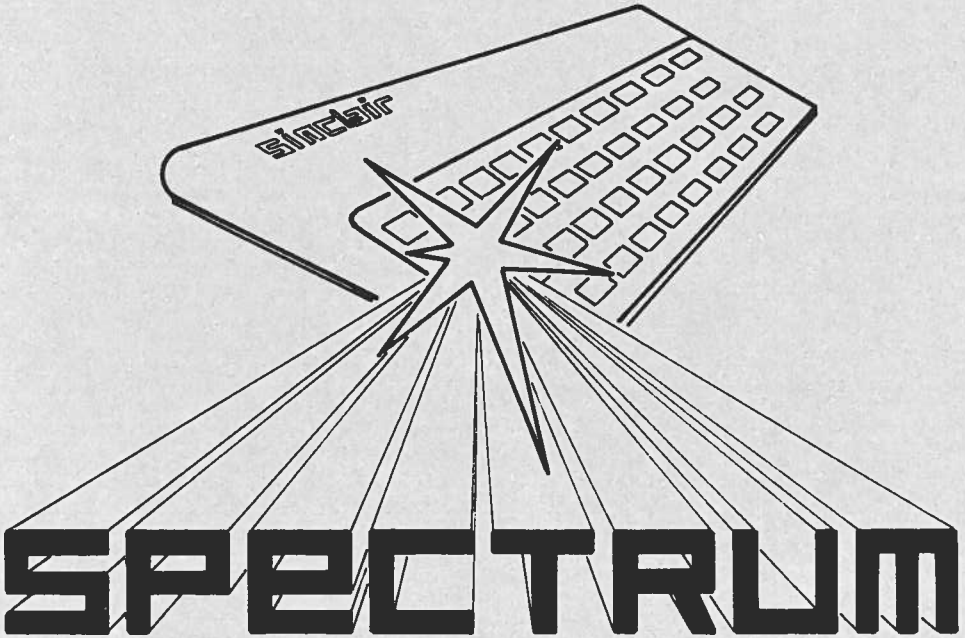
ZO SVAZARM KAROLINKA

# SBORNÍK č. 12

# PASCAL HP 4T



# MANUAL HISOFT PASCAL 4T

HISOFT
13 Gooseacre, Cheddingron
Leighton Buzzard, Beds.
Postcode: LU7 OSR
Tel: (0296) 668995

TRANSL. 1985 SPECTRUM ASSOC. BRNO

## HISOFT PASCAL 4 - VERZE 1.4

Označení Hisoft Pascal 4D a Hisoft Pascal 4T je nyní 1.4, s účinností od 31 října 1982.
Rozdíly mezi předchozí verzí Hisoft Pascal 4 a verzi 1.4 jsou udány níže:
1.Závada ve vyhodnocování výrazů (která vedla k nesprávnému vyhodnocování výrazů jako 1+SQR(2) ) byla opravena.
2.Závada ve vyhodnocování výrazů (jako I+(I+1) ) byla opravena.
3.Závada,která zapříčinila nesprávné vyhodnocování výsledků při srovnávání integer-to-real byla opravena.
4.POUZE PRO HISOFT PASCAL 4T: byla opravena závada v editorovém sub-povelu 'S' (viz str. 38 Programátorské příručky). 'S' nyní může být kdykoliv použito.
5.POUZE PRO HISOFT PASCAL 4T:při použití editorového povelu 'F' k nalezení znakového řetězce, je editorový kurzor umístněn na počátek nalezeného stringu, viz str. 37 a 39 Programátorské příručky.
6.POUZE PRO HISOFT PASCAL 4T: byl přidán nový editorový povel 'X', který zobrazuje v hexadecimální soustavě konečnou běžnou adresu kompilátoru. Tato informace umožňuje uživateli, aby si vyhotovil pracovní kopii kompileru, čímž minimalizuje nebezpečí poškození hlavního záznamu programu. JSTE VŠAK HISOFTEM ZPLNOMOCNĚN KE ZHOTOVENÍ POUZE JEDNÉ PRACOVNÍ KOPIE HP4T. Pro zhotovení pracovní kopie napřed nalezněte počáteční adresu programu ve vašich Poznámkách k implementaci a potom nalezněte konečnou adresu použitím 'X'. Nyní použijte pomocného povelu v operačním systému k nahrání paměťového bloku na pásku. Všimněte si, že nebudete používat HP4T loader k vložení kompilátoru, uloženého touto cestou - používejte pomocný operační systém load-from-tape k přehrání z pásky a potom použijte studený start do editoru podle vašich Poznámek k implementaci.
UPOZORNĚNÍ: Uživatelé ZX Spectra nemohou použít shora popsaný způsob zhotovení kopie a musí nahlédnout do Pozámek k implementaci na podrobné informace o zhotovení pracovní kopie HP4T.

Jako obvykle HISOFT vítá dotazy týkající se zde uvedeného.

## HISOFT PASCAL 4T - VERZE 1.5

Od 1.dubna 1983 existuje Hisoft Pascal 4T ve verzi 1.5. Vlastníci dřívějších verzí HP4T by proto měli přejít na vyšší úroveň verze 1.5 tím, že by vrátili svou starou pásku spolu s manipulačním poplatkem £3 + VAT (luxusní daň z výrobku). Rozdíly mezi verzí 1.4 a 1.5 jsou následující:

1.Funkce nyní mohou vracet POINTER výsledek.

2.Závada v předdefinované proceduře NEW, která měla občas za následek nesprávné umístnění paměti pro dynamickou proměnnou byla nyní opravena.

3.Byl přidán nový editorový povel 'V' (bez argumentu), který zobrazuje běžné implicitní hodnoty rozsahu řádku, string Find a Substitue string (Hledat a Nahradit). Běžná implicitní hodnota rozsahu řádku je zobrazena nejdříve a je následována dvěma řetězci na odlišných řádcích (které mohou zůstat prázdné). Mělo by být připomenuto, že určité povely editoru (jako 'D' a 'N') nepoužívají tyto implicitní hodnoty, ale musejí mít hodnoty specifikované na příkazovém řádku - viz Příručka programátora.

4.POUZE PRO ZX SPECTRUM: 'Include' volba ('$F' viz Sekce 3.2 Programátorské příručky HP4T) je nyní k dispozici ve verzi HP4T pro ZX Spectrum. Když si přejete zvolit tuto možnost, potom zdrojový text, který má být v dalším postupu bezprostředně zahrnut, musí být uložen na pásku použitím editorového povelu 'W' (platí jen pro Spectrum). 'W' funguje přesně jako 'P' s vyjímkou, že neukládá na pásku standardní formát HP4T a začíná ukládat text na pasku bezprostředně po ukončení příkazového řádku. Proto musí být magnetofon nastaven na nahrávání před stisknutím ENTER na konci povelového řádku. Když ukládáte text použitím 'W', nebo čtete text použitím 'F' povelu kompilátoru, potom můžete použít klávesu BREAK kdykoliv. Použití této klávesy vás navrátí do editoru.

Všimněte si, že když si přejete načíst text z editoru povelem 'G', potom text musí být nahrán použitím 'P' a nikoliv 'W'!

Příklad použití:

Vypsání části uživatelského programu:

```
W50,120,PLOT                          ;vypsat PLOT proceduru.
```

Vložení této části do jiného programu:

```
100 END;
110
120 {$F PLOT    vloží sem PLOT proceduru .}
130
140 PROCEDURE MORE;                   {zbytek programu}
150
```

Hisoft doufá, že shora uvedená zlepšení a opravy zvýší použitelnost HP4T a jako obvykle vítá jakoukoliv korespondenci týkající se tohoto dokumentu.

## SEKCE 0. ÚVOD

### 0.0 SPUŠTĚNÍ.

Hisoft Pascal 4T (HP4T) je rychlou, snadno použitelnou a výkonnou verzí jazyku Pascal, jak je specifikováno v příručce Pascal User Manual and Report (Jensen/Wirth Second Edition). Výjimky z této specifikace jsou následující:

FILES nejsou implementovány, i když proměnné mohou být ukládány na pásku.
Typ RECORD nemusí mít součást VARIANT.
PROCEDURE a FUNCTION neplatí jako parametry.
Jsou zahrnuty mnohé mimořádné funkce a procedury, aby odrazily různé podmínky, ve kterých jsou kompilátory používány. Mezi nimi jsou POKE, PEEK, TIN, TOUT a ADDR.
Kompiler zabírá cca 12K paměti, zatím co 'zpracovatel' (runtimes) zaujímá asi 4K. Obojí jsou na dodávané kazetě ve formátu používaném 'runtimes'. Veškeré spojení mezi HP4T a hostitelským systémem je uskutečňováno vektory umístněnými na začátku 'runtimes' (viz HP4T Instrukce o změnách) - to usnadňuje uživateli napsat své vlastní I/O rutiny podle potřeby. Hisoft Pascal 4T používá různé řídící kódy ponejvíce v editoru. Různé systémy však mohou mít různé konstrukce klaviatury a tak i různé způsoby dosahování kontrolních kódů. V této příručce kontrolní znaky budou uváděny takto: RETURN, CC, CH, CI, CP, CS, a CX. Připojená implementační dokumentace vám sdělí odpovídající klávesy u vašeho systému.
Kdykoliv HP4T očekává vstupní řádek, mohou být použity kontrolní znaky:

| | |
|---|---|
| RETURN | je používáno k ukončení řádku. |
| CC | návrat do editoru. |
| CH | vymazání posledního natypovaného znaku. |
| CI | posunuje k následující TAB pozici. |
| CP | řídí výstup na tiskárnu, je-li k dispozici, nebo když výstup jde na tiskárnu, vrací jej na obrazovku. |
| CX | maže celý řádek, který byl napsán. |

