlišují definiční výskyty a jejich použití. Definiční výskyt návěští je v úseku deklarací návěští (viz 4.1), použitím je návěští příkazu a výskyt návěští v příkazu skoku.

2.7 Řetězce znaků

Řetězce znaků jsou literály označující posloupnosti znaků.

řetězec znaků



element řetězce

Znakem řetězce je libovolný ze znaků ASCII s výjimkou znaku apostrof; znak apostrof je v řetězci reprezentován dvojicí "´´".

Řetězec znaků obsahující jediný element řetězce označuje hodnotu typu char (viz 4.3.2.2). Řetězec znaků obsahující n elementů řetězce, n>1, označuje hodnotu typu <u>packed array</u> [1..n] <u>of</u> char, tj. hodnotu typu řetězec délky n (viz 4.3.3.2).

Příklady:

´A´	literál typu char
´;´	- " -
´´´´	- " -
´Pascal´	literál typu <u>packed</u> <u>array</u> [1..6] <u>of</u> char
´TOTO JE RETEZEC´	literál typu <u>packed</u> <u>array</u> [1..15] <u>of</u> char

2.8 Komentáře

Komentářem je konstrukce

{posloupnost znaků}

ve které posloupnost znaků neobsahuje znak "}" a znak "{" se nevyskytuje v řetězci znaků ani v komentáři. Protože se znaky "{", "}" jsou na některých terminálech problémy, doporučujeme používat pro oddělovače komentářů alternativní symboly (viz 2.11).

Příklad: {toto je komentar}

2.9 Operační kódy

Jako lexikální symboly jsou přípustné operační kódy instrukcí assembleru, před které je předřazen znak ´"´. Za jeden operační kód se považuje také celá instrukce MOV s parametry (viz 6.2.2.2).

Příklad:

"MOV A,B   "LHLD

2.10 $-poznámky

Pomocí $-poznámek se nastavují hodnoty parametrů překladu během kompilace. V jedné $-poznámce lze definovat hodnoty několika parametrů. $-poznámka má tvar

{$p1,p2,...pn posloupnost znaků, která nezačíná ","}

kde p1, p2,...pn jsou definice hodnot parametrů překladu a posloupnost znaků je komentářem. $-poznámky je možno umístit i mimo komentářové závorky. V tomto případě musí být v každé $-poznámce uveden jediný parametr překladu. Seznam všech parametrů překladu, jejichž hodnoty lze definovat $-poznámkami, je uveden v kap. 12.

Příklady:

{$R+,X-} $R+ $X- {$L+} $L+ {$Z2+} $Z2+ $J3+ $P

2.11 Lexikální alternativy

Některé speciální symboly je možno nahradit jinými symboly nebo dvojicemi znaků podle následující tabulky:

^	@
[	(.
]	.)
{	(*
}	*)

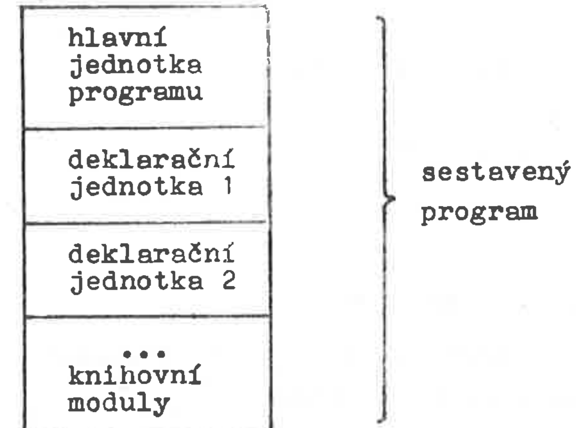
3. KOMPILAČNÍ JEDNOTKY

3.1 Struktura programu

Program v jazyku Fel-Pascal-80 může být rozdělen do několika kompilačních jednotek, které kompilátor překládá samostatně a nezávisle. Kompilačními jednotkami jsou:

- hlavní jednotka programu,
- deklarační jednotka

Každý pascalský program musí obsahovat právě jednu hlavní jednotku programu. V této jednotce mohou být využívány externí proměnné, procedury a funkce, které jsou deklarovány a specifikovány jako globální v deklaračních jednotkách (totéž platí i naopak a vzájemně mezi deklaračními jednotkami). Po samostatném překladu všech potřebných jednotek vznikne sestavením (při němž se připojí knihovní moduly) cílový nesegmentovaný program (viz obr.).



Obr. Struktura nesegmentovaného programu

Pascalský program může být rovněž segmentován. Sestavený program se v tom případě skládá z kořene, který začíná hlavní jednotkou programu a může, kromě knihovních modulů, obsahovat deklarační jednotky, a řady segmentů, které

se v paměti vzájemně překrývají. Každý segment obsahuje jednu nebo více deklaračních jednotek a eventuelně další knihovní moduly. Další podrobnosti o segmentaci viz 10.2.

3.2 Hlavní jednotka programu

Hlavní jednotka programu se skládá z hlavičky programu, specifikací a bloku a je zakončena tečkou. Sama o sobě může tvořit program.

hlavní jednotka programu

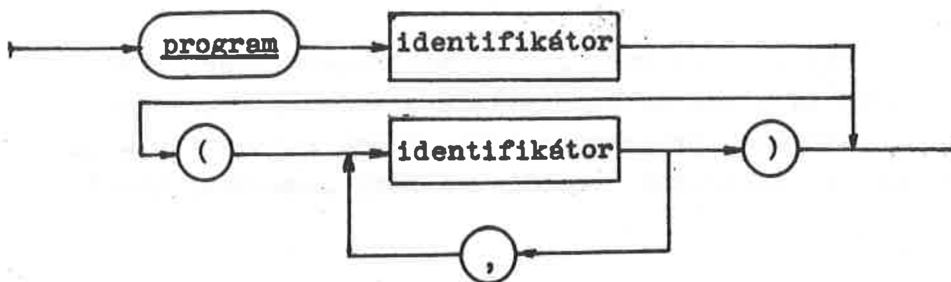
Příklad jednoduchého programu:

```
program SIMPLE(input, output);
var N : integer; S : real;
begin
   read(input, N); S := 0;
   repeat
      S := S+1/N; N := N-1
   until N=0;
   write(output, S)
end.
```

3.2.1 Hlavička programu

Hlavička programu obsahuje identifikátor programu a případně seznam parametrů programu.

hlavička programu



Identifikátor programu nemá žádný význam pro vlastní program. Pokud je kompilační jednotka přeložena do assembleru, je jeho prvních šest znaků použito v pseudoinstrukcích TITLE a NAME.

Každý parametr programu musí být identifikátor proměnné typu soubor. Pokud se nejedná o standardní soubory input nebo output, musí každému parametru odpovídat deklarace proměnné typu soubor na základní úrovni bloku programu. S výjimkou souborů input a output, mají parametry programu pouze mnemotechnický význam (viz.kap.7).

Seznam parametrů programu nesmí obsahovat dva stejné identifikátory.

Příklady hlavičky programu:

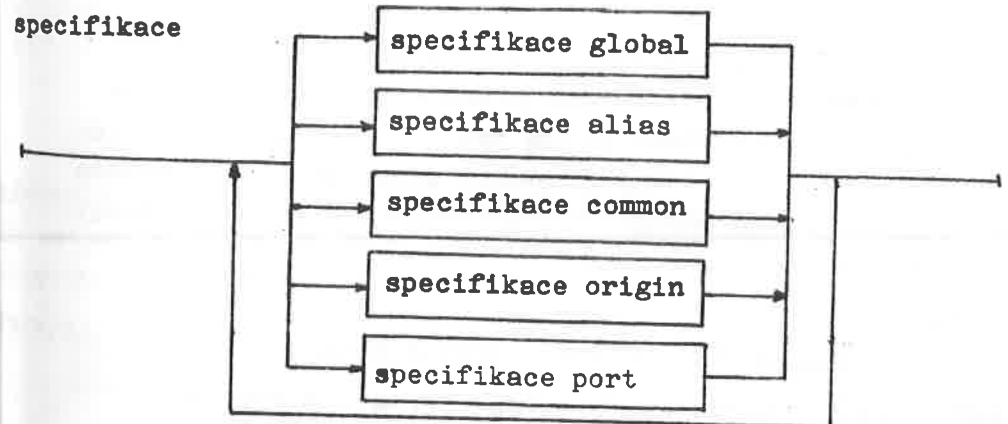
```
program TEST
program AREA (input, output)
program FILE_MERGE(MASTER_FILE, UPDATES_FILE, NEW_MASTER)
```

Poznámka: Výskyty identifikátorů v hlavičce programu nejsou definičními místy s výjimkou identifikátorů input a output (viz kap.7).

3.2.2 Specifikace

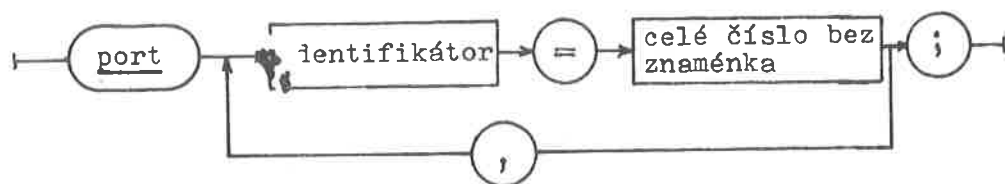
Specifikace určuje vazby dané kompilační jednotky na její okolí, ať už je tvořeno jinými kompilačními jednotkami (v Pascalu, assembleru atd.) nebo hardwarem.

specifikace



3.2.2.1 Specifikace port

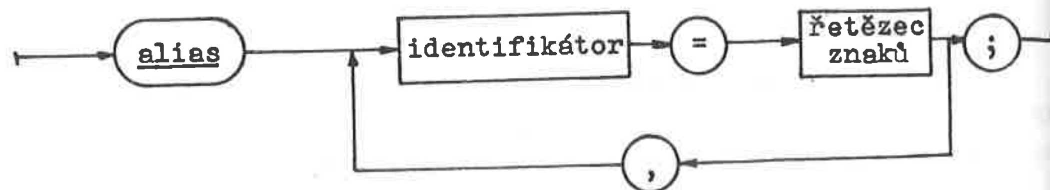
specifikace port



Identifikátory uvedené v seznamu musí být identifikátory externích proměnných (viz 4.4). Proměnné musí být typu s délkou 1 byte. Tyto proměnné budou v programu chápány jako porty určených adres. Tyto proměnné nesmí být předávány jako skutečné parametry za parametry volané odkazem (viz 4.5.4) a nelze na ně provádět operaci ref (viz 8.2) s výjimkou použití v proceduře inline v instrukcích IN a OUT (viz 6.2.2.2).

3.2.2.2 Specifikace alias

specifikace alias



Identifikátory uvedené v seznamu musí být identifikátory externích či globálních proměnných, procedur nebo funkcí; jde-li o proměnné, mohou být i v seznamu specifikace common. Jméno objektu v produktu kompilátoru pak nebude určeno prvými šesti znaky identifikátoru, ale prvými šesti znaky řetězce.

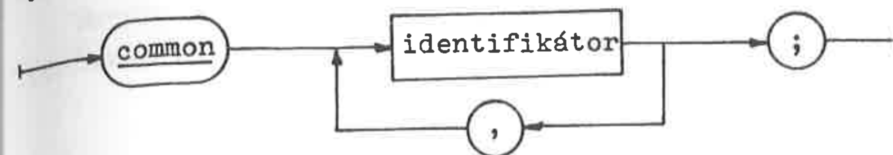
Např.

alias XXX='$$XXX', YYY='F.RDX$';

V assembleru budou namísto identifikátorů XXX,YYY identifikátory $$XXX a F.RDX$.

3.2.2.3 Specifikace common

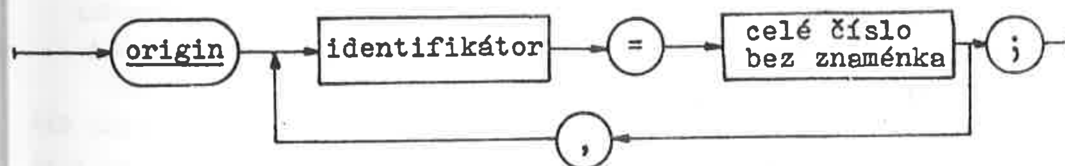
specifikace common



Identifikátory uvedené v seznamu musí být identifikátory externích proměnných (viz 4.4). Tyto proměnné nesmí být ve specifikaci global v jiných kompilačních jednotkách. Prvních 6 znaků jména proměnné pak tvoří jméno společného bloku proměnných, tvořeného právě touto jednou proměnnou. V každé kompilační jednotce, ve které je tato proměnná specifikována jako common, musí být stejného typu (nebo mít alespoň stejnou délku).

3.2.2.4 Specifikace origin

specifikace origin



Identifikátory uvedené v seznamu musí být identifikátory externích proměnných, procedur nebo funkcí. Těmto externím objektům bude v produktu kompilátoru přiřazena absolutní adresa namísto jejich identifikátoru.

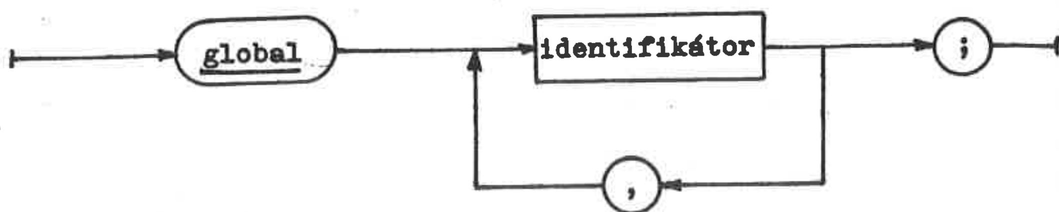
Příklad:

origin BDOS = 5, FDT = 5CH;

3.2.2.5 Specifikace global

Specifikace global obsahuje seznam těch identifikátorů proměnných, procedur a funkcí, které jsou deklarovány v dané kompilační jednotce a mohou být využity i v jiných kompilačních jednotkách (jsou-li v nich deklarovány jako externí).

specifikace global



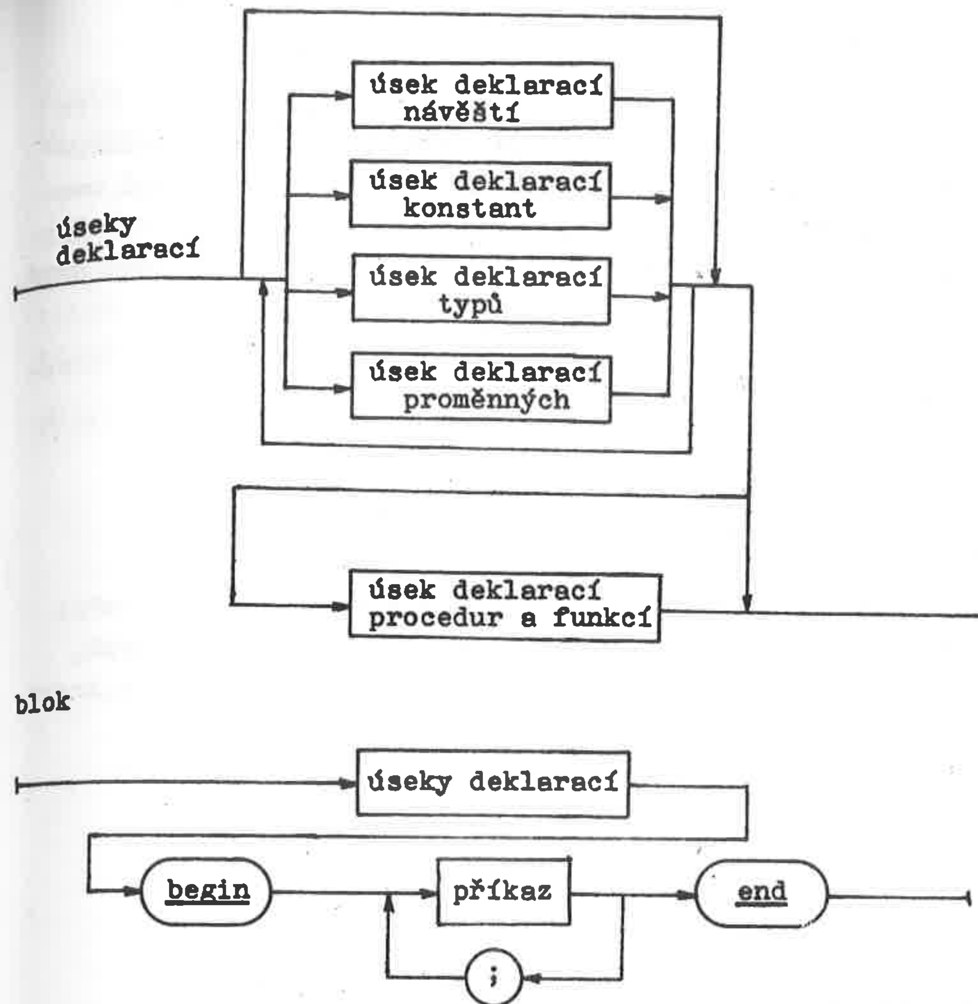
Globálem hlavní jednotky programu může být libovolný identifikátor proměnné, procedury nebo funkce, který je definován na základní úrovni jejího bloku, nebo standardní identifikátory input a output (jsou-li uvedeny mezi parametry programu). V rámci jednoho programu se musí všechny globály vzájemně lišit v prvních 6 znacích. Seznam globálů nesmí obsahovat dva stejné identifikátory.

Příklady specifikace global:

<u>global</u> INIT, SOTZ;
<u>global</u> output;

3.2.3 Blok

Blok je konstrukce, která obsahuje deklarace a příkazy a tvoří rámec platnosti deklarací v ní uvedených.



Blok je základem nejen hlavní jednotky programu, ale též deklarací procedur a funkcí. Bloky tedy mohou být do sebe vnořovány.

Řekneme, že identifikátor konstanty, typu, proměnné apod. je definován na základní úrovni bloku B, jestliže jeho definiční místo v bloku B není obsaženo v bloku, který je do B vnořen.

Jednotlivé úseky deklarací obsahují definiční místa identifikátorů a návěští; rozsahy platnosti těchto definic nepřesahují daný blok. Struktura úseků deklarací je popsána v kap.4.

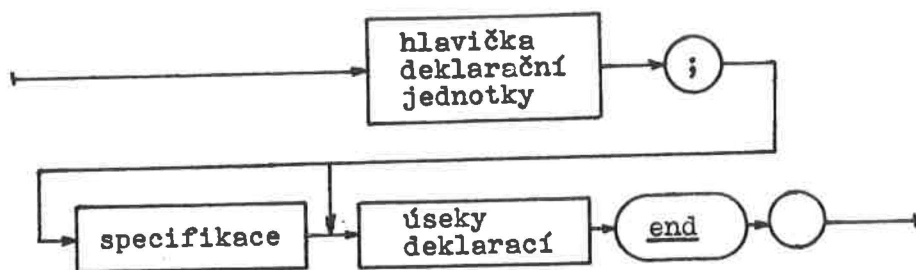
Příkazová část bloku popisuje vlastní výpočet. Tato část má strukturu složeného příkazu, tzn. je to posloupnost příkazů oddělovaných středníkem a uzavřená mezi omezovače begin a end. Při provádění bloku se jako první provede příkaz, který je v této posloupnosti uveden jako první, provádění končí provedením posledního příkazu.

Provedení programu znamená provedení bloku jeho hlavní jednotky.

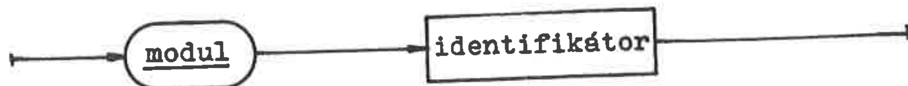
3.3 Deklarační jednotky

Deklarační jednotka se skládá z hlavičky deklarační jednotky, specifikací a úseků deklarací konstant, typů, proměnných a funkcí. Neobsahuje příkazovou část. Deklarace v ní uvedené platí pouze v dané jednotce.

deklarační jednotka



hlavička deklarační jednotky



Stejnou syntaxi jako deklarační jednotka má hlavní jednotka segmentu.

Identifikátor uvedený v hlavičce deklarační jednotky se chová stejně jako identifikátor v hlavičce programu.

Řekneme, že identifikátor je definován na základní úrovni deklarační jednotky, jestliže jeho definiční místo

v této jednotce není obsaženo v bloku, který je do jednotky vnořen.

Specifikace global deklarační jednotky obsahuje seznam jejích globálů a má stejnou syntaxi, jako v případě hlavní jednotky programu. Globálem deklarační jednotky může být libovolný identifikátor proměnné, procedury nebo funkce, který je definován na základní úrovni jednotky.

Příklad deklarační jednotky:

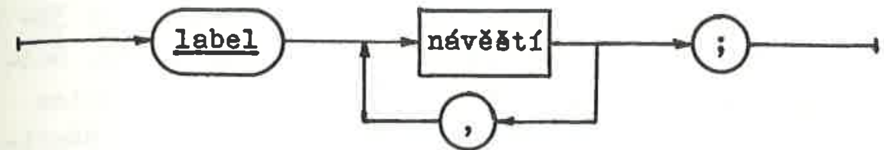
```
modul STACK;
global INIT, PUSH, POP, EMPTY;
const
   MAXLEN = 100;
var
   STC : array [1..MAXLEN] of integer;
   TOP : 0..MAXLEN;
procedure ERRSTC(NER  : char); external;
procedure INIT;
begin TOP:=0 end;
procedure PUSH(X:integer);
begin
   if TOP=MAXLEN then ERRSTC('F') else
   begin TOP:=TOP+1; STC[TOP]:=X end
end;
procedure POP(var X:integer);
begin
   if TOP=0 then ERRSTC('E') else
   begin X:=STC[TOP]; TOP:=TOP-1 end
end;
function EMPTY:boolean;
begin EMPTY:=TOP=0 end;
end.
```

4. DEKLARACE

4.1 Deklarace návěští

Úsek deklarací návěští v bloku b obsahuje seznam všech návěští, která v příkazové části bloku jsou použita jako návěští příkazu. Každému deklarovanému návěští musí v příkazové části bloku b odpovídat právě jeden příkaz, který je tímto návěštím označen.

úsek deklarací návěští



Výskyt návěští v úseku deklarací návěští je definičním místem návěští.

Příklad:

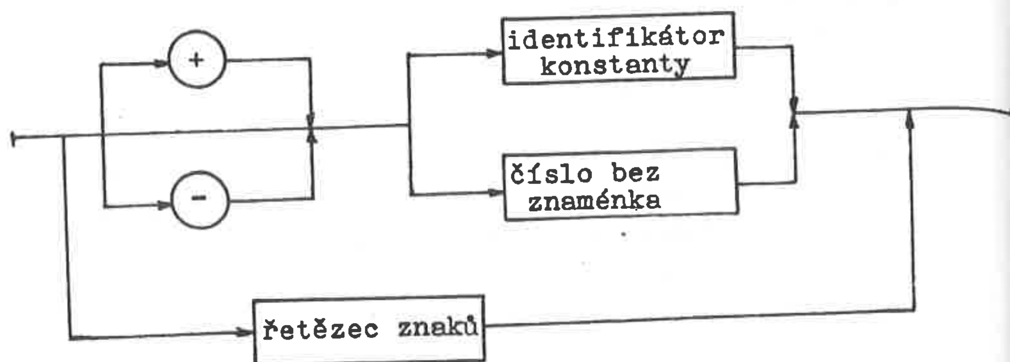
label 0, 10, 999;

4.2 Deklarace konstant

Úsek deklarací konstant je posloupnost deklarací konstant oddělovaných středníkem. Každou deklarací konstanty se zavádí identifikátor konstanty označující hodnotu.

úsek deklarací konstant

konstanta



Výskyt identifikátoru v deklaraci konstanty je jeho definičním místem jako identifikátoru konstanty. Hodnota označená tímto identifikátorem je dána konstantou v deklaraci konstanty. Konstanta nesmí obsahovat identifikátor, jehož hodnotu definuje; znaménko může obsahovat pouze tehdy, označuje-li hodnotu typu integer nebo real.

Předdefinované identifikátory

maxint, false a true

označují standardní konstanty a jsou popsány v odst. 4.3.2.2.

Příklad úseku deklarací konstant:

```
const
    Delka_stranky = 60;
    CELKPOCET     = 100;
    MAX           = 1000;
    MIN           = -MAX;
    KONCZNAK      = '.';
    TITUL         = 'Programovaní v jazyku Pascal';
```

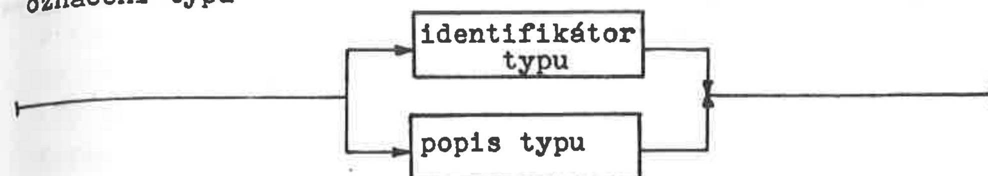
4.3 Deklarace typů

4.3.1 Obecně

Každá proměnná a každá hodnota je svázána s určitým typem. Typ určuje množinu hodnot a množinu operací s těmito hodnotami. K zavedení identifikátoru označujícího typ slouží deklarace typu. Posloupnost deklarací typu oddělovaných středníkem tvoří úsek deklarací typu.

úsek deklarací typu

označení typu



Výskyt identifikátoru v deklaraci typu je jeho definičním místem jako identifikátoru typu. Typ označený tímto identifikátorem je dán označením typu v deklaraci typu. V tomto označení nesmí být použit identifikátor, jehož význam se definuje.

Každým popisem typu se zavádí nový typ, který je odlišný od všech ostatních typů.

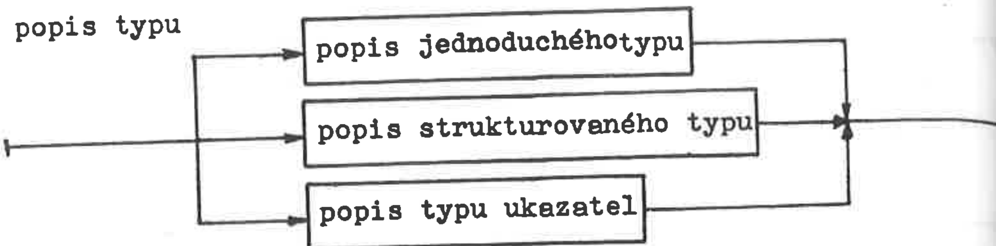
Příklad úseku deklarací typů:

```
type
    STAV   = (SVOB, ZENAT, ROZV, VDOV);
    KLADNE = 1..maxint;
    VEKTOR = array [1..3] of real;
    ALFA   = packed array [1..20] of char;
    BETA   = packed array [1..20] of char;
    GAMA   = ALFA;
    PRVEK  = record
                SLOVO : ALFA; POCET : integer
             end;
    ZNAKY  = set of char;
    INTSB  = file of integer;
    SPOJ   = ↑PRVEK;
```

Poznámka: typ ALFA a GAMA jsou totožné (jeden typ je označen dvěma identifikátory); typ BETA je jiný typ.

Typy dělíme na jednoduché, strukturované a typy ukazatel. Zvláštní třídou jednoduchých typů jsou ordinální typy.

popis typu



Předdefinované identifikátory

integer, real, boolean, char a text

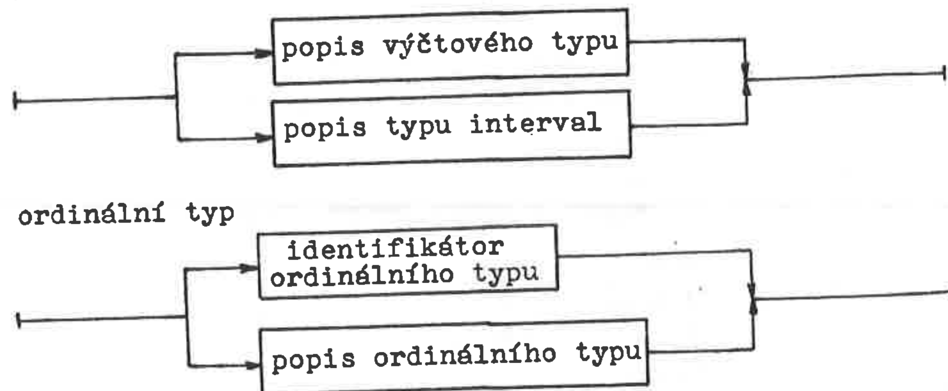
označují standardní typy a jsou popsány v odst.4.3.2.2 a 4.3.3.5.

4.3.2 Jednoduché typy

4.3.2.1 Obecně

Jednoduchý typ určuje uspořádanou množinu hodnot. Standardní jednoduché typy jsou integer, real, boolean a char; další jednoduché typy se zavádějí popisem ordinálního typu.

popis ordinálního typu



Ordinální typy jsou všechny jednoduché typy s výjimkou typu real. Každé hodnotě ordinálního typu přísluší celé ordinální číslo. Uspořádání každé množiny hodnot ordinálního typu je dáno uspořádáním příslušných ordinálních čísel.

Množinu hodnot každého ordinálního typu lze použít pro indexaci prvků pole. Dále lze každý ordinální typ použít jako typ řídící proměnné v příkazu for (viz 6.3.6), typ rozlišovacích konstant v příkazu case (viz 6.3.3) a v typu záznam (viz 4.3.3.3) a jako bázový typ v typu množina (viz 4.3.3.4). S hodnotami jednoho ordinálního typu lze provádět relační operace (viz 5.4.5) a jsou pro ně definovány standardní funkce succ, pred a ord (viz 5.5.2).

4.3.2.2 Standardní jednoduché typy

V tomto odstavci jsou specifikovány množiny hodnot standardních jednoduchých typů a uveden přehled operátorů a funkcí, které s těmito typy souvisí. Podrobnosti o operátorech a funkcích obsahuje kapitola 5.

a) Typ integer

Množinu hodnot typu integer tvoří celá čísla z uzavřeného intervalu

<-maxint-1, maxint>

kde maxint je standardní konstanta typu integer označující hodnotu 32767. Typ integer je ordinální typ, ordinálním číslem hodnoty typu integer je tato hodnota sama.

Kromě vlastností společných všem ordinálním typům, jsou pro typ integer definovány operace +,-,*,<u>div</u>,<u>mod</u> (viz 5.4) a standardní funkce odd(funkční hodnota je typu boolean), chr (funkční hodnota je typu char), abs, sqr, trunc a round (funkční hodnota je typu integer, argument typu real) (viz 5.5.2). Hodnotu typu integer je možno použít všude tam, kde je přípustná hodnota typu real. Konverze na hodnotu typu real se provede automaticky.

b) Typ real

Množina hodnot typu real obsahuje nulu a dále aproximace reálných čísel, jejichž absolutní hodnoty leží přibližně v intervalu $2.7105 * 10^{-20} .. 9.2234 * 10^{18}$. Největší relativní chyba zobrazení je přibližně 0.003 % (viz 9.3). Překladač je koncipován tak, aby změna standardního typu real za uživatelský byla snadno proveditelná.

S reálnými hodnotami lze provádět relační operace (viz 5.4.5), aritmetické operace +,-,*,/ (viz 5.4.2) a jsou pro ně definovány standardní funkce (viz 5.5.2) abs, sqr, sqrt, trunc a round (funkční hodnota typu integer) a standardní matematické funkce sin, cos, exp, ln, arctan (viz 5.5.2).

c) Typ boolean

Množina hodnot typu boolean obsahuje dvě hodnoty označené předdefinovanými identifikátory konstant false a true. Pro tyto hodnoty platí false < true. Ordinálním číslem hodnoty false je 0, ordinálním číslem hodnoty true je 1.

Kromě vlastností společných všem ordinálním typům jsou pro typ boolean definovány boolovské operace <u>not</u>, <u>and</u> a <u>or</u> (viz 5.4.3) , relační operace (viz 5.4.5) a standardní funkce odd (viz 5.5.2.3), eoln a eof (viz 7.1.1.3), mající funkční hodnotu typu boolean.

d) Typ char

Množinou hodnot typu char je množina znaků ASCII a to i těch, které nemají grafickou reprezentaci. Ordinální číslo první hodnoty je 0, ordinální číslo poslední hodnoty typu char je 127. Přehled hodnot typu char a jejich ordinálních čísel je v tabulce.

Pro získání hodnoty typu char pro dané ordinální číslo je definována standardní funkce chr(viz 5.5.2.3).

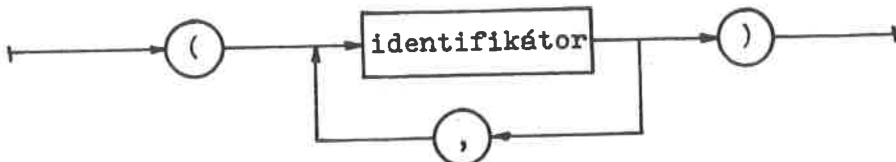
znak	ord_{16}	ord_{10}	znak	ord_{16}	ord_{10}	znak	ord_{16}	ord_{10}
␣	20	32	@	40	64	`	60	96
!	21	33	A	41	65	a	61	97
"	22	34	B	42	66	b	62	98
#	23	35	C	43	67	c	63	99
$	24	36	D	44	68	d	64	100
%	25	37	E	45	69	e	65	101
&	26	38	F	46	70	f	66	102
'	27	39	G	47	71	g	67	103
(	28	40	H	48	72	h	68	104
)	29	41	I	49	73	i	69	105
*	2A	42	J	4A	74	j	6A	106
+	2B	43	K	4B	75	k	6B	107
,	2C	44	L	4C	76	l	6C	108
-	2D	45	M	4D	77	m	6D	109
.	2E	46	N	4E	78	n	6E	110
/	2F	47	O	4F	79	o	6F	111
0	30	48	P	50	80	p	70	112
1	31	49	Q	51	81	q	71	113
2	32	50	R	52	82	r	72	114
3	33	51	S	53	83	s	73	115
4	34	52	T	54	84	t	74	116
5	35	53	U	55	85	u	75	117
6	36	54	V	56	86	v	76	118
7	37	55	W	57	87	w	77	117
8	38	56	X	58	88	x	78	118
9	39	57	Y	59	89	y	79	119
:	3A	58	Z	5A	90	z	7A	120
;	3B	59	[	5B	91	{	7B	121
<	3C	60	\	5C	92	|	7C	122
=	3D	61	]	5D	93	}	7D	123
>	3E	62	↑	5E	94	~	7E	124
?	3F	63	_	5F	95			

Tab. Hodnoty typu char s grafickou reprezentací.

4.3.2.3 Výčtové typy

Množina hodnot výčtového typu je stanovena popisem výčtového typu a sice výčtem identifikátorů, jimiž jsou jednotlivé hodnoty označeny.

popis výčtového typu



Výskyt identifikátoru v popisu výčtového typu je jeho definičním místem jako identifikátoru konstanty tohoto typu. Pořadím identifikátorů v popisu výčtového typu je dáno uspořádání množiny hodnot a ordinální čísla. Obsahuje-li popis výčtového typu celkem n identifikátorů

$$id_1, id_2, \dots, id_n$$

v tomto pořadí, pak ordinální čísla těchto hodnot jsou

$$0, 1, \dots, n-1 .$$

Kromě standardních funkcí, které jsou definovány pro všechny ordinální typy je pro výčtový typ T ve Fel-Pascalu definována funkce

T(e)

kde e je výraz typu integer a hodnotou funkce je ta hodnota x výčtového typu T pro niž platí (viz 5.5.2.3)

ord(x)=e

Příklad:

```
type
  Tyden = (PONDELI, UTERY, STREDA, CTVRTEK, PATEK,
           SOBOTA, NEDELE);
```

4.3.2.4 Typy interval

Typ interval definuje neprázdnou souvislou podmnožinu hodnot ordinálního typu. Dolní a horní mez této podmnožiny udávají dvě konstanty, které se uvádějí v popisu typu interval.

popis typu interval

Obě konstanty v popisu typu interval musí být téhož ordinálního typu. Jejich typ se nazývá hostitelským typem typu interval. Interval, jehož hostitelským typem je T, nazýváme intervalem z T. První konstanta (udávající dolní mez intervalu) nesmí být větší než druhá konstanta (udávající horní mez intervalu).