Jednoduchý loader je také dodáván v programové sadě, takže uživatel může vkládat z pásky data, která má zaznamenaná ve formátu HP4T.
K vložení kompileru a 'runtimes' z programové pásky dodávané Hisoftem, musí se nejprve vložit loader - je-li to možné, je dodáván ve formě vhodné pro uživatelův operační systém. Když není uživatel schopen nalézt přístup k operačnímu systému počítače, potom loader musí být zaveden přímo do paměti počítače, buď použitím assembleru, nebo vyšším programovacím jazykem jako BASIC. Detaily jsou uvedeny v HP4T Instrukcích o změnách.
Jakmile loader byl zaveden, vyhledá řetězec zaznamenaný ve formátu HP4T na pásce. Jakmile jej nalezne, vloží řetězec do paměti. Když je v kterémkoliv stádiu zjištěna chyba při čtení z pásky, ukáže se odpovídající hlášení na obrazovce. Potom musíte převinout pásku na začátek záznamu a pokusit se vložit jej znovu s jinou úrovní hlasitosti. Když ani to nepomůže, vraťte prosím pásku Hisoftu a my vám ji vyměníme.
Takto loader automaticky uloží překladač a 'runtimes' do paměti vašeho počítače.

Když byl překladač úspěšně vložen, bude automaticky odstartován a dá hlášení:

Top of RAM? (vrchol RAM?)

Na to by se mělo odpovědět buďto kladným dekadickým číslem až do hranice 65536 (následováno RETURN), nebo stisknutím RETURN. (Viz Poznámky k implementaci).

Když vložíte číslo, potom je považováno za nejvyšší paměťové místo RAM + 1, jinak je nejdříve automaticky vypočítána první adresa, která již neleží v RAM. Zasobník překladače je nastaven na tuto hodnotu a tak můžete uchovat paměťová místa s vysokou adresou (třeba pro rozšíření kompileru) záměrným zadáním hodnoty nižší, než skutečná hodnota vrcholu paměti RAM. Ve verzi ZX Spectra je za vrchol RAM považován začátek oblasti UDG (viz manual Sinclair).

Potom se objeví dotaz:

Top of RAM for 'T' (vrchol RAM pro 'T')

Zde můžete zadat dekadické číslo, které nedosahuje vrcholné adresy RAM zadané dříve. Zadaná hodnota bude brána jako zásobník pro strojový kód po použití editorového povelu 'T' (viz pro detaily Sekce 4). Budete muset definovat zásobník pro 'runtimes' odlišný od vrcholu RAM, když jste si například napsali rozšíření týkající se 'runtimes' a přejete si je bezpečně umístnit na horních adresách paměti.

Nakonec budete dotázáni:

Table size? (rozměr tabulky?)

Zde zadávaná hodnota specifikuje rozsah paměti pro tabulku symbolů kompileru.

Zadejte znovu buďto kladné dekadické číslo následované RETURN, nebo jednoduše RETURN samotný. V druhém případě bude brána implicitní hodnota (t.j. dostupná RAM dělená 16) a ta bude brána pro rozměr tabulky symbolů. Téměř ve všech případech hodnoty dané implicitně zajišťují dostatek prostoru pro symboly. Tabulka symbolů nesmí překročit strojovou adresu '8000*(32768 dec.). Když si specifikujete tak velkou hodnotu, že se tak stane, dostanete opět dotaz 'Top of RAM?' atd.

Podle své vůle můžete vložit 'E' před číslo po tomto dotazu. Když to učiníte, potom nebude vnitřní řádkový editor uchován pro použití kompilerem. Udělejte to, když si přejete použít svůj vlastní editor u kompileru (pro detaily viz HP4T Instrukce o změnách).

V tomto případě překladač a editor (když je uchován) bude umístněn na konec symbolové tabulky a provádění bude přeneseno na udržovaný editor.

* Upozornění: Všude v této příručce je znak '`' nahrazen číslem znaku, nebo 'hash' (#, 35 dec., 23 hex., SHIFT '3') na všech systémech, které nepoužívají kód ASCII. Čísla, před kterými tento symbol stojí, jsou v šesnáctkové soustavě.

## 0.1 ROZSAH TÉTO PŘÍRUČKY

Tento manuál není zamýšlen jako učebnice Pascalu. Pokud jste nováčkem v programování v jazyce Pascal, v bibliografii budete odkázán na vhodné knihy.

Tato příručka je pouze srovnávacím dokumentem, který detailně rozvádí zvláštnosti Hisoft Pascal 4.

Sekce 1 vám udává syntaxi a sémantiku očekávanou kompilerem.

Sekce 2 vám detailně rozvádí různé předem definované identifikátory, které jsou u Hisoft Pascal 4 k dispozici od CONST až po FUNCTION.
Sekce 3 obsahuje informace o různých volitelných možnostech kompilátoru, které jsou k dispozici a také o formátu komentářů.
Sekce 4 ukazuje použití řádkového editoru, který je integrální součástí HP4T. Když nechcete používat tento editor, ale chcete připojit svůj vlastní editor, potom viz HP4T Instrukce o změnách.
Shora jmenované sekce by si měli všichni uživatelé velmi dobře prostudovat.
Dodatek 1 detailně upřesňuje chybová hlášení kompileru i 'runtimes'.
Dodatek 2 obsahuje předdefinované identifikátory a rezervovaná slova.
Dodatek 3 udává detaily o vnitřní reprezentaci dat v Hisoft Pascal 4 - je to prospěšné pro programátory, kteří chtějí pracovat i na úrovni strojního kódu.
Dodatek 4 udává některé příklady programů v jazyce Pascal, při problémech se psaním programu pro Hisoft Pascal 4 si je prostudujte.

## 0.2 PŘEKLAD A ZPRACOVÁNÍ

Pro detaily, jak vytvářet, upravovat, překládat a spouštět programy pro HP4T užíváním integrálního řádkového edtoru viz sekce 4 této příručky. Pro informaci jak postupovat při použití vašeho vlastního editoru viz HP4T Instrukce o změnách.
Když již byl kompiler spuštěn, generuje soupisku v této formě:

xxxx nnnn text zdrojového řádku

kde: xxxx je počáteční adresa strojového překladu řádku
nnnn je číslo řádku s vynechanými počátečními nulami

Když řádek obsahuje více než 80 znaků, potom kompilátor vkládá znaky nového řádku, takže délka řádku není nikdy větší, než 80 znaků.
Listing může být vypisován tiskárnou. Je-li to požadováno, použijte 'P', je-li připojena (viz sekce 3).

V kterémkoliv stádiu můžete udělat přestávku v listingu stlačením CS. Ihned potom použijte CC k návratu do editoru, nebo nějakou jinou klávesu k znovuspuštění listingu.
Když je zjištěn omyl během překladu, potom hlášení '*ERROR*' bude vypsáno na obrazovce a bude následováno '^' ukazující za (!) symbol, který vyvolal chybové hlášení a číslem chyby (viz Dodatek 1). Listing se zastaví, natypujte 'E' k návratu do editoru k editaci řádku, který je ukázán na obrazovce, 'P' k návratu do editoru a k editaci předešlého řádku pokud existuje, a nebo jakoukoliv jinou klávesu k pokračování kompilace.
Když program končí nesprávně (např. chybí 'END'), potom je zobrazena zpráva 'No more text' (není více textu) a řízení se vrátí editoru.
Když překladač vyběhne z tabulkového prostoru, potom se objeví hlášení 'No Table Space' (není místo v tabulce) a kontrola se vrátí editoru. Normálně programátor potom uloží program na pásku a znovu založí kompiler. Potom specifikuje větší 'Table size' (viz sekci 0.0).

Jestliže překlad skončí bez správně, ale když obsahovala chyby, potom bude počet detekovaných chyb ukázán na obrazovce a strojový překlad bude vymazán. Když je překládání úspěšné, potom bude zobrazen dotaz 'Run?' na obrazovce. Pokud si přejete ihned spustit program, potom odpovězte 'Y', jinak bude kontrola navrácena editoru.
Během zpracování objektového kódu jsou generována rozmanitá chybová hlášení (viz Dodatek 1). Chod můžete zastavit použitím CS, potom použijte CS k předčasnému ukončení chodu, nebo nějakou jinou klávesu k obnovení chodu.

## 0.3 STANOVENÍ TYPU PROMĚNNÝCH

Různé jazyky mají různé způsoby zajištění, aby uživatel nepoužíval takové datové elementy, které se neslučují s jejich definicí.
Na jednom konci škály stojí strojový kód, kde není prováděna žádná sémantická kontrola typu proměnných,na které je odkazováno v programu. Nejblíže dále stojí Byte 'Tiny Pascal', u kterého mohou byt volně směšována data typu INTEGER a BOOLEAN bez vzniku chyby. Dále na stupnici stojí BASIC, který rozlišuje mezi čísly, řetězci a někdy mezi čísly INTEGER a REAL (např. použitím znaku '%' k označení celých čísel). Potom přichází Pascal, který zachází až tak daleko, že dovoluje různé typy, vyjmenované uživatelem. Na vrcholu škály je v současné době jazyk jako ADA, ve kterém lze definovat různé neslučitelné numerické typy.

Existují zásadně dva přístupy používané implementacemi Pascalu pro zadávání typu: implicitní a explicitní. Hisoft Pascal 4 používá explicitní ekvivalenci pro RECORD a ARRAY. Důsledky jsou objasněny v sekci 1 - na tomto místě jen jeden příklad. Dejme tomu, že proměnné jsou definovány následovně:

```
VAR A:ARRAY['A'..'C'] OF INTEGER;
    B:ARRAY['A'..'C'] OF INTEGER;
```

potom to může svádět uživatele, aby si myslel, že může psát A:= B: , ale to by vyvolalo chybu (*ERROR* 10) u Hisoft Pascalu 4, protože byly vytvořeny dva různé 'TYPE records' nahoře uvedenými definicemi. Jinými slovy, uživatel nerespektoval rozhodnutí, že A a B mají reprezentovat jeden typ dat. Mohlo by se to učinit takto:

```
VAR A,B : ARRAY['A'..'C'] OF INTEGER;
```

a nyní může uživatel volně přičleňovat A k B a naopak B k A, protože byl vytvořen pouze jeden 'TYPE record'.
I když, povrchně vzato, se tato explicitní definice typu může zdát poněkud komplikovaná, všeobecně to však vede k méně častým chybám v programu, protože to vyžaduje lepší předběžné promyšlení úlohy programátorem.

## SEKCE 1 SYNTAXE A SÉMANTIKA

Tato sekce detailně rozpracovává syntaxi a sémantiku Hisoft Pascalu 4 - a to v odlišnostech implementace od standardní verze Pascalu (viz Pascal User Manual and Report (Jensen,Wirth druhé vydání)).

## 1.1 IDENTIFIKATOR.

```
                                        _______
                                 _____ |písmeno|_____
                                |      |_______|     |
 __________ _______ __________ |_____________________|___________>
           |písmeno|          |      _______
           |_______|          |_____ |číslice|_____|
                                     |_______|
```

Pouze prvních 10 znaků identifikátoru je významových.
Identifikátory mohou obsahovat velká, nebo malá písmena. Velká písmena nejsou měněna na velká, takže identifikátory HELLO, HELlo a hello jsou od sebe odlišné. Rezervovaná slova a předem definované identifikátory musí být zadávány velkými písmeny!

## 1.2 CELÉ ČISLO BEZ ZNAMÉNKA.

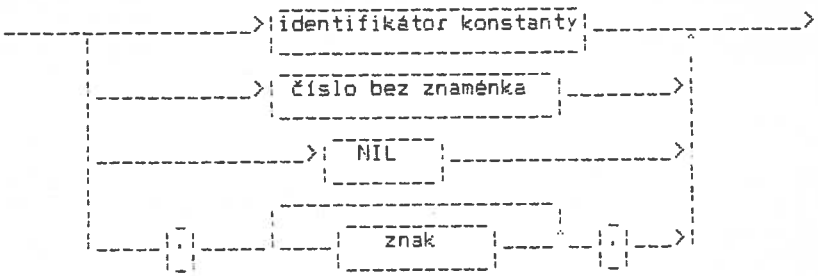
```
                 ______________________
                |                      |
 _______________|______>|_______|______^_____________________>
                        |číslice|
                        |_______|
```

## 1.3 ČISLO BEZ ZNAMENKA.

```
                 ___________________________________________
                |                                           |
                |  ________________                         |
  ___________   | |                |        +   ___________ |
 |INTEGER bez|  | |  _       _____ |  _   | |  |INTEGER bez| |
_>|  znaménka |__^_^_|.|_>|číslice|_|>|E|_^_|_|  znaménka |_|_>
 |___________|     |_|^ |_____|  | |_|  |^|    |___________|^
 |                  |____________|        | |                |
 |                                        _                  |
 |  _   ___________                                          |
 |__>|^|__|_>|  hex  |_^_________________________________________|
     |_|     |číslice|
             |_______|
```

Celá čísla mají absolutní hodnotu menší nebo rovnou 32767 ve verzi Hisoft Pascal 4. Větší celá čísla jsou zpracovávána jako reálná.
Mantisa reálných čísel je 23 bitová. Přesnost při použití reálných čísel je tedy cca 7 platných míst. Povšimněte si, že přesnost je ztracena, když je výsledek výpočtu mnohem menší, než absolutní hodnota jeho argumentu, např. 2.00002-2 nedá 0.00002. Je to důsledkem nepřesnosti, způsobené reprezentováním zlomku desetinného čísla ve tvaru binárního zlomku. Nestane se to, když jsou menší celá čísla reprezentována jako reálná, např. 200002-200000=2 přesně.
Největší zobrazitelné reálné číslo je 3.4E38, nejmenší reálné číslo je 5.9E-39.
Nemá zde smysl používat více než 7 číslic v mantise, když se specifikují reálná čísla, protože další místa jsou ignorována s výjimkou jejich umístnění.
Když je důležitá přesnost, vyhýbejte se tomu, abyste uváděli vedoucí nuly, protože ty se také počítají jako číslice. Tak 0.000123456 je reprezentováno méně přesně, než 1.23456E-4.
Hexadecimální čísla jsou programátorovi k dispozici, aby specifikovala podrobně adresy paměti pro jazyk symbolických adres pro jeho vazbu s jinými. Všimněte si, že zde musí být alespoň jedna hexadecimální číslice po '$', jinak je hlášena chyba (*ERROR* 51).

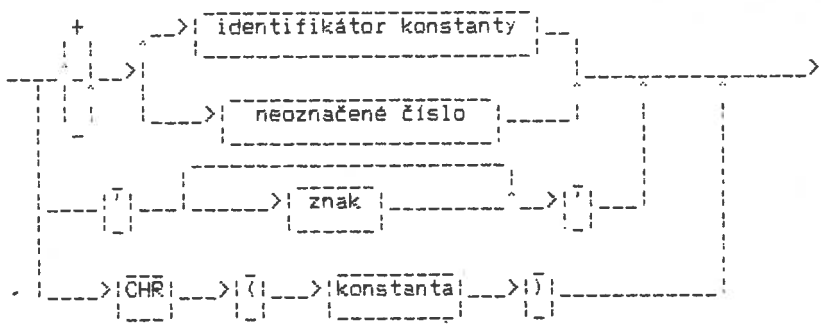
## 1.4 KONSTANTA BEZ ZNAMÉNKA




Všimněte si, že řetězce nesmějí obsahovat více, než 255 znaků. řetězcové typy jsou ARRAY [1..N] OF CHAR kde N je celé číslo mezi 1 až 255 včetně. Písmenové řetězce by neměly obsahovat znak end-of-line (CHR(13)), jinak je vyvolán omyl '*ERROR* 68'.

Znaky, které jsou k dispozici, jsou úplnou množinou ASCII s 256 prvky. Pro zachování kompatibility se standardním Pascalem není znak NUL reprezentován jako ''; místo toho by mělo být použito CHR(0).

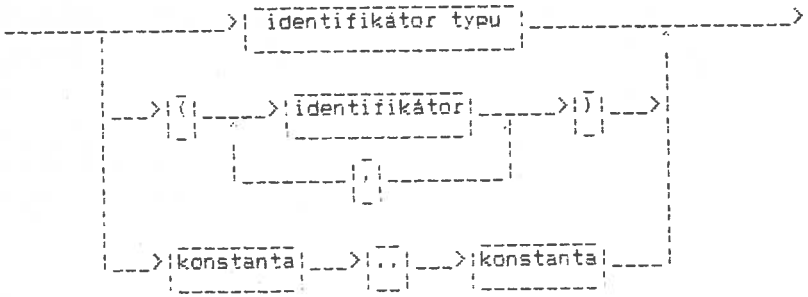
## 1.5 KONSTANTA.




Nestandardní CHR konstrukce je zde tak provedena, že konstanty mohou být používány pro řídící znaky. V tom případě konstanta v závorkách musí být typu INTEGER.