Pro proměnné typu interval z T jsou definovány všechny operátory a funkce, jako pro proměnné typu T. Proměnné typu interval však nesmí být přiřazena hodnota, která neleží v daném intervalu (viz kompatibilita vzhledem k přiřazení - 4.3.5).

Příklad:

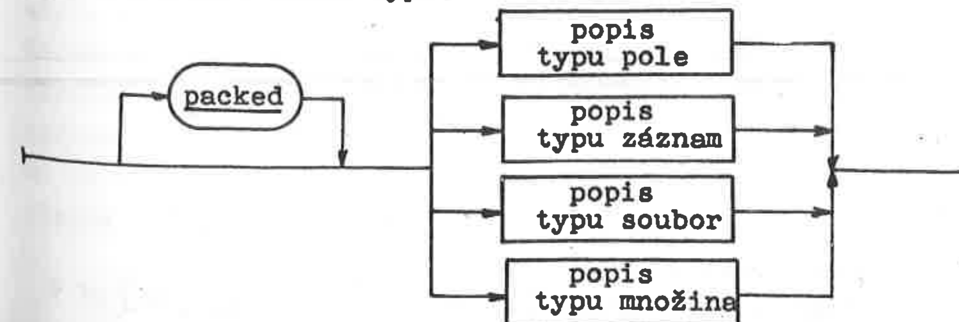
```
type
    Prac_den = PONDELI .. PÁTEK;
    MESICE   = 1 .. 12;
    PISMENA  = 'A' .. 'Z' ;
```

4.3.3 Strukturované typy

4.3.3.1 Obecně

Strukturované typy určují strukturované objekty, tj. objekty skládající se z dílčích složek (podobjektů). Strukturované typy se dělí na typy pole, typy záznam, typy soubor a typy množina.

popis strukturovaného typu



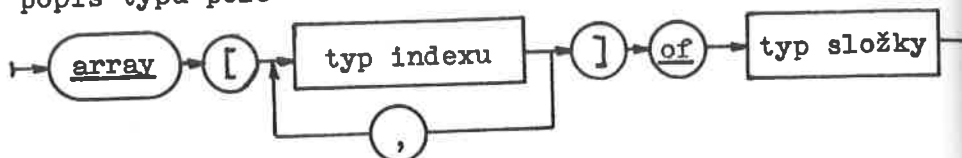
Výskyt slova packed před popisem strukturovaného typu je příznak zhuštěného typu. Příznak zhuštěného typu se přitom týká pouze daného typu. Je-li složka typu také strukturovaný typ, pak bude zhuštěná pouze tehdy, je-li typ této složky zaveden jako zhuštěný.

Ve Fel-Pascalu-80 je zhuštěný typ reprezentován stejně jako nezhuštěný, avšak pro používání složek objektů zhuštěného typu platí některá omezení (viz 4.5.4 a 8.2), která jsou stanovena normou jazyka Pascal.

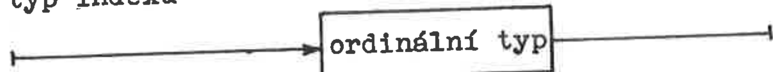
4.3.3.2 Typy pole

Objekty typu pole se skládají z pevného počtu složek stejného typu. Složky pole, nazýváme je též prvky pole, se rozlišují pomocí indexů, což jsou hodnoty určených ordinálních typů. Typy indexů a typ složek se zadávají v popisu typu pole.

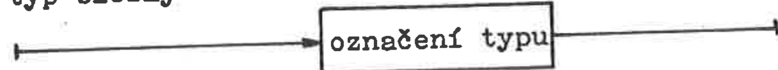
popis typu pole



typ indexu



typ složky



Příklady:

```
type
    VEKTOR = array [1 .. 3] of real;
    VETA   = array [0 .. 79] of char;
    KOD    = array [char] of char;
```

Popis typu pole ve tvaru

array [T1, T2, ... , Tn] of T

se považuje za zkratku rozepsaného popisu

array [T1] of array [T2] of ... of array [Tn] of T

Příklad tří variant zavedení typu dvojrozměrného pole:

```
const
    POCETR = 10;
    POCETS = 20;
```

a)
```
type
    MATICE = array [1..POCETR, 1..POCETS] of integer;
```

b)
```
type
    MATICE = array [1..POCETR] of
                array [1..POCETS] of integer;
```

c)
```
type
    RADEK  = array [1..POCETS] of integer;
    MATICE = array [1..POCETR] of RADEK;
```

Pokud je zkrácený zápis vícerozměrného typu pole označen jako packed, je tento zápis ekvivalentní nezkrácenému zápisu, ve kterém je jako packed označen jak typ pole, tak typ jeho složek.

Příklad dvou ekvivalentních zápisů pole:

```
packed array [1..3, 1..2] of char
packed array [1..3] of packed array [1..2] of char
```

Pole, jehož typy indexů jsou T1, T2 ... , Tn, obsahuje celkem T1 * T2 * ... * Tn prvků, kde Ti je mohutnost množiny hodnot typu Ti.

Typy řetězec

Typem řetězec délky n se nazývá každý typ, jehož popis je packed array [d..h] of char

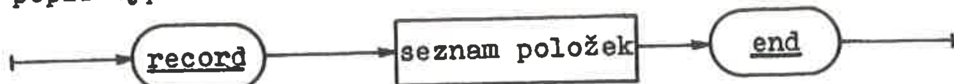
kde $n = h-d+1>1$

Hodnotou typu řetězec délky n je řetězec znaků, který obsahuje n znaků ($n>1$). Pro hodnoty typu řetězec jsou definovány relace =, <> ,<,>,<=,>= (viz 5.4.5). Kromě toho má typ řetězec všechny vlastnosti typu pole. Řetězce stejné délky jsou navíc kompatibilní (viz 4.3.5).

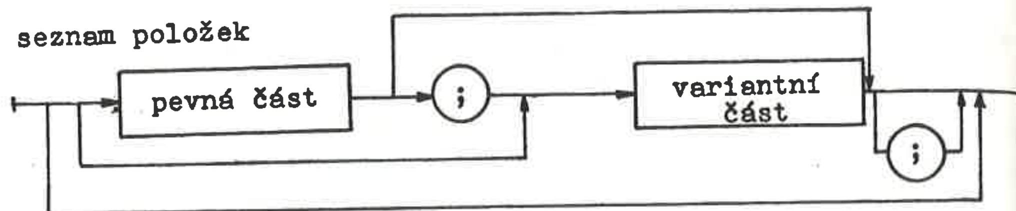
4.3.3.3 Typy záznam

Objekty typu záznam se skládají z pevného počtu složek, které nemusí být stejného typu. Složky záznamu, nazýváme je též položkami záznamu, se rozlišují pomocí identifikátorů položek. Identifikátory položek a jejich typy jsou určeny v popisu typu záznam.

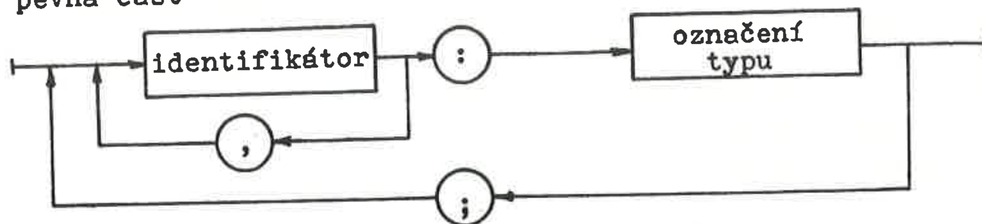
popis typu záznam



seznam položek



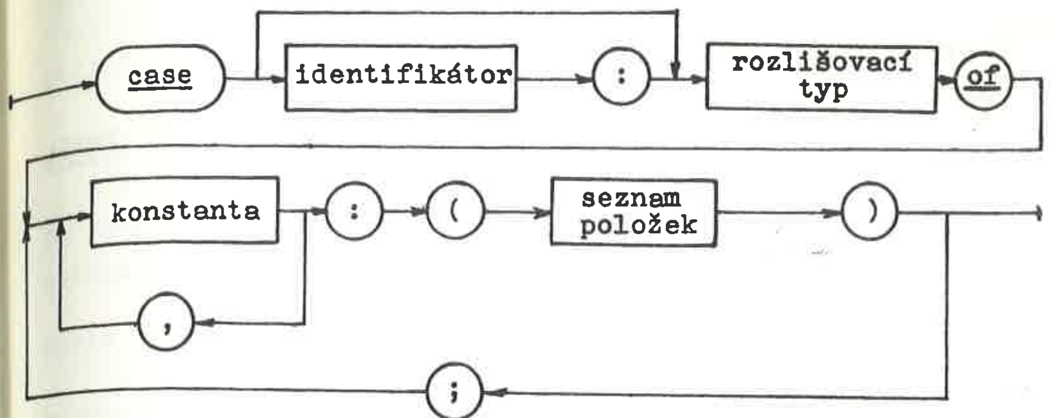
pevná část



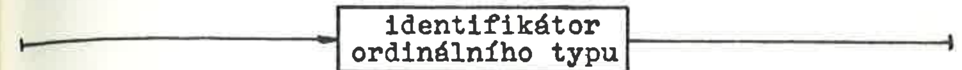
Výskyt identifikátoru v seznamu položek je jeho definičním místem jako identifikátoru položky pro příslušný typ záznam.

Seznam položek má dvě části (z nichž jedna nebo obě mohou chybět): pevnou část a variantní část. Variantní část umožňuje specifikovat různé varianty struktury záznamu (při zachování téhož typu).

variantní část



rozlišovací typ



Ve variantní části je určen rozlišovací typ, jehož hodnoty jsou přidruženy k jednotlivým variantám. Ke každé variantě je přidružena jedna nebo několik hodnot rozlišovacího typu. Součástí variantní části může být rozlišovací položka, která nabývá hodnot rozlišovacího typu a tím v konkrétním objektu indikuje příslušnou variantu (není-li identifikátor rozlišovací položky ve variantní části definován, pak rozlišovací položka v záznamu není). Jednotlivé varianty jsou určeny seznamem položek, který může být prázdný. Seznam konstant rozlišovacího typu před variantou udává hodnoty, které jsou k této variantě přidruženy.

Příklady:

Typ záznam bez variantní části:

```
type
   DATUM = record
               DEN : 1..31;
             MESIC : 1..12;
               ROK : 1..2000
           end;
```

Typ záznam s pevnou i variantní částí:

```
type
   TYPPOHL = (MUZ, ZENA);
   OSOBA = record
             PRIJMENI, JMENO : packed array [1..20] of char;
             STAV : (SVOB, ZENAT, ROZV, VDOV);
             case POHLAVI : TYPPOHL of
               MUZ : (VAHA : real;
                      VOUS : boolean);
               ZENA : (MIRY : array [1..3] of integer)
           end;
```

Typ záznam s variantní částí bez rozlišovací položky:

```
type
   TYPELEM = (INT, DCHAR);
   ELEM = record
            case TYPELEM of
              INT : (CISLO : integer);
              DCHAR : (ZNAKY : packed array [1..3] of char)
          end;
```

Typ záznam s vnořenou variantní částí:

```
type
   ALFA    = packed array [1..10] of char;
   ADRESA = (ZNAMA, NEZNAMA);
   DELIKVENT = record
                 JMENO, PRIJMENI : ALFA;
                 case VEZNEN : boolean of
                   true : (KDE : ALFA;
                           DOKDY : DATUM);
                   false: (case ADR:ADRESA of
                             ZNAMA    : (BYDLISTE : ALFA);
                             NEZNAMA : () )
               end;
```

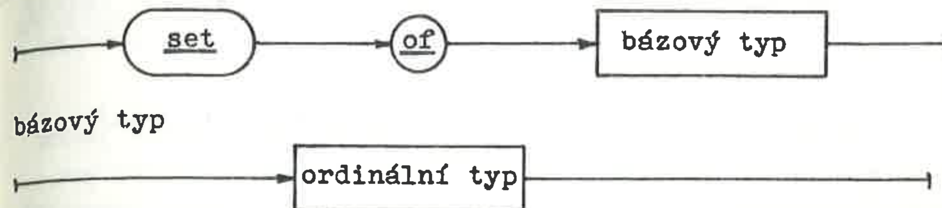
Proměnné typu variantní záznam mají pro jednotlivé varianty v paměti rezervované stejné místo. Jednotlivé varianty se tedy vzájemně překrývají. Délka záznamu je

dána délkou nejdelší varianty (nejedná-li se o proměnnou vytvořenou dynamicky procedurou new s udáním varianty) (viz 8.1). Součástí variantní části záznamu nesmí být položka obsahující soubor.

4.3.3.4 Typy množina

Hodnotou typu množina z T, kde T je ordinální typ, je podmnožina hodnot typu T. Typ množina z T tedy určuje potenční množinu typu T.

popis typu množina



bázový typ

ordinální typ

Příklady:

```
type
   CHARSET = set of char;
   ODSTIN  = set of (CERVENA, ZLUTA, MODRA, ZELENA, SEDA);
   INTSET  = set of 0..127;
   ALFASET = set of 'A' 'Z';
```

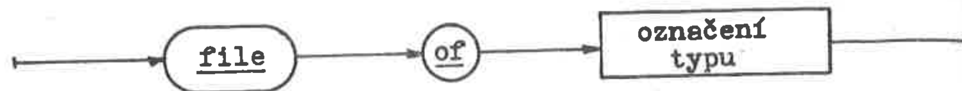
Ke každému typu množina set of T je definován kanonický typ množina následujícím způsobem: je-li T interval z typu integer, potom kanonický typ je set of 0..SETM, kde konstanta SETM je dána parametrem překladu a její maximální hodnota je 255 (viz 12.3). Je-li T interval z typu S a S není integer, pak kanonický typ je set of S, jinak je kanonický typ přímo set of T. Všechny hodnoty typu množina se považují za hodnoty jejího kanonického typu. Typy CHARSET a ALFASET mají tedy stejný kanonický typ a jejich hodnoty jsou v paměti uloženy stejným způsobem.

Pro hodnoty typu množina jsou definovány relace in, =,<>, množinová inkluze (<=,>=) a dále sjednocení (+), průnik (*) a rozdíl množin (-) (viz 5.4.5).

4.3.3.5 Typy soubor

Objekt typu soubor je strukturován jako posloupnost obsahující libovolný, předem neurčený počet složek stejného typu. Popisem typu soubor je tedy stanoven pouze typ složek a ne jejich počet.

popis typu soubor



Typem složek souboru nesmí být typ soubor ani takový typ pole nebo záznam, jehož některá složka (na libovolné úrovni) je typu soubor.

Z hodnoty typu soubor je přístupná vždy právě jedna složka, kterou nazýváme aktuální (pomocí přístupové proměnné souboru). Pro práci s objektem typu soubor jsou definovány základní standardní procedury reset, rewrite, get, put, read, write a boolovská funkce eof. Popis těchto a ostatních procedur pro práci se soubory je v kap.7. Hodnoty typu soubor na rozdíl od všech ostatních typů nelze přiřazovat a nemohou být součástí variantní části typu záznam.

V Pascalu je definován standardní typ text, který je souborem, jehož složkami jsou znaky (typ char) uspořádané do řádků. Pro typ text jsou definovány další procedury - readln, writeln, page a booleovská funkce eoln.

Typ text (textový soubor) slouží pro vstup a výstup ve vnější (znakové, textové, řádkové) reprezentaci dat; uživatelem definované typy soubor mají složky ve vnitřní reprezentaci hodnot.

Příklady:

```
type
   OSOBA      = record
                   JMENO:packed array [1..30] of char;
                   VEK  : integer
                end;
   OSOBY      = file of OSOBA;
   INTSOUBOR = file of integer;
```


Pro textové soubory jsou dále definovány vstupní a výstupní konverze hodnot některých typů.

Vstup z textového souboru procedurou read, resp. readln, umožňuje provádět konverzi hodnot typů char, integer a real.

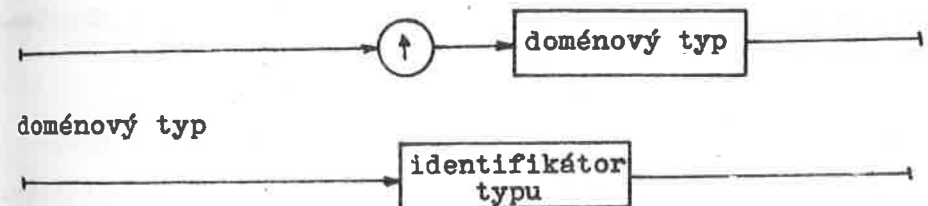
Výstup do textového souboru procedurou write, resp. writeln, umožňuje provádět konverzi hodnot typů char, integer, real, boolean, řetězec a ukazatel.

Další podrobnosti o zpracování textových souborů jsou uvedeny v kap.7.

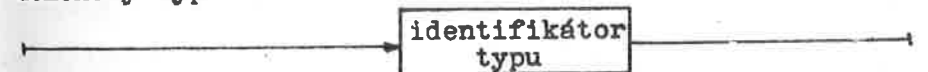
4.3.4 Typy ukazatel

Hodnotami typu ukazatel jsou vnitřní identifikace (adresy) proměnných určitého typu. Typ proměnných, které jsou hodnotami konkrétního typu ukazatel identifikovány, se nazývá doménový typ příslušného typu ukazatel. Doménový typ je určen v popisu typu ukazatel.

popis typu ukazatel



doménový typ



Použití identifikátoru typu pro určení doménového typu v popisu typu ukazatel je jedinou výjimkou, kdy použití identifikátoru může předcházet jeho definičnímu místu.

Příklady:

```
type
   SPOJ1 = ↑ UZEL1;
   SPOJ2 = ↑ UZEL2;
   UZEL1 = record
              V1, V2 : integer;
              SPOJ : SPOJ1
           end;
   UZEL2 = record
              V1, V2 : real;
              SPOJ : SPOJ2
           end;
```

Pro objekty typu ukazatel jsou definovány relace =,<>,<=,>=, <, > a dále operace přičtení a odečtení doplňku a rozdíl dvou ukazatelů (viz 5.4.5 a 5.4.6).

Zvláštní hodnotou, která patří do každého typu ukazatel a neidentifikuje žádnou proměnnou, je prázdný ukazatel. Tato hodnota je označena vyhrazeným slovem <u>nil</u>.

Pro vytváření hodnot typu ukazatel slouží procedury new a ref, pro rušení hodnot procedura dispose (viz kap.8).

4.3.5 Kompatibilita typů

Typy T1 a T2 jsou kompatibilní, jestliže splňují alespoň jednu z těchto čtyř podmínek:

- T1 a T2 je tentýž typ.
- T1 je intervalem z T2, nebo T2 je intervalem z T1, nebo T1 i T2 jsou intervaly z téhož hostitelského typu.
- T1 a T2 jsou typy množina s kompatibilními bázovými typy. Pokud bázové typy typů T1 a T2 jsou intervaly z typu integer a typy T1 a T2 se nacházejí v různých programových jednotkách, jsou kompatibilní pouze za předpokladu, že obě programové jednotky jsou přeloženy se stejným přepínačem překladu/SET (viz kap.12).
- T1 a T2 jsou typy řetězec stejné délky.
- T1 a T2 jsou typy ukazatel se stejnými doménovými typy.

Je-li v syntaktické konstrukci požadována kompatibilita typů, kontroluje ji kompilátor při překladu.

Hodnota typu T2 je kompatibilní vzhledem k přiřazení s typem T1, je-li splněna alespoň jedna z těchto podmínek:

- T1 a T2 je tentýž typ a tento typ není typem soubor ani strukturovaným typem se složkou typu soubor. Tento typ dále nesmí být typ záznam s variantní částí, který přísluší proměnné vzniklé dynamicky procedurou new s udáním varianty. (Tuto kontrolu neprovádí ani překladač, ani není prováděna v době běhu programu a tato chyba může způsobit nedefinovanou činnost programu).

- T1 je typ real a T2 je typ integer.
- T1 a T2 jsou kompatibilní ordinální typy a hodnota typu T2 patří do intervalu, který je specifikován typem T1.
- T1 a T2 jsou kompatibilní typy množina a všechny prvky hodnoty typu T2 patří do intervalu, který je specifikován bázovým typem z typu T1,
- T1 a T2 jsou typy řetězec. Pokud T2 je typem konstanty řetězec znaků, může mít tato konstanta menší počet znaků než typ T1; jinak musí mít oba typy stejný počet znaků.
- T1 a T2 jsou kompatibilní typy ukazatel.

Je-li v syntaktické konstrukci požadována kompatibilita vzhledem k přiřazení, pak kontrola příslušnosti přiřazované hodnoty k danému intervalu může být provedena až při výpočtu. Kompilátor generuje instrukce pro provedení této kontroly v závislosti na tom, zda je či není nastaven parametr překladu R(přepínač překladu /RNG).

Příklady:

```
type
    PRIROZENY = 0..maxint;
    CITAC     = integer;
    ROZSAH    = integer;
    ROK       = 1900..1999;
    POLE1     = array [1..10] of integer;
    POLE2     = POLE1;
    POLE3     = array [1..10] of integer;
    ISET1     = set of 0..15;
    ISET2     = set of 0..31;
    ALFA1     = packed array [1..6] of char;
    ALFA2     = packed array [1..6] of char;
    POINT1    =  POLE1;
    POINT2    =  POLE1;
```

Takto definované identifikátory typu mají následující vlastnosti:

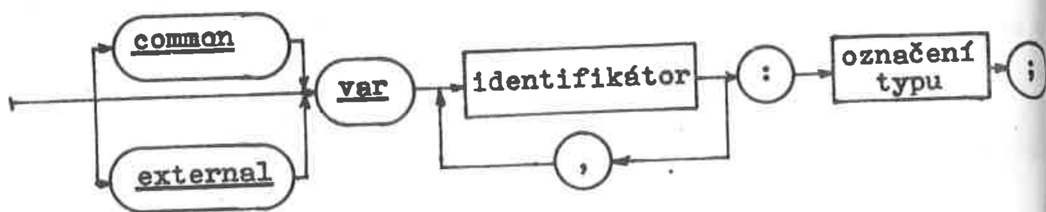
- CITAC, ROZSAH a integer označují týž typ,
- CITAC, PRIROZENY a ROK označují různé, avšak kompatibilní typy,

- POLE1 a POLE2 označují týž typ
- POLE1 a POLE3 označují různé nekompatibilní typy
- ISET1 a ISET2 označují různé, avšak kompatibilní typy
- ALFA1 a ALFA2 označují kompatibilní typy
- POINT1 a POINT2 označují různé kompatibilní typy

4.4 Deklarace proměnných

Deklaracemi proměnných se zavádějí identifikátory označující proměnné. Proměnná může být lokální nebo externí.

úsek deklarací proměnných



Výskyt identifikátoru v deklaraci proměnných je jeho definičním místem jako identifikátoru proměnné. Začíná-li úsek deklarací proměnných slovem var, pak identifikátory proměnných v něm definované označují lokální proměnné, začíná-li slovem external nebo common, pak identifikátory proměnných označují externí proměnné. Typ proměnné, která je definovaným identifikátorem označena, je dán označením typu v deklaraci proměnné.

Lokálním proměnným deklarovaným na základní úrovni bloku hlavní programové jednotky nebo deklarační jednotky je přidělena paměť staticky. Lokálním proměnným deklarovaným na základní úrovni bloku procedury (resp. funkce) je paměť přidělena dynamicky (v systémovém zásobníku), pokud je procedura (resp. funkce) dynamická nebo staticky, pokud je procedura (resp. funkce) statická (viz 4.5).

Externí proměnné uvedené v úseku deklarací proměnných označeném jako common se nacházejí v common bloku beze jména, který je společný všem sestaveným kompilačním jednotkám. V různých kompilačních jednotkách nemusí mít tento úsek deklarací stejnou délku, avšak kratší musí svojí strukturou odpovídat delšímu a musí být sestavován později.

Příklad dvou úseků deklarací proměnných common bloku beze jména ze dvou kompilačních jednotek:

```
common var                      common var
    I,J: integer;                   I: integer;
    B  : boolean;                   J: integer;
    C  : char;
```

Více úseků deklarací označených jako common má stejný význam jako jeden úsek, ve kterém jsou všechny proměnné umístěny ve stejném pořadí.

Umístění proměnných do common bloku beze jména lze rovněž předepsat $-poznámkou B (viz 12.3.2).

Pro externí proměnné uvedené v úseku deklarací proměnných označeném jako external musí platit právě jedna z uvedených podmínek:

- externí proměnná je určená specifikací common nebo origin (viz 3.2.2.3 a 3.2.2.4).
- externí proměnné odpovídá v jiné kompilační jednotce lokální proměnná deklarovaná na základní úrovni a určená specifikací global (viz 3.2.2.5).
  Identifikátory uvedené ve specifikaci global resp. v deklaraci external var se při sestavování úlohy programu rozlišují pouze podle prvých šesti znaků. Různé identifikátory tohoto druhu se tedy musí lišit v prvých šesti znacích.

Příklady úseků deklarací proměnných:

```
var
  MAX     : integer;
  I, J    : 0..100;
  POMTAB1 : array [1..100] of integer;
  POMTAB2 : array [1..100] of integer;
  VEKT1,
  VEKT2   : POLE1;
  VEKT3   : POLE1;
```

Poznámka: Proměnné POMTAB1 a POMTAB2 jsou různého typu. Proměnné VEKT1, VEKT2, VEKT3 jsou téhož typu a proměnné I a J jsou téhož typu.

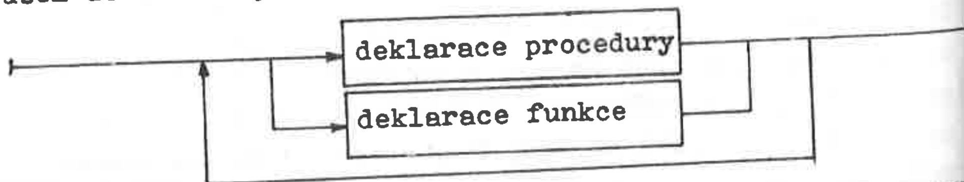
```
external var
  TAB   : array [1..100] of integer;
  DISPL : text;
```

4.5 Deklarace procedur a funkcí

4.5.1 Obecně

Procedura je část programu, která může být vyvolána příkazem procedury. Funkce je část programu, která může být vyvolána zápisem funkce. Posloupnost deklarací procedur a funkcí tvoří úsek deklarací procedur a funkcí.

úsek deklarací procedur a funkcí



Procedura (resp. funkce) může být lokální nebo extern[í.] Lokální procedura (resp. funkce) má blok vnořen do dané ko[m]pilační jednotky, externí procedura (resp. funkce) má blok vnořen do jiné kompilační jednotky.

Je-li procedura (resp. funkce) deklarovaná jako exter[ní,] pak musí platit právě jedna z následujících podmínek:

- procedura (resp. funkce) je specifikována jako origin (viz 3.2.2.4).
- proceduře (resp. funkci) odpovídá v jiné kompilační jednotce procedura (resp. funkce) specifikovaná jako global (viz 3.2.2.5). Identifikátory těchto procedur či funkcí se při sestavování rozlišují pouze podle prvých šesti znaků.

Procedura (resp. funkce) může být klasifikována jako static, dynamic a system. Klasifikace dynamic určuje, že procedura (resp. funkce) má proměnné a parametry uloženy na zásobníku a může být tedy volána rekurzivně.

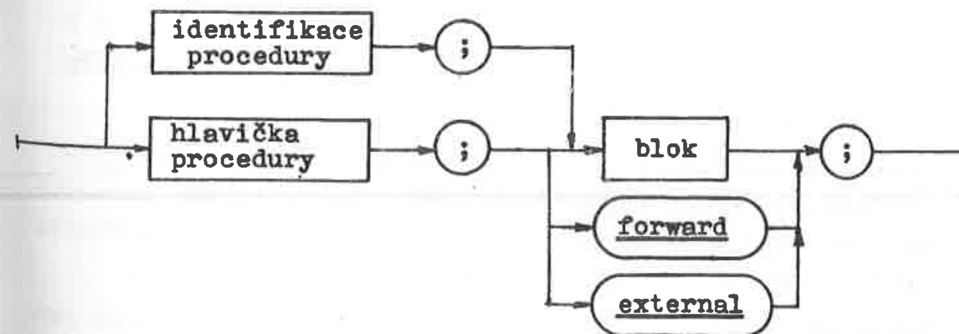
Implicitně jsou všechny neklasifikované procedury přeloženy jako statické. Klasifikace static se používá tehdy, je-li tento implicitní způsob změněn Ø-poznámkou D (viz 12.3.2).

Klasifikace system slouží k vazbě pascalského programu na procedury napsané v jazyku PL/M a k jiným speciálním účelům (viz 9.2.5).

Existuje řada standardních procedur a funkcí, které není třeba deklarovat a které mají zvláštní způsob zpracování skutečných parametrů (viz 5.5.2 ,6.2.2.2). Kromě toho je možno používat knihovní procedury a funkce (viz 7.2.5, 10.3), které je ovšem nutno deklarovat jako externí.

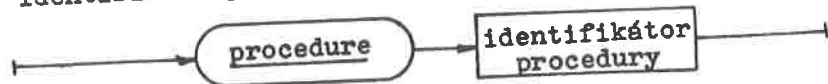
4.5.2 Deklarace procedury

deklarace procedury

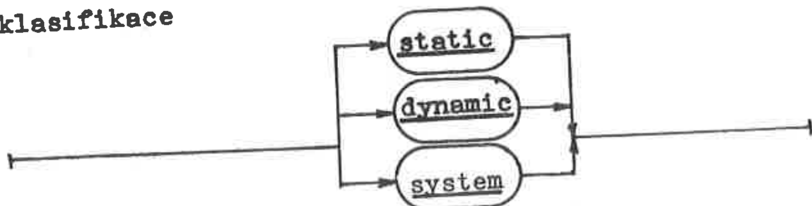


hlavička procedury

identifikace procedury



klasifikace



Výskyt identifikátoru v hlavičce procedury je jeho definičním místem jako identifikátoru procedury. Rozlišujeme čtyři případy deklarace procedury:

- úplnou deklaraci lokální procedury,
- deklaraci externí procedury,
- předběžnou deklaraci lokální procedury,
- dokončující deklaraci lokální procedury.

a) Úplná deklarace lokální procedury obsahuje hlavičku procedury a blok této procedury.

b) Deklarace externí procedury obsahuje hlavičku procedury a slovo <u>external</u>.

c) Předběžná deklarace lokální procedury obsahuje hlavičku procedury a slovo <u>forward</u>. K předběžné deklaraci procedury P se musí ve stejném úseku deklarací procedur a funkcí ještě vyskytovat dokončující deklarace procedury P.

d) Dokončující deklarace procedury obsahuje identifikaci procedury a její blok.

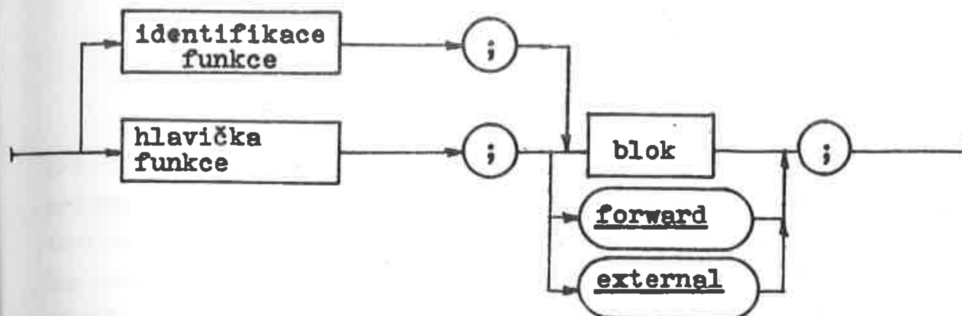
Rozdělení deklarace procedury na předběžnou a dokončující je nutné při deklaraci vzájemně rekurzivních procedur.

Příklady:

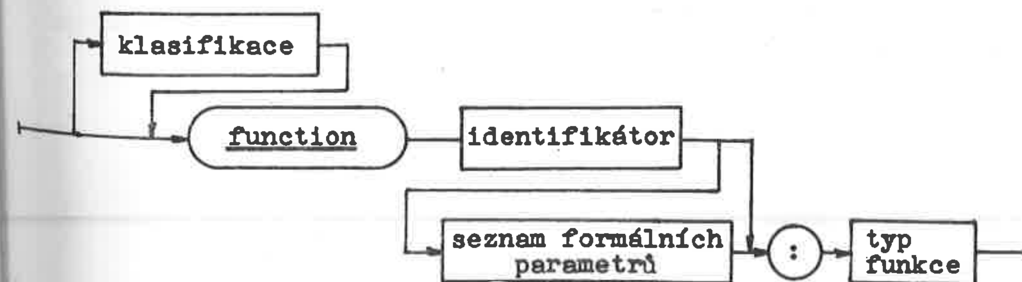
```
procedure P1(X,Y:integer);              {lokální}
  begin  ... end;
procedure P2(X,Y:integer);              {externí}
external;
procedure P3(X,Y:integer);              {předběžná deklarace}
forward;
  ...
procedure P3;                           {dokončující deklarace}
  begin ... end;
```

4.5.3 Deklarace funkce

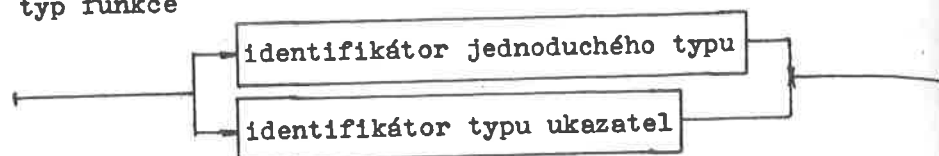
deklarace funkce



hlavička funkce



typ funkce



identifikace funkce

Výskyt identifikátoru v hlavičce funkce je jeho definičním místem jako identifikátoru funkce. Typ funkce může být pouze jednoduchý typ nebo typ ukazatel a je specifikován identifikátorem typu. Pravidla vymezující vlastnosti funkce jsou stejná jako u procedur.

Uvnitř bloku funkce musí být přiřazovací příkaz, který se přiřazuje hodnota identifikátoru funkce. Tímto příkazem je definována hodnota funkce při jejím vyvolání.