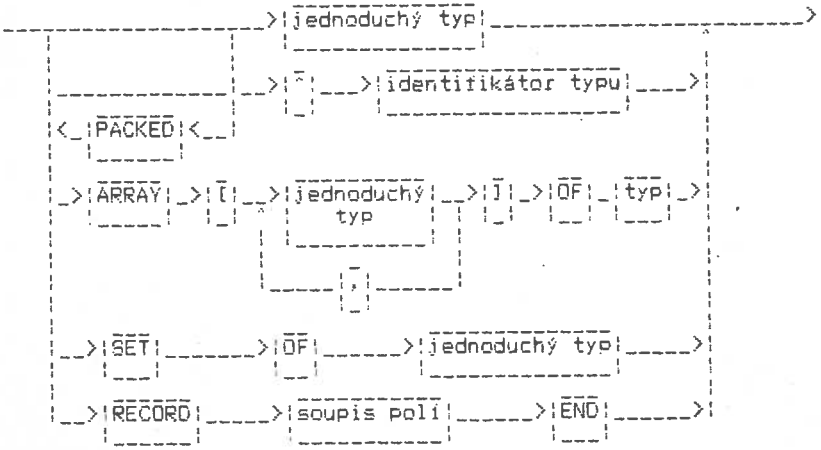
Příklad: CONST bs=CHR(10);
cr=CHR(13);

## 1.6 JEDNODUCHÝ TYP




Skalárně vyčíslené typy (identifikátor, identifikátor, .....) nesmějí mít víc než 256 prvků.

## 1.7 TYP.




Rezervované slovo PACKET je akceptováno, ale je ignorováno, protože ke zhušťování již dochází pro řetězce znaků atd. Jen v případě Booleových řetězců by toto zhušťování mělo smysl, ale to je přirozenější vyjádřit jako množinu, když je zhušťování požadováno.

## 1.7.1 POLE A SOUBORY.

Základní typ souboru smí mít až 256 prvků. To umožňuje, aby soubory (SETs) znaků (CHAR) byly deklarovány společně se soubory jakýchkoliv uživatelem vyjmenovaných typů. Všimněte si ale, že jenom podřetězce celých čísel mohou být používány jako základní typy. Všechny podsoubory celých čísel jsou zpracovávány jako soubory 0..255.
Celá pole polí, pole souborů, recordů souborů atd. jsou přípustné.
Dva ARRAY typy jsou zpracovávány jako ekvivalentní, pokud jejich definice vychází z téhož použití rezervovaného slova ARRAY. Tak nemohou být ekvivalentní následující typy:

```
TYPE

      tablea = ARRAY[1..100] OF INTEGER;
      tableb = ARRAY[1..100] OF INTEGER;
```

Takovým způsobem proměnná typu tablea nesmí být přiřazena k proměnné typu tableb. To umožňuje detekovat chyby jako například přiřleňování dvou tabulek reprezentujících různá data. Zhora uvedené omezení neplatí pro speciální případ pole typu řetězcového, protože pole tohoto typu jsou vždy používána k tomu účelu, aby představovala podobná data.

## 1.7.2 UKAZATELE.

Hisoft Pascal 4 dovoluje vytváření dynamických proměnných použitím standardní procedury NEW (viz sekce 2). Na dynamickou proměnnou, na rozdíl od statické proměnné, která má paměťový prostor umístněn v bloku, ve kterém je deklarována, nemůže být činěn odkaz identifikátorem, protože jej nemá. Místo toho se používá proměnná typu ukazatel. Proměnná typu ukazatel 'pointer' je proměnnou statickou, obsahuje adresu dynamické proměnné a dynamická proměnná sama dostává zprostředkovaný přístup použitím znaku '^' za ukazatelovou proměnnou. Příklady použití proměnné typu ukazatel viz Dodatek 4.
Užívání ukazatele mají některá omezení v Hisoft Pascalu 4. Zde jsou některá z nich:

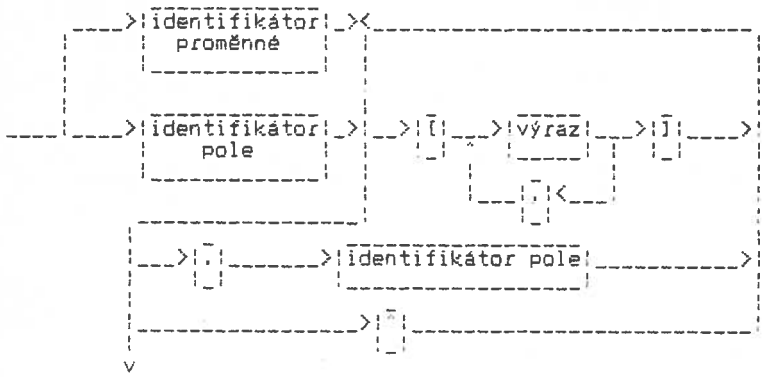
Ukazatele na typy, které nebyly deklarovány nejsou dovoleny. To nebrání konstrukci sřetězených soupisových struktur, protože symbolové definice mohou obsahovat ukazatele na sebe samé.
Např.:

```
TYPE
   item = RECORD
          value : INTEGER;
          next : ^item
          END
    link = ^item;
```

Ukazatele směrem na ukazatele nejsou přípustné.
Ukazatele na ten samý symbol jsou považovány za ekvivalenty.
Např.:

```
VAR
   first : link;
   current : ^item;
```

## 1.9 PROMĚNNÁ




V Hisoft Pascalu 4 jsou možné dva druhy proměnných: statické a dynamické. Statické proměnné jsou deklarovány explicitně užitím VAR a je pro ně přidělována paměť v průběhu celého provádění bloku, ve kterém jsou deklarovány.

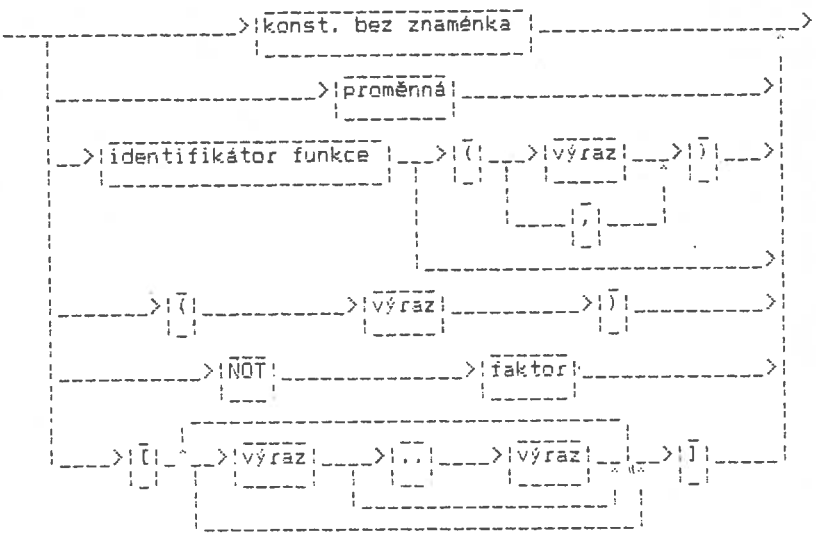
Dynamické proměnné jsou vytvářeny dynamicky při provádění programu procedurou NEW.Nejsou deklarovány explicitně, nemůže na ně být prováděn odkaz identifikátorem. Je na ně činěn odkaz nepřímo statickou proměnnou typu pointer, která obsahuje adresu dynamické proměnné.

Viz sekce 1.7.2 a sekce 2, kde je uvedeno více detailů o používání dynamických proměnných a příklad viz Dodatek 4.

Při specifikování elementů mnohorozměrných polí programátor není nucen používat tutéž formu indexové specifikace k odkazům jako při deklaraci - to je změna oproti Hisoft Pascal 2.

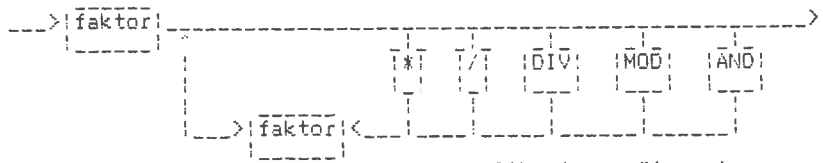
Například: když je proměnná popisována jako ARRAY[1..10] OF ARRAY[1..10] OF INTEGER potom buďto a[1][1] nebo a[1,1] může být použito pro přístup k elementu (1,1) tohoto řetězce.

## 1.10 FAKTOR.




Detaily viz VÝRAZY v sekci 1.13 a FUNKCE v sekci 8.

## 1.11 TERM (člen)

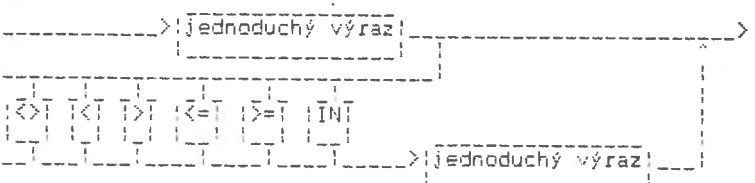



Dolní hranice množiny je vždy 0 a velikost množiny je roven maximu základního typu množiny. Takto SET OF CHAR (množina znaků) vždy zaujímá 32 bytes (je možných 256 elementů - jeden bit pro každý element). Podobně je SET OF 0..10 ekvivalentní SET OF 0..255.