Příklady deklarací funkcí:

```
type
  PRIROZENE = 0..maxint;
  INDEX = 1..10;
  VEKTOR = array [INDEX] of real;
function POWER(X:real; N:PRIROZENE):real;
{POWER := X ** N }
var W,Z : real; I:integer;
begin
  W:=X; Z:=1; I:=N;
  while I >0 do
    begin     {Z*(W**I)=X**N}
      if odd(I) then Z:=Z*W;
      I:=I div 2; W:=sqr(W)
  end;
  { Z=X**N }
  POWER:=Z
end;
```

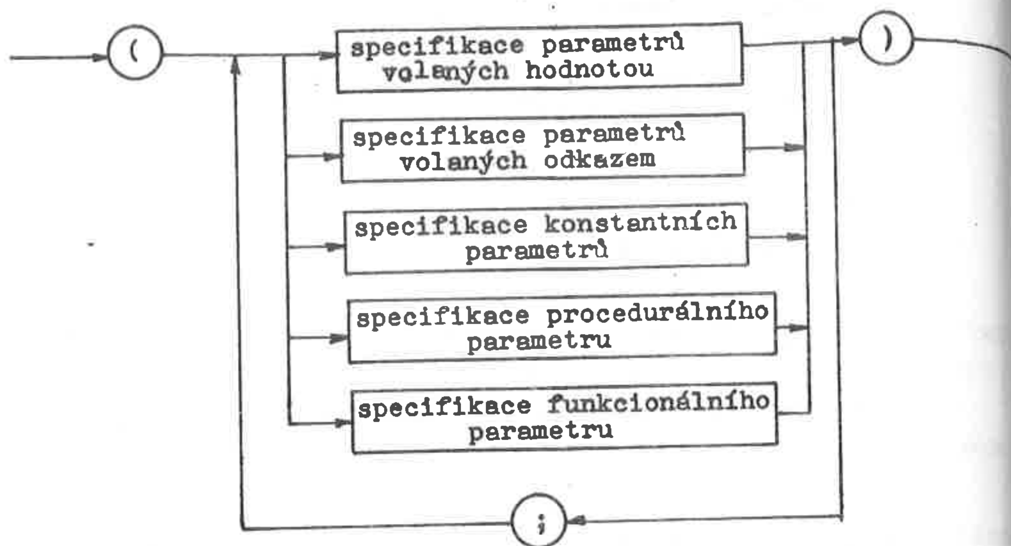


```
function MAX(A:VEKTOR; POCET:INDEX):real;
{ MAX := max(A [1] ,...A [POCET] ) }
var X:REAL; I:INDEX;
begin X:=A [1] ;
  for I:=2 to POCET do
    if X < A[I] then X:=A[I];
  MAX:=X
end;

function Y(I:integer):integer;
forward;

function Z(I:integer):integer;
begin
  if I<=0 then Z:=1 else Z:=Y(I-1)+3
end;

function Y;
begin
  if I<=0 then Y:=-1 else Y:=2*Z(I-1)
end;
```

4.5.4 Formální parametry

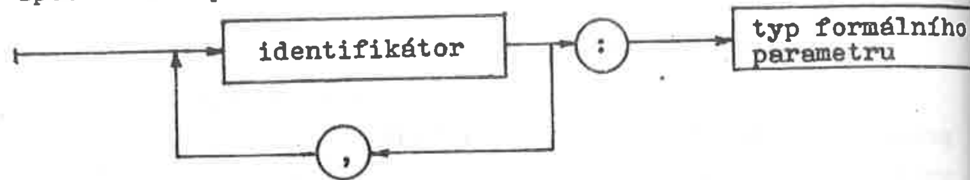
Formální parametry jsou identifikátory, které v bloku procedury (resp.funkce) reprezentují objekty, které jsou určeny skutečnými parametry při vyvolání procedury (resp. funkce). Fel-Pascal rozlišuje pět druhů parametrů: parametry volané hodnotou, parametry volané odkazem, konstantní parametry, procedurální parametry a funkcionální parametry. Druh, typ a identifikátor parametru jsou dány specifikací parametru v seznamu formálních parametrů.

seznam formálních parametrů

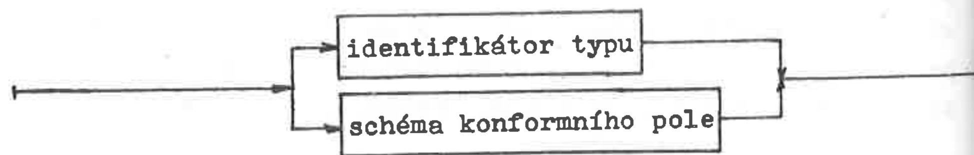


Parametry volané hodnotou

specifikace parametrů volaných hodnotou



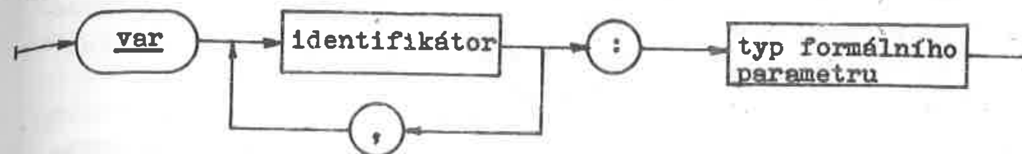
typ formálního parametru



Výskyt identifikátoru ve specifikaci parametrů volaných hodnotou je jeho definičním místem jako parametru volaného hodnotou. Identifikátor parametru volaného hodnotou označuje v bloku procedury (funkce) lokální proměnnou, jejíž typ je dán typem parametru a které je na začátku provádění bloku procedury (resp.funkce) přiřazena hodnota skutečného parametru. Přípustným skutečným parametrem je výraz, jehož hodnota je kompatibilní vzhledem k přiřazení s typem parametru (viz 4.3.5).

Parametry volané odkazem

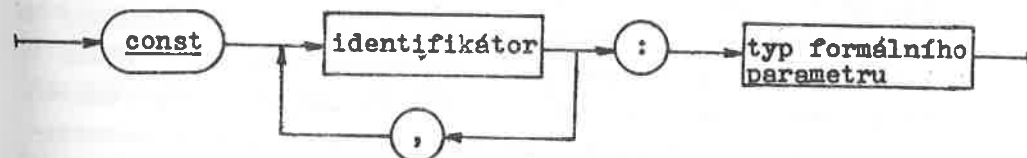
specifikace parametrů volaných odkazem



Výskyt identifikátoru ve specifikaci parametrů volaných odkazem je jeho definiční místo jako identifikátoru parametru volaného odkazem. Takto specifikovaný parametr představuje v bloku procedury (resp. funkce) proměnnou, která je určena skutečným parametrem. Přípustným skutečným parametrem je proměnná téhož typu, jakého je formální parametr. Je-li skutečným parametrem proměnná, která je určena pomocí selektoru, pak všechny přístupové operace se provedou pouze jednou na začátku provádění bloku procedury (resp. funkce). Složky zhuštěných typů nesmí být skutečnými parametry volanými odkazem.

Konstantní parametry

specifikace konstantních parametrů



Výskyt identifikátoru ve specifikaci konstantních parametrů je jeho definičním místem jako identifikátoru konstantního parametru. Typ parametru je dán identifikátorem typu ve specifikaci. Identifikátor konstantního parametru označuje v bloku procedury (resp. funkce) lokální proměnnou, které je na začátku provádění bloku přiřazena hodnota skutečného parametru. Hodnota konstantního parametru nesmí být v bloku ohrožena (viz 5.2.1). Jako skutečný parametr je přípustný výraz, jehož hodnota je kompatibilní vzhledem k při-

řazení s typem parametru. Konstantní parametry jednotlivých typů, typu ukazatel a typu množina se předávají jako parametry volané hodnotou, konstantní parametry ostatních typů se předávají jako parametry volané odkazem. Místo vyhrazeného slova <u>const</u> je možno zadat specifikaci konstantních parametrů parametrem překladu X.

Příklad tří různých zápisů specifikace konstantních parametrů:

<u>const</u> X,Y:real
ØX+X,Y:real ØX-
{ØX+} X,Y:real{ØX-}

<u>Procedurální parametry</u>

specifikace procedurálního parametru

Výskyt identifikátoru v hlavičce procedury, která tvoří specifikaci procedurálního parametru, je jeho definičním místem jako identifikátoru procedurálního parametru. Tomuto parametru přísluší seznam formálních parametrů uvedený v téže hlavičce. V bloku procedury (resp.funkce) označuje takto specifikovaný parametr proceduru, která je určena skutečným parametrem při vyvolání procedury (resp.funkce). Přípustným skutečným parametrem je identifikátor procedury, jejíž seznam formálních parametrů je kongruentní se seznamem formálních parametrů, který přísluší identifikátoru procedurálního parametru.

Dva seznamy formálních parametrů jsou kongruentní, jestliže obsahují stejný počet specifikací formálních parametrů a pro stejně položené specifikace platí jedna z uvedených podmínek:

- obě jsou současně buď specifikace parametrů volaných hodnotou, odkazem nebo konstantních parametrů. Obě obsahují stejný počet parametrů a typy formálních parametrů jsou buď tytéž typy, nebo schémata konformních polí, která definují stejnou třídu typů pole (viz dále);
- obě jsou specifikace buď procedurálního nebo funkcionálního parametru a jejich seznam formálních parametrů je kongruentní. U funkcionálních parametrů je typ funkcí týž.

<u>Funkcionální parametry</u>

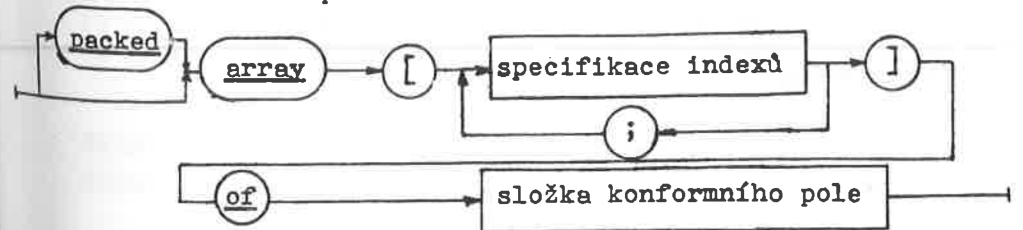
specifikace funkcionálního parametru

Výskyt identifikátoru v hlavičce funkce, která tvoří specifikaci funkcionálního parametru, je jeho definičním místem jako identifikátoru funkcionálního parametru. Tomuto parametru přísluší typ funkce a seznam formálních parametrů uvedené v téže hlavičce. V bloku procedury (resp. funkce) označuje takto specifikovaný parametr funkci, která je určena skutečným parametrem při vyvolání procedury (resp. funkce). Přípustným skutečným parametrem je identifikátor funkce téhož typu, jaký přísluší identifikátoru funkcionálního parametru, a jejíž seznam formálních parametrů je kongruentní se seznamem formálních parametrů, který přísluší identifikátoru funkcionálního parametru.

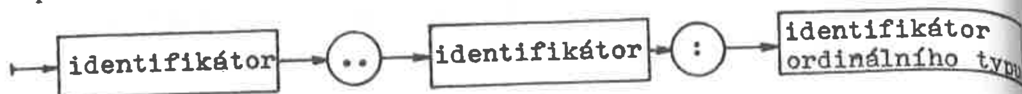
<u>Schéma konformního pole</u>

Schéma konformního pole je popis typu formálního parametru, který umožňuje předávat proceduře (resp. funkci) jako skutečné parametry pole s různými mezemi indexů.

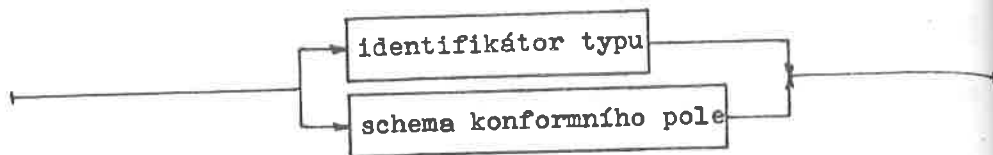
schema konformního pole



specifikace indexů



složka konformního pole



Pokud je složka konformního pole opět schéma konformního pole, lze použít zkrácený zápis podobně jako u typu pole.

Je-li zkrácený zápis schématu konformního pole označen jako packed, je to ekvivalentní nezkrácenému zápisu, ve kterém je jako packed označeno jak schéma konformního pole, tak jeho složka.

Příklady ekvivalentních zápisů schématu konformního pole:

array[U..V:T1] of array [J..K:T2] of T3

je ekvivalentní

array [U..V:T1; J..K:T2] of T3

packed array [U..V:T1] of packed array [J..K:T2] of T3

je ekvivalentní

packed array [U..V:T1; J..K:T2] of T3

Je-li složkou schématu konformního pole opět schéma konformního pole, pak musí být obě schémata současně buď označena, anebo neoznačena jako packed.

Příklad nedovoleného schématu konformního pole:

packed array [U..V:T1] of array [J..K:T2] of T3

Výskyt identifikátoru ve specifikaci indexů je jeho definičním místem jako identifikátoru meze pro hlavičku a eventuální blok procedury (resp. funkce). V tomto bloku označuje identifikátor proměnnou, která je typu uvedeného ve specifikaci indexů a které je na začátku aktivace bloku přiřazena hodnota příslušné meze indexu typu pole skutečného parametru. Hodnota této proměnné nesmí být v bloku ohrožena, takže se chová jako konstantní parametr.

Schéma konformního pole neudává konkrétně typ formálního parametru, ale určuje pouze třídu typů pole, které jsou konformní se schematem konformního pole a které mohou být typem skutečného parametru dosazeného za formální parametr.

Typ pole je konformní se schématem konformního pole právě když platí současně všechny dále uvedené podmínky. Přitom předpokládáme, že ve schématu konformního pole je pouze jedna specifikace indexů, což umožňuje ekvivalence zkráceného a rozšířeného zápisu schématu konformního pole:

- typ pole i schéma konformního pole jsou současně buď označeny jako packed nebo nejsou označeny jako packed,
- meze indexů typu pole jsou kompatibilní pro přiřazení s typem uvedeným ve specifikaci indexů schématu konformního pole,
- je-li složka konformního pole popsána schématem konformního pole, pak složka typu pole je s ní konformní,
- je-li složka konformního pole identifikátor typu, pak je tento typ totožný s typem složky typu pole.

Příklad:

Typ T2 je konformní s oběma schématy konformních polí:

```
type
  T1 = array [1..2] of char;
  T2 = array ['A'..'Z'] of T1;
   ...
  array [C1..C2:char; I..J:integer] of char;
  array [C1..C2:char] of T1;
```

Za formální parametry popsané jedním schématem konformního pole je třeba dosadit skutečné parametry, které jsou všechny stejného typu. Pokud jsou formální parametry volané odkazem nebo jsou konstantní mohou být za ně jako skutečné parametry dosazeny formální parametry konformní pole. U parametrů volaných hodnotou tato možnost není.

Příklady:

```
procedure P (var A : array [I..J:integer] of char);
begin
  ...
  P(A); { rekurzivní volání procedury P, kde jako skutečný
          parametr je použit formální parametr A }
  ...
end;

procedure SOUCVEKT(var A,B,C:array [I..J:integer] of real);
  var K:INTEGER;
begin
  for K:=I to J do
    A[K]:=B[K]+C[K]
end;
```

4.6 Rozsahy platnosti

a) Každé návěští nebo identifikátor obsažený v bloku programu nebo v deklarační jednotce musí mít své definiční místo. Každému definičnímu místu přísluší oblast, jež je částí textu kompilační jednotky a rozsah platnosti, kterým je buď celá tato oblast, nebo její část.

b) Oblasti příslušející definičním místům jsou vymezeny následujícími pravidly:

b1) oblastí příslušející definičnímu místu
- návěští (v úseku deklarací návěští)je blok bezprostředně obsahující toto definiční místo;

b2) oblastí příslušející definičnímu místu
- identifikátoru konstanty (v úseku deklarací konstant nebo v popisu výčtového typu),
- identifikátoru typu (v úseku deklarací typu),
- identifikátoru proměnné (v úseku deklarací proměnných),
- identifikátoru procedury nebo funkce (v úseku deklarací procedur a funkcí)

je blok nebo deklarační jednotka bezprostředně obsahující toto definiční místo;

b3) oblastí příslušející definičnímu místu
- identifikátoru formálního parametru (v seznamu specifikací parametrů)
- identifikátoru meze (ve schématu konformního pole)

je seznam specifikací parametrů bezprostředně obsahující toto definiční místo. Jedná-li se přitom o seznam specifikací parametrů deklarované procedury nebo funkce (a nikoliv o seznam specifikací parametrů formální procedury nebo funkce), pak oblastí příslušející tomuto definičnímu místu je rovněž blok dané procedury nebo funkce;

b4) oblastí příslušející definičnímu místu
- identifikátoru položky (v popisu typu záznam)

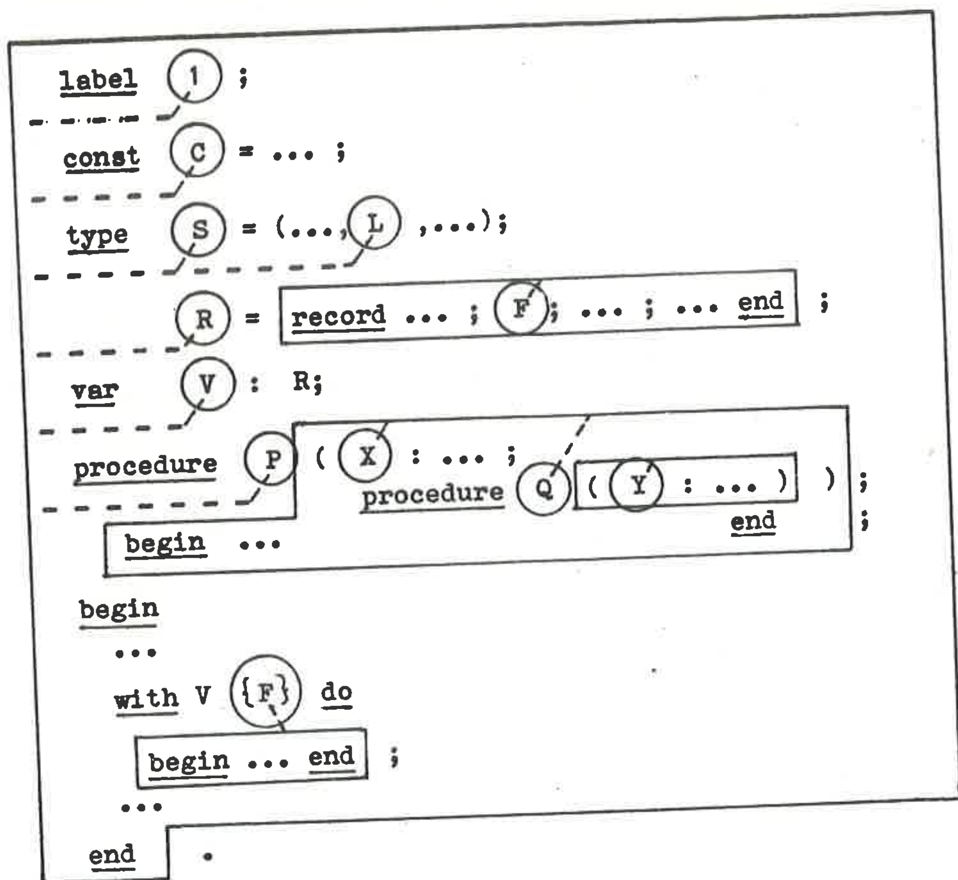
je popis typu záznam bezprostředně obsahující toto definiční místo;

b5) oblastí příslušející definičnímu místu
- identifikátoru určení položky (v příkazu with)

je příkaz obsažený v příkazu with.

Příklad:
Výše uvedená pravidla ilustruje obrázek, na němž je každé definiční místo uvedeno v kroužku a spojeno s hranicí oblasti, která tomuto definičnímu místu přísluší.

program ... ;



Obr. Definiční místa a jim příslušející oblasti

c) Rozsahem platnosti každého definičního místa je oblast příslušející tomuto definičnímu místu včetně všech oblastí do ní vnořených, avšak s výjimkou těch oblastí, které jsou z něho vyjmuty na základě následujícího pravidla:

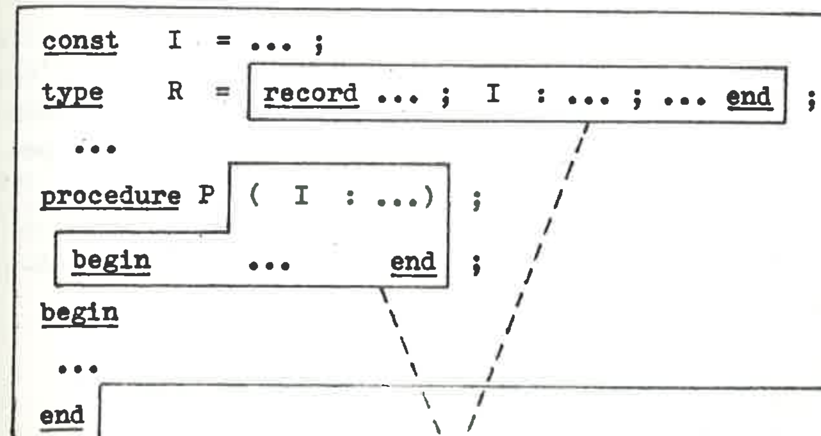
Má-li identifikátor nebo návěští definiční místo D pro oblast A a má-li další definiční místo pro nějakou oblast B, která je do A vnořena, pak oblast B a všechny

oblasti do ní vnořené jsou vyjmuty z rozsahu platnosti definičního místa D.

Příklad:

Ilustrací k tomuto pravidlu je obrázek.

program ... ;



tyto oblasti nepatří do rozsahu platnosti definičního místa identifikátoru konstanty I

Obr. Vyjmutí oblasti z rozsahu platnosti

Poznámka: Z rozsahu platnosti se vyjímají celé oblasti a nikoliv jen jejich části počínaje vnořeným definičním místem. Jinými slovy, lokálním definičním místem identifikátoru je zastíněn nelokální význam tohoto identifikátoru v celé oblasti, a ne jen v té její části, která začíná lokálním definičním místem. Použití identifikátoru MAX v následujícím příkladu deklarací je tedy chybné.

```
const MAX = 100;
type ZAZN = record
              P:1..MAX;     {nedovolené použití}
            MAX:integer
            end;
```

d) Rozsah platnosti definičního místa identifikátoru (resp. návěští) nesmí obsahovat žádná jiná definiční místa téhož identifikátoru (resp. návěští).

e) Všechny výskyty identifikátoru (resp. návěští) uvnitř rozsahu platnosti definičního místa, s výjimkou výskytu který tvoří definiční místo, jsou použití identifikátoru (resp.návěští) odpovídající příslušnému definičnímu místu. Výskyt identifikátoru (resp. návěští) vně rozsahu platnosti definičního místa není použití odpovídající příslušnému definičnímu místu.

f) Definiční místo identifikátoru musí předcházet všem použitím tohoto identifikátoru s jedinou výjimkou: v popisu typu ukazatel může být použit identifikátor typu, jehož definiční místo je v témž bloku na základní úrovni.

Příklady:

```
1. type
       TP = ↑SEZN;
       SEZN = record
                  X:integer
                  NXT:TP
              end;

2. var P : ↑LIST;
   type
       LIST = record
                  X,Y:integer
              end;
```

g) Identifikátory označující standardní konstanty, typy, proměnné, procedury a funkce se používají tak, jako by jejich definičním místům příslušela oblast obsahující celou programovou jednotku.

Poznámka: Z předchozího bodu vyplývá, že významy standardních identifikátorů lze změnit deklarací.

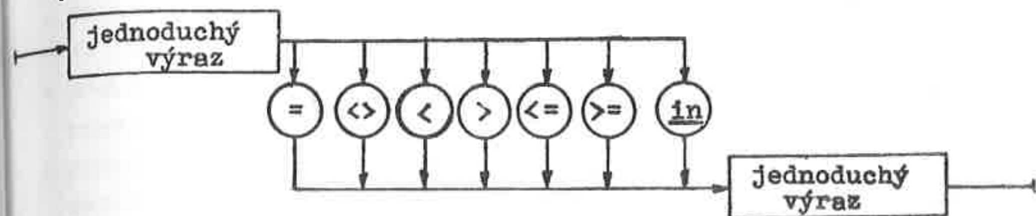
h) Použití identifikátoru označuje vždy to, co je tímto identifikátorem označeno v odpovídajícím definičním místě.

5. VÝRAZY

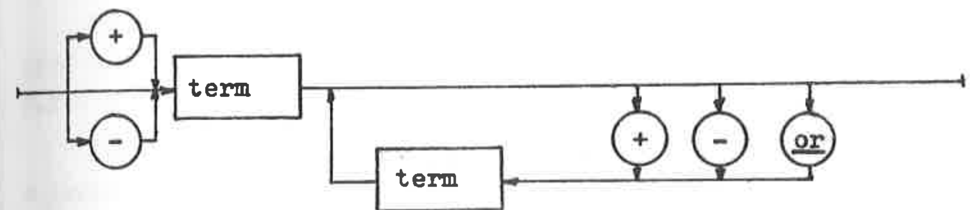
5.1 Obecně

Výpočet nové hodnoty ze známých hodnot operandů se provádí unárními a binárními operacemi, zápisem standardní nebo deklarované funkce a konstruktorem množiny. Operandem může být konstanta bez znaménka nebo proměnná, zápis funkce nebo konstruktor množiny. Všechny výpočty se provádějí v rámci výrazu. Každý výraz je určitého typu.

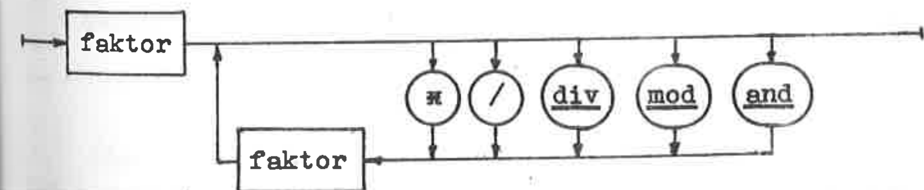
výraz



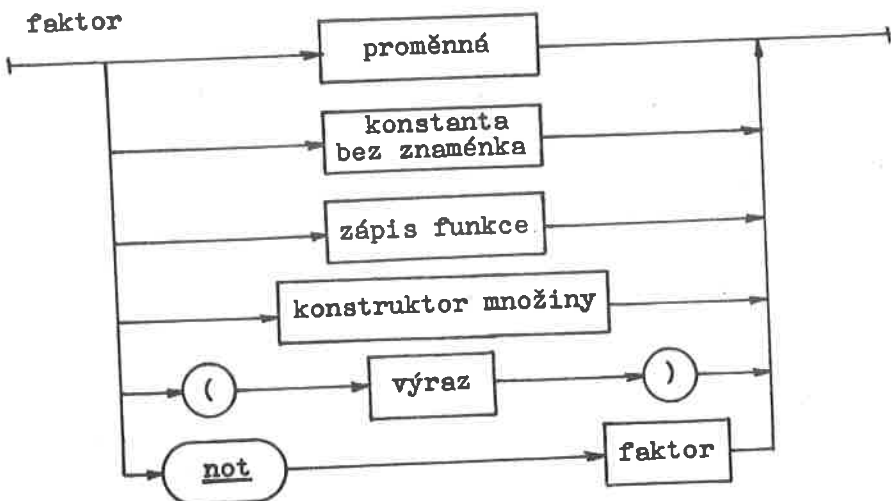
jednoduchý výraz



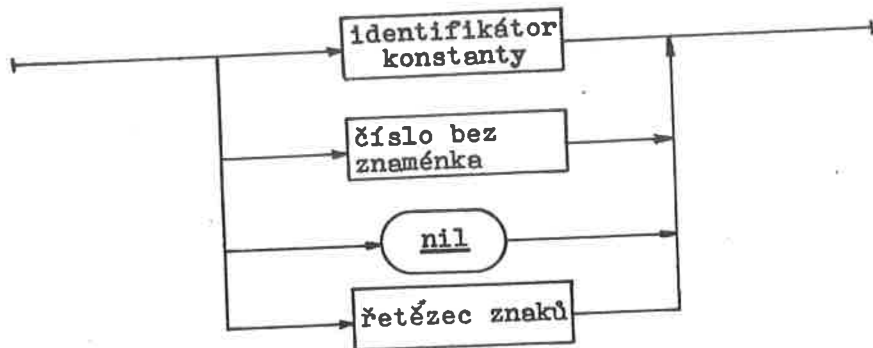
term



faktor



konstanta bez znaménka



S faktorem typu interval z T se pracuje stejně jako s faktorem typu T. S faktorem typu množina se pracuje stejně jako s faktorem příslušného kanonického typu množina (viz 4.3.3.4).

5.2 Proměnné

5.2.1 Obecně

Jako proměnné se souhrnně označují tyto objekty:

- proměnné deklarované v úseku deklarací proměnných (viz 4.4),
- parametry volané odkazem, hodnotou nebo konstantní parametry (viz 4.5.4),
- meze ve specifikaci indexů konformního pole (viz 4.5.4),
- proměnné identifikované ukazateli (viz kap.8),
- složky proměnných typu pole a záznam (viz 4.3.3.2 a 4.3.3.3),
- přístupové proměnné souborů (viz 4.3.3.5).

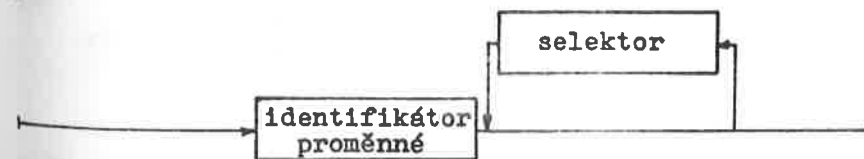
Proměnným je možno přiřazovat hodnotu příslušného typu. Přiřazení je možné těmito způsoby:

- proměnná je použita na levé straně přiřazovacího příkazu (viz 6.2.1),
- proměnná je parametrem standardní procedury read nebo readln (viz 7.1.1.4 a 7.1.2.2),
- proměnná je použitá jako skutečný parametr za formální parametr volaný odkazem (viz 4.5.4),
- proměnná je použita jako řídící proměnná cyklu for (viz 6.2.6).

Použití proměnné v některé z těchto konstrukcí se nazývá ohrožení hodnoty proměnné a je někdy zakázáno (viz 6.2.6 a 4.5.4).

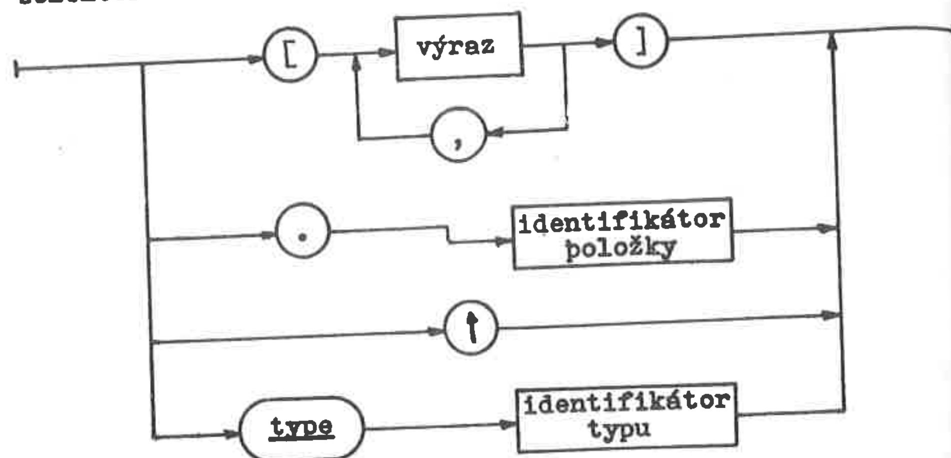
Přístup k proměnné je dán identifikátorem proměnné a posloupností selektorů (může být prázdná).

proměnná



Identifikátorem proměnné je nejen identifikátor, jehož definiční místo je v deklaraci proměnných, ale v příkazu with též identifikátor položky, která patří do příslušného typu záznam (viz 6.3.7). Proměnnou bez selektorů nazýváme úplnou proměnnou.

selektor



Pomocí selektorů se označují:

- složky proměnných typu pole (selektor pole)
- složky proměnných typu záznam (selektor záznamu)
- proměnné identifikované ukazateli (dereference ukazatele)
- přístupové proměnné souborů
- změna typu proměnné

V následujících příkladech odstavce 5.2 budeme předpokládat platnost těchto deklarací:

```
type
   POLE1 = array [1..10] of integer;
   POLE2 = array [1..20] of POLE1;
   UKAZP = ↑POLE1;
   UKAZZ = ↑ZAZN2;
   ZAZN1 = record
             S, T : char;
             U : POLE1
           end;
   SOUB  = file of integer;
   ZAZN2 = record
             F : integer;
             G : ZAZN1;
             N : POLE1;
             Q : UKAZZ;
            FF : SOUB
           end;
```



```
   POLEZ = array [1..3] of ZAZN1;
   POLEU = array [1..10] of UKAZP;
   POLES = array [1..5] of file of ZAZN1;

var
   I : 1..3;                    UP : UKAZP;
   A : POLE1;                   UZ : UKAZZ;
   AA: POLE2;                   PZ : POLEZ;
   R : ZAZN2;                   PU : POLEU;
   FL: SOUB;                    PS : POLES;
```

Příklady označení úplných proměnných:

```
   I             { proměnná typu 1..3 }
   A             { proměnná typu POLE1 }
   R             { proměnná typu ZAZN2 }
   UP            { proměnná typu UKAZP }
   FL            { proměnná typu SOUB }
```

5.2.2 Selektor pole

Selektor pole je tvořen výrazem, který udává index prvku pole a je uzavřen do hranatých závorek.

Jestliže

x	je proměnná typu **array** [T1] **of** T2 a
ind	je výraz, jehož hodnota je kompatibilní vzhledem k přiřazení s typem T1,

pak

x[ind]	je typu T2 a označuje ten prvek pole x, jehož index je roven hodnotě výrazu ind.

Je-li typ výrazu ind kompatibilní s typem T1, avšak hodnota výrazu ind nepatří do množiny hodnot typu T1 a parametr překladu R je nastaven na +, hlásí se při výpočtu chyba.

Zápis x[ind1, ind2,..., indn] má stejný význam jako zápis x[ind1][ind2]...[indn].

Příklady:

A[10]	{proměnná typu integer}
A[I*25 <u>mod</u> 3+1]	{proměnná typu integer}
AA[1]	{proměnná typu POLE1 }
AA[1,20]	{proměnná typu integer }
AA[1][20]	{tentýž význam }
R.N[I]	{proměnná typu integer }
PZ[2]	{proměnná typu ZAZN1 }
PZ[2].U[I]	{proměnná typu integer }

5.2.3 Selektor záznamu

Selektor záznamu je tvořen identifikátorem položky, před níž je uvedena tečka.

Jestliže

x   je proměnná typu **record** ... p:T ... **end** ,

pak

x.p   je typu T a označuje tu položku záznamu x, jejíž identifikátor je p.

Příklady:

R.F	proměnná typu integer
R.G.S	proměnná typu char
PZ[1].U	proměnná typu POLE1
UZ .F	proměnná typu integer

5.2.4 Dereference ukazatele

Dereference ukazatele, tzn. přístup k proměnné, kterou ukazatel identifikuje, vyjadřuje operátor ↑.

Jestliže

x   je proměnná typu ↑T, jejíž hodnota není **nil**,

pak

x↑   je typu T a označuje proměnnou, kterou hodnota proměnné x identifikuje.

Pokud je hodnota proměnné x **nil** a je nastaven parametr R na + hlásí se při výpočtu chyba.

Příklady:

UP↑	{proměnná typu POLE1}
R.Q↑	{proměnná typu ZAZN2}
PU[1]↑	{proměnná typu POLE1}
R.Q↑.Q↑	{proměnná typu ZAZN2}

5.2.5 Přístupová proměnná souboru

S každou proměnnou typu soubor je sdružena přístupová proměnná souboru, pomocí níž se pracuje s jednotlivými složkami souboru.

Jestliže

x   je proměnná typu **file of** T,

pak

x↑   je typu T a označuje přístupovou proměnnou souboru x.

Příklady:

FL↑	{přístupová proměnná typu integer souboru FL}
R.FF↑	{přístupová proměnná typu integer souboru R.FF}
PS[1]↑	{přístupová proměnná typu ZAZN1 souboru PS[1]}

5.2.6 Přetypování proměnné

Zvláštním případem selektoru je přetypování proměnné. Je zprávou pro kompilátor, aby vybranou proměnnou považoval za proměnnou uvedeného typu.

Jestliže

x   je proměnná typu T1,

pak

x **type** T2   je proměnná typu T2.

Příklad:

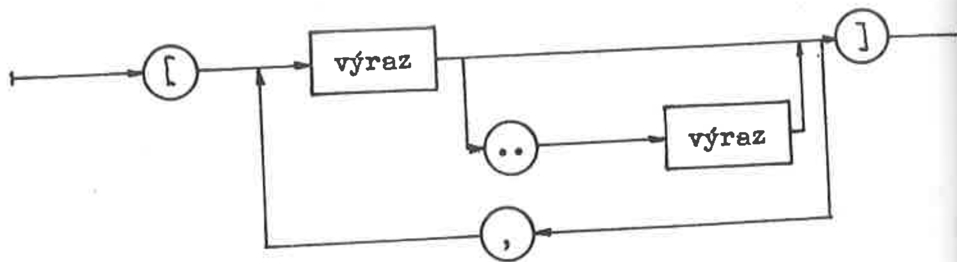
Přepínač překladu /SET:15

```
type
  T = set of 0..15;
var
  I:integer;
begin
  ...
  I type T:=I type T-[0..2]; {zamaskování tří nejniž-
                               ších řádů celého čísla
                                              maskou }
  ...
  I type T:=I type T-[3..15]; {I:=I mod 8}
  :
```

5.3 Konstruktor množiny

Konstruktor množiny označuje hodnotu typu množina určenou výčtem jejích prvků.

konstruktor množiny



Konstruktor množiny ve tvaru [] označuje prázdnou množinu, která patří do každého typu množina.

Všechny výrazy vyskytující se v konstruktoru množiny musí být kompatibilních ordinálních typů a musí označovat přípustné hodnoty prvků množin. Pokud jsou tyto hodnoty typu integer, nepatří do intervalu 0 až SETM a je nastaven parametr překladu R na +, hlásí se při výpočtu chyba (konstanta SETM viz kap.12). Konstruktor množiny označuje hodnotu kanonického typu set of T, jestliže všechny výrazy

v něm se vyskytující mají typ kompatibilní s T. Každá dvojice výrazů x..y v konstruktoru množiny označuje všechny hodnoty z uzavřeného intervalu <x,y> pokud x<=y (je-li x>y, pak dvojice x..y neoznačuje žádnou hodnotu). Každý výraz, který není součástí dvojice x..y, označuje jeden prvek množiny s udanou hodnotou.

Příklady konstruktorů množin:

```
var  ZN1, ZN2: '0'..'9';

     ['A', 'E', 'I', 'O', 'U', 'X']     {set of char}
     ['A'..'Z', '0'..'9']               {set of char}
     [ZN1, '+', ZN2..'9']               {set of char}
     [1, 2, 10]                         {set of 0..SETM}
     [0..15]                            {set of 0..SETM}
```

5.4 Operace

5.4.1 Obecně

Operace výpočtu hodnoty na základě hodnot operandů se dělí na

- aritmetické operace (+, -, *, /, div, mod)
- boolovské operace (not, and, or)
- množinové operace (+, -, *)
- relační operace (=,<>, <, >, <=, >=, in)
- operace s ukazateli (+, -)

Operátory označující tyto operace se dělí podle priority do čtyř skupin:

4. not (nejvyšší priorita)
3. *, /, div, mod, and
2. +, -, or
1. =, <> , < , > , <=, >=, in (nejnižší priorita)

Posloupnost operátorů se stejnou prioritou se vyhodnocuje zleva doprava.

Příklady výrazů:

A+B*C	{= A+(B*C)}
X/Y/Z	{= (X/Y)/Z}
not P or Q and R	{= (not P) or (Q and R)}
(1<=I) and (I<=10)	{1<=I and I<=10 je špatně}

5.4.2 Aritmetické operace

Typy operandů a výsledků pro binární a unární aritmetické operace jsou uvedeny v tabulkách.

operátor	operace	typy operandů	typ výsledku
+	sčítání	integer nebo real	integer pro oba operandy typu integer, jinak typ real
-	odčítání	integer nebo real	
*	násobení	integer nebo real	
/	dělení	integer nebo real	real
div	celočíselné dělení	integer	integer
mod	modulo	integer	integer

Tab. Binární aritmetické operace

operátor	operace	typ operandu	typ výsledku
+	identita	integer	integer
		real	real
-	změna znaménka	integer	integer
		real	real

Tab. Unární aritmetické operace

Výsledek operace div je definován takto:

a) Jestliže j≠0, pak pro hodnotu i div j platí:
abs(i) - abs(j) < abs((i div j) * j) <= abs(i);
tato hodnota je nulová, jestliže abs(i) < abs(j), v ostatních případech bude mít kladné znaménko, když i a j mají stejná znaménka a bude mít záporné znaménko, když i a j mají různá znaménka.

b) Jestliže j=0, pak hodnota i div j není definovaná. Parametrem překladu J3 (viz kap.12) je možno určit, že operace i div j, kde j je konstanta 2^n, (2,4,8..) se provádí aritmetickým posuvem. V tom případě platí výše uvedené vztahy jen pro nezáporné hodnoty i.

Výsledek operace mod je definován takto:

a) Jestliže j>0, pak hodnota i mod j se rovná hodnotě (i-(k*j)) pro takové celé k, pro nějž je 0 <= i mod j < j.

b) Jestliže j <= 0, pak hodnota i mod j není definovaná.

Poznámka: Relace (i div j) * j + i mod j = i
platí pouze pro i>=0 a j>0.

Příklady:

5 div 2 = 2	5 mod 2 = 1
3 div 4 = 0	3 mod 4 = 3
-7 div 3 = -2	-7 mod 3 = 2

Každá aritmetická operace, pro kterou tabulka udává typ výsledku integer, probíhá jako operace celočíselné aritmetiky v pevné řádové čárce. Nedefinovaný výsledek (přetečení) je hlášen jako chyba, pokud je parametr překladu R nastaven na +.

Každá aritmetická operace, pro kterou tabulka udává typ výsledku real, probíhá jako operace reálné aritmetiky v pohyblivé řádové čárce (je-li přitom jeden z operandů typu integer, převede se nejprve na odpovídající hodnotu typu real). Přesnost výsledku závisí na přesnosti zobrazených hodnot typu real.

5.4.3 Boolovské operace

Boolovské operace mají operandy typu boolean a výsledky rovněž typu boolean. Jsou označeny těmito operátory:

not	(unární) negace
or	(binární) disjunkce
and	(binární) konjunkce

Disjunkce a konjunkce se někdy vyhodnocují zkráceným způsobem:

a) při vyhodnocení disjunkce x or y se výraz y vyhodnotí pouze tehdy, když x má hodnotu false,

b) při vyhodnocení konjunkce x and y se výraz y vyhodnotí pouze tehdy, když x má hodnotu true.

5.4.4 Množinové operace

Množinové operace jsou definovány pro kompatibilní typy množina, typ výsledku je příslušný kanonický typ množina. Jsou označeny těmito operátory:

+	sjednocení množin
-	rozdíl množin
*	průnik množin

Příklady množinových operací:

```
var   CS : set of char;
      CS1: set of '0'..'9';
      IS : set of 0..127;
      IS1: set of 0..15;

      CS + ['A']              set of char
      CS * CS1 - ['0']        set of char
      IS + IS1                set of 0..SETM
      IS1 - [10..15]          set of 0..SETM
```


5.4.5 Relační operace

Relační operace dávají výsledky typu boolean, další jejich vlastnosti se liší pro různé třídy typů operandů.

a) Jednoduché typy

Pro jednoduché typy jsou definovány následující relační operace:

=	rovnost
<>	nerovnost
<	menší než
>	větší než
<=	menší nebo rovno
>=	větší nebo rovno

Typy operandů musí být kompatibilní nebo jeden operand může být typu integer a druhý typu real. Je-li alespoň jeden operand typu real, operace se provádí v pohyblivé řádové čárce.

b) Typy ukazatel

Z hlediska těchto relací se hodnoty typu ukazatel chápou jako celá čísla bez znaménka z intervalu ⟨0, 2*maxint +1⟩. Konstanta nil se chápe jako 0.

c) Typy řetězec

Pro typy řetězec jsou definovány tytéž relační operace jako pro jednoduché typy. Typy operandů musí být kompatibilní, tzn. musí to být řetězce stejných délek. Relační operace s hodnotami s1 a s2 typu řetězec délky n se vyhodnocuje podle těchto pravidel:

i) s1 = s2, jestliže pro všechna i∈⟨1, n⟩ platí s1[i] = s2[i];

ii) s1 < s2, jestliže existuje takové p∈⟨1,n⟩, pro které s1[p]<s2[p] a pro všechna i∈⟨1,p-1⟩ platí s1[i] = s2[i].

Příklady:

'ABC' < 'ABD'   'ABC1'<'ABCD'

d) Typy množina

Pro typy množina jsou definovány následující relační operace:

=	rovnost	
<>	nerovnost	
<=	je podmnožinou	(X<=Y znamená $X \subseteq Y$)
>=	je nadmnožinou	(X>=Y znamená $X \supseteq Y$)
in	je prvkem	(P in X znamená $P \in X$).

Typ levého operandu operátoru in musí být kompatibilní s bázovým typem množiny uvedené jako pravý operand. Ostatní množinové relace vyžadují operandy kompatibilních typů množina.

Je-li nastaven parametr překladu R na + a levý operand v relaci in je typu integer a jeho hodnota je větší než konstanta SETM, hlásí se chyba.

5.4.6 Operace s ukazateli

Pro hodnoty typu ukazatel jsou definovány následující operace:

+	přičtení doplňku
-	odečtení doplňku nebo rozdíl ukazatelů.

Přípustné typy operandů, typ výsledku a významy jsou uvedeny v tabulce.

operátor	operace	typ levého operandu	typ pravého operandu	typ výsledku
+	přičtení doplňku	↑T	integer	↑T
-	odečtení doplňku	↑T	integer	↑T
-	rozdíl ukazatelů	↑T	↑T	integer

Tab. Binární operace s ukazateli

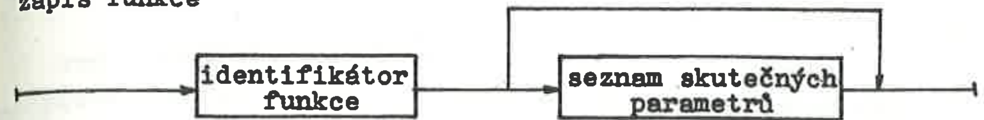
Při těchto operacích se pracuje s ukazateli jako s celými čísly, která představují adresy bytů logické paměti. Uvedené operace mají význam v souvislosti se standardními funkcemi ref a size (viz kap.8).

5.5 Zápis funkce

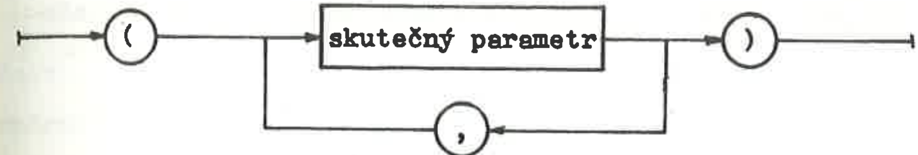
5.5.1 Obecně

Zápis funkce je výraz označující funkční hodnotu. Tato hodnota se vypočte vyvoláním funkce s příslušnými skutečnými parametry.

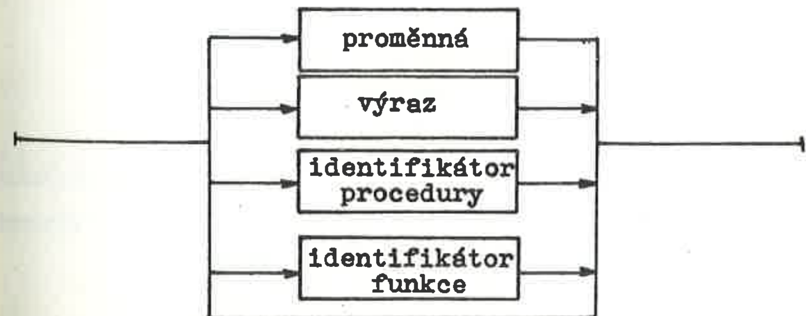
zápis funkce



seznam skutečných parametrů



skutečný parametr



Jestliže funkce má formální parametry, potom zápis funkce musí obsahovat seznam skutečných parametrů, které budou vázány s odpovídajícími formálními parametry. Korespondence mezi skutečnými a formálními parametry je dána

pořadím parametrů v seznamu skutečných resp. formálních parametrů. Počet skutečných parametrů musí být stejný jako počet formálních parametrů. Skutečný parametr musí být buď přípustným skutečným parametrem (viz 4.5.4 a 5.5.2), a nebo může být u nestandardních funkcí prázdným skutečným parametrem, pokud pro dané volání funkce nemá skutečný parametr význam.

Příklad zápisu funkce, kde je druhý skutečný parametr prázdný:

F(I*3, {prázdný parametr}, true);

5.5.2 Standardní funkce

5.5.2.1 Obecně

Standardní funkce jsou jazykem definované funkce pro výpočet hodnoty nebo pro zjišťování stavu proměnných během výpočtu. Zápis těchto funkcí znamená buď vyvolání knihovního podprogramu anebo se výpočet přímo generuje do kódu. Přípustné skutečné parametry jsou pro každou funkci určeny zvláštním způsobem.

Seznam standardních procedur:

dispose	(viz 8.1)
get	(viz 7.1.1.3 a 7.1.2.2)
new	(viz 8.1)
pack	(viz dále)
page	(viz 7.1.1.1)
put	(viz 7.1.1.1 a 7.1.2.1)
read	(viz 7.1.1.4 a 7.1.2.2)
readln	(viz 7.1.1.4)
reset	(viz 7.1.1.3 a 7.1.2.2)
rewrite	(viz 7.1.1.1 a 7.1.2.1)
unpack	(viz dále)
write	(viz 7.1.1.2 a 7.1.2.1)
writeln	(viz 7.1.1.1)
inline	(viz dále)

sin	(viz 5.5.2.2)
size	(viz 8.2)
sqr	(viz 5.5.2.2)
sqrt	(viz 5.5.2.2)
succ	(viz 5.5.2.3)
trunc	(viz 5.5.2.3)

Zvláštní druh standardních funkcí je označen identifikátorem výčtového typu (viz 5.5.2.3).

5.5.2.2 Aritmetické funkce

Poznámka: x dále označuje výraz příslušného typu.

abs(x) - hodnotou je absolutní hodnota x. x může být typu integer nebo real. Výsledek je stejného typu jako argument.

sqr(x) - hodnota je x^2. Argument může být typu integer nebo real. Výsledek je stejného typu jako argument. Jestliže je výsledek mimo rozsah příslušného typu hlásí se chyba.

sin(x), cos(x), exp(x), ln(x), sqrt(x) (odmocnina),

arctan(x) - hodnotou je aproximace příslušné matematické funkce pro daný argument. Argument může být typu integer nebo real, výsledek je typu real. Argument sin, cos a arctan je v radiánech. Jestliže je argument nepřípustný nebo je výsledek mimo rozsah typu real,hlásí se chyba.

5.5.2.3 Ostatní funkce

trunc(x) - argument musí být typu real, výsledek je typu integer. Jestliže x>=0, potom platí 0≤x-trunc(x)<1; jinak -1<x-trunc(x)≤0. Jestliže výsledek je mimo rozsah typu integer, hlásí se chyba.

round(x) - argument musí být typu real, výsledek je typu integer. Jestliže x>=0, potom platí

round(x)= trunc(x+0.5)

jinak round(x) = trunc(x-0.5).

Jestliže výsledek je mimo rozsah typu integer, hlásí se chyba.

Příklady relací, jejichž hodnota je true:

trunc (3.5) = 3
trunc (-3.5) = -3
round (3.5) = 4
round (-3.5) = -4

ord(x) - argument musí být ordinálního typu, výsledek je typu integer. Hodnotou je ordinální číslo hodnoty výrazu x.

chr(x) - argument musí být typu integer; výsledek je typu char. Hodnotou je znak (nemusí mít grafickou reprezentaci), jehož ordinální číslo odpovídá hodnotě argumentu. Chyba se hlásí pokud takový znak neexistuje (hodnota x je mimo interval < 0,127>) a parametr překladu R je nastaven na +.

T(x) - T je identifikátor výčtového typu nebo jeho intervalu. Funkce T je inverzní funkce k ord - argument je typu integer, výsledek je typu T. Chyba se hlásí pokud je parametr překladu R nastaven na + a hodnota T(x) neexistuje.

Příklad:

```
type
  BARVA = (BILA, ZELENA, MODRA, ZLUTA);
var
  X : BARVA;
begin
  X := BARVA(0);
  ...
```

succ(x), pred(x) - argument musí být ordinálního typu, výsledek je stejného typu jako argument. Ordinální číslo hodnoty výsledku je o jedna vyšší resp. nižší než ordinální číslo argumentu. Jestliže výsledek neexistuje

a je nastaven parametr překladu R na +, hlásí se chyba.

odd(x) - argument musí být typu integer, výsledek je typu boolean. Pro výsledek platí relace:

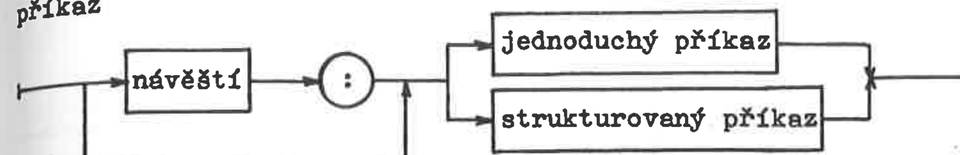
odd(x) = ((x mod 2) = 1)

6. PŘÍKAZY

6.1 Obecně

Každý příkaz označuje algoritmickou akci, která se uskuteční provedením příkazu. Příkaz může obsahovat návěští.

příkaz



Návěští vyskytující se v příkazu se nazývá návěštím příkazu. V daném rozsahu platnosti na něj může směřovat příkaz skoku.

Příkazy se dělí na jednoduché a strukturované. Jednoduché příkazy neobsahují jiné příkazy. Jednoduchými příkazy jsou:

- přiřazovací příkaz,
- příkaz procedury,
- příkaz skoku,
- prázdný příkaz.

Strukturované příkazy se skládají z dílčích příkazů a předepisuje se jimi postupné, podmíněné či opakované provádění dílčích příkazů. Příkaz with je speciální strukturovaný příkaz, pomocí něhož se optimalizují přístupy k položkám proměnné typu záznam. Strukturovanými příkazy jsou

- složený příkaz,
- podmíněný příkaz (příkazy if a case),
- příkaz cyklu (příkazy repeat, while a for),
- příkaz with.

Následující příklady příkazů v této kapitole předpokládají platnost těchto deklarací:

```
type
  PRIROZENE = 1..maxint;
  BARVA = (CERVENA, ZLUTA, ZELENA, MODRA);
  POLE  = array [1..100] of integer;
  STRING4 = packed array [1..4] of char;
  STRING6 = packed array [0..5] of char;
  DATUM   = record
              MESIC : 1..12; ROK : PRIROZENE
            end;
  DATUMY  = array [1..10] of DATUM;

var
  X, Y, Z : real;
  I, J, MAX : integer;
  ODST : set of BARVA;
  K : PRIROZENE;
  ZN : char;
  P, Q : boolean;
  OP : (PLUS, MINUS, KRAT);
  V : BARVA;
  A, B : POLE;
  C : array [1..100] of integer;
  S4 : STRING4;
  S6 : STRING6;
  M, M1, M2 : array [1..10, 1..10] of real;
  DAT : DATUM;
  SB1, SB2 : file of POLE;
  R,S :↑DATUMY
  DATY : DATUMY;

procedure CHYBA(N:integer);
  ...
procedure CTIHEX(var F:text; var N:PRIROZENE);
  ...
procedure TISKHEX(X:integer);
  ...
procedure VEMZNAK(var Z:char);
  ...
```



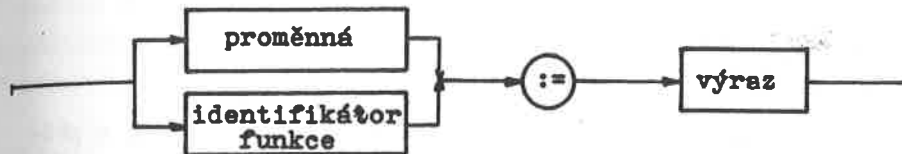
```
procedure TRIDENI(var P:POLE; N:integer);
  ...
procedure CTIRETEZ(var S:STRING4);
  ...
procedure PISRETEZ(S:STRING6; L:integer);
  ...
procedure VYJIMKA(procedure ER(N:integer));
  ...
```

6.2 Jednoduché příkazy

6.2.1 Přiřazovací příkaz

Přiřazovací příkaz přiřazuje hodnotu výrazu proměnné nebo funkci.

přiřazovací příkaz:



Přiřazovací příkaz, na jehož levé straně je identifikátor funkce, se může vyskytovat pouze uvnitř bloku této funkce (buď v jeho příkazové části nebo ve vnořeném bloku). Tímto příkazem se definuje funkční hodnota, která je výsledkem vyvolání funkce.

Hodnota výrazu musí být kompatibilní vzhledem k přiřazení s typem proměnné (resp. funkce) - viz 4.3.5. Je-li proměnná (resp. funkce) typu interval nebo množina s bázovým typem interval, pak při provádění příkazu se kontroluje přípustnost hodnoty pouze tehdy, má-li v místě výskytu přiřazovacího příkazu parametr překladu R hodnotu +. Použití proměnné na levé straně přiřazovacího příkazu je jeden ze způsobů ohrožení její hodnoty (viz 5.2.1).

Příklady správných přiřazovacích příkazů:

```
X := Y+Z
Y := 2*I
P := (1<=Z) and (Z<=100)
ODST := [MODRA, succ(V)]
K := K+1                 { má-li parametr překladu R hodnotu +,
                           provede se při výpočtu kontrola
                           přiřazované hodnoty }

A := B
SB1↑ := A
S6 := 'ABCDEF'
S6 := '      '           { přiřadí se šest mezer }
```

Příklady nesprávných přiřazovacích příkazů:

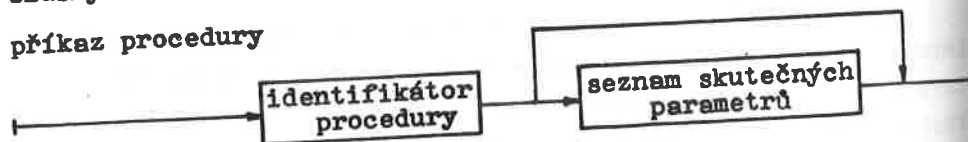
```
I := X          { konverze real - integer není implic.}
SB1 := SB2      { přiřazení není definováno pro typ soubor }
A := C          { proměnné A a C nejsou téhož typu }
```

6.2.2 Příkaz procedury

6.2.2.1 Obecně

Provedením příkazu procedury se vyvolá procedura s příslušnými skutečnými parametry.

příkaz procedury



Pro seznam skutečných parametrů v příkazu procedury platí stejná pravidla jako v zápisu funkce (viz 5.5.1).

Příklady správných příkazů procedury:

```
CHYBA(I)
CTIHEX(input, K)
TISKHEX(J-1)
VEMZNAK(ZN)
TRIDENI(B, 50)
CTIRETEZ(S4)
```



```
PISRETEZ(S6, 6)
PISRETEZ('XXXX')     { hodnota skutečného parametru se
                       doplní zprava o dvě mezery }
VYJIMKA(CHYBA)
```

Příklady nesprávných příkazů procedury:

```
TISKHEX(1.5)
CTIHEX(input, I)     { I není typu PRIROZENE }
VEMZNAK(S4[1])       { skutečným parametrem je prvek
                       zhuštěného pole a parametr je
                       volán odkazem }
TRIDENI(C,100)       { C není typu POLE }
VYJIMKA(VEMZNAK)     { formální procedura nemá s proce-
                       durou TISKHEX kompatibilní seznam
                       formálních parametrů }
```

6.2.2.2 Standardní procedury

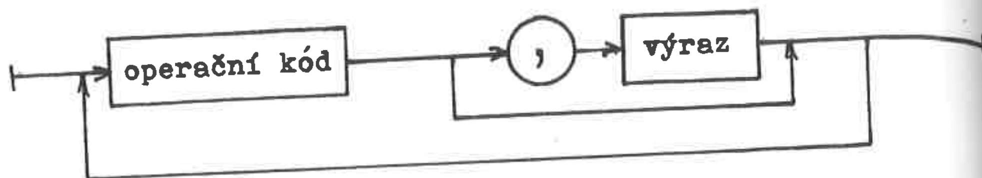
Standardní procedury jsou jazykem definované procedury, které není třeba deklarovat. Příkaz standardní procedury může znamenat volání knihovních procedur nebo se příslušná akce přímo generuje do kódu. Přípustné skutečné parametry jsou pro každou proceduru určeny zvlášť.

Seznam standardních procedur:

dispose	(viz 8.1)
get	(viz 7.1.1.1.3 a 7.1.2.2)
new	(viz 8.1)
pack	(viz dále)
page	(viz 7.1.1.1)
put	(viz 7.1.1.1 a 7.1.2.1)
read	(viz 7.1.1.4 a 7.1.2.2)
readln	(viz 7.1.1.4)
reset	(viz 7.1.1.1.3 a 7.1.2.2)
rewrite	(viz 7.1.1.1 a 7.1.2.1)
unpack	(viz dále)
write	(viz 7.1.1.1.2 a 7.1.2.1)
writeln	(viz 7.1.1.1)
inline	(viz dále)

Procedura inline

parametry procedury inline



Pomocí standardní procedury inline je možno vkládat do pascalského programu části kódu napsané v assembleru. Parametry procedury inline jsou instrukce assembleru a některé pseudoinstrukce pro definici konstant. Tvar operačního kódu instrukcí a pseudoinstrukcí je popsán v čl.2.

Výraz v parametrech procedury inline musí být konstantní,t.zn. vyhodnotitelný v době překladu. Tento výraz reprezentuje parametr assemblerovské instrukce nebo pseudoinstrukce. Parametrem může být, kromě aritmetického výrazu, také adresa statické proměnné. Statická proměnná je taková, která je deklarovaná na základní úrovni kompilační jednotky nebo lokální proměnná statické procedury nebo funkce. Za lokální proměnnou považujeme v tomto případě i parametr, jehož hodnota je předávána v tabulce parametrů (parametry volané hodnotou a konstantní parametry jednoduchých typů, typu ukazatel a typu množina). Adresa statické proměnné musí být získána operací ref.

Pomocí pseudoinstrukcí můžeme do pascalského programu vkládat konstanty různých typů. Dovolené pseudoinstrukce mají následující sémantiku:

DEFB - parametrem pseudoinstrukce musí být hodnota, která je možno zobrazit na jednom bytu. To je ordinální hodnota, jejíž ordinální číslo leží v intervalu 0..255 a dále množina, která nemá více než osm prvků.

DEFW - parametrem pseudoinstrukce musí být hodnota,kterou je možno zobrazit na jednom slově. Jsou to všechny ordinální hodnoty, adresy statických promměnných, dvojznakové řetězy a množiny, které mají devět až šestnáct prvků.

DEFSS - parametrem pseudoinstrukce je hodnota, kterou je možno zobrazit na třech a více bytech (s výjimkou hodnoty typu real). Jsou to víceznakové řetězy a množiny, jejichž počet prvků je větší než šestnáct.

DEFR - parametrem pseudoinstrukce je hodnota typu real.

Pozn.: Počtem prvků množiny se myslí počet prvků jejího kanonického typu (viz 4.3.3.4).
Použité pseudoinstrukce nejsou zcela ekvivalentní obdobným pseudoinstrukcím assembleru.

Příklad:

```
static procedure X;
var ADR : ↑char;
begin
  inline(                {uložení návratové adresy}
    "POPH                {procedury X do lokální  }
    "PUSHH               {proměnné ADR            }
    "SHLD, ref(ADR)
       );
  .
  .
  .
end;
```

Procedury pack a unpack

Procedury pack a unpack umožňují obecně provádět přesuny mezi poli. Jejich jména jsou zachována z důvodů kompatibility se standardním Pascalem. Předpokládejme, že a je proměnná typu <u>array</u> [s1] <u>of</u> T nebo typu <u>packed</u> <u>array</u> [s1] <u>of</u> T, z je proměnná typu <u>array</u> [s2] <u>of</u> T nebo typu <u>packed</u> <u>array</u> [s2] <u>of</u> T a x je výraz typu <u>array</u> [s2] <u>of</u> T nebo <u>packed</u> <u>array</u> [s2] <u>of</u> T (může to být i konstanta řetězec znaků). s1 a s2 jsou kompatibilní typy a i je výraz, jehož hodnota je kompatibilní pro přiřazení s typem s1. u resp. v je nejmenší resp. největší hodnota typu s2. Přitom příkaz pack (a,i,z) je ekvivalentní příkazu:

```
for J:= u to v do z[J]:= a[i+J-u]
```

a příkaz unpack (x,a,i) je ekvivalentní příkazu

```
begin
  X:=x;
  for J:= u to v do   a[i+J-u] :=X[J]
end
```

J a X jsou pomocné proměnné, které se v programu jinak nevyskytují. Skutečná realizace příkazů pack a unpack je samozřejmě optimálnější než náhradní příkazy. Pokud příkazy nelze provést (proměnná J nabývá hodnot mimo rozsah určený typem indexu) a je nastaven parametr překladu R na +, hlásí se při výpočtu chyba.

6.2.3 Příkaz skoku

Provedením příkazu skoku se předá řízení tomu příkazu, který je opatřen návěštím uvedeným v příkazu skoku.

Návěští uvedené v příkazu skoku musí být návěštím nějakého příkazu (kromě toho musí být deklarováno v úseku dekla-

rací návěští toho bloku, v jehož příkazové části je použito jako návěští příkazu).

Nejčastějším použitím příkazu skoku je předčasné ukončení strukturovaného příkazu, procedury nebo funkce. Toto použití ilustruje následující příklad:

```
program UKAZKAGOTO(input, output);
label 99;
var X:integer;...
procedure PRESKOCMEZERY;
  label 10;
  begin
    while input↑=' ' do
      begin get(input);
        if eof(input) then goto 10
      end;
   10:
  end;
procedure CHYBA;
  begin
    writeln('CHYBA VE VSTUPNICH DATECH');
    goto 99
  end;
begin
  while not eof(input) do
    begin read(X);
      if ... then CHYBA;
      {zpracování hodnoty}
      if ... then CHYBA;
      PRESKOCMEZERY
    end;
  writeln('VSTUPNI DATA BEZ CHYBY');
 99:writeln('KONEC PROGRAMU')
end.
```

Příkaz skoku nesmí směřovat dovnitř strukturovaného příkazu. Je-li návěštím označen příkaz ze sekvence příkazů,

potom příslušný příkaz skoku musí být součástí této sekvence, anebo být vnořen do některého příkazu této sekvence.

Příklad možných skoků:

```
begin
  9:A:=B;
   ...
   goto 9;
   ...
end
```

```
repeat
   ...
   if B1 then goto 9;
   ...
9:{prázdný příkaz (viz 6.2.4)}
until B2
```

Jestliže příkaz označený návěštím není součástí sekvence příkazů, musí být příkaz skoku do něho vnořen.

Příklad:

```
if X>Y then
9 : begin
      ...
      goto 9
      ...
    end
```

Skok, kterým se opouští procedura nebo funkce, může směřovat pouze na návěští, kterým je označen příkaz na nejvyšší úrovni vnoření příkazů v bloku. Pokud je v programu deklarováno návěští, které se nevyskytuje v žádném příkazu skoku, je hlášena chyba.

6.2.4 Prázdný příkaz

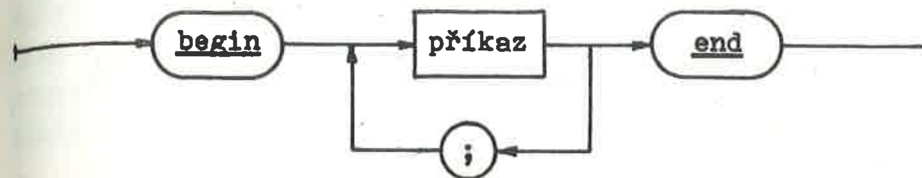
Prázdný příkaz neobsahuje žádný symbol a neoznačuje žádnou akci. Používá se nejčastěji v souvislosti s návěštím, které označuje konec složeného příkazu. Toto použití ilustruje procedura PRESKOCMEZERY z předchozího příkladu, v jejíž příkazové části označuje návěští 10 prázdný příkaz, který je posledním příkazem této příkazové části.

6.3 Strukturované příkazy

6.3.1 Složený příkaz

Složený příkaz předepisuje postupné provedení dílčích příkazů, které jsou v něm uvedeny.

složený příkaz



Stejnou syntaxi jako složený příkaz má též příkazová část bloku.

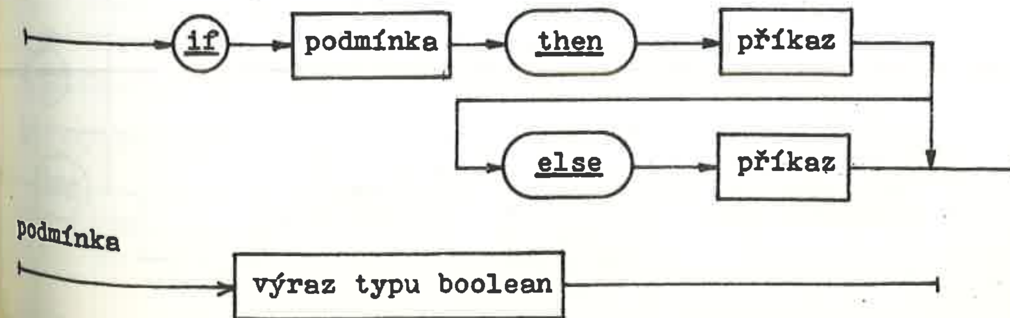
Dílčí příkazy ve složeném příkazu jsou oddělovány středníkem. Vzhledem k tomu, že příkazem je též prázdný příkaz, může být středník uveden i za posledním (neprázdným) příkazem. Oba následující příklady složeného příkazu jsou tedy správné:

```
begin X:=Y; Y:=Z; Z:=X end
begin X:=Y; Y:=Z; Z:=X; end
```

6.3.2 Příkaz if

Příkazem if se předepisuje podmíněné provedení dílčího příkazu resp. provedení jednoho ze dvou dílčích příkazů v závislosti na splnění nějaké podmínky.

příkaz if



Jestliže výraz tvořící podmínku má hodnotu true, provede se příkaz za slovem then. V opačném případě se provede příkaz za slovem else, pokud je tato část příkazu if uvedena. Jsou-li příkazy if do sebe vnořeny, přičemž jeden má a druhý nemá část else, řeší se tato konstrukce tak, že slovo else tvoří dvojici s nejblíže předcházejícím slovem then. Následující příkazy jsou tedy ekvivalentní:

```
if podm1 then if podm2 then příkaz1 else příkaz2
if podm1 then
  begin if podm2 then příkaz1 else příkaz2 end
```

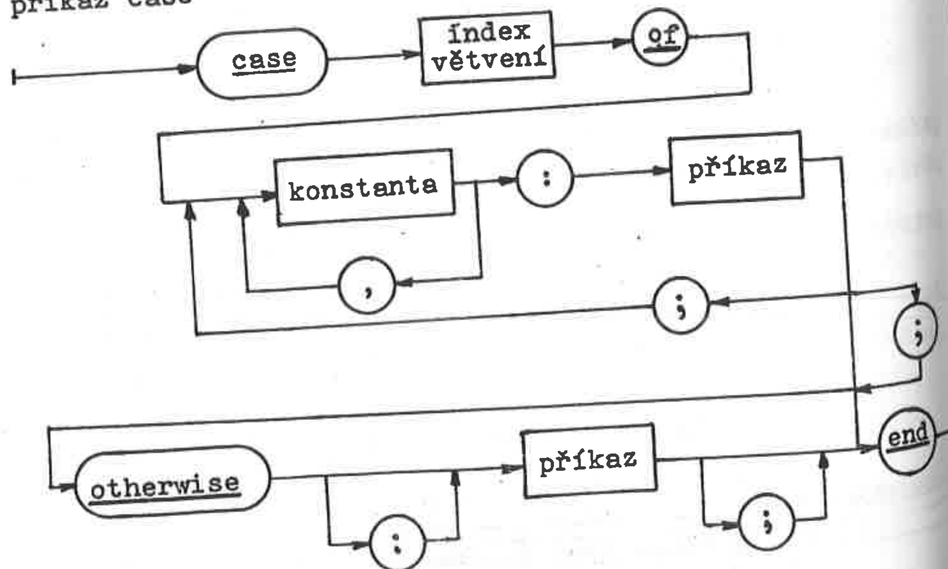
Příklady příkazu if:

```
if X<1.5 then Z:=X+Y else Z:=1.5
if X<Y then begin Z:=X; X:=Y; Y:=Z end
if J=0 then
  if I=0 then writeln('NEDEFINOVANY')
  else writeln('NEKONECNY')
else writeln(I/J)
```

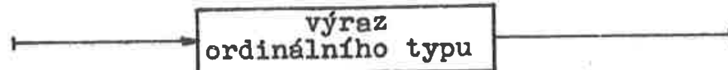
6.3.3 Příkaz case

Příkazem case se předepisuje provedení jednoho příkazu z několika alternativních.

příkaz case



index větvení



Při provádění příkazu case se nejprve vypočte hodnota indexu větvení (musí být ordinálního typu) a pak se provede ten příkaz, před nímž je uvedena konstanta označující vypočtenou hodnotu. Jestliže takový příkaz mezi alternativami není, pak v případě, že je uvedena část otherwise, provede se příkaz z této části,v ostatních případech se hlásí chyba, pokud je parametr překladu R nastaven na +.