## 1.12 JEDNODUCHÝ VÝRAZ.

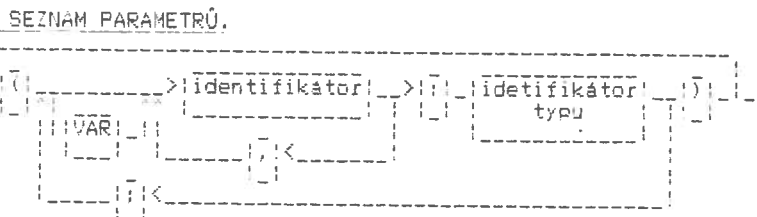

Tytéž poznámky, uvedené v sekci 1.11 pokud se týče soustav, se také vztahují na jednoduché výrazy.

## 1.13 VÝRAZY.




Když používáme IN, potom atributy množiny jsou stejného typu jako typ jednoduchého výrazu s výjimkou integer argumentu pro který jsou atributy omezeny v rozsahu (0..255).
Horní syntaxe se vztahuje na srovnávání stringu té samé délky, ukazatele a všechny skalárního typy. Soustavy mohou být srovnávány použitím >=, <=, <>, nebo =, pointery mohou být porovnávány pouze použitím = a <>.

## 1.14 SEZNAM PARAMETRŮ.




Identifikátor typu musí být použit za dvojtečkou, jinak vznikne *ERROR* 44.
Proměnné parametry, stejně i hodnotové parametry jsou plně přijímány.
Procedúry a funkce neplatí jako parametry.

## 1.15 PŘÍKAZ.

Srovnej syntaktický diagram na str. 16 - 17.

Přiřazovací příkaz: viz sekce 1.7 ohledně nepřípustných přiřazovacích příkazů.

CASE příkazy: úplně prázdný CASE seznam je nepřípustný, znamená to CASE OF END; to generuje *ERROR* 13.

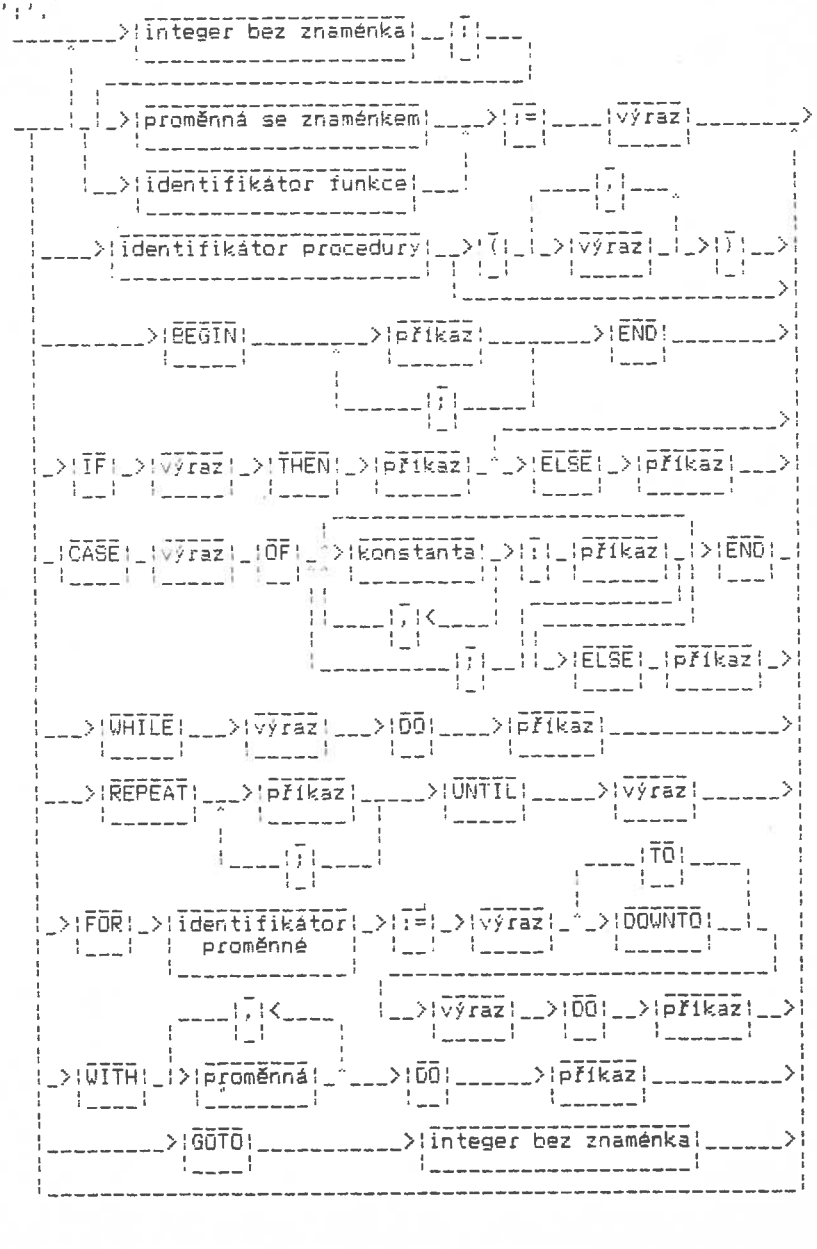
ELSE klauzule, alternativní k END, je vykonávána,když selektor ('výraz' mimo ) se nenachází v CASE seznamu ('konstanta' mimo).

Je-li použit terminátor END a když nebyl nalezen selektor, pak je kontrola předávána na příkaz následující po END.

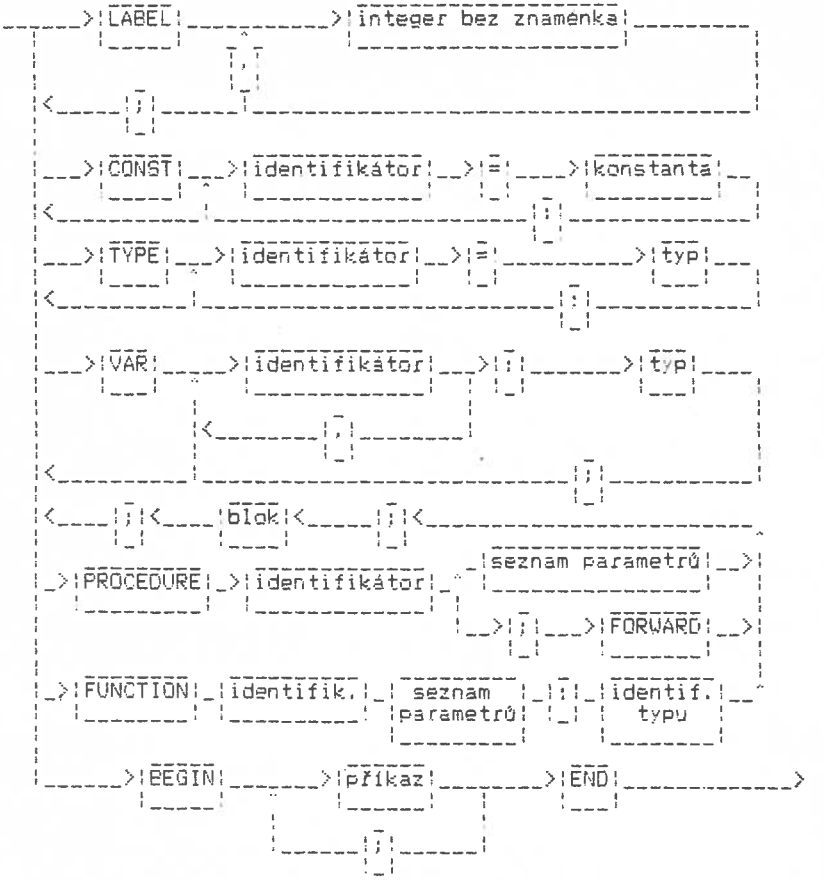
Příkaz FOR: kontrolní řídící proměnná příkazu FOR smí být pouze nestrukturovaná, nikoliv parametr. To je kompromisem mezi standardem Jensen/Wirth a návrhem ISO standardní definice.

Příkaz GOTO: jedinou možností pro GOTO je návěští přítomné v tomtéž bloku a na stejné úrovni jako příkaz GOTO.

Návěští musí být popisována použitím rezervovaného slova LABEL v bloku, ve kterém jsou používána. Návěští se skládá z nejméně jedné až do čtyř číslic. Když je použito návěští k označení příkazu, musí se objevit na počátku příkazu a musí být následováno ':'.



## 1.15 BLOK




## PŘEDSUNUTÉ DEKLARACE.

Tak, jak je uvedeno v Pascal User Manual and Report (Sekce 11. C.1), může být odkazováno na procedury a na funkce dříve, než byly popsány, použitím rezervovaného slova FORWARD. Např.:

```
PROCEDURE a(y:t); FORWARD;    {procedura, která má být popsána}
PROCEDURE b(x:t);             {před tímto příkazem}
 BEGIN
 ....
 a(p);                        {procedura, na kterou je odkazováno}
 ....
 END;
```

```
PROCEDURE a;                    (vlastní popis procedury a)
 BEGIN
 ....
 b(q);
 ....
 END;
```

Všimněte si, že parametry a výsledný typ procedury a jsou popisovány s FORWARD a nejsou opakovány v hlavním popisu procedury. Připomeňme si, že FORWARD je rezervované slovo.