Všechny konstanty označující alternativní příkazy musí být kompatibilní s typem indexu větvení a rozdíly jejich ordinálních čísel nesmí být větší než 255.

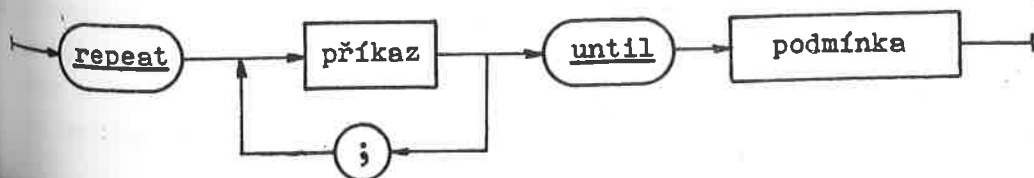
Příklady příkazu case:

```
case OP of
  PLUS : X:=X+Y;
  MINUS: X:=X-Y;
  KRAT : X:=X*Y
end
case I of
  1:X:=sin(X);
  2:X:=cos(X);
  3:X:=exp(X);
otherwise:X:=abs(X)
end
```

6.3.4 Příkaz repeat

Příkazem repeat se předepisuje opakované provádění posloupnosti příkazů až do splnění nějaké podmínky.

příkaz repeat



Posloupnost příkazů v příkazu repeat se provádí opakovaně (s výjimkou změny v provádění způsobené příkazem skoku) tak dlouho, až po jejím provedení má výraz uvedený jako podmínka hodnotu true. Posloupnost příkazů se provede vždy alespoň jednou, neboť podmínka za until se vyhodnocuje až po provedení této posloupnosti příkazů.

Příklad:

```
repeat
  K:=I mod J; I:=J; J:=K
until J=0
```

6.3.5 Příkaz while

Příkazem while se předepisuje opakované provádění příkazu, pokud platí nějaká podmínka.

příkaz while

Příkaz while ve tvaru

```
while podm do příkaz
```

je ekvivalentní příkazu

```
begin
  if podm then
    repeat příkaz until not podm
end
```

Příkaz uvedený v příkazu while se nemusí provést ani jednou, neboť výraz uvedený jako podmínka se vyhodnocuje před provedením příkazu.

Příklady:

```
while I>0 do
  begin
    if odd(I) then Z:=Z*X;
    I:=I div 2; X:=sqr(X)
  end
```

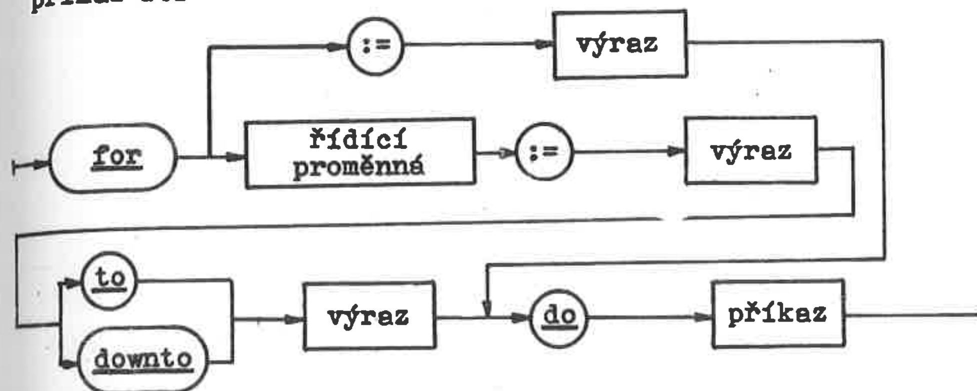


```
while not eof(SB1) do
  begin
    read(SB1,A); TRIDENI(A,100); write(SB2,A)
  end
```

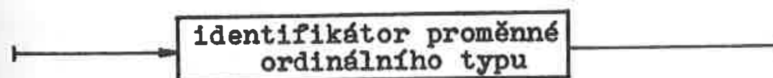
6.3.6 Příkaz for

Příkazem for se předepisuje opakované provádění příkazu, přičemž se řídící proměnné cyklu mají postupně přiřazovat hodnoty z nějakého intervalu.

příkaz for



řídící proměnná



Příkaz může nebo nemusí mít řídící proměnnou. Pokud řídící proměnná existuje, musí to být proměnná v bloku lokální a nesmí být uvnitř příkazu for ani ve vnořených blocích ohrožena její hodnota (viz 5.2.1). Oba výrazy v příkazu for musí být ordinálního typu a musí být kompatibilní vzhledem k přiřazení s typem řídící proměnné. První výraz udává první hodnotu řídící proměnné a druhý výraz poslední hodnotu řídící proměnné, pro kterou má být příkaz proveden. Pokud v příkazu for není uvedena řídící proměnná, musí mít výraz hodnotu typu integer a jeho hodnota udává, kolikrát se cyklus provede.

Příkaz for ve tvaru

```
    for rp := výraz1 to výraz2 do příkaz
```

má týž význam, jako příkaz

```
    begin
      pom1 := výraz1;
      pom2 := výraz2;
      if pom1 <= pom2 then
        begin
          rp := pom2;
          příkaz;
          while rp <> pom1 do
            begin
              rp := succ(rp);
              příkaz
            end
        end
    end
```

a příkaz for ve tvaru

```
    for rp := výraz1 downto výraz2 do příkaz
```

má týž význam jako příkaz

```
    begin
      pom1 := výraz1;
      pom2 := výraz2;
      if pom1 >= pom2 then
        begin
          rp := pom1;
          příkaz;
          while rp <> pom2 do
            begin
              rp := pred(rp);
              příkaz
            end
        end
    end
```

kde pom1 a pom2 jsou pomocné proměnné téhož typu, jakého je řídící proměnná.

Příkaz for ve tvaru

```
    for := výraz do příkaz;
```

má týž význam jako příkaz

```
    for pom := 1 to výraz do příkaz
```

kde pom je pomocná proměnná typu integer.

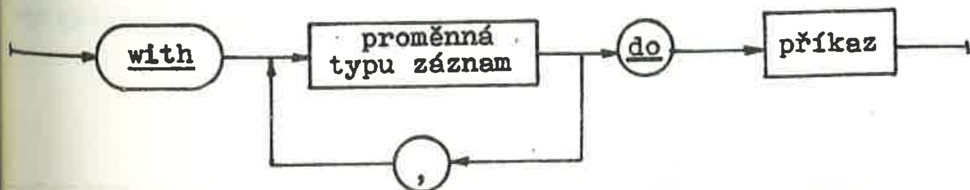
Příklady:

```
    for I := 2 to 100 do
      if A[I] > MAX then MAX := A[I]

    for I := 1 to 10 do
      for J := 1 to 10 do
        begin
          X := 0;
          for K := 1 to 10 do
            X := X + M1[I,K]* M1[K,J];
          M[I,J] := X
        end

    for I := 2 to 10 do M[I] := M[1]
    for V := MODRA downto CERVENA do
      if V in ODST then I := I + 1
```

6.3.7 Příkaz with

Příkaz with umožňuje optimalizovat přístup k položkám proměnné typu záznam.

příkaz with



Nechť příkaz with má tvar with z do p a proměnná z je typu T. Potom výskyt této proměnné v příkazu with tvoří pro každý identifikátor položky, která patří do typu T, jeho definiční místo jako identifikátoru proměnné, který ozna-

čuje příslušnou položku v proměnné z. Rozsahem platnosti tohoto definičního místa je příkaz p.

Příkaz with ve tvaru

```
    with z1, z2,...,zn do p
```

je ekvivalentní příkazu

```
    with z1 do with z2 do ... with zn do p.
```

Příklad:

```
with DAT do
  if MESIC = 12 then
    begin
      MESIC := 1; ROK := ROK+1
    end else
      MESIC := MESIC+1
```

Tento příkaz je ekvivalentní příkazu

```
if DAT.MESIC = 12 then
  begin
    DAT.MESIC := 1; DAT.ROK := DAT.ROK+1
  end else DAT.MESIC := DAT.MESIC+1
```

Přístup k proměnné v příkazu with se provede pouze jednou a to na začátku provádění příkazu with. To znamená, že pokud je v přístupu k proměnné z indexový výraz nebo dereference ukazatele, není příkaz with totožný s příkazem p, ve kterém by každé použití identifikátoru x položky typu T bylo nahrazeno z.x.

Příklad:

```
with R↑[I] do
  begin
    R := S;
    I := I+1;
    MESIC := 10
  end
```


Tento příkaz není ekvivalentní příkazu:

```
begin
  R := S;
  I := I+1;
  R↑[I].MESIC := 10
end
```

Ŝ-poznámkou J2 (viz 12.3) je možno předepsat, že přístup k proměnné z se provede na každém místě, kde se vyskytuje identifikátor položky proměnné z. Potom jsou výše uvedené příkazy ekvivalentní.

7. SOUBORY

Soubory jsou v pascalském programu reprezentovány pomocí proměnných typu soubor. Tyto proměnné budeme dále nazývat logickými soubory. Konkrétní implementace logických souborů budeme nazývat fyzickými soubory. Každému logickému souboru odpovídá fyzický soubor. Vznik fyzického souboru a jeho vazba na příslušný logický soubor je buď automatická nebo může být explicitně předepsána uživatelem.

7.1 Logické soubory

Logický soubor může být deklarován na základní úrovni kompilační jednotky nebo lokálně v proceduře resp. funkci a nebo může vzniknout dynamicky procedurou new. Ve všech případech může být soubor úplnou proměnnou nebo složkou strukturované proměnné (nemůže být složkou proměnné typu soubor).

Logickému souboru, jehož označení je F, přísluší přístupová proměnná souboru, jejíž označení je F↑ . Přístupová proměnná souboru reprezentuje aktuální složku souboru, tzn. složku souboru, která je v daném okamžiku přístupná. Je-li soubor typu <u>file</u> <u>of</u> T, pak přístupová proměnná je typu T.

Rozlišují se dvě třídy logických souborů:

a) textové soubory, tj. soubory standardního typu text;
b) binární soubory, tj. soubory typu <u>file</u> <u>of</u> T.

7.1.1 Textové soubory

Textový soubor je posloupnost znaků členěná na řádky. Z hlediska logické struktury mají řádky proměnnou délku a každý řádek (tedy i poslední) je zakončen oddělovačem řádku. Přístupová proměnná F↑ textového souboru F je typu char.

Zvláštní postavení mezi textovými soubory mají standardní soubory input a output. Deklarace těchto souborů se provede v hlavní jednotce programu uvedením v seznamu parametrů programu a v deklarační jednotce (tj. modulu) v sekci deklarace externích proměnných:

```
program X(input,output);        modul Y;
   .
   .                            external var
   .                              input,output : text;
                                    .
                                    .
                                    .
```

Jejich další vlastnosti jsou popsány dále.

V následujících odstavcích budeme předpokládat, že F je soubor typu text.

7.1.1.1 Vytváření textových souborů

Pro vytváření textových souborů slouží standardní procedury rewrite, put, writeln, page a write.

Procedura rewrite

Příkaz procedury rewrite má tvar:

rewrite(F)

a způsobí otevření textového souboru na výstup. Do takto otevřeného souboru lze zapisovat procedurami put, writeln, page a write. Pro standardní soubor output se operace rewrite provede automaticky.

Procedura put

Příkaz procedury put přidá znak přiřazený přístupové proměnné do výstupního souboru. Příkaz má tvar:

put(F)

Po provedení příkazu je hodnota přístupové proměnné nedefinovaná.

Procedura writeln

Procedura writeln má proměnný počet parametrů a může mít jeden z následujících tvarů:

```
writeln(F,p1,p2,...,pn)
writeln(p1,p2,...,pn)
writeln(F)
writeln
```

kde p1 až pn jsou parametry stejného tvaru jako u procedury write (viz dále). Příkazem writeln(F) se do souboru F zapíše znak konce řádky (ukončí se vytvářená řádka).

Příkaz

writeln(F,p1,p2,...,pn)

je ekvivalentní příkazu

```
begin
  write(F,p1,p2,...,pn);
  writeln(F)
end
```

Není-li v příkazu procedury uveden parametr F, aplikuje se procedura na standardní soubor output.

Procedura page

Příkaz procedury page má tvar:

page(F)

resp.

page

Příkazem page je ukončena řádka (je-li neprázdná) a je zajištěno, že další řádka bude jako první na nové stránce.

Není-li uveden parametr F, je procedura page aplikována na standardní soubor output.

7.1.1.2 Výstupní konverze

Procedura write

Procedura write má proměnný počet parametrů. Příkaz této procedury má tvar

write(F, p1, p2, ... , pn)

resp.

write(p1, p2, ... , pn)

kde F je textový soubor otevřený pro vytváření a p1, p2, ... , pn jsou parametry, které mohou mít jeden z následujících tvarů.

e   e:m   e:m:n

Zde e je výraz, jehož hodnota se do souboru F zapisuje, a m resp. n jsou výrazy typu integer jejichž význam závisí na typu výrazu e (viz.dále). Chybí-li v příkazu procedury parametr F, aplikuje se procedura na standardní soubor output.

Procedura write má následující efekt:

a) Příkaz write(F, p1, p2, ... , pn) je ekvivalentní příkazu begin write(F, p1); write(F, p2); ... write(F, pn) end

b) Jestliže e je výraz typu ukazatel, pak se do souboru F zapíše:
   b1) příkazem ve tvaru write(F,e) hexadecimální hodnota výrazu e v délce čtyři znaky (nejprve dvě hexadecimální číslice pro vyšší slabiku, pak dvě pro nižší).
   b2) příkazem ve tvaru write(F,e:m) obsah paměti od hodnoty výrazu e v délce m slabik, vždy dvě hexadecimální číslice na slabiku.

c) Jestliže e je výraz typu char, pak příkazem write(F,e:m)

se do souboru F zapíše:
   m-1 mezer (je-li m<=1, pak žádná mezera),
   znak, který je hodnotou výraz e.
Příkaz write(F,e) je v tomto případě ekvivalentní příkazu write(F,e:1)

d) Jestliže e je výraz typu integer a dekadický zápis jeho hodnoty obsahuje p číslic, pak příkazem write(F,e:m) se do souboru F zapíše:
   c1) je-li m>=p+1, pak
      m-p-1 mezer (je-li m=p+1, pak žádná mezera),
      znaménko ´-´, je-li e<0, jinak mezera,
      p číslic dekadického zápisu hodnoty e;
   c2) je-li m<p+1, pak
      znaménko ´-´, je-li e<0, jinak žádné znaménko
      p číslic dekadického zápis hodnoty e.

Příkaz write(F,e) je v tomto případě ekvivalentní příkazu write(F,e:1).

e) Jestliže e je výraz typu real, pak příkazem write(F,e:m) se do souboru F zapíše:
   d1) je-li m>=8, pak
      znaménko ´-´, je-li e<0, jinak mezera,
      prvá číslice normalizované mantisy x, pro níž platí $x.10^z=|e|$, $1 <= x < 10$,
      znak ´.´,
      m-7 číslic desetinné části mantisy x se zaokrouhlenou poslední číslicí,
      znak ´E´,
      znaménko exponentu z ("+" nebo "-"),
      2 číslice exponentu;
   d2) je-li m<8, pak se zapíše stejná posloupnost znaků jako v předchozím případě, ale pouze s jednou číslicí desetinné části. Příkaz je v tomto případě ekvivalentní příkazu write(F,e:8);
   d3) Příkaz write(F,e), kde e je výraz typu real, je ekvivalentní příkazu write(F,e:13).

e) Jestliže e je výraz typu real, pak příkazem write(F,e:m:n) se do souboru F zapíše:

- tolik mezer, aby celkový počet znaků zapsaných při konverzi byl roven m.
  Pokud je počet následujících znaků konverze větší nebo roven m, nezapíše se žádná mezera.
- znaménko ´-´, je-li e< 0
- číslice ´0´, je-li |e|< 1; jinak dekadická reprezentace celé části e
- znak ´.´
- n číslic desetinné části e se zaokrouhlením poslední číslice

Pozn.: Pro obě výstupní konverze typu real tedy platí: Nelze-li zaokrouhlenou hodnotu výrazu e zapsat předepsaným počtem znaků, zapíše se takovým počtem znaků, aby hodnota e byla správně vyjádřena.

f) Je-li e výraz typu řetězec délky n, pak příkazem write(F,e:m) se do souboru F zapíše:

f1) je-li m>n, pak
m-n mezer,
první až n-tý znak hodnoty e;

f2) je-li m<=n, pak
první až m-tý znak hodnoty e.
Příkaz write (F,e) je v tomto případě ekvivalentní příkazu write(F,e:n).

g) Je-li e výraz typu boolean, pak příkaz write(F,e:m) je ekvivalentní:
- příkazu write(F,´TRUE´:m), má-li e hodnotu true,
- příkazu write(F,´FALSE´:m), má-li e hodnotu false.

Příkaz write(F,e) je v tomto případě ekvivalentní příkazu write(F,e:5).

7.1.1.3 Čtení textových souborů

Čtení textových souborů umožňují standardní procedury reset, get, read, readln a standardní funkce eof a eoln.

Procedura reset

Příkazem standardní procedury reset se textový soubor otevírá na čtení. Tvar příkazu je:

reset(F)

Byl-li soubor před provedením příkazu otevřen pro výstup, je nejprve uzavřen (je-li poslední řádka neprázdná, provede se writeln(F)). Po provedení procedury reset má přístupová proměnná hodnotu prvého znaku další řádky souboru. Pro standardní soubor input se operace reset provede automaticky.

Procedura get

Pomocí procedury get se přiřadí přístupové proměnné souboru další znak souboru. Příkaz této procedury má tvar

get(F)

kde F je textový soubor otevřený pro čtení. Příkaz může být proveden pouze tehdy, platí-li eof(F) = false. Provedením příkazu se posune ukazatel aktuální složky na další složku. Dále má tento příkaz následující efekt:

- Je-li po provedení příkazu get(F) aktuální složkou další znak téhož řádku, pak tento znak je hodnotou přístupové proměnné F↑ a funkce eoln(F) má hodnotu false.

- Je-li po provedení příkazu get(F) aktuální složkou oddělovač řádku, pak hodnotou přístupové proměnné F↑ je znak ´␣´ (mezera) a funkce eoln(F) má hodnotu true.

- Je-li před provedením příkazu get(F) aktuální složkou oddělovač řádku, za nímž následuje další řádek, pak provedením příkazu se aktuální složkou stane první znak tohoto řádku a tento znak je rovněž hodnotou přístupové proměnné F↑ . Funkce eoln(F) má přitom hodnotu false.
- Je-li před provedením příkazu get(F) aktuální složkou oddělovač řádku, za nímž již není další řádek, pak po provedení příkazu má funkce eof(F) hodnotu true (ve všech předchozích případech má hodnotu false) a hodnota přístupové proměnné F↑ není definovaná.

Funkce eoln

Funkce eoln je standardní boolovská funkce, která indikuje, že aktuální složkou souboru je oddělovač řádku. Tvar zápisu je

eoln(F)

resp. eoln

Je-li parametr vynechán, aplikuje se funkce na standardní soubor input. Hodnotou funkce je true právě tehdy, je-li aktuální složkou souboru oddělovač řádku.

Funkce eof

Funkce eof je standardní boolovská funkce, která indikuje konec souboru. Konec souboru je nalezen, jestliže je aktuální složkou souboru poslední oddělovač řádky a je zavolána procedura get(F) nebo je aktuální složkou znak z poslední řádky a je zavolána procedura readln(F). Tvar zápisu je

eof(F)

resp. eof

Je-li parametr vynechán, aplikuje se funkce na standardní soubor input.

7.1.1.4 Vstupní konverze

Při vstupní konverzi se přečte posloupnost znaků, která je vnější reprezentací hodnoty typu char, integer nebo real, vytvoří se odpovídající vnitřní reprezentace a přiřadí se zadané proměnné. Vstupní konverzi provádějí procedury read a readln.

Procedura read

Procedura read má proměnný počet parametrů. Příkaz procedury read má tvar

read(F, V1, V2,..., Vn)

resp. read(V1, V2, ...,Vn)

kde F je textový soubor otevřený pro čtení a V1, V2, ..., Vn jsou proměnné, jejichž typy jsou kompatibilní s typy char, integer nebo real.
Není-li uveden parametr F, procedura read se aplikuje na standardní soubor input.

Procedura read má následující efekt:

a) Příkaz read(F, V1, V2, ... , Vn) je ekvivalentní příkazu begin read(F, V1); read(F, V2); ... read(F, Vn) end
b) Jestliže V je proměnná typu char (nebo interval z char), pak příkaz read(F, V) je ekvivalentní příkazu begin V:=F↑ ; get(F) end
c) Jestliže V je proměnná typu integer (nebo interval z integer), pak provedením příkazu read(F, V) se ze souboru F přečte posloupnost znaků. Počáteční mezery a oddělovače řádků se ignorují. Zbývající část posloupnosti musí být dekadickým zápisem celého čísla. Čtení končí, jakmile hodnotou přístupové proměnné F↑ je znak, který již nepatří do zápisu celého čísla. Hodnota takto přečteného celého čísla se přiřadí proměnné V, přičemž musí být kompatibilní vzhledem k přiřazení s typem proměnné V (kontrola kompatibility vzhledem k přiřazení se provádí pouze

tehdy, jestliže v místě výskytu příkazu má parametr překladu R hodnotu +).

d) Jestliže V je proměnná typu real, pak provedením příkazu read(F, V) se přečte ze souboru F posloupnost znaků. Počáteční mezery a oddělovače řádku se ignorují. Zbývající část posloupnosti musí být zápisem čísla (viz syntaktický diagram číslo v odst. 2.5). Čtení končí, jakmile hodnotou přístupové proměnné F↑ je znak, který již nepatří do zápisu čísla. Hodnota takto přečteného čísla se přiřadí proměnné V.

Skutečnými parametry procedury read mohou být též složky zhuštěných struktur.

Procedura readln

Procedura readln má proměnný počet parametrů. Příkaz této procedury může mít jeden z následujících tvarů:

readln(F, V1, V2, ... , Vn)
resp. readln(V1, V2, ... , Vn)
resp. readln

kde F je textový soubor otevřený pro čtení a V1, V2, ... , Vn jsou proměnné, jejichž typy jsou kompatibilní s typy char, integer nebo real.

Není-li uveden parametr F, aplikuje se procedura readln na standardní soubor input.

Procedura readln má následující efekt:

a) Příkaz readln(F) je ekvivalentní příkazu begin while not eoln(F) do get(F); get(F) end Tímto příkazem se tedy přeskočí zbytek řádku a první znak následujícího řádku se přiřadí přístupové proměnné.

b) Příkaz readln(F, V1, V2, ... , Vn) je ekvivalentní příkazu begin read(F, V1, V2, ... , Vn); readln(F) end

Jestliže v souboru F již další řádek není, pak po provedení příkazu readln(F) má funkce eof(F) hodnotu true a hodnota přístupové proměnné F↑ není definovaná.

7.1.2 Binární soubory

Binárním souborem se rozumí logický soubor typu file of T. Dále budeme předpokládat, že F je binární soubor.

7.1.2.1 Vytváření binárních souborů

Pro vytváření binárních souborů se používají standardní procedury rewrite, put a write.

Procedura rewrite

Provedením příkazu

rewrite(F)

se soubor F otevře na výstup. Funkce eof(F) má hodnotu true.

Procedura put

Před provedením příkazu procedury put musí mít přístupová proměnná F↑ definovanou hodnotu a soubor F musí být otevřen na výstup. Příkazem

put(F)

je hodnota přístupové proměnné F↑ zapsána do souboru F a přístupová proměnná se stane nedefinovanou. Funkce eof(F) má hodnotu true.

Procedure write

Příkaz procedury write na binární soubor má tvar

write(F, e_1,...,e_n)

kde e_1 až e_n jsou výrazy typu T. Příkaz je ekvivalentní příkazu

```
begin
  F↑ := e1; put(F);
  .
  .
  F↑ := en; put(F);
end
```

7.1.2.2 Čtení binárních souborů

Pro čtení binárních souborů se používají standardní procedur reset,get, read a standardní funkce eof.

Procedura reset

Čtení binárních souborů se zahajuje příkazem

reset(F)

Je-li soubor neprázdný je po provedení příkazu hodnota přístupové proměnné F↑ rovna hodnotě první složky souboru a funkce eof(F) má hodnotu false. Pro prázdný soubor je hodnota proměnné F↑ nedefinovaná a eof(F) má hodnotu true.

Funkce eof

Zápis standardní boolovské funkce

eof(F)

indikuje konec souboru F. Konec souboru F je nalezen, jestliže je hodnotou proměnné F↑ poslední složka souboru a je provedena operace get(F). Hodnota proměnné F↑ je potom nedefinovaná.

Procedura get

Pomocí příkazu procedury

get(F)

se přístupové proměnné souboru F↑přiřadí další složka souboru. Před provedením příkazu musí být hodnota eof(F) false.

Procedura read

Příkaz procedury

read(F,a_1,...,a_n)

kde a_1 až a_n jsou proměnné typu T, je ekvivalentní příkazu

```
begin
  a1:= F↑; get(F);
  .
  .
  an:= F↑; get(F)
end
```


7.2 Fyzické soubory

Fyzické soubory jsou vnější implementací logických souborů. Podle doby existence lze soubory dělit na pracovní a permanentní. Pracovní soubory vznikají při vytvoření příslušného logického souboru a existují pouze po dobu běhu programu. Permanentní soubory existují i před a (nebo) po ukončení běhu programu. Jak pracovní tak permanentní soubory mohou být textové nebo binární. Existují tedy čtyři druhy fyzických souborů:

- pracovní textové
- pracovní binární
- permanentní textové
- permanentní binární

Všechny fyzické soubory musí být diskové. Pouze permanentní textové soubory mohou být navíc reprezentovány systémovými zařízeními (LST:, RDR:, PUN:, CON:).

7.2.1 Pracovní soubory

Pracovní soubory jsou soubory na diskové jednotce, která byla jednotkou aktuální v době spuštění programu. Každý logický soubor je v okamžiku vzniku automaticky inicializován (při aktivaci bloku, kde je deklarován nebo při zavolání procedury new). Je-li na takto inicializovaný soubor aplikována procedura rewrite, vznikne příslušný pracovní soubor. Jména pracovních souborů mají tvar ØPWFxxxx.ØØØ, kde xxxx je pořadové číslo souboru. Vzhledem k tomu, že deklarační jednotka (modul) neobsahuje blok, nejsou inicializovány logické soubory, deklarované na její základní úrovni. Existují-li takové soubory, je třeba je explicitně inicializovat procedurou IWFDB resp. IWFDT (viz. 7.2.5).

Po ukončení běhu programu jsou všechny pracovní soubory automaticky zrušeny. Kromě toho lze pracovní soubory rušit explicitně při běhu programu procedurou DELETE (např. pro uvolnění místa na disku resp. v adresáři).

7.2.2 Permanentní soubory

Jako permanentní soubory označujeme fyzické soubory, které existují mimo vlastní běh programu. Vazba permanentního souboru na nějaký logický soubor provedeme procedurou ASSIGN (viz. 7.2.5). Permanentní soubor přitom nemusí ještě existovat. Logický soubor již musí být inicializován. Po provedení procedury ASSIGN můžeme nad příslušným logickým souborem provést operaci reset nebo rewrite. Jestliže permanentní soubor dosud neexistoval, pak se aplikací procedury rewrite vytvoří a při provedení procedury reset nastane chyba. Jestliže permanentní soubor již existoval, pak provedením procedury rewrite se permanentní soubor zruší a znovu vytvoří jako prázdný.

Logickým binárním souborům lze přiřazovat pouze diskové soubory, textovým souborům lze navíc přiřazovat některé ze zařízení CON:, LST:, RDR: a PUN:.

Diskový permanentní soubor otevřený na výstup musí být před skončením existence příslušného logického souboru (opuštění bloku, ve kterém je deklarován; volání procedury dispose) uzavřen voláním procedury CLOSE (viz. 7.2.5.). Stejného efektu lze dosáhnout aplikací procedury reset .

7.2.3 Fyzické textové soubory

7.2.3.1 Diskové textové soubory

Textové soubory jsou standardní diskové textové soubory. Řádky jsou proměnné délky ukončené znaky CR (0DH) a LF (0AH). Délka řádky není omezena.

Volání procedury page způsobí ukončení řádky (byla-li neprázdná) a vložení znaku FF (0CH) na začátek další řádky

Soubor je ukončen znakem ctrl/Z (1AH) za ukončenou poslední řádkou.

7.2.3.2 Textový soubor na systémovém logickém zařízení

Voláním procedury ASSIGN lze logickému textovému souboru přiřadit jako permanentní fyzický soubor také některé z logických systémových zařízení LST:, PUN:, RDR: a CON:.

Výstup na zařízení LST: je realizován voláním systémové funkce č.5 (List Output). Volání procedury rewrite není nutné. Je-li provedeno,má stejný efekt jako volání procedury page. Výstup se provádí po znacích. Soubor není třeba uzavírat.

Výstup na zařízení PUN: se provádí voláním systémové funkce č.4 (Punch Output). Voláním procedury rewrite se v případě, že byl soubor otevřen,doplní znak ctrl/Z (1AH) a vyděruje se cca 15 cm vodící perforace.

Vstup ze zařízení RDR: odpovídá volání systémové funkce č.3 (Reader Input). Prázdné znaky jsou vynechávány, paritní bit je nulován. Za konec souboru je považováno nalezení znaku ctrl/Z (1AH).

Zařízení CON: je implicitně přiřazeno standardním logickým textovým souborům input a output. Je-li zařízení CON: přiřazeno jinému textovému souboru, bude se soubor chovat tak, jak je popsáno v kapitole 7.2.3.3.

7.2.3.3 Fyzická reprezentace souborů input a output

Logické soubory input a output jsou implementovány pomocí systémového logického zařízení CON:. Jedná se tedy o jeden soubor. Tento soubor je neustále otevřen na vstup i výstup (aplikace procedur reset a rewrite je prázdná operace). Aby bylo možno provádět vstup i výstup na jednom zařízení jsou řádky souboru output členěny ještě na zprávy. Jedna zpráva je vyslána do souboru output příkazem put nebo příkazem write. Výstup se provádí voláním systémové funkce č.2 (Console Output) znak po znaku na místo momentální polohy kurzoru. Systém zajišťuje expandování tabulátorů (ctrl/I) příslušným počtem mezer. Ukončení výstupní řádky a posun

kurzoru na novou řádku se provádí procedurou writeln.

Po vyslání každé zprávy a také na začátku běhu programu je zařízení CON: připraveno buď vyslat další zprávu nebo přijmout vstupní řádku. Tomuto stavu říkáme, že je indikován konec vstupní řádky a je možno ho testovat funkcí eoln (nabývá hodnoty true). Přístupová proměnná input↑ má hodnotu znak mezera. Kurzor přitom nemusí být v první pozici na obrazovce.

Je-li ve stavu "indikován konec vstupní řádky" zavolána procedura get (přímo a nebo prostřednictvím procedury read) je vyvolána systémová funkce č.10 (Read Console Buffer) a program očekává vložení vstupního řádku. Vstupní řádek může mít maximálně 80 znaků a při jeho vytváření se provádí echo od okamžité polohy kurzoru. Přitom platí:

- stisk klávesy ctrl/C na první pozici vstupní řádky způsobí okamžité ukončení programu a opětovné zavedení systému
- stisk klávesy ctrl/Z na první pozici vstupní řádky je inerpretován jako konec vstupního souboru (funkce eof má hodnotu true). Ve vstupu však lze pokračovat.
- stisk klávesy ctrl/H ruší poslední platný znak a vrací kurzor o jednu pozici zpět (nejvýše však do první pozice vstupního řádku)
- stisk klávesy ctrl/X ruší celou zadanou řádku a vrací kurzor do první pozice vstupní řádky
- tabulátory (ctrl/I) jsou při echu na obrazovce expandovány na mezery.

Vstup je ukončen stiskem klávesy cr (return a pod.) nebo zadáním 80. znaku. Poté je kurzor přesunut na první pozici následující řádky obrazovky.

Program pokračuje v činnosti a hodnotou přístupové proměnné input↑ je první znak vstupní řádky. Následující volání procedury get způsobí, že přístupová proměnná input↑ nabývá hodnoty jednotlivých znaků vstupní řádky. Vstupní řádka se předává až po editaci, tedy bez řídících znaků.

Po vyčerpání celé délky vstupní řádky je zařízení opět ve stavu "indikován konec vstupní řádky".

Pokud je vyslána výstupní zpráva na zařízení předtím, než je celá vstupní řádka přečtena, je zbytek vstupní řádky ztracen. Obdobný efekt má provedení operace readln.

Pokud je nalezena syntaktická chyba vstupu při vstupní konverzi (při čtení celého čísla není na vstupu znaménko nebo číslice a pod.), provedou se následující akce:

- zapomene se zbytek vstupní řádky
- na obrazovku se napíše znak ´?´
- zavolá se funkce Read Console Buffer
- po zadání nové vstupní řádky se konverze zopakuje

Uživatel musí napsat vstupní řádku znova od místa, kde byla ve vstupu chyba.

Příklad:

```
        .
        .
        .
   repeat
      write(´ZADEJ POCET: ´);
      read(I);
      if I<=0 then goto 9;
      writeln(´POCET MUSI BYT VETSI NEZ 0´);
   until false;
 9:     .
        .
```

Na obrazovce se může objevit např. tento text:

ZADEJ POCET: <u>XX</u>
?<u>0</u>
POCET MUSI BYT VETSI NEZ 0
ZADEJ POCET: <u>4</u>

Pozn.: Podtržené znaky píše uživatel

7.2.4 Fyzické binární soubory

Binární soubory jsou soubory typu file of T. Chovají se zcela standardně podle normy jazyka Pascal. Vytvářejí se pomocí procedur rewrite,put a write; prohlížejí se pomocí procedur reset,get,read a funkce eof. Permanentním binárním souborům mohou být pomocí procedury ASSIGN přiřazeny pouze diskové soubory.

Vzhledem k tomu, že v systému CP/M je velikost souboru dána v logických sektorech po 128 bytech, bylo pro správnou činnost funkce eof zvoleno uložení složek logických souborů v zvislosti od délky složky.

a) soubory s délkou složky size(T)< 64

V každém sektoru může být uloženo maximálně 127 div size(T) složek souboru, přičemž v prvním bytu je uložen jejich skutečný počet. Zbytek sektoru není využit.

b) soubory s délkou složky 64<= size(T)<= 128

Každá složka souboru je uložena právě v jednom sektoru. Je-li size(T)< 128, není zbytek sektoru využit.

c) soubory s délkou složky size(T)>128

Každá složka souboru je uložena v (size(T)-1) div 128+1 sektorech. Zbytek posledního sektoru ze sektorů, ve kterých je uložena jedna složka, může zůstat nevyužit.

7.2.5 Nestandardní procedury systému ovládání souborů

Nestandardní procedury systému ovládání souborů zajišťují většinou přímou vazbu pascalského programu na operační systém. Vzhledem k tomu, že nejsou standardní, musí být nadeklarovány jako externí. Pro jednoduchost je vhodné (podobně jako v následujících příkladech) deklarovat parametr určující soubor jako typu text. Je-li skutečný parametr jiného typu, lze použít konstrukce přetypování proměnné.

Procedura IWFDB

procedure IWFDB(**var** F : TF; L : integer); **external**;

kde TF = **file of** T

Tato inicializační procedura je volána automaticky při vzniku logického binárního souboru (vstup do bloku, kde je deklarován nebo volání procedury new). Proměnné deklarované na základní úrovní deklarační jednotky (modulu) je třeba inicializovat explicitně. Při volání musí být hodnotou skutečného parametru L délka složky souboru (t.j. size(T)).

Procedura IWFDT

procedure IWFDT(**var** F : text); **external**;

Procedura je obdobou procedury IWFDB avšak pro textové soubory.

Procedura ASSIGN

procedure ASSIGN(**var** F : text; **const** NAME : TNAME);
external;

kde TNAME = **packed array**[1..14] **of** char;

Procedura ASSIGN provádí přiřazení permanentního souboru inicializované proměnné typu soubor. Struktura řetězu znaků dosazeného za formální parametr NAME musí být následující:

x:filename.filetyp

kde

x: - označení diskové jednotky, na které se soubor nachází. Může nabývat hodnot A,B až P (podle konfigurace systému). Je-li parametr vynechán, bude uvažována aktuální jednotka.

filename - jméno souboru v délce 1 až 8 znaků. Podle konvencí systému nesmí jméno obsahovat znaky ':', ',' a '.'.

.filetype - typ souboru v délce 1 až 3 znaky. Je-li typ kratší než 3 znaky nebo je-li vynechán, je **doplněn** mezerami.

Pozn.: Jak typ tak jméno souboru může být nejednoznačné, t.j. může obsahovat znak '?'. Logický soubor, kterému bylo přiřazeno nejednoznačné jméno, lze použít pouze v proceduře DELETE pro rušení souborů v adresáři.

Procedura DELETE

procedure DELETE(var F : text); external;

Procedura DELETE zajistí při vyvolání zrušení souboru, který byl přiřazen logickému souboru F a uvolní místo zabrané tímto souborem na disku i v adresáři.

Procedura CLOSE

procedure CLOSE(var F : text); external;

Procedurou CLOSE je třeba uzavřít permanentní soubor otevřený na výstup, končí-li existence příslušného logického souboru, ke kterému je přiřazen (opuštění bloku procedury nebo funkce, volání procedury dispose, konec běhu programu).

Popis dalších nestandartních procedur zajišťujících např. přímý přístup k diskovým souborům je uveden v textovém souboru dodávaném spolu s překladačem.

8. PROMĚNNÉ IDENTIFIKOVANÉ UKAZATELEM

8.1 Ukazatele dynamických proměnných

Dynamické proměnné se vytvářejí v průběhu výpočtu procedurou new a jsou uloženy ve volné (logické paměti za programem. Adresy těchto proměnných jsou z hlediska jazyka Pascal hodnotami typu ukazatel.

Pro vytváření dynamických proměnných slouží standardní procedura new, pro zrušení dynamických proměnných slouží jednak standardní procedura dispose a dále dvojice knihovních procedur MARK a RELEASE.

Procedura new

Příkaz procedury new má tvar

new(P)

resp. new(P, c1, c2, ..., cn)

kde P je proměnná typu ↑T a
c1, c2, ..., cn jsou konstanty ordinálních typů.

Příkazem ve tvaru new(P) se vytvoří dynamická proměnná typu T a její adresa se uloží do proměnné P. Říkáme pak, že hodnota proměnné P identifikuje takto vytvořenou dynamickou proměnnou. Dynamická proměnná má po svém vytvoření nedefinovanou hodnotu. Je-li dynamická proměnná typu variantní záznam, může jí být dalšími příkazy přiřazena hodnota odpovídající libovolné variantě.

Příkaz ve tvaru new(P, c1, c2, ... , cn) lze použít tehdy, je-li doménovým typem ukazatele P variantní záznam, v němž jsou vnořeny varianty odpovídající rozlišovacím konstantám c1, c2, ..., cn. V příkazu procedury musí být rozlišovací konstanty uvedeny v pořadí postupného vnořování variantních částí. Nespecifikovaná varianta může být pouze na hlubší úrovni vnoření, než je úroveň vnoření odpovídající parametru cn. Takto vytvořené dynamické proměnné může být dalšími příkazy přiřazena pouze hodnota odpovídají-

cí specifikované variantě. Takto vytvořená dynamická proměnná dále nesmí tvořit levou stranu přiřazovacího příkazu.

Příklady:

```
type
  PRVNI_UROVEN = (P1, P2);
  DRUHA_UROVEN = (D1, D2, D3);
  ZAZNAM = record
             A : integer;
             case P : PRVNI_UROVEN of
               P1 : (B : char);
               P2 : (case D : DRUHA_UROVEN of
                       D1 : (X : integer);
                       D2 : (Y : char);
                       D3 : ()
                    )
           end;

var
     T, U, V, W :  ZAZNAM;
   ...
   new(T);        {proměnné T↑ lze přiřadit hodnoty odpo-
                   vídající libovolné variantě}
   new(U,P1);     {proměnné U↑ lze přiřadit pouze hodnoty
                   odpovídající variantě s položkami A P B}
   new(V,P2);     {proměnné V↑ lze přiřadit hodnoty odpo-
                   vídající variantě A P D X nebo A P D Y
                   nebo A P D}
   new(W,P2,D2);  { proměnné W  lze přiřadit pouze
                    hodnoty odpovídající variantě
                    A P D Y }
```

Procedura dispose

Příkaz procedury dispose má tvar

dispose(P)

resp.

dispose(P, c1, c2, ..., cn)

kde P je proměnná typu ukazatel a

c1, c2, ..., cn jsou konstanty ordinálních typů.

Provedením příkazu se zruší dynamická proměnná, kterou identifikuje ukazatel P. Konstanty c1, c2, ..., cn musí být v příkazu procedury uvedeny právě tehdy, když byly uvedeny i v příkazu procedury new, kterým se dynamická proměnná P↑ vytvořila.

Procedury MARK a RELEASE

Na místo zrušení dynamických proměnných procedurou dispose je v některých případech výhodnější použít procedury MARK a RELEASE.

Procedury MARK a RELEASE jsou nestandardní a nemají žádné parametry. Provedením příkazu procedury RELEASE se zruší všechny dynamické proměnné, které byly vytvořeny od posledního vyvolání procedury MARK a současně se zruší efekt tohoto vyvolání procedury MARK. Jestliže před provedením procedury RELEASE nebyla provedena procedura MARK, je hlášena chyba.

Příklad:

```
procedure P;
{všechny dynamické proměnné, které se vytvoří během provádění této procedury, se mají po jejím ukončení zrušit}
...
   procedure MARK; external;
   procedure RELEASE; external;
begin
  MARK;
  {následuje vlastní výpočet}
   ...
   RELEASE
end;
```

Efekt procedur MARK a RELEASE při opakovaném vyvolání:

```
     new(P1);
     MARK;
       new(P2);
       MARK;
         new(P3); new(P4);
       RELEASE; {zruší P3↑ a P4↑}
       new(P5);
     RELEASE; {zruší P2↑ a P5↑}
     {zbývá P1↑ , další RELEASE by byl chybou}
```

8.2 Ukazatele statických proměnných

Statickou proměnnou se rozumí proměnná, která byla zavedena deklarací. Jestliže statická proměnná (resp. její složka) je typu T, pak její adresa (resp. adresa její složky) může být přiřazena proměnné typu ukazatel na T.

Adresy statických proměnných a jejich složek se získávají pomocí standardní funkce ref.

Funkce ref

Jestliže prom je proměnná typu T, pak hodnotou zápisu funkce ref ve tvaru

ref(prom)

je adresa proměnné prom. Tato hodnota se z praktických důvodů chápe (podobně jako hodnota nil) jako ukazatel kompatibilní pro přiřazení se všemi typy ukazatelů. Je dovoleno, aby parametrem funkce ref byla složka zhuštěné struktury, avšak je hlášena varovná zpráva. Toto použití funkce ref má ve Fel-Pascalu smysl proto, že každá složka i zhuštěné struktury má adresu (při zhušťování se nejde pod úroveň byte). Je-li skutečným parametrem funkce ref lokální proměnná procedury nebo funkce, je třeba počítat s tím, že po skončení procedury (funkce) přestanou její lokální proměnné existovat.

Skutečným parametrem funkce ref může být kromě proměnné také konstanta typu řetězec znaků. V tomto případě je hodnotou zápisu funkce adresa uložení řetězce v paměti.

Jestliže proměnné typu ukazatel byla přiřazena adresa statické proměnné, pak tato proměnná nesmí být skutečným parametrem procedury dispose.

Hodnota typu ukazatel může být zvětšena resp. zmenšena o celé číslo (viz 5.4.6). Nejčastěji toto číslo udává délku datové struktury určitého typu a lze jej získat pomocí standardní funkce size.

Funkce size

Zápis funkce size má tvar

size(T)

resp.

size(T, c1, c2, ..., cn)

kde T je identifikátor typu a
c1, c2, ... cn jsou ordinální konstanty.

Hodnotou zápisu size(T) je celé číslo, které udává délku vnitřní reprezentace hodnot typu T v bytech. Délka platí pro uložení hodnot typu T v nezpakované struktuře. Je-li T variantní záznam, pak se jedná o délku nejdelší varianty.

Zápis ve tvaru size(T, c1, c2, ..., cn) lze použít tehdy, je-li T variantní záznam, v němž jsou vnořeny varianty odpovídající rozlišovacím konstantám c1, c2, ..., cn. Hodnotou zápisu je pak délka vnitřní reprezentace pro specifikovanou variantu.

Poznámka: Je-li P proměnná typu ↑T, pak příkazem new(P) se vytvoří dynamická proměnná, jejíž reprezentace v paměti zaujímá size(T) bytů a příkazem new(P, c1, c2, ..., cn) se vytvoří proměnná, jejíž reprezentace v paměti zaujímá size(T, c1, c2, ..., cn) bytů.

Příklad:

```
var A : array [1..100] of real;
    I : integer;
    P, Q : ↑real;
```

Chceme-li vynulovat pole A, můžeme na místo standardního postupu

```
for I := 1 to 100 do A [I]:= 0;
```

použít efektivnější způsob:

```
begin P := ref(A[1] ); Q := ref(A[100] );
  while P<= Q do
    begin P↑ := 0; P := P + size(real) end
  end
```

Pozn.: Pro zcela speciální účely je možno provádět operaci ref na identifikátor procedury nebo funkce. Výsledkem je adresa začátku složeného příkazu z bloku této procedury nebo funkce.

9. STRUKTURA GENEROVANÉHO KÓDU

9.1 Znakový assembler

Při vytváření znakového assembleru platí následující pravidla:

a) procedury a funkce se číslují 1, 2 atd. v pořadí, v jakém jsou deklarovány (externí procedury a funkce se nepočítají). Blok hlavní jednotky programu má číslo 0.
   Vstupní body procedur a funkcí jsou označeny identifikátory ve tvaru
   F?nnnn
   kde nnnn je číslo procedury nebo funkce hexadecimálně. Vstupní bod programu je F?0000.

b) zóny lokálních proměnných statických procedur a funkcí jsou umístěny v datovém segmentu a jsou označeny identifikátory
   V?nnnn
   kde nnnn je číslo příslušné procedury nebo funkce hexadecimálně. Zona proměnných deklarovaných na základní úrovni kompilační jednotky je označena V?0000.

c) návěští deklarovaná uživatelem jsou deklarovaná ve taru
   U?kkkk
   kde kkkk je pořadové číslo návěští hexadecimálně.
   Návěští generovaná překladačem mají tvar L?kkkk; v některých generovaných sekvencích je navíc použit adresní výraz ve tvaru
   $ + d resp. $ - d

d) zóna konstant je na konci programu a je označena návěštím S?0000.

Jména podprogramů v systémových knihovnách, které uživatel nevolá přímo, začínají prefixem

P@

Je-li použito generování do znakového assembleru, nesmí se jako identifikátor externího nebo globálního objeku použít klíčové slovo assembleru (platí i pro použití specifikace alias).

Je-li při překladu uveden přepínač /MXA je do znakového assembleru zahrnut zdrojový text jako komentář. V opačném případě jsou v komentáři uváděna čísla řádků zdrojového textu, které odpovídají generovanému kodu

9.2 Struktura procedur a funkcí

9.2.1 Statické procedury a funkce

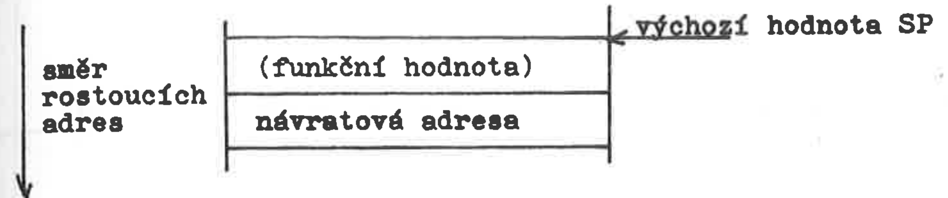
Parametry se předávají na zásobníku. Na zásobník jsou ukládány v opačném pořadí, než se zapisují do programu t.zn. zprava doleva. Nad parametry je při vstupu do procedury uložena návratová adresa. Způsob předávání jednotlivých druhů parametrů viz kap.4. Po vstupu do procedury jsou parametry nebo jejich adresy přesunuty do zóny statických proměnných. Na zásobníku zůstane pouze návratová adresa, která je přesunuta na místo prvního uloženého parametru. U funkcí je nad návratovou adresou vymezeno místo na funkční hodnotu. Na zásobníku je každý parametr uložen na sudém počtu bytů. Při přesunu do zóny statických proměnných se uloží pouze na potřebném počtu bytů.

Směřuje-li do statické procedury (resp. funkce) skok z vnořené procedury (resp. funkce), je při vstupu do procedury hodnota registru SP uložena v zóně lokálních proměnných.

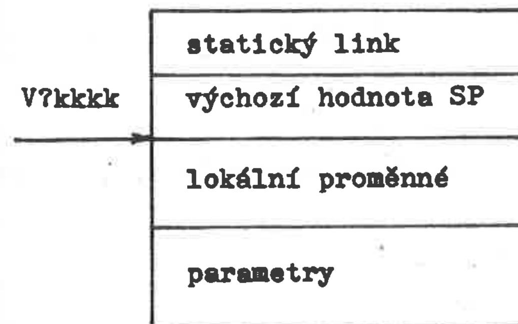
Je-li statická procedura vnořena do dynamické procedury (resp. funkce) je do zóny lokálních proměnných uložena také hodnota tzv. statického linku, který je proceduře předán v registru DE. Tento link slouží

k adresaci nelokálních objektů (parametrů a proměnných).

Během činnosti procedury (resp. funkce) se na zásobník ukládají adresy získané výpočtem adresního výrazu v příkazu with, mezivýsledky při výpočtech výrazů a parametry volaných procedur a funkcí.



Obr.: Struktura zásobníku po vstupu do statické procedury nebo funkce



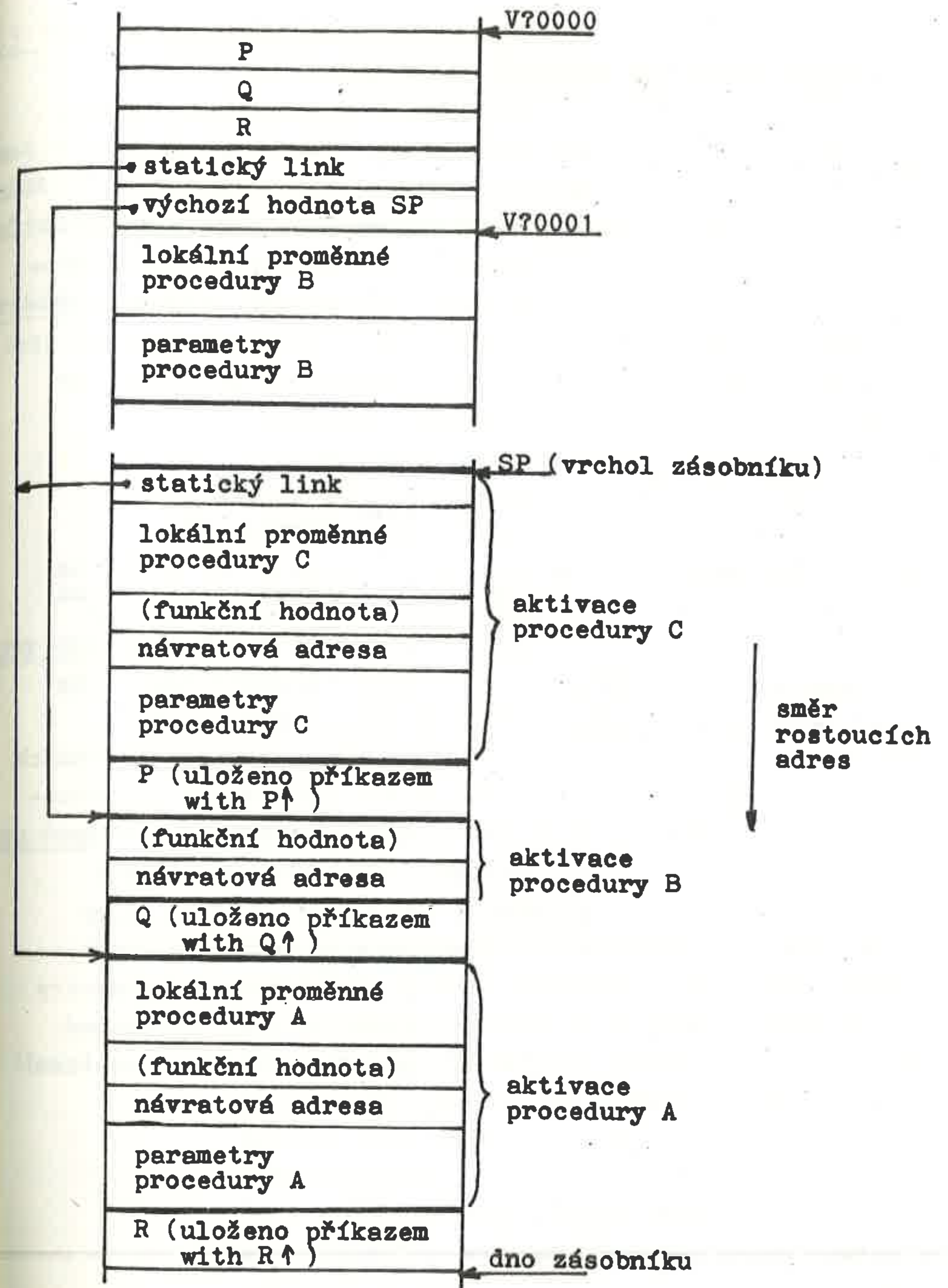
Obr.: Zóna lokálních proměnných statické procedury nebo funkce číslo kkkk

9.2.2 Dynamické procedury nebo funkce

Parametry se předávají stejným způsobem jako u statických procedur, zůstávají však na zásobníku po celou dobu aktivace procedury (resp. funkce). Nad návratovou adresou je vymezeno místo na lokální proměnné a (resp.)na funkční hodnotu. Nad případnou funkční hodnotou je uložen statický link. Práce se zásobníkem a předávání funkční hodnoty jsou stejné jako u procedur statických

Příklad:

```
program X;
   type
      REC = record ... end;
   var
      P, Q, R : ↑REC;
   dynamic procedure A(...);
      var
         ...
      static procedure B(...);
         label 9;
         var
            ...
         dynamic procedure C(...);
            var
               ...
            begin
               {stav zachycený následujícím obrázkem}
               goto 9
            end; {procedury C}
         begin
            with P↑ do C(...);
         end; {procedury B}
      begin
         with Q↑ do B(...)
      end; {procedury A}
   begin
      with R↑ do A(...)
   end .
```

Obr.: Vazba dynamických a statických procedur nebo funkcí (viz předcházející příklad).

9.2.3 Systémové procedury a funkce

Klasifikace **system** má dvojí význam. U externí procedury (resp.funkce) znamená, že tělo deklarované procedury (resp. funkce) se nachází v kompilační jednotce napsané v jazyku PL/M (nebo konvence volání podprogramů jsou shodné s jazykem PL/M). Takto deklarované proceduře se paramtry předávají podle těchto konvencí:

jeden parametr	- v reg. C resp. BC
dva parametry	- první v reg. E resp. DE, druhý v reg. C resp. BC
více parametrů	- prvé dva v registrech, další na zásobníku, přičemž třetí je pod návratovou adresou a případné další v pořadí pod ním

Hodnotové parametry mohou být pouze takového typu, aby jejich hodnota nebyla delší než dva byty.

U lokálních procedur (resp. funkcí) klasifikovaných jako **system** se negeneruje žádný kód při vstupu do procedury (resp. funkce) ani při opouštění procedury (resp. funkce). To znamená, že skutečné parametry předávané proceduře(funkci) zůstávají při vstupu do procedury (funkce) na zásobníku. V proceduře je možno používat pouze globální proměnné, nelokální proměnné a parametry statických procedur a lokální proměnné této systémové procedury (funkce). Skutečné parametry je třeba odstranit ze zásobníku pomocí inline assembleru.

9.3 Vnitřní reprezenzace proměnných

Vnitřní reprezentace proměnných je způsob uložení hodnot v paměti. Pro každý typ je jednoznačně určen způsob tohot uložení. Základem vnitřní reprezentace je zobrazení celých čísel a to buď v přímém nebo doplňkovém kódu.

ordinální typy - hodnoty jsou reprezentovány svými ordinálními čísly. Jestliže ordinální čísla hodnot daného typu leží v intervalu 0..255, jsou hodnoty reprezentovány na jednom bytu v přímém kodu. Jestliže alespoň jedno ordinální číslo leží mimo interval 0..255, jsou hodnoty reprezentovány na jednom slově v doplňkovém kódu. Na prvním bytu je uložena část low na druhém část high (viz 5.5.2.3).

typy ukazatel - hodnoty těchto typů jsou reprezentovány čísly z intervalu 0..65535, které jsou adresami příslušných referencovaných proměnných. Adresy jsou uloženy na slově v přímém kódu.

typ real - hodnota typu real je zobrazena na třech bytech ve tvaru

$$m * 2^{e-79}$$

kde e je exponent, jehož hodnota leží v intervalu 0..127 a je zobrazen na prvním bytu reprezentace hodnoty real (8. bit prvního bytu je vždy nulový). m je normalizovaná mantisa, která je uložena v doplňkovém kodu na dalších dvou bytech. Hodnota mantisy leží v intervalech -32768..-16383 a 16384..32767 t.j. polouzavřených intervalech $-2^{15}..-2^{14}$ a $2^{14}..2^{15}$. Nejmenší zobrazitelná absolutní hodnota je tedy

$$2^{14} * 2^{0-79} = 2^{-65} = 2.7105 * 10^{-20}$$

a největší

$$2^{15} * 2^{127-79} = 2^{63} = 9.2234 * 10^{18}$$

Největší relativní chyba je 2^{-15} t.j. $3.052 * 10^{-5}$

<u>typ množina</u> - hodnota je vždy reprezentována hodnotou příslušného kanonického typu množina (viz 4.3.3.4). Je-li n počet prvků kanonického bázového typu, pak délka typu množina je (n-1) <u>div</u> 8 + 1. Je-li bázový typ interval z typu integer, pak n je rovno přepínači překladu SET zvětšenému o 1 (implicitně/SET:255.-viz kap.12). Přítomnost prvku v množině je reprezentována nastavením příslušného bitu na hodnotu 1. Relativní adresa bytu v množině, ve kterém se nachází bit reprezentující přítomnost prvku m je dána vzorcem

d = ord(m) <u>div</u> 8

Číslo bitu v tomto bytu je dána vzorcem

c = ord(m) <u>mod</u> 8

Příklad:

```
type
TB = (B0,B1,B2,B3,B4,B5,B6,B7,B8,B9,B10,B11,
     B12,B13,B14,B15);
var
  B : TB;
  S : set of TB;
  SUM, PWR : integer;
```

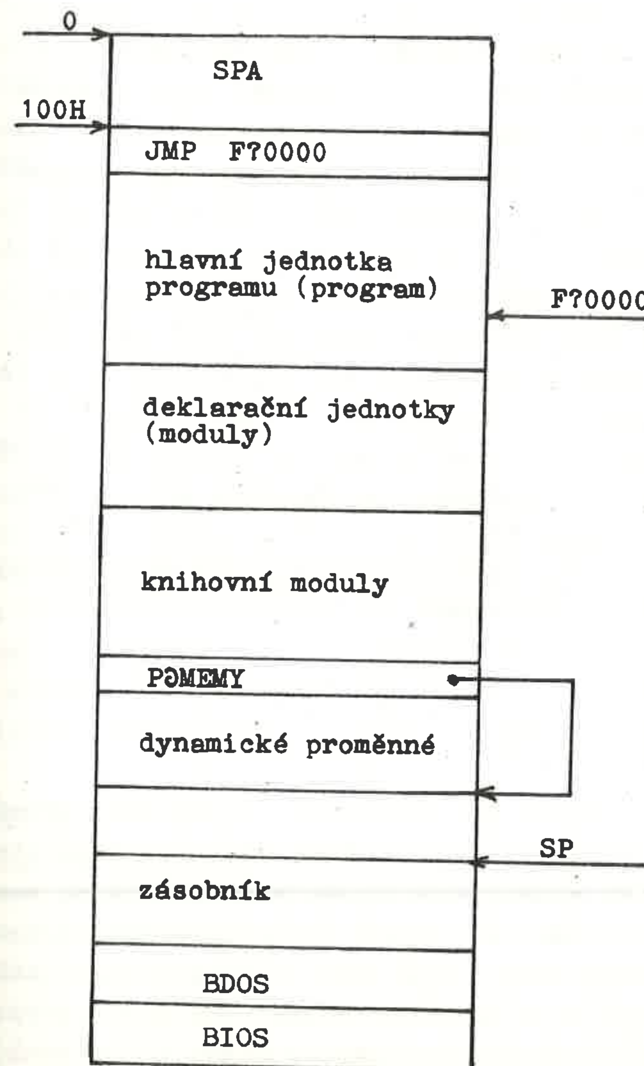
Dva následující příkazy přiřadí proměnné SUM stejnou hodnotu:

```
a) SUM := S type integer
b) begin { příkaz nesmí být přeložen s kontrolou
           přeplnění }
     SUM := 0; PWR := 1;
     for B := B0 to B15 do
       begin
         if B in S then  SUM := SUM+PWR;
         PWR := PWR*2
       end
     end
```


10. <u>Sestavování přeloženého programu /pod CP/M /</u>.

Překladem zdrojového textu vznikají moduly znakového asembleru /ty je třeba dále přeložit překladačem M80 / nebo rovnou relativní moduly.Jednotlivé relativní moduly se potom sestavují společně s knihovnami sestavovacím programem L80.

10.1 <u>Nesegmentované programy</u>.



Obr.: Uložení nesegmentovaného programu v paměti

Sestavený program se skládá z hlavní programové jednotky (programu), deklaračních jednotek (modulů) a knihovních modulů. Program je sestaven tak, aby začínal na adrese 100H. Na tuto adresu doplní sestavovací program L80 instrukci skoku na vlastní vstupní bod programu F?0000 (t.j. na začátek složeného příkazu na základní úrovni hlavní programové jednotky). Před vlastním prováděním tohoto složeného příkazu se systém inicializuje voláním procedur z knihovního modulu PSCI z knihovny PFCLIB. Provádí se jednak inicializace souborů, jednak nastavení hodnoty registru SP, aby ukazoval na dno zásobníku (do SP se přiřadí obsah adres 6 a 7, t.j. adresní část instrukce JMP BDOS). Používá-li program dynamickou paměť (t.j. používá-li proceduru new), musí být součástí programu proměnná P@MEMY, která obsahuje adresu paměti pro vytváření dynamických proměnných. Proměnná se nachází v knihovním modulu YQMEMY v knihovně PFLIB. Program se musí sestavit tak, aby tato proměnná byla umístěna zcela na konci programu (modul YQMEMY se nachází na konci knihovny PFLIB a ta se musí sestavovat jako poslední) a sestavovací program ji inicializuje adresou o dva větší než je adresa, na které je uložena. Obsah této proměnné je využíván procedurami new a RELEASE (viz kap. 8).

Z obrázku vyplývá, že zásobník a dynamická paměť jdou při běhu programu "proti sobě".

10.2 Segmentované programy

Sestavovací program L80 neumožňuje standardně provádět segmentaci, avšak uživatel si může vytvořit segmentovaný program pomocí jednoduchých prostředků. Dále je popsán jeden z možných způsobů, který byl použit pro segmentaci vlastního překladače a byl dán k dispozici uživatelům PF80. Uvedená segmentace umožňuje pouze standardní tvar hrábí (t.j. jeden rezidentní segment a několik segmentů

navzájem se překrývajících). Data společná různým segmentům mohou být uložena pouze v rezidentní části nikoliv v v dynamické paměti. Dynamické proměnné jsou přemazány zavlečením delšího segmentu a navíc je ztracena hodnota proměnné P@MEMY.

Rezidentní segment

Rezidentní segment se chová obdobně jako nesegmentovaný program. Vznikne sestavenám hlavní programové jednotky a případně deklaračních jednotek a knihovních modulů. Musí obsahovat tyto deklarace:

```
type
  TNAME = packed array [1..14] of char;
   .
   .
   .
procedure LDSGM(const NAME : TNAME); external;
```

Deklarace procedury LDSGM v rezidentním segmentu způsobí, že na konec rezidentního segmentu se připojí z knihovny PFLIB (musí být přisestavena jako poslední) knihovní modul YQOVRL obsahující pouze globální symbol P@OVRL. Tento symbol pak představuje adresu, na kterou se zavlékají jednotlivé překrývající se segmenty. Parametrem procedury LDSGM je specifikace souboru, ve kterém je uložen zavlékaný segment. Procedura zavleče segment do paměti na adresu P@OVRL a zavolá proceduru bez parametrů, která musí mít na této adrese vstupní bod.

Překryvné segmenty

Každý překryvný segment se skládá z jedné nebo více deklaračních jednotek a knihovních modulů. Vstupní bod každého segmentu musí ležet na začátku tohoto segmentu (neboť tam předá řízení procedura LDSGM). To znamená, že první deklarační jednotka segmentu nesmí mít žádné proměnné na základní úrovni a první deklarovaná procedura musí tvořit

vstupní bod segmentu.

Příklad:

```
modul SEGENT;
  procedure SEGACT; external;

  procedure SEGENT;
    begin
      SEGACT   {volání vlastní výkonné procedury segmentu}
    end;
end .
```

Aby byl překryvný segment sestaven na adrese, na kterou bude zavlékán procedurou LDSGM, a aby byly správně vyřešeny vazby mezi rezidentním a překryvným segmentem, musí sestavení překryvného segmentu začít kompletním sestavením rezidentního segmentu včetně knihoven. Poté je třeba připojovat jednotlivé deklarační jednotky překryvného segmentu a sestavení zakončit opětným přisestavením knihoven. Nyní je třeba ze sestaveného programu "odříznout" rezidentní část. Sestavený segment se uloží na disk a znovu se programem DDT načte do paměti. Příkazem:

M adr1,adr2,100

se vlastní segment přesune na adresu 100H.(adr1 je hodnota symbolu P@OVRL ve výpisu mapy sestavení a adr2 je hodnota NEXT vypsaná programem DDT při načtení segmentu). Stiskne se ctrl/C a po zavedení systému se příkazem SAVE uloží segment na disk. Délka uloženého segmentu je délka původně sestaveného segmentu minus délka rezidentní části. Specifikace souboru, do kterého je překryvný segment uložen, je potom paramterem procedury LDSGM.

Činnost takto segmentovaného programu je následující: Při vyvolání příkazem pro CCP je do paměti zavedena rezidentní část a je jí předáno řízení. Vyvoláním procedury LDSGM je z disku načten určený soubor, je uložen bezprostředně za rezidentní segment a je předáno řízení na jeho začátek. Segment může využívat globální proměnné a procedury (resp. funkce) rezidentního segmentu. Skončí-li svou činnost normálně, vrací řízení do procedury LDSGM. Nenormální ukončení může být realizováno např. vyvoláním globální procedury rezidentního segmentu, která je potom opuštěna skokem do hlavního programu.

10.3 Knihovny

10.3.1 Obecně

Pozn.: Vše co je dále uvedeno pro procedury platí stejně i pro funkce.

Z hlediska uživatele se všechny procedury dělí na standardní a nestandardní. Standardní procedury může uživatel používat, aniž by je musel deklarovat. Jejich činnost a způsob předávání parametrů je určen pro každou proceduru zvlášť (viz 5.5.2 a 6.2.2.2).

Ostatní procedury musí být vždy deklarovány jako lokální v dané kompilační jednotce nebo jako externí. Těmto procedurám se předávají skutečné parametry pascalským způsobem nebo konvencí PL/M. Ke každé proceduře, která je deklarovaná jako externí, musí existovat kompilační jednotka, ve které je tato procedura globální nebo musí být tato procedura uložena na knihovně.

Volání standardní procedury může překladač interpretovat následujícími způsoby:

a) volání standardní procedury slouží pouze pro upřesnění sémantiky programu a negeneruje se žádný kód (chr, ord, ref, "identifikátor výčtového typu").

b) požadovaná akce se generuje přímo do kódu, nevolá se tedy žádný podprogram (size, succ, pred, inline, pack, unpack, high, low, odd).

c) volání standardní procedury se přeloží na volání jednoho nebo více knihovních podprogramů. Přitom se může modifikovat seznam skutečných parametrů (eof, eoln, page, read, write, readln, writeln, new, dispose).

d) volání standardní procedury se přeloží tak, jakoby standardní procedura byla nedeklarována jako externí, avšak s modifikovaným jménem (obvykle je před jméno předřazen prefix P@ (get, put, reset, rewrite, sin, ... arctan, trunc, round).

Knihovna podprogramů obsahuje tedy tyto druhy podprogramů:

1) Podprogramy volané překladačem implicitně pro akce, které se nevyplatí generovat (příkazy case a for, některé aritmetické operace atd.).

2) Podprogramy, jejichž volání je generováno při volání standardních procedur (viz body c) a d) výše).

3) Podprogramy, které rozšiřují možnosti systému, které nepatří standardně do jazyka Fel-Pascal (ASSIGN, MARK, LDSGM atd.).

Jména podprogramů z bodu 2) začínají vždy prefixem P@. Podprogramy z bodů 2) a 3) mají způsob předávání parametrů pascalský a nebo konvencí PL/M.

Pro uživatele z toho vyplývá:

- může přímo využívat všech knihovních podprogramů z bodů 2) a 3), jestliže je nadeklaruje jako externí. Ty, které mají způsob předávání parametrů konvencí PL/M klasifikuje jako system. Podprogram jehož jméno není identifikátor, musí přejmenovat specifikací alias.

Příklad:

```
    program X;
```



```
    alias  QNEW = 'P@NEW';
  type
    TPCH = ↑char;
  var
    P : TPCH;
  procedure QNEW(var Q : TPCH; L : integer);
  external;
  begin
    .
    .
    new(P);  {generuje se P@NEW(P, size(char))}
    .
    QNEW(P, 5);{generuje  se P@NEW(P, 5)}
    .
    .
  end .
```

- uživatel si může sám vytvořit procedury, které se obvykle připojují z knihovny, a dát jim vlastní sémantiku.

Příklad:

```
  modul HEAP;
  alias  QNEW = 'P@NEW';
  global  QNEW;
  type
    TPCH =  char;
  procedure QNEW(var P : TPCH; L : integer);
    begin
    .
    .
    end;
  end .
```

10.3.2 Knihovna PFRLIB

Knihovna PFRLIB je knihovna aritmetických podprogramů. Obsahuje:

- podprogramy reálné aritmetiky
- podprogramy řešící standardní reálné funkce

  <u>function</u> P@SIN(R : real) : real;
  •
  •
  •
  <u>function</u> P@A : real

- podprogramy dalších reálných funkcí

  <u>function</u> EXP10(R : real) : real; 10^R
  <u>function</u> LOG (R : real) : real; $LOG_{10}(R)$
  <u>function</u> LN2 (R : real) : real; LN_2

- podprogramy aritmetiky s typem FRACTION

Typ FRACTION je název zobrazení hodnot v pevné řádové čárce v doplňkové kódu na 16-ti bitech. Rozsah zobrazení je od -1.0 (8000H) do $1.0-2^{15}$ (7FFFH). Hodnota typu FRACTION je 2^{15} krát menší, než stejně zobrazená hodnota typu integer. Takto lze většinou interpretovat hodnoty snímané z ADC převodníků, nebo zapisované na DAC převodníky.