### 1.17 PROGRAM

```
___>|PROGRAM|__>|identifikator|__>|;|__>|blok|__>|END|__>|.|
    |_______|   |_____________|   |_|   |____|   |___|   |_|
```

Protože nejsou implementovány soubory dat, neexistují žádné formální parametry programu.

## SEKCE 2. PŘEDEFINOVANÉ IDENTIFIKÁTORY

### 2.1 KONSTANTY

MAXINT  Největší celé číslo, které je k dispozici, t.j. 32767.
TRUE, FALSE  Konstanty typu Boolean.

### 2.2 TYPY

INTEGER  Viz sekce 1.3.
REAL  Viz sekce 1.3.
CHAR  úplný rozšířený kód ASCII s 256 prvky.
BOOLEAN  (TRUE, FALSE). Tento typ je používán v logických operacích, obsahuje výsledky porovnávání.

### 2.3 PROCEDURY A FUNKCE

#### 2.3.1 VSTUPNÍ A VÝSTUPNÍ PROCEDURY

##### 2.3.1.1 WRITE

Procedura WRITE je používána pro výstup údajů na obrazovku, nebo na tiskárnu. Když je výraz, který má být napsán, jednoduchého typu character, potom WRITE (e) vyjádří 8 bitovou hodnotu jako znak e, podle okolností na tiskárně, nebo obrazovce.

Poznámka:

CHR(8) (CTRL H) udává destruktivní zpětný krok na obrazovce.
CHR(12) (CTRL L) maže obrazovku, nebo udává novou stránku na tiskárně.
CHR(13) (CTRL M) provádí návrat vozu a posun řádku.
CHR(16) (CTRL P) normálně nasměruje výstup na tiskárnu, je-li právě používaná obrazovka, nebo naopak.

Všeobecně:

```
WRITE (P1,P2,......Pn);  je ekvivalentní:
BEGIN WRITE (P1);WRITE(P2);......WRITE(Pn) END;
```

Parametry P1,P2,.....Pn mohou mít jednu z následujících podob:
(e) nebo (e:m) nebo (e:m:n) nebo (e:m:H)
kde e,m a n jsou výrazy a H je literál (konstanta).

Příklady:

1] e je typu integer: je používáno buďto (e), nebo (e:m). Hodnota integer výrazu e je přeměněna na znakový řetězec s následnými mezerami. Délka řetězce může být zvětšována (úvodními mezerami) použitím m, které specifikuje celkový počet znaků, které mají přijít na výstup. Jestliže m není postačující pro napsání e, nebo m není přítomno, potom je e napsáno cele s následnou mezerou a je ignorováno m. Všimněte si, že je-li m specifikováno jako délka e bez následné mezery, potom žádná následná mezera nepřijde na výstup.

2] e je typu integer a je použito tvaru (e:m:H). V tomto případě je e výstup v hexadecimální soustavě. Když m=1 nebo m=2, potom hodnota (e MOD 16 m) je výstup v délce m znaků. Když m=3, nebo m=4, potom výstup je plná hodnota e v hexadecimální soustavě. Když m>4, potom jsou vkládány úvodní mezery před plnou hexadecimální hodnotu e. Úvodní nuly budou vkládány tam, kde je to možné. Příklad:

```
 WRITE(1025:m:H);
m=1    vystup:  0
m=2    vystup:  01
m=3    vystup:  0401
m=4    vystup:  0401
m=5    vystup:   0401
```

3] e je typu real. Jsou použity tvary (e), (e:m) nebo (e:m:n). Hodnota e je přeměňována na znakový řetězec reprezentující reálné číslo. Formát je určen použitím n. Když n není použito, potom číslo znamená výstup ve vědecké notaci s mantisou a exponentem. Když je číslo záporné, potom znaménko '-' je ve výstupu udáno před mantisou, jinak je výstupem mezera. číslo vždy vystupuje nejméně s jedním decimálním, až do maxima 5 decimálních míst a exponent je vždy se znaménkem plus nebo minus. To znamená, že minimální šířka při exponenciálním zobrazení je 9 znaků. Je-li šířka m<9, potom na výstupu bude vždy plná šířka 12 znaků. Když m>=9, potom bude vystupovat jedno, nebo více desetinných míst, až do maxima 5 desetinných míst (m=12). Pro m>12 jsou před číslo vkládány mezery. Příklady:

```
 WRITE(-1.23E 10:m)
m=7   dává:  -1.2300E+10
m=8   dává:  -1.2E+10
m=9   dává:  -1.23E+10
m=10  dává:  -1.220E+10
m=11  dává:  -1.2300E+10
m=12  dává:  -1.23000E+10
m=13  dává:   -1.23000E+10
```

Když je použita forma (e:m:n), potom e bude vypsáno jako číslo ve tvaru s pevnou desetinnou tečkou s n desetinnými místy na výstupu. Před číslo nebudou zařazeny žádné mezery, ledaže by byla šířka pole m dostatečná. Je-li n=0, potom e vystupuje jako integer. Je-li e příliš velké pro výstup ve specifikované šířce pole, potom je výstup udán ve vědecké notaci s šířkou pole m (viz nahoře). Příklady:

```
WRITE(1E2:6:2)      dává:  100.00
WRITE(1E2:8:2)      dává:    100.00
WRITE(23.455:6:1)   dává:    23.5
WRITE(23.455:4:2)   dává:   2.34550E+01
WRITE(23.455:4:0)   dává:    23
```

4] e je typu char nebo typu string.
Může být použito buďto (e), nebo (e:m) a znak, nebo řetězec znaků, bude na výstupu v minimalní šíři pole 1 (pro znaky), nebo v délce řetězce (pro typ string). Je-li m dostatečně velké, jsou vpředu vkládány mezery.

5] e typu boolean
Může být použito buďto (e), nebo (e:m) a na výstupu bude buďto TRUE, nebo FALSE v závislosti na Booleovské hodnotě e, s použitím minimální šíře výstupního pole 4, respektive 5.

### 2.3.1.2 WRITELN

WRITELN výstupy jsou ukončeny newline. To je ekvivalentní WRITE (CHR(13)). Všimněte si, že je zde zahrnut posuv řádku.

```
WRITELN(P1,P2,......Pn);  je ekvivalentní:
BEGIN WRITE(P1,P2,......Pn);WRITELN END;
```

### 2.3.1.3 PAGE

Procedura PAGE je ekvivalentní WRITE(CHR(12)); vyčistí obrazovku a tiskárnu posune na vrchol nové stránky.

### 2.3.1.4 READ

Procedura READ je používána pro vstup dat z klávesnice. Děje se to prostřednictvím vyrovnávací paměti obsažené ve zpracovateli (runtimes) - bufer je zpočátku prázdný (s výjimkou znaku end of line). Můžeme brát v úvahu,že jakékoliv přístupy k tomuto buferu se provádí prostřednictvím textového okna nad vyrovnávací pamětí, čímž můžeme vždy najednou pozorovat jeden znak. Když je toto textové okno umístněno nad znakem end-of-line, potom před ukončením operace čtení bude do vyrovnávací paměti čten nový řádek textu z klávesnice. V průběhu čtení na tomto řádku jsou rozpoznávány všechny řídící kódy, detailně popsané v sekci 0.0.

```
READ(V1,V2,......Vn); je ekvivalentní:
BEGIN READ(V1); READ(V2); ......; READ(Vn) END;
```

kde V1, V2, atd. mohou být typu character, string, integer, nebo real.

Příkaz READ(V); má různé účinky v závislosti na typu V. Musí se zde brát v úvahu 4 případy:

1] V je typu character.
V tomto případě READ(V) jednoduše čte znak ze vstupní vyrovnávací paměti a přiřeďuje ho k V. Když je textové okno na vyrovnávací paměti umístněno na znaku řádku (CHR(13)), potom funkce EOLN vrátí hodnotu TRUE a je čten nový řádek textu z klávesnice. Když je nato provedena operace čtení, potom bude textové okno umístněno na počátku nového řádku.

Důležitá poznámka: Všimněte si, že EOLN je TRUE na počátku programu. To znamená, že jestliže první READ je typu charakter, tak hodnota CHR(13) bude vrácena a je následována čtením z klávesnice do nového řádku. Následující čtení typu character vrátí první znak z nového řádku za předpokladu, že není prázdný. Viz také dole proceduru READLN.

2] V je typu string.
řetězec znaků může být čten použitím READ a v tomto případě bude čtena série znaků až počet znaků vymezených řetězcem bude přečten, nebo EOLN = TRUE. Když řetězec není zaplněn čtením, to je konec řádku byl dosažen před přidělením celého řetězce, potom konec řetězce je zaplněn znaky CHR(0) - to umožňuje programátorovi vyhodnotit délku řetězce, který byl přečten.
Poznámka nahoře pro 1] plně platí i zde.