1 0
2^{-14} 2^{-15}

Obr.: Typ FRACTION

Protože typ FRACTION není standardní, je třeba ho deklarovat např.:

<u>type</u> FRACTION = integer;

Takovéto zobrazení typu FRACTION má výhodu, že se s proměnnými tohoto typu dají provádět aditivní operace (+, -) a relační operace (=, , ...), které jsou pro typ FRACTION stejné jako pro typ integer. Pro ostatní operace jsou v knihovně PFRLIB následující podprogramy:

<u>function</u> FRACF3(X : real) : FRACTION;

Převod hodnoty typu real na hodnotu typu FRACTION. Dojde-li k přetečení, je vyvolána procedura R@ROER (viz knihovna PFCLIB).

<u>function</u> F3FRAC(X : real) : real;

Převod hodnoty typu real na hodnotu typu FRACTION.

<u>system</u> <u>function</u> FRCADD(X, Y : FRACTION) : FRACTION;

Sečtení dvou hodnot typu FRACTION. Dojde-li k přetečení, je výsledkem maximální přípustná hodnota příslušného znaménka.

<u>system</u> <u>function</u> FRCNEG(X : FRACTION) : FRACTION;

Změna znaménka hodnoty typu FRACTION. Hodnota -1.0 (8000H) je převedena na hodnotu $1.0-2^{-15}$ (7FFFH).

<u>system</u> <u>function</u> FRCMUL(X, Y : FRACTION) : FRACTION;

Násobení dvou hodnot typu FRACTION. Násobení dvou hodnot -1.0 dá výsledek $1.0-2^{-15}$.

<u>system</u> <u>function</u> FRCPWR(X : FRACTION; Y : integer) : FRACTION;

Výpočet výrazu $X * 2^Y$. Dojde-li k přetečení, je výsledkem maximální hodnota příslušného znaménka.

10.3.3 Knihovna PFCLIB

Knihovna PFCLIB obsahuje systémově závislé podprogramy, a proto nemůže být přisestavena k aplikačním programům (viz 10.4). Kromě jiného obsahuje knihovna podprogramy pro základní práci se soubory, podprogramy vstupních a výstupních konverzí, podprogramy pro ovládání dynamické paměti a podprogramy hlášení chyb zjištěných během výpočtu.

Předpokládejme následující deklarace:

```
type  TP1 = 0..255;
var I : integer; C : char; R : real; FT : text;
```

Dále předpokládejme, že e je výraz typu real, k, m, n jsou výrazy typu integer, ch je výraz typu char a s je řetěz délky ls. Dále předpokládejma, že TFL je libovolný typ soubor, TPP je libovolný typ ukazatel a p je výraz typu TPP.

- podprogramy pro základní práci se soubory

```
function P@EOF (var F : TF  ) : boolean;
function P@EOLN(var F : text) : boolean;
procedure P@RST (var F : TF);
procedure P@RWRT(var F : TF);
procedure P@PUT (var F : TF);
procedure P@GET (var F : TF);
procedure P@WRLN(var F : text);
```

- podprogramy vstupních a výstupních konverzí (význam parametrů a způsob volání je vyjádřen pomocí příslušné standardní funkce)

```
function P@RDI(var F : text) : integer;
read(FT,I) .... I := P@RDI(FL)
```



```
function P@RDC(var F : text) : char;
read(FT, C) .... C := P@RDC(FT);

function P@RDR(var F : text) : real;
read(FT, R) .... R := P@RDR(FT);

procedure P@WRI1(I : integer; L : TP1; var F : text);
write(FT, k:m) .... P@WRI1(k, m, FT)

procedure P@WRH0(P : TPP; var F : text);
write(FT, p) .... P@WRH0(p, FT)

procedure P@WRH1(P : TPP; L : TP1, var F : text);
write(FT, p:m) .... P@WRH1(p, m, FT)

procedure P@WRC0(C : char; var F : text);
write(FT, ch) .... P@WRC0(ch, FT)

procedure P@WRC1(C : char; L : TP1; var F : text);
write(FT, ch:m) .... P@WRC1(ch, m, FT)

procedure P@WRS0(P : TPP; L : TP1; var F : text);
write(FT,s) ..... P@WRS0(ref(s), ls, FT)

procedure P@WRS1(P : TPP; LS, L : TP1; var F : text);
write(FT, s:m) .... P@WRS1(ref(s), ls, m, FT)

procedure P@WRR0(R : real; var F : text)
write(FT, e) .... P@WRR0(e, FT)

procedure P@WRR1(R : real; L : TP1; var F : text);
write(FT, e:m) .... P@WRR1(e, m, FT)

procedure P@WRR2(R : real; L1, L2 : TP1; var F : text);
write(FT, e:m:n) .... P@WRR2(e, m, n, FT)
```

- podprogramy pro ovládání dynamické paměti

```
procedure P@NEW (var P : TPP; L : integer);
procedure P@DISP(P : TPP; L : integer);
procedure MARK;
procedure RELEAS;
```

- podprogramy pro hlášení chyb zjištěných v průběhu výpočtu (tyto podprogramy zjistí na zás níku návratovou adresu do uživatelova programu na které došlo k chybě, vypíší ji s příslušnou hláškou na output, a ukončí výpočet)

procedure P@ROER; - chyba v reálné operaci
procedure P@LNER; - chybný argument funkce ln
procedure P@EXER; - chybný argument funkce exp
procedure P@CSER; - hodnotě výrazu v příkazu case neodpovídá žádné rozlišovací konstantě
procedure P@RNER; - hodnota výrazu je mimo rozsah příslušného ordinálního nebo množinového typu
procedure P@DRER; - dereference hodnoty nil
procedure P@RDER; - chyba ve vstupní konverzi (je-li vstupní soubor soubor input, pracuje procedura způsobem popsaným v 7.2.3.3)

10.3.4 Knihovna PFLIB

Knihovna PFLIB obsahuje systémově nezávislé podprogramy volané z vygenerovaného kódu (např. podprogramy aritmetických operací na 16 bitech, podprogramy volané na začátku a na konci příkazu for atd.).

Na konci knihovny PFLIB jsou moduly, které svým umístěním určují konec rezidentního segmentu (počáteční adresu pro zavádění překryvných segmentů) resp. začátek dynamické paměti (při volání procedury new).

10.4 Programy pracující bez operačního systému (aplikační programy)

Při tvorbě aplikačního programu je třeba zajistit, aby program nepoužíval služby operačního systému. Normální program používá tyto služby v těchto případech:

a) pro práci se soubory

b) pro spuštění programu a pro inicializaci zásobníku

c) pro hlášení chyb vzniklých během výpočtu

d) po skončení programu k návratu do operačního systému

Aplikační program nesmí obsahovat proměnné typu soubor. Může být sestaven z jedné nebo z více kompilačních jednotek a může, avšak nemusí obsahovat hlavní jednotku programu (program). Pokud aplikační program obsahuje hlavní jednotku programu, musí uživatel zajistit jeho spuštění skokem na návěští F?0000. Na tomto návěští program vždy volá knihovní proceduru P@PSCI z knihovního modulu PSCI. Tato procedura provádí inicializaci souborů a zásobníku. Protože však dosud není nastaven registr SP, volá se tato procedura skokem a návratová adresa se ji předává v registru HL. Tuto proceduru musí uživatel napsat sám např. podle dále uvedeného vzoru.

Jestliže má aplikační program skončit, t.zn. dojít až na konec složeného příkazu v hlavní jednotce programu, musí být na dno zásobníku uložena návratová adresa, protože program končí instrukcí RET.

```
        NAME  ('PSCI')
        ENTRY  P@PSCI
        EXTRN  F?0000
; tento modul bude uložen od adresy 0 a bude prováděn
; po odeznění signálu reset
```

```
        JMP  F?0000       ; skok do programu
;
P@PSCI: LXI  SP,STACK     ; nastaveví SP
        CALL  IPCHL       ; uložení návratové adresy na
                          ; zásobník a skok na adresu z HL
        DI                ; konec aplikačního programu
        HLT
IPCHL:  PCHL
        END
```

Jestliže v aplikačním programu není použita hlavní jednotka programu, stačí ve vstupní sekci nastavit registr SP a zavolat obdobným způsobem příslušnou proceduru, která realizuje činnost programu.

Při sestavení aplikačního programu nesmí být použita knihovna PFCLIB, ve které jsou shromážděny sytémově závislé moduly. Po sestavení aplikačního prgramu bez této knihovny by měly zůstat nevyřešené pouze reference P@PSCI a reference procedur pro hlášení chyb vzniklých v průběhu výpočtu. Pokud si je uživatel jist, že k těmto chybám nemůže dojít, může tyto reference nechat nevyřešené. V opačném případě musí procedury vhodným způsobem doplnit.

11. OPTIMALIZACE

Optimalizační fáze tvoří čtvrtou fázi překladače. Provádí úpravy na vnitřní formě jazyka. Kromě toho, že usnadňuje práci generátoru kódu některými formálními úpravami v datové struktuře, provádí ve vnitřní formě určité změny, které umožní vygenerovat efektivnější kód. Protože řadu těchto úprav může uživatel ovlivnit způsobem zápisu příkazu, je užitečná alespoň jejich orientační znalost. V následujícím textu budou optimalizace, na které má uživatel vliv, popsány a vysvětleny na příkladech, které předpokládají tyto nadefinované typy proměnných:

```
type TREC1 = record
                R1 : real;
                I1 : integer;
             end;
     TREC2 = record
                REC1 : TREC1;
                I2 : integer;
             end;

var REC2 : TREC2;
    I : integer;
    POLE : array [1..10] of integer;
    M : set of char;
    R : real;
    B : boolean;
```

Poznámka: Všechny níže popsané úpravy se provádějí i ve vzájemných kombinacích, tj. např. vyhodnocením standardní funkce s konstantním argumentem vznikne opět konstanta, které se může použít např. při zjednodušení indexace.

Záměna pořadí operandů

Optimalizace provádí vhodnou záměnu pořadí operandů v relacích a ve všech komutativních operacích. Tato záměna je taková, aby 2. operand byl konstantní anebo alespoň jednodušší (jednoduchá proměnná). Protože se touto úpravou

změní pořadí vyhodnocování operandů, může být jejím důsledkem ovlivnění vedlejšího efektu funkce, která by se vyskytovala ve složitějším výrazu.

Částečný výpočet adresy

Další úprava je částečný výpočet doplňku adresy v adresních výrazech. Výsledným efektem je, že přístup k proměnné REC2.REC1.I1 je stejně rychlý, jako např. přístup k proměnné I. Tuto úpravu ovšem nelze provést nebo ji lze provést jen částečně, vyskytuje-li se v tomto zápisu adresní dereference.

Provedení aritmetických operací s konstantami

Optimalizační fáze vyhodnocuje všechny operace s konstantními operandy včetně množinových. Znamená to, že již v době překladu se vypočítávají výrazy, jejichž strom má takovou vlastnost, že z uzlu, který je tvořen operátorem vycházejí 2 větve zakončené konstantními operandy. Z tohoto hlediska je tedy nevhodný zápis I := I + size(TREC1) +1, neboť strom tohoto výrazu nemá požadovanou vlastnost. Naproti tomu zápis I := I + (size(TREC1)+1) nebo
I := size(TREC1)+1+I částečné vyhodnocení výrazu umožní.

Vyhodnocení standardních funkcí s konstantami

Během překladu se také vyhodnocují všechny standardní funkce s konstantním argumentem, jejichž výsledek není typu real. Vyhodnotí se tedy např. výraz odd(trunc(2.5)), ale nevyhodnotí se výraz sqrt(8).

Zjednodušení vyhodnocení indexových výrazů

Napíšeme-li v programu indexový výraz s konstantním indexem, je možno relativní adresu příslušného prvku pole vypočítat již během překladu. Důsledkem je stejná rychlost přístupu k proměnné I a např. POLE[7].

Vyskytne-li se při zápisu indexového výrazu v indexu výraz, jehož strom má tu vlastnost, že kořen je tvořen operátorem+ nebo- a napravo je konstantní operand, potom se adresa pole zmodifikuje takovým způsobem, že tato aritmetická operace zcela odpadne. V praxi to znamená, že přístup k proměnné POLE[I+3] nebo POLE [I-3] je stejně rychlý jako přístup k proměnné POLE[I] . Z tohoto hlediska je tedy doporučený zápis POLE [-I+5] na rozdíl od zápisu POLE [5-I] . Tato úprava se ovšem neprovádí, je-li program překládán s parametrem R+, neboť výhody, které z ní vyplývají zcela zanikají.

Vyhodnocení relací s konstantními operandy

Stejným způsobem jako aritmetické operace s konstantami se vyhodnocují i všechny typy relací včetně relací mezi množinami a znakovými řetězci. Jsou tedy naprosto totožné zápisy B := ´ABC´ = ´ABC´ a B := true.

Zjednodušení přiřazení

K nejefektivnějším úpravám patří rozpoznávání výrazů typu A:=A± kladná konstanta, kde A jsou stejné, libovolně složité přístupy k jednobytové proměnné (i množinové), neobsahující volání funkce. Přístup tohoto typu je přeložen tak, že se adresní výraz vyhodnocuje pouze jednou a navíc odpadnou některé další manipulace. Výsledkem je výrazné zkrácení vygenerovaného kódu i času výpočtu.

Vyhodnocení výrazu typu 0 * výraz

Výraz tohoto typu je vypuštěn a nahrazen nulou a adresní výraz se vůbec nevyhodnocuje. Z toho vyplývá, že jsou potlačeny vedlejší efekty funkcí, které by byly v tomto výrazu volány. Z tohoto důvodu je tato skutečnost hlášena varovnou zprávou.

Zjednodušení strukturovaných příkazů

Tato optimalizace odstraní nebo zjednoduší podmíněný příkaz nebo příkaz cyklu je-li řízen konstantou takové hodnoty, že je známo, jak se bude strukturovaný příkaz zpracovávat.

a) příkaz if
   je-li podmínka true resp. false příkaz se odstraní a nahradí se příkazem v části resp.

b) příkaz case
   Je-li index příkazu ordinální konstanta, nahradí se celý příkaz příkazem za příslušnou rozlišovací konstantou nebo se ohlásí chyba, že index neodpovídá žádné rozlišovací konstantě.

   Např. **case** 1 **of** je nahrazen příkazem:

```
case 1 of                je nahrazen příkazem:
   1 : I := 1;                I := 1
   2 : I := 2;
   3 : I := 3
end;
```

c) příkaz for
   jsou-li meze příkazu konstantní a takové, že by se příkaz neprovedl ani jednou, celý příkaz se vypustí. Jsou-li meze konstantní a stejné, nahradí se příkaz příkazem těla cyklu.

d) příkaz repeat
   je-li podmínka true resp. false, zpracuje se příkaz takovým způsobem, že se podmínka již v době výpočtu nevyhodnocuje.

e) příkaz while
   je-li podmínka false resp. true, celý příkaz se vypustí, resp. se zpracuje obdobným způsobem jako příkaz s konstantou false (tj. nekonečný cykl).

12. VOLÁNÍ PŘEKLADAČE A ŘÍZENÍ PRŮBĚHU PŘEKLADU

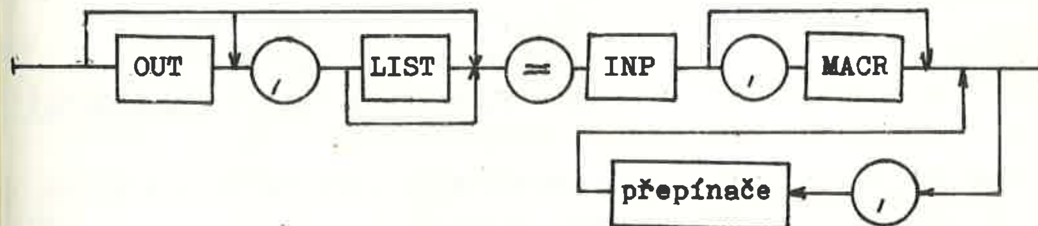
12.1 Volání překladače pod operačním systémem CP/M

Před spuštěním překladače je třeba disk, který obsahuje segmenty překladače, stanovit jako aktuální (segmenty lze zavlékat pouze z aktuálního disku). Potom lze napsat příkazovou řádku v jednom ze dvou možných tvarů:

a) PF80 parametry - překlad jedné kompilační jednotky

b) PF80 - překladač se ohlásí znakem '*' na začátku nové řádky a uživatel může napsat parametry pro překlad jedné kompilační jednotky. Po skončení překladu se překladač ohlásí znovu a je možno psát další parametry nebo skončit zavlečením systému pomocí ctrl/C.

Parametry překladu jsou tvořeny specifikacemi souborů a přepínači:



OUT - specifikace výstupního souboru
LIST - specifikace souboru protokolu o překladu
INP - specifikace vstupního souboru
MACR - specifikace vstupního souboru maker

Implicitní typ souboru INP je PSC. Implicitní typ souboru OUT je REL; pokud je použit přepínač /MAC je typu .MAC. Oba soubory musí být diskové. Soubor LIST, pokud je diskový, má implicitní typ PRN. Je-li při jeho specifikaci v příkazové řádce

použito systémové zařízení LST: nebo CON:, bude protokol vystupovat na patřičném zařízení dané aktuálním stavem IOBYTE (tiskárna, konzole).

Vynechání specifikací souborů OUT, LIST nebo OUT i LIST způsobí, že během překladu nedojde k jejich vytváření. Začínají-li parametry znakem ´=´, znamená to totéž, jako kdybychom pro specifikaci výstupního souboru a souboru protokolu použili diskové soubory stejného jména jako u vstupního souboru a implicitních typů.

Soubor MACR je implicitně typu MCR a jeho uvedení v příkazové řádce způsobí expanzi maker z tohoto souboru do zdrojového textu.

Příklady:

PF80␣=T	- překladem zdrojového souboru T.PSC vznikne výstupní soubor T.REL a soubor T.PRN protokolu o překladu.
PF80 X,=T,M/MAC	- překladem zdrojového souboru T.PSC, do kterého byly expandovány makra ze souboru M.MCR, vznikne výstupní soubor X.MAC (překlad do assembleru způsobený přepínačem /MAC). Soubor protokolu není specifikován a proto bude potlačen.

12.2 Použití souboru maker

Uvedení souboru maker v příkazové řádce způsobí expanzi maker z tohoto souboru do zdrojového textu. K expanzi dojde v okamžiku, kdy je v první pozici zdrojového textu nalezen znak ´$´. Identifikátor následující za tímto znakem je identifikátor makra, které se má expandovat.

Makro v souboru maker je tvořeno jednou nebo více řádky textu a je zakončeno řádkou, která obsahuje v první pozici znak ´.´. Tato řádka se již do makra nepočítá. První řádka makra musí obsahovat identifikátor makra (nemusí být od první pozice). Je-li tento identifikátor uveden znakem $, pak se tato první řádka nezahrnuje do rozvoje makra.

Příklady použití maker:

	zdrojový text	soubor maker	výsledný zdrojový text
a)	$XYZ	$XYZ type XYZ=(X,Y,Z); .	type XYZ=(X,Y,Z);
b)	$RST	RST=(R,S,T); .	RST=(R,S,T);

12.3 Řízení překladu

Překladač obsahuje řadu proměnných, které slouží uživateli pro řízení překladu. Proměnné jsou buď celočíselného nebo boolovského typu. Na začátku překladu každé kompilační jednotky jsou těmto proměnným přiřazeny implicitní hodnoty. Uživatel má možnost změnit tyto hodnoty buď pomocí parametrů příkazové řádky pro celou kompilační jednotku nebo pomocí $-poznámek pro části zdrojového textu. Proměnné MXA, RNG (viz dále) a proměnná ovládající tisk protokolu mají zvláštní postavení. Jejich hodnoty mohou být ovládány pomocí $-poznámek pouze v případě, že byla parametrem příkazové řádky nastavena jejich hodnota na true.

12.3.1 Přepínače

Přepínače jsou konstrukce používané v příkazové řádce pro změnu implicitních hodnot proměnných, které řídí překlad. U číselných přepínačů se uvádí dekadická hodnota.

Logické přepínače:

/STN - překladač zpracuje program ve standardním Pascalu; implicitně překladač akceptuje rozšíření Fel-Pascalu.

/RNG - budou se generovat a v době běhu provádět kontroly rozsahů; implicitně bez kontrol (kontroly lze dynamicky měnit $-poznámkou R+,R-).

/MAC - cílovým jazykem je assembler; implicitně je cílový produkt relativní modul.

/MXA - je-li /MAC, bude protokol kopírován jako komentář do assembleru; implicitně se jako komentář generují pouze čísla zdrojových řádek (lze dynamicky měnit $-poznámkou M+, M-).

/Wxx - kde x je označení disku (může být A, B až P podle dané systémové konfigurace). Překladač potřebuje pro svou práci dva pracovní soubory, přičemž každá jeho fáze jeden tento soubor zpracovává a druhý generuje. Tyto soubory se jmenují 0.$$$ a 1.$$$ a jsou implicitně vytvářeny na aktuálním disku. Tímto přepínačem je možno určit jiné umístění těchto souborů.

Přepínače s číselnou hodnotou:

/SET:n - horní mez ordinálního čísla posledního prvku množiny s kanonickým typem <u>set of</u> integer (n <u>in</u> 1..255); implicitně n=255.(viz $-poznámka S).

/CND:n - podmíněný překlad úseků mezi $-poznámkami Zk+ a Zk- (k <u>in</u> 0..7, n <u>in</u> 0..255); implicitně n=0.
Číslo n se chápe jako

$$A[7]*2^7 + A[6]*2^6 + \dots + A[0]$$

kde A[k] má hodnotu 0 nebo 1.

Je-li A[k] =1, pak se překládají úseky mezi $-poznámkami Zk+ a Zk-. Úseky, které neleží mezi žádnými $-poznámkami,se překládají vždy.

/PAG:n - počet řádků na stránku protokolu; implicitně n=61 (je-li n větší než počet řádků programu, je protokol bez stránkování).

/FIN:n - přerušení překladu po n-tém průchodu (n <u>in</u> 0..6). Tento parametr slouží pro ladění systému. Při předčasném přerušení se zachovají pracovní soubory, které se jinak ruší.

12.3.2 $-poznámky

J2+ J2- - začátek a konec speciálního překladu příkazu with. Adresa proměnné typu záznam v příkazu with se neukládá na zásobník, ale počítá se vždy znovu při každém přístupu k položkám tohoto záznamu. Ve speciálních případech to může značně zkrátit vygenerovaný kód. Je však třeba dávat pozor na odlišnou sémantiku takto přeloženého programu.

J3+ J3- - začátek a konec speciálního překladu binární operace <u>div</u> , kde druhý argument je konstanta 2^k. Operace se provede pomocí posunů a je-li první argument záporný, je výsledek chybný.

S=n - horní mez ordinálního čísla posledního prvku množiny s kanonickým typem <u>set of</u> integer (n <u>in</u> 1..255). Tato poznámka může být v kompilační jednotce uvedena pouze jednou. Je-li uvedena, ignoruje se přepínač překladu /SET.

P - přechod na novou stránku protokolu (existuje-li soubor LIST).

W - označení rozvoje makra v protokolu (nemá sémantický význam).

L+ - povolení a potlačení výstupu protokolu o překladu
L- (existuje-li soubor LIST), implicitně L+.

R+ - povolení a potlačení kontrol rozsahu intervalů, množin
R- a přeplnění při ordinálních operacích (jen je-li přepínač /RNG), implicitně R+.

M+ - povolení a potlačení výstupu prvých 80 znaků protoko-
M- lu do souboru OUT jako komentář (jen je-li přepínač /ASM a /MXA), implicitně M+.

U+ - povolení a potlačení hlášení varovných zpráv,
U- implicitně U+.

Zk+ - začátek a konec podmíněného překladu zdrojového textu
Zk- (k in 0..7).

B+ - začátek a konec úseku, ve kterém se deklarované pro-
B- měnné (na základní úrovni kompilační jednotky) umisťují do common bloku beze jména - též <u>common</u> <u>var</u>.

D+ - začátek a konec úseku, ve kterém jsou deklarované.
D- procedury a funkce dynamické - též <u>dynamic</u> <u>procedure</u> resp. <u>function</u>.

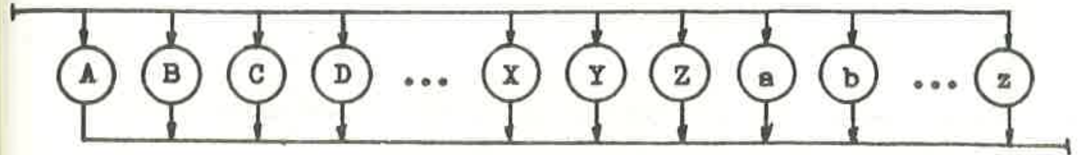
F+ - začátek a konec úseku,ve kterém jsou externí deklaro-
F- vané procedury a funkce systémové - též <u>system</u> <u>procedure</u> resp. <u>function</u>.

X+ - začátek a konec úseku, ve kterém jsou hodnotové para-
X- metry nahrazeny konstantními parametry - též <u>const</u>.

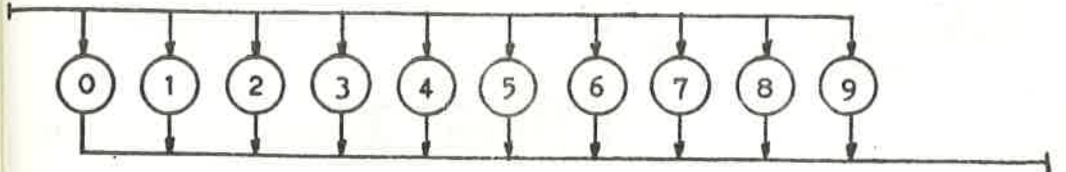
Pozn.: $-poznámky B,D,F,X jsou pouze kvůli kompatibilitě s Fel-Pascalem-ADT.