3] V je typu integer.
V tomto případě série znaků, reprezentující celé číslo, jak je to definováno v sekci 1.3, je čtena. Všechna úvodní prázdná místa a znaky end-of-line jsou přeskakovány. To znamená, že celá čísla mohou být okamžitě čtena (srovnej poznámku k odstavci 1] nahoře).

Když čtení celého čísla má absolutní hodnotu větší, než MAXINT (32767), potom bude vydáno chybové hlášení 'Number too large' a provádění bude ukončeno.
Když první znak po space a end-of-line není číslice, nebo znaménko ('+' nebo '-'), potom bude chybové hlášení 'Number expected' a program bude přerušen.

4] V je typu real.
V tomto případě je čtena série znaků, reprezentující reálné číslo v souhlase se syntaxí podle sekce 1.2.
Všechny úvodní mezery a znaky end-of-line jsou přeskočeny a stejně jako pro integer čísla shora, první znak potom musí být číslice, nebo znaménko. Jestliže čtené číslo je příliš veliké, nebo příliš malé (viz sekce 1.3), potom bude hlášena chyba 'Over- flow'. Když 'E' je přítomno bez následujícího znaménka, nebo číslice, potom vznikne chyba 'Exponent expected'. Když je přítomna desetinná tečka bez následjící číslice, potom bude chybové hlášení 'Number expected'.
Reálná čísla, stejně jako celá čísla mohou být čtena bezprostředně, viz odstavce 1] a 2].

2.3.1.5 READLN

READLN(V1,V2,.....Vn); je ekvivalentní :
BEGIN READ(V1,V2,.... ..Vn); READLN END;

READLN jednoduše čte do vyrovnávací paměti z klávesnice: vkládáte-li do bufferu, můžete použít řídící funkce popsané podrobně v sekci 0.0. Takto EOLN se stává FALSE po provedení READLN, pokud by následující řádek nebyl prázdný.
READLN může být použito k přeskočení prázdného řádku, přítomného na počátku provádění objektového kódu, to znamená, že vyvolá čtení do nové vyrovnávací paměti. To bude užitečné, když si přejete číst složku typu character na začátku programu, ale není to nutné, když právě čtete integer, nebo real číslo (protože znaky end-of-line jsou přeskočeny), nebo když právě čtete znaky z následujících řádků.

2.3.2 FUNKCE VSTUPU.

2.3.2.1 EOLN

Funkce EOLN je booleovskou funkcí, která vrací hodnotu TRUE když následující znak, který má být čten je znakem end-of-line (CHR(13)). V ostatních případech vrací hodnotu FALSE.

2.3.2.2 INCH

Funkce INCH vyvolá snímání klávesnice počítače a když byla stisknuta klávesa, vrací znak, reprezentovaný stisknutou klávesou. Když nebyla stisknuta žádná klávesa, potom vrací CHR(0). Tato funkce tedy vrací výsledek typu charakter.

2.3.3 FUNKCE PŘENOSU.

2.3.3.1 TRUNC(X)

Parametr X musí být typu real, nebo integer a hodnota vrácená TRUNC je největší celé číslo menší nebo rovné X když X je kladné nebo nejmenší celé číslo větší nebo rovné X, když X je záporné.
Příklady:
TRUNC(-1.5) vrací -1 TRUNC(1.9) vrací 1

2.3.3.2 ROUND(X)

X musí být typu real, nebo integer a tato funkce vrací 'nejbližší' celé číslo k X (podle standardních zaokrouhlovacích pravidel). Příklady:
ROUND(-6.5) vrací -6 ROUD(11.7) vrací 12
ROUND(-6.51) vrací -7 ROUND(23.5) vrací 24

2.3.3.3 ENTIER(X)

X musí být typu real nebo integer - ENTIER vrací největší celé číslo menší, nebo rovné X pro všechna X. Příklady:
ENTIER(-6.5) vrací -7 ENTIER(11.7) vrací 11
Poznámka: ENTIER není standardní Pascalovskou funkcí, ale je ekvivalentní Basicovské funkci INT. Funkce je užitečná při psaní rychlých programů pro mnoho matematických aplikací.

2.3.3.4. ORD(X)

X smí být skalárního typu s výjimkou real. Vrácená hodnota je celé číslo, reprezentující pořadí hodnoty X v množině definující typ X.
Je-li X typu integer, potom ORD(X)=X. Toho by se normálně mělo vyvarovat. Příklady:
ORD('a') vrací 97 ORD('@') vrací 64

2.3.3.5 CHR(X)

X musí být typu integer. CHR vrací znak odpovídající ASCII hodnotě X. Příklady:
CHR(49) vrací '1' CHR(91) vrací '['

2.3.4 ARITMETICKÉ FUNKCE

Ve všech funkcích této podsekce musí být parametr X typu real, nebo integer.

2.3.4.1 ABS(X)

Navrací absolutní hodnotu X (např. ABS(-4.5) vrací 4.5). Výsledek je stejného typu jako X.

2.3.4.2 SQR(X)

Navrací druhou mocninu X. Výsledek je stejného typu jako X.

2.3.4.3 SQRT(X)

Navrací druhou odmocninu X. Vrácená hodnota je vždy typu real. Je-li argument X záporný, je generováno chybové hlášení 'Maths Call Error'.

2.3.4.4 FRAC(X)

Navrací zlomkovou část X: FRAC(X) = X - ENTIER(X). Jako u funkce ENTIER je tato funkce užitečná k napsání mnoha rychlých matematických rutin. Příklady:

FRAC(1.5) vrací 0.5 FRAC(-12.56) vrací 0.44.

2.3.4.5 SIN(X)

Navrací sinus X, kde X je v radiánech. Výsledek je vždy typu real.

2.3.4.6 COS(X)

Navrací cosinus X, kde X je v radiánech. Výsledek je typu real.

2.3.4.7 TAN(X)

Navrací tangens X, kde X je v radiárech. Výsledek je vždy typu real.

2.3.4.8 ARCTAN(X)

Navrací úhel v radiánech, jehož tangens je dané číslo X. Výsledek je typu real.

2.3.4.9 EXP(X)

Navrací hodnotu e^X, kde e = 2.71828. Výsledek je vždy typu real.

2.3.4.10 LN(X)

Vrací přirozený logaritmus (t.j. se základem e) čísla X. Výsledek je vždy typu real. Je-li X <= 0, potom je generováno hlášení 'Maths Call Error'.

2.3.5 DALŠÍ PŘEDDEFINOVANÉ PROCEDURY

2.3.5.1 NEW(p)

Tato procedura NEW(p) umístňuje prostor pro dynamickou proměnnou. Proměnná p je ukazatelového typu, po provedení NEW(p) obsahuje p adresu nově umístněné dynamické proměnné. Typ dynamické proměnné je stejný, jako u pointer proměnné p a může být jakéhokoliv druhu.
K nalezení přístupu k dynamické proměnné se používá p^ - viz o tom příklady použití ukazatele k odkazu na dynamické proměnné v Dodatku 4.
Pro nové umístnění prostoru užitého pro dynamické proměnné používejte procedury MARK a RELEASE (viz níže).

2.3.5.2 MARK(vi)

Tato procedura uchovává adresu posledního obsazeného bytu dynamické proměnné, uložené v proměnné typu ukazatel vi. Stav může být obnoven po provedení procedury MARK pomocí procedury RELEASE (viz níže).

Typ proměnné na kterou vi odkazuje nerozhoduje, protože vi může být použito pouze ve spojení s MARK a RELEASE, nikdy s NEW.
Ukázkový program s použitím MARK a RELEASE viz v dodatku 4.

2.3.5.3 RELEASE(v1)

Tato procedura uvolňuje prostor v paměti pro použití dynamických proměnných. Obsah této části paměti může být znovu obnoven po použití MARK(v1) - takto se účinně zruší všechny dynamické proměnné vytvořené procedurou MARK. Proto by mělo být této procedury používáno velmi opatrně.
Viz nahoře a další detaily v Dodatku 4.

2.3.5.4 INLINE(C1,C2,C3,.....)

Tato procedura umožňuje vložit do programu v jazyce Pascal strojový kód Z80; hodnoty (C1 MOD 256, C2 MOD 256, C3 MOD 256, ....) jsou vkládány do výsledného programu při běžném umístňování čítačové adresy, podržované překladačem. C1, C2, C3 atd. jsou integer konstanty a může jich být libovolný počet. Příklad použití INLINE viz Dodatek 4.

2.3.5.5 USER(V)

USER je procedura s jedním integer argumentem V. Volá pamětovou adresu V. Protože Hisoft Pascal 4 obsahuje integer čísla ve formě dvojkového doplňku (viz Doplněk 2), potom k odkazu na adresy větší než '7FFF (32767) musí být použito záporných hodnot V. Například 'C000 je -16384 a tedy USER(-16384); bude volat paměťovou adresu 'C000. Pokud se k odkazu na paměťovou adresu používá konstanta, je vhodnější používat hexadecimální soustavu.
Vyvolávaná rutina by měla končit Z80 instrukcí RET ('C9) a musí zachovat registr IX.

2.3.5.6 HALT

Tato procedura zastaví provádění programu se zprávou 'Halt at PC=XXXX' kde XXXX je hexadecimální adresa paměti, na které je povel HALT. Společně s listingem kompilace může být použito HALT k trasování programu.
Tak se to obvykle použije během ladění programu.

2.3.5.7 POKE(X,V)

POKE ukládá výraz V do paměti počítače počínaje adresou X. X je typu integer a může být jakéhokoliv druhu mimo SET. Viz Sekce 2.3.5.5 nahoře, kde je vysvětleno používání integer čísel k vyjádření paměťové adresy. Příklady:
POKE('6000,'A') umístní '41 na adresu '6000.
POKE(-16384,3.6E3) umístní 00 0B 80 70 (v hex.) od adresy 'C000.

2.3.5.8 TOUT(NAME,START,SIZE)

TOUT je procedura, použitá k uložení proměnných na pásku. První parametr je typu ARRAY[1..8] OF CHAR a je to jméno souboru dat, který má být uložen. SIZE bytes jsou vypisovány z paměti od adresy START. Oba tyto parametry jsou typu integer.
Například k uložení proměnné V na pásku pod jménem 'VAR
užijte: TOUT('VAR ',ADDR(V),SIZE(V))

Používání aktuálních adres paměti dává uživateli možnost pružnější všestrannosti,než pouze schopnosti ukládat řetězce. Například má-li systém zamapovanou obrazovkovou paměť, mohou se přímo ukládat plné obrazovky. Viz příklad použití TOUT v Dodatku 4.

### 2.3.5.9 TIN(NAME,START)

Tato procedura se používá ke čtení proměnných a pod. z pásky, uložených pomocí TOUT. NAME je typu ARRAY[1..8] OF CHAR a START je typu integer. Na pásce je hledán soubor NAME, který je vložen od paměťové adresy START. Počet bytů k vložení je brán z pásky a je uložen na pásce prostřednictvím TOUT.
Např. k načtení proměnné uložené v příkladu v sekci 2.3.5.8 nahoře je třeba použít: TIN('VAR ',ADDR(V)).
Protože zdrojové soubory dat jsou ukládány editorem s použitím stejného formátu jako u TIN a TOUT, TIN se může použít k načtení textových souborů do ARRAYs OF CHAR pro další zpracování.
Viz HP4T Průvodce změnami.
Příklady použití procedury TIN viz Dodatek 4.

### 2.3.5.10 OUT(P,C)

Tato procedura se používá k přímému přístupu k výstupnímu portu Z80 bez použití procedury INLINE. Hodnota integer parametru P je vkládána do BC registru, znakový parametr C je vložen do registru A a je vykonána assemblerovská instrukce OUT (C),A.
Například: OUT(1,'A') dává na Z80 výstupním portu 1 znak 'A'.

## 2.3.6 DALŠÍ PŘEDDEFINOVANÉ FUNKCE

### 2.3.6.1 RANDOM

Tato funkce vrací pseudonáhodné číslo mezi 0 a 255 včetně. Ačkoliv je tato rutina velmi rychlá, dává chabé výsledky při opakovaném použití v cyklech, které neobsahují I/O operace.

Když jsou třeba lepší výsledky než umožňuje tato funkce, potom by měla být napsána rutina v Pascalu, nebo ve strojovém kódu, přizpůsobená k danému účelu.

### 2.3.6.2 SUCC(X)

X může být jakéhokoliv skalárního typu s vyjímkou real a SUCC (X) vrací následovníka X.

Např.: SUCC('A') vrací 'B', SUCC('5') vrací '6'.

### 2.3.6.3 PRED(X)

X může být jakéhokoliv skalárního typu s vyjímkou real a PRED (X) vrací předchůdce X.

Např.: PRED('j') vrací 'i', PRED(TRUE) vrací FALSE

2.3.6.4 ODD(X)

X musí být typu integer, ODD vrací booleovský výsledek TRUE, je li X liché a FALSE, je-li sudé.

2.3.6.5 ADDR(V)

Tato funkce bere identifikátor proměnné jakéhokoliv typu za paramer a vrací integer adresu proměnné s identifikátorem V. Pro informaci jak jsou proměnné udržovány v runtime v Hisoft Pascal 4 viz Dodatek 3. Příklad použití ADDR viz Dodatek 4.

2.3.6.6 PEEK(X,T)

První parametr této funkce je typu integer. Používá se k vymezení paměťové adresy (viz sekce 2.3.5.5). Druhý argument je typ, to je výsledný typ funkce.
PEEK je používán k vracení údaje z paměti počítače a výsledek může být jakéhokoliv typu.
Ve všech procedurách PEEK a POKE ( opak PEEK) jsou data uváděna ve vnitřní reprezentaci Hisoft Pascal 4, detaily viz Dodatek 3.
Například: když paměť obsahuje od adresy `5000 hodnoty 50 61 73 63 61 6C (hex.) potom:
WRITE(PEEK(`5000,ARRAY[1..6] OF CHAR)) dává 'Pascal'
WRITE(PEEK(`5000,CHAR)) dává 'P'
WRITE(PEEK(`5000,INTEGER)) dává 24912
WRITE(PEEK(`5000,REAL) dává 2.46227E+29
Pro další detaily o vnitřní reprezentaci typu v Hisoft Pascal 4 viz Dodatek 3.

2.3.6.7 SIZE(V)

Parametr této funkce je nějaká proměnná. Integer výsledek je rozsah uložení v bytech, kterou zaujímá uvedená proměnná.

2.3.6.8 INP(P)

INP je používáno k přímému přístupu k vstupnímu portu Z80 bez použití procedury INLINE. Hodnota integer parametru je vkládána do BC registru a znakový výsledek funkce je potom ve střádači, takže se provádí Z80 instrukce IN A,(BC).

## SEKCE 3 KOMENTÁŘE A MOŽNOSTI VOLBY U PŘEKLADAČE

### 3.1 KOMENTÁŘE

Komentář se může vyskytnout mezi dvěma vyhrazenými slovy, čísly identifikátory nebo speciálními symboly - viz Dodatek 2. Komentář začíná znakem '{' a nebo dvojicí znaku '(*'. Když následujícím znakem není '$', jsou potom všechny znaky ignorovány až do dalšího znaku '}' a nebo až do dalšího znakového páru '*)'. Když bylo nalezeno '$', potom překladač vyhledává sérii volitelných možností překladače (viz dole), po nichž jsou ostatní znaky přeskočeny až po '}', nebo '*)'.

## 3.2 MOŽNOSTI VOLBY U PŘEKLADAČE

Syntaxe pro specifikaci volitelných možností překladače:

```
                                  __>|+|__
                                  |_|    |
__>|$|__>|písmeno|_________|_______>|_____________>
   |_|   |_______|         |       |
         |                 |__>|-|__|
         |                         |_|
         |___________|,|<____________________|
                     |_|
```

Jsou k dispozici následující volitelné možnosti:

VOLBA L:

Ovládá řízení listingu programového textu a adres object kódu překladačem.
Když je vloženo L+, potom je žádan úplný listing.
Když je vloženo L-, potom jsou řádky listovány, když je nalezena chyba.
IMPLICITNĚ: L+

VOLBA O:

Kontroluje provádění kontroly přeplnění. Násobení a dělení celými čísly a všechny aritmetické operace s reálnými čísly jsou VŽDY kontrolovány na přeplnění.
Když je zadáno O+, potom je kontrola prováděna při sečítání a odčítání celých čísel.
Když je zadáno O-, potom zhora uvedené kontroly nejsou prováděny.
IMPLICITNĚ: O+

VOLBA C:

Zapíná, nebo vypíná čtení klávesnice během provádění object code programu. Je-li C+, potom při stisknutí CC se provádění přeruší se zprávou HALT. Viz sekce 2.3.5.6.
Tato kontrola se provádí na počátku všech cyklů, všech procedur a všech funkcí. Toho může uživatel použít při ladění programu ke zjištění, který cyklus (procedura, funkce, ...) se správně neukončuje. Tento proces nemůžete použít, pokud si přejete rychlý průběh cílového programu.
Je-li zadáno C-, potom zhora uvedená kontrola není prováděna.
IMPLICITNĚ: C+

VOLBA S:

Určuje provádění zásobníkových kontrol.
Objeví-li se S+, potom na začátku každého vyvolání procedury nebo funkce je prováděna kontrola, zda by v tomto bloku mohlo dojít k přeplnění. Když během pracovní doby dojde k přeplnění dynamické proměnné, nebo je přeplněn program, potom je zobrazena zpráva 'Out of RAM at PC=XXXX' a provádění je přerušeno.

Přirozeně to není jednoduché; má-li procedura velký rozsah používání zásobníku, potom může dojít k 'crash' programu. Naopak, používá li funkce málo zásobníku při použití rekurzí, potom je možné, že funkce bude nesprávně zastavována.

Když je zadáno S-, kontroly zásobníku nejsou prováděny.
IMPLICITNĚ