Přehled syntaxe

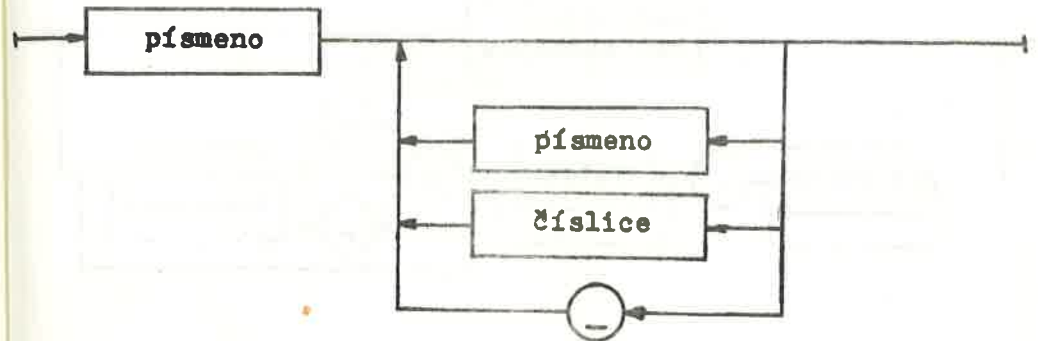
písmeno



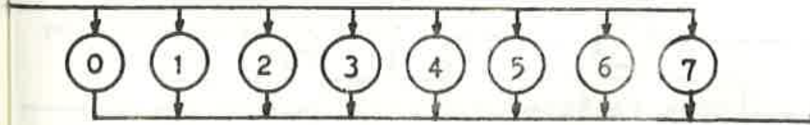
číslice



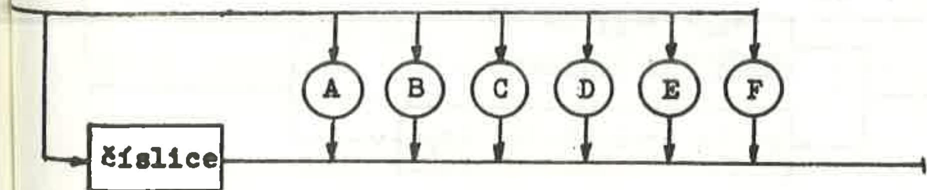
identifikátor



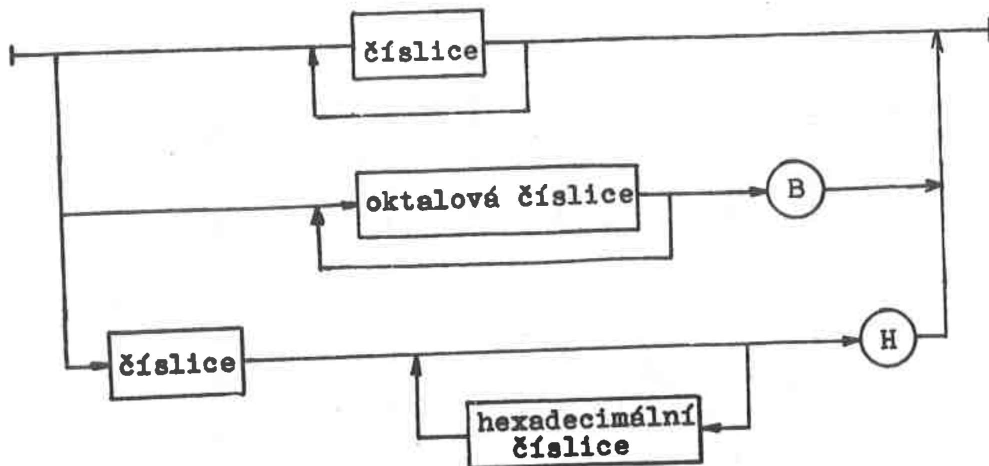
oktalová číslice



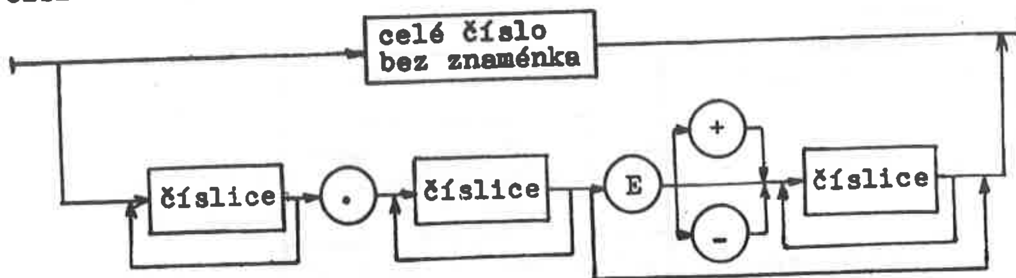
hexadecimální číslice



celé číslo bez znaménka

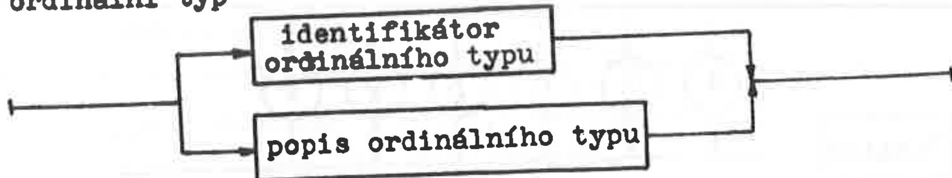


číslo bez znaménka

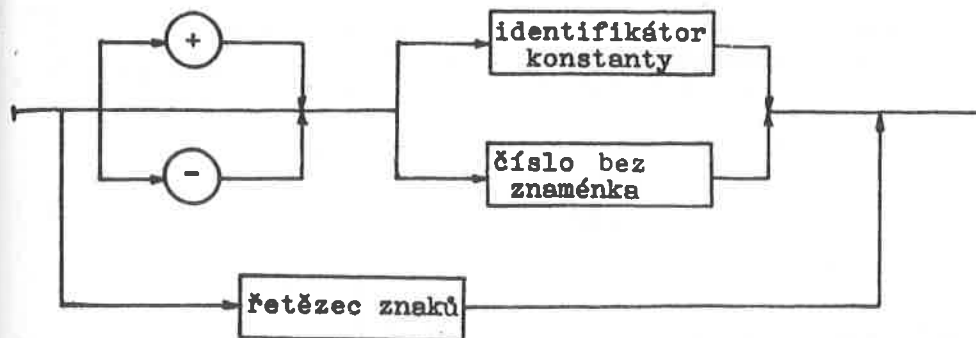


řetězec znaků

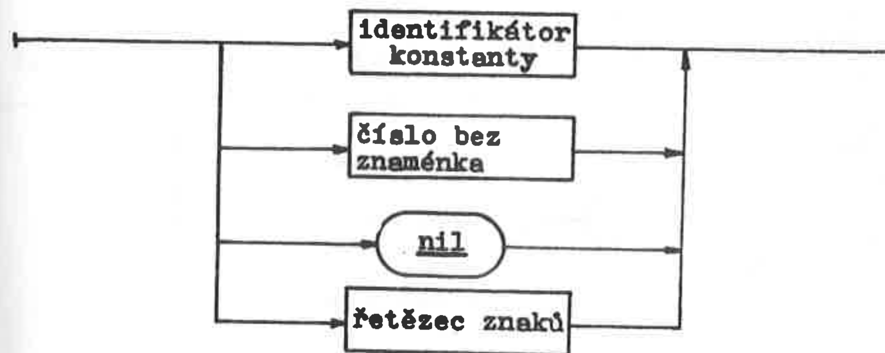
ordinální typ



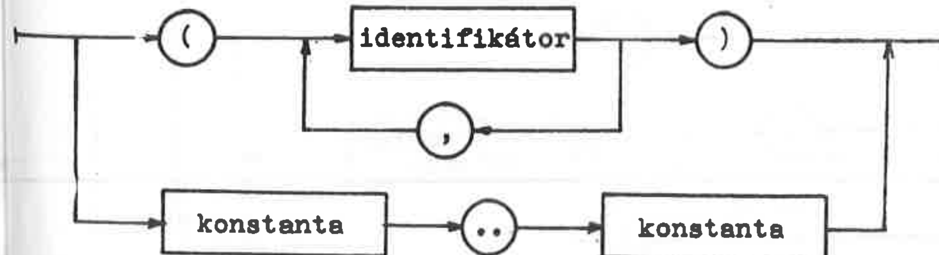
konstanta



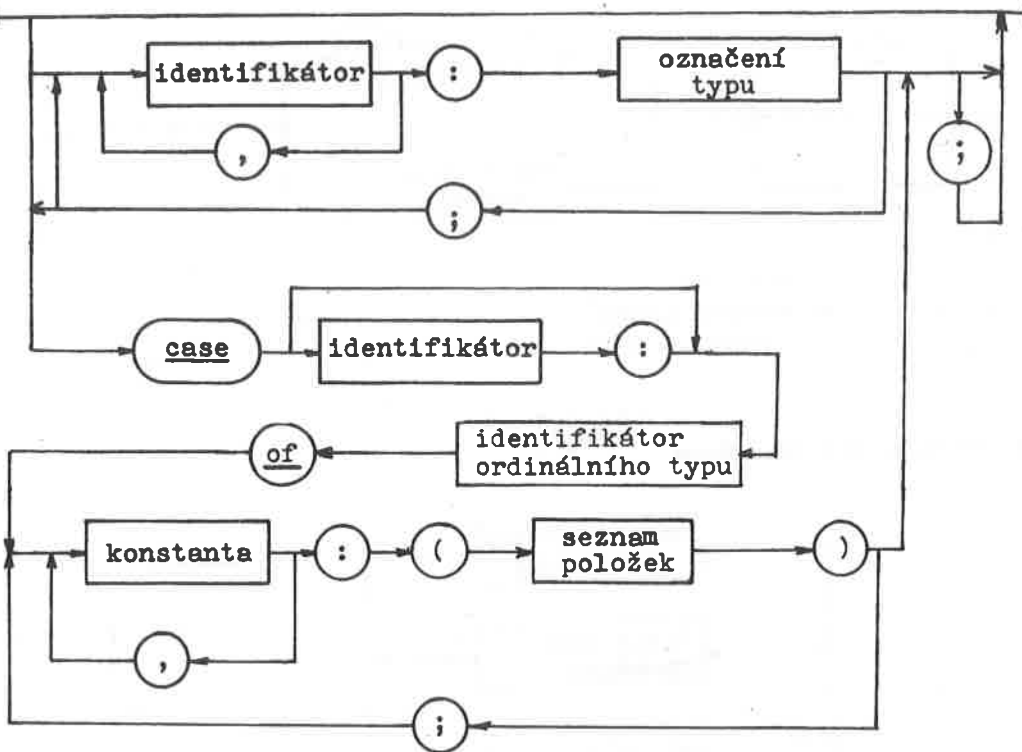
konstanta bez znaménka



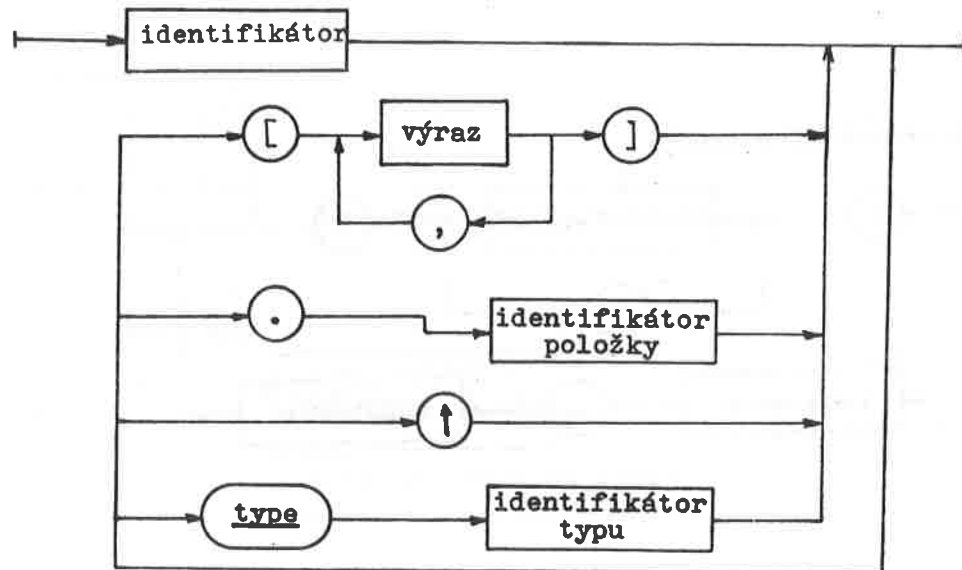
popis ordinálního typu



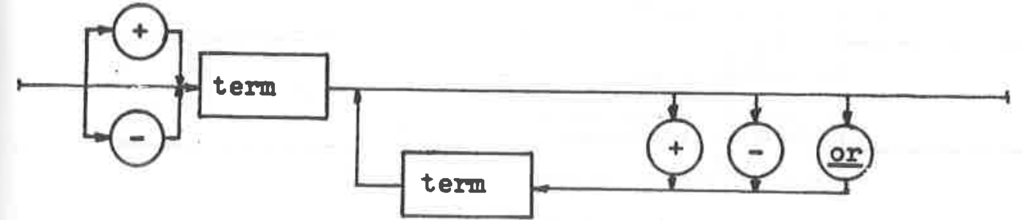
seznam položek



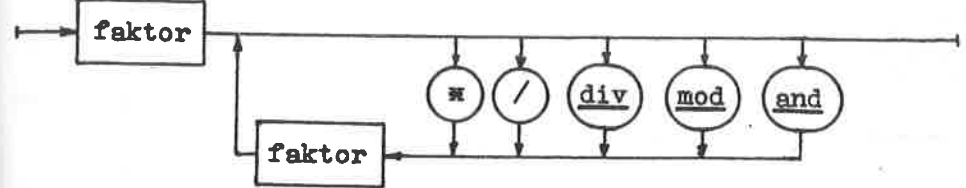
proměnná



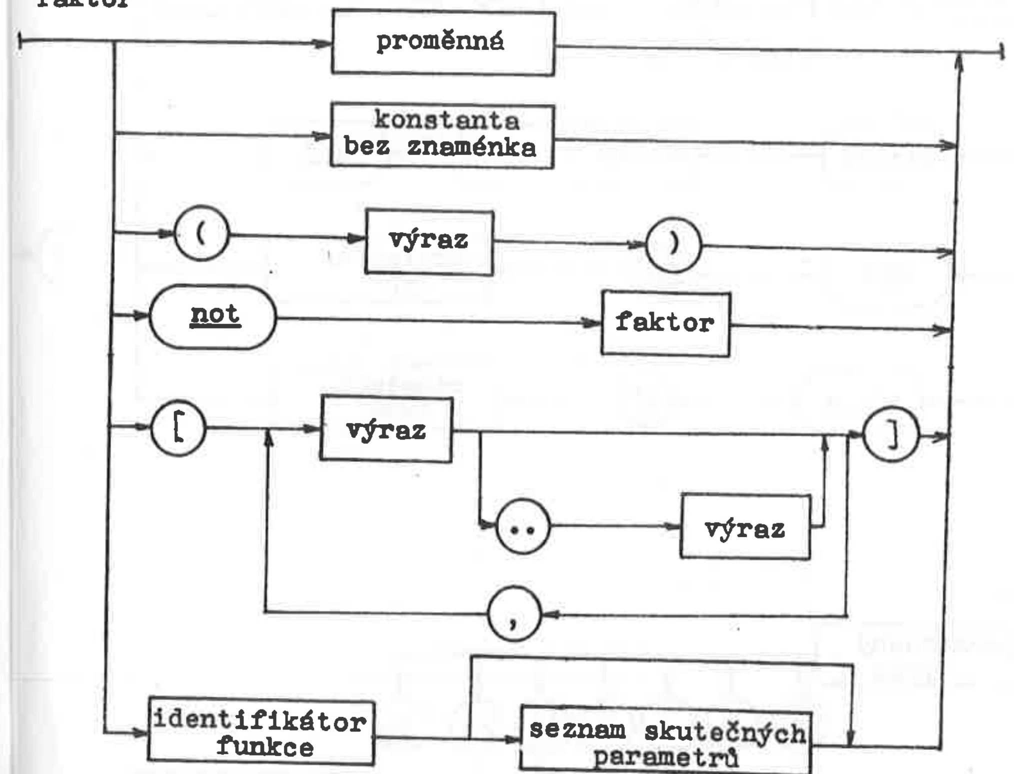
jednoduchý výraz



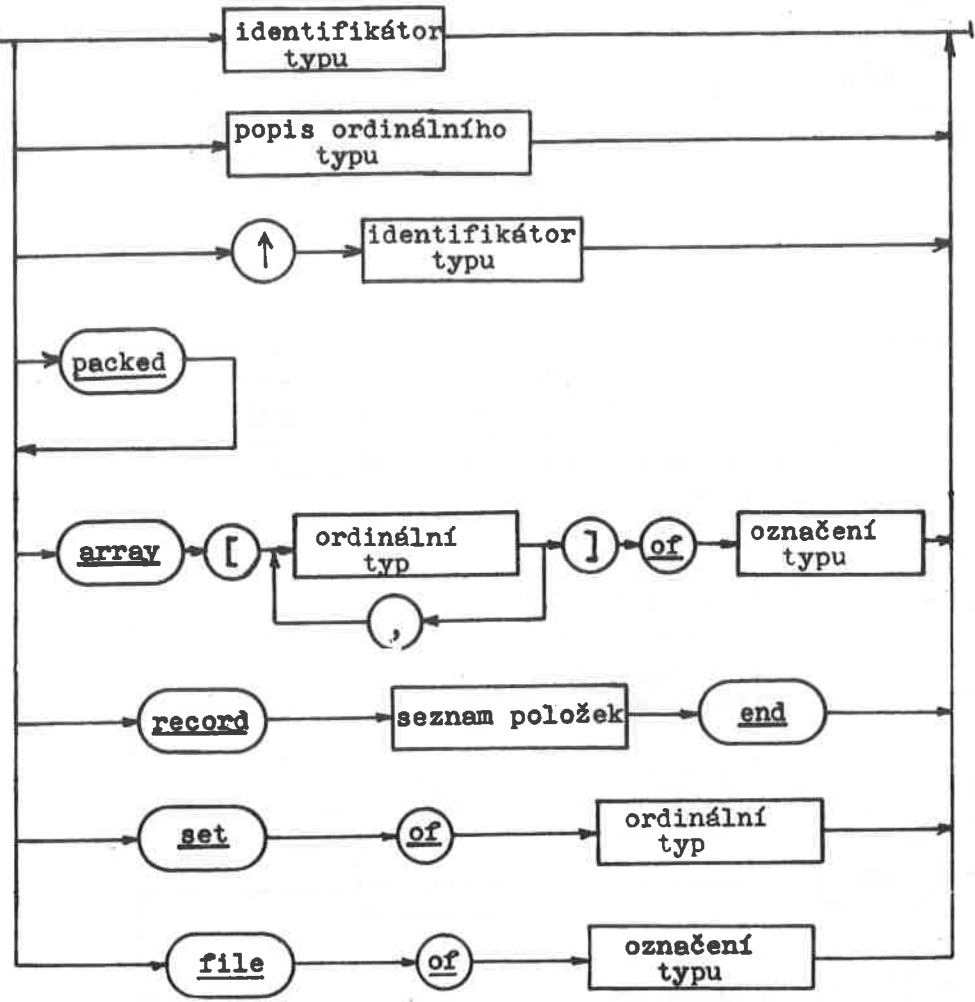
term



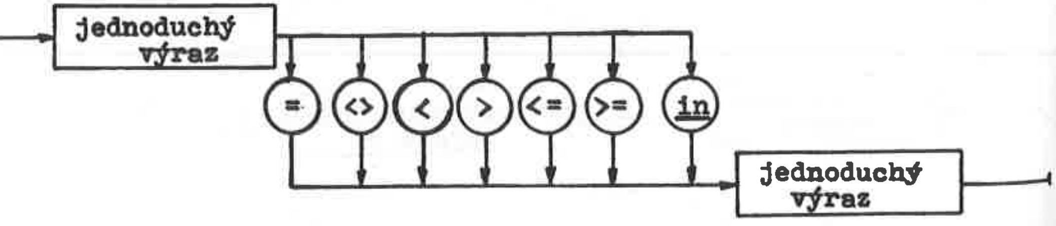
faktor



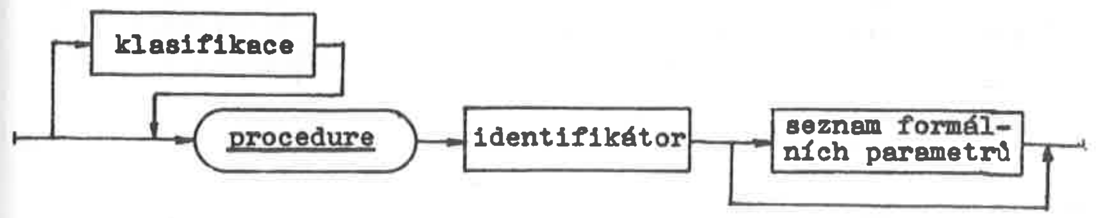
označení typu



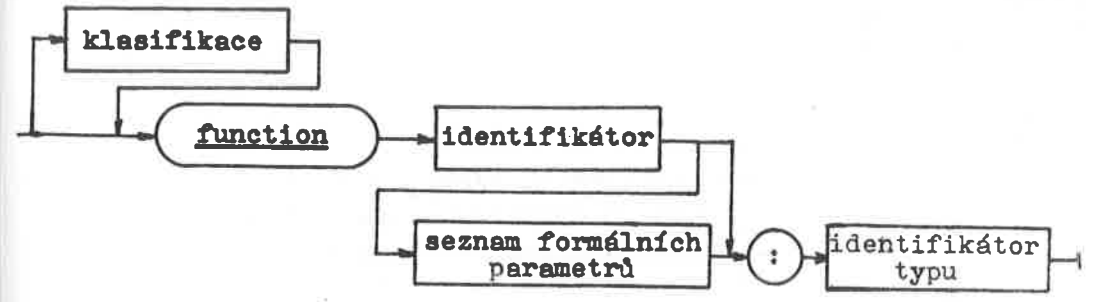
výraz



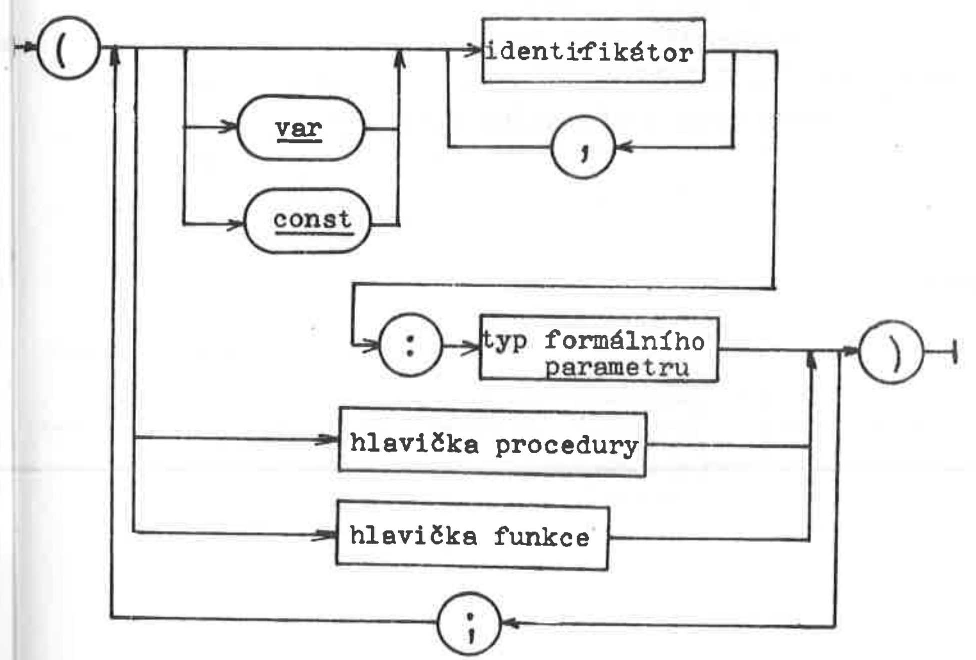
hlavička procedury



hlavička funkce

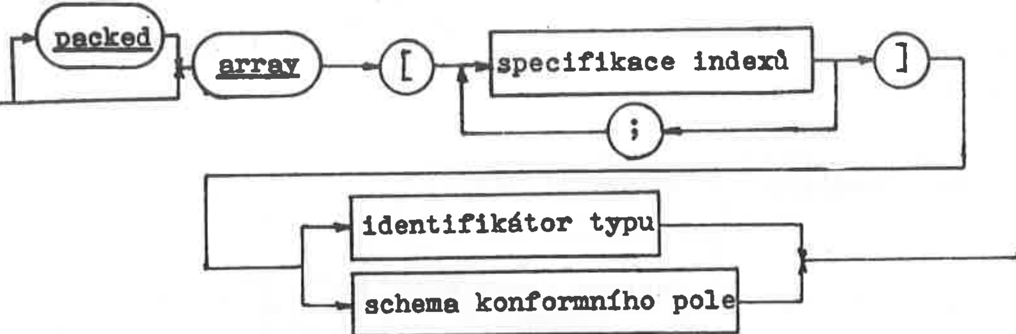


seznam formálních parametrů

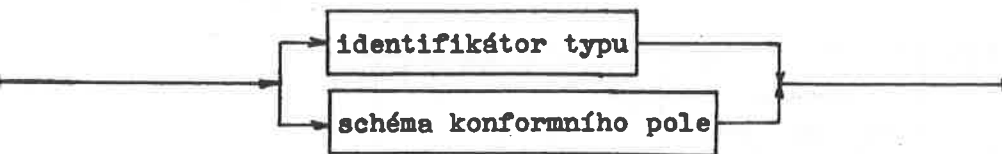


specifikace indexů

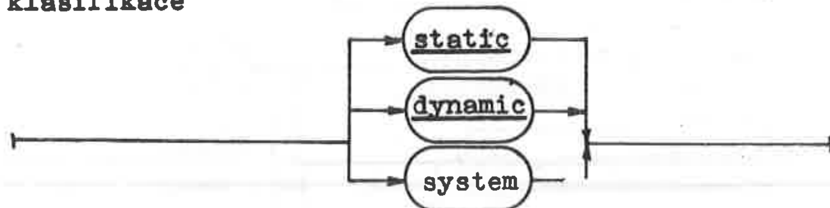
schema konformního pole



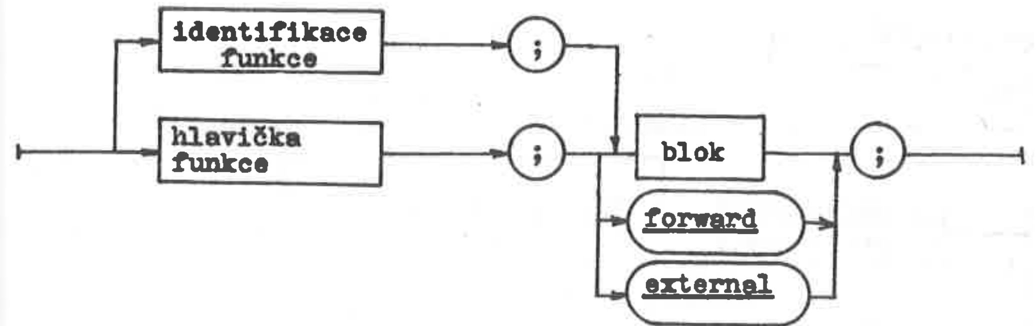
typ formálního parametru



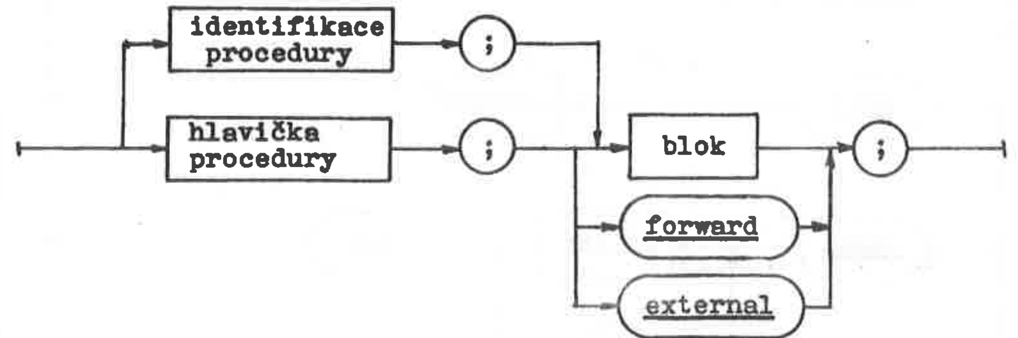
klasifikace



deklarace funkce



deklarace procedury



identifikace procedury

identifikace funkce

příkaz

návěští

identifikátor procedury
seznam skutečných parametrů

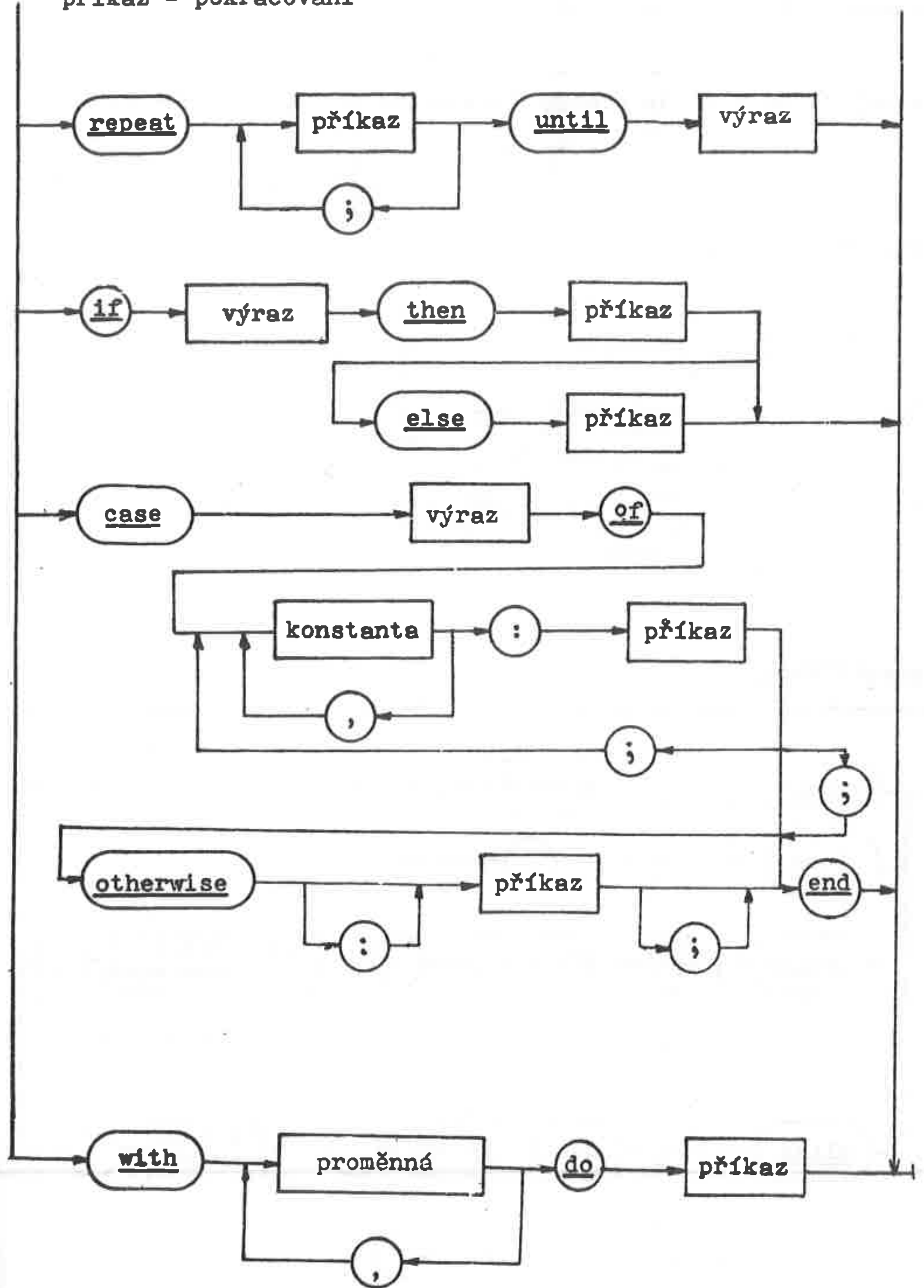
proměnná
identifikátor funkce
:=
výraz

goto
návěští

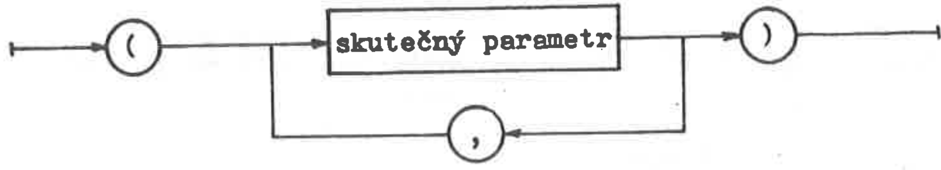
begin
příkaz
;
end

while
výraz
do
příkaz

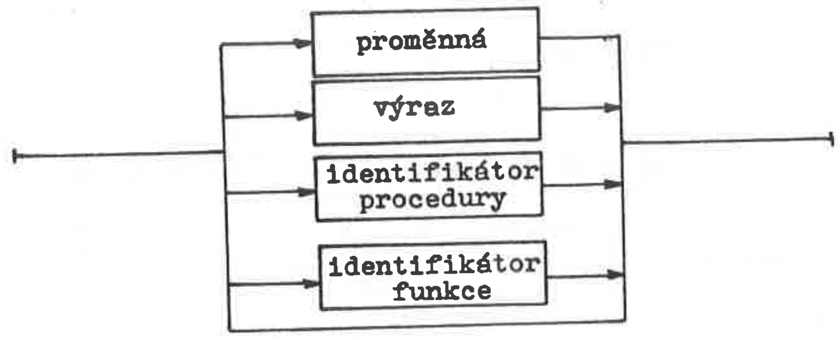
for
:=
výraz
proměnná
:=
výraz
to
downto
výraz
do
příkaz



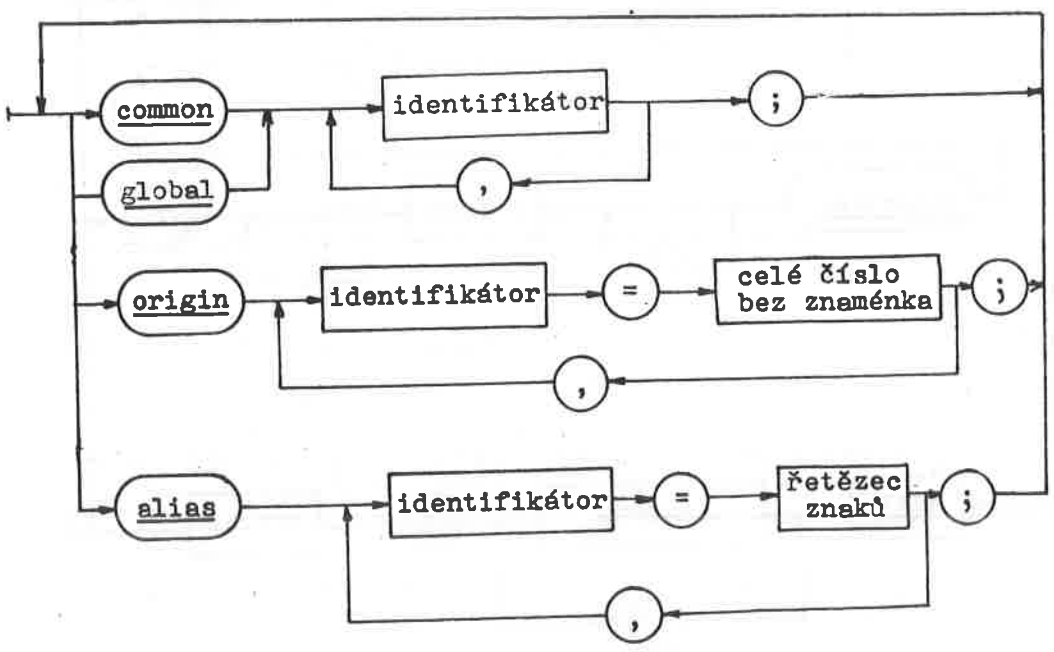
seznam skutečných parametrů



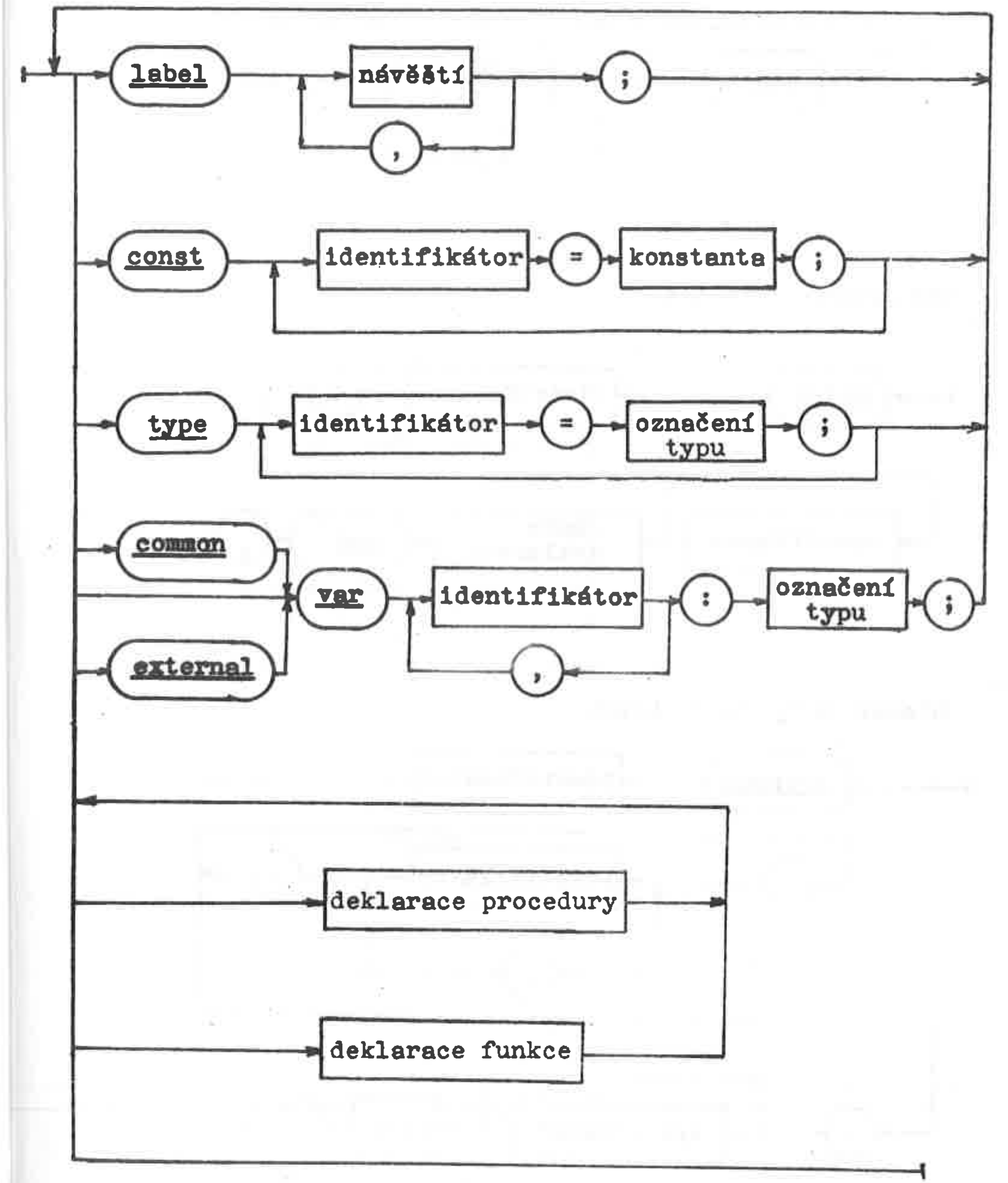
skutečný parametr



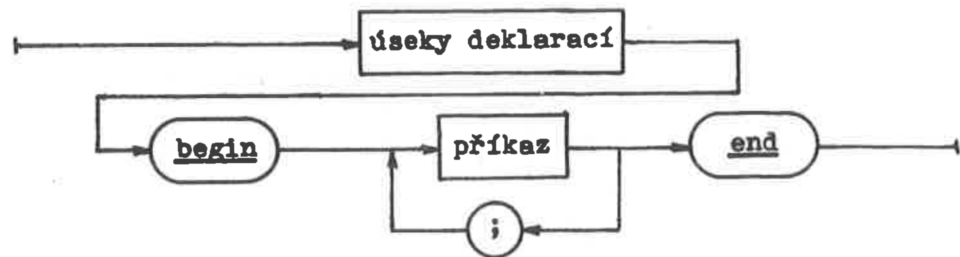
specifikace



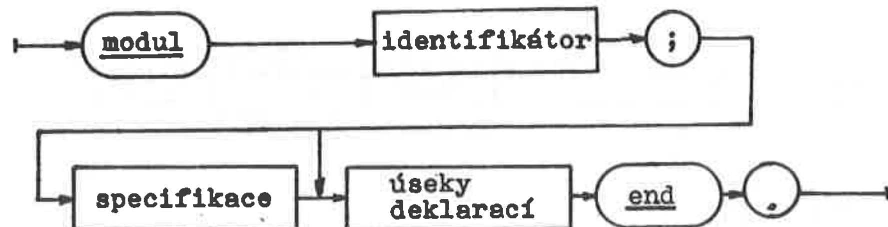
úseky deklarací



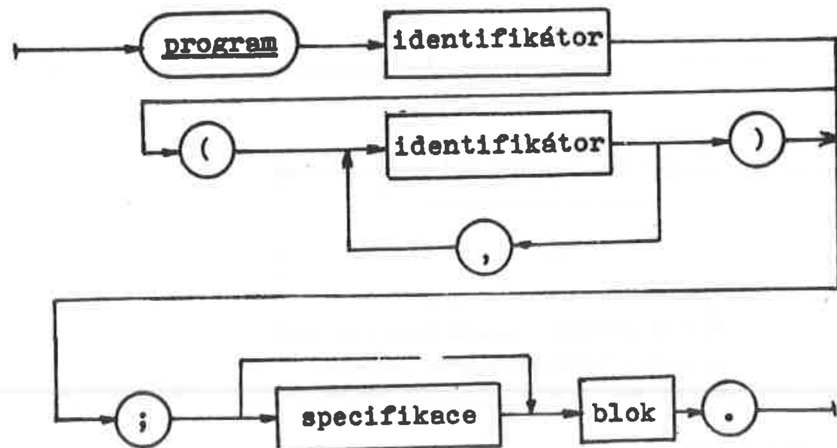
blok



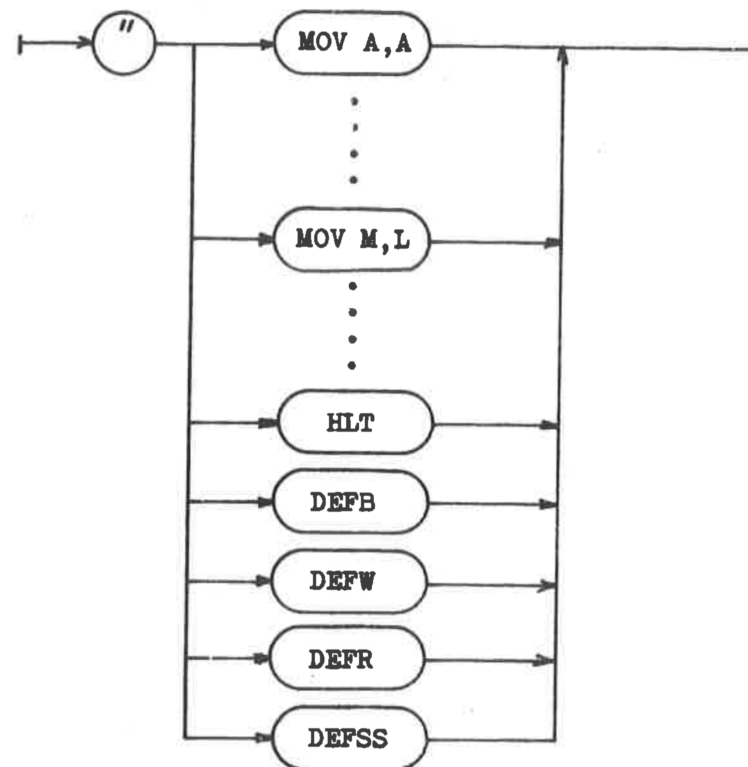
deklarační jednotka



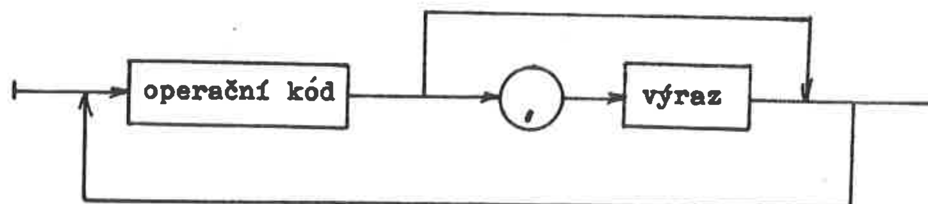
hlavní jednotka programu



operační kód



parametry procedury inline



2002

Název Popis jazyka FEL PASCAL-80 a překladače PF 80

Zpracoval kolektiv autorů : ing.Fr. Drdák, ing.J. Daněček,
ing. Zd. Muzikář, ing. Jan Zajíc

Náklad 150 ks

Tisk Jihočeské tiskárny Strakonice

Publikace je určena pro vnitřní potřebu pobočky ČSVTS-FEL,
Školicí stř. 01-01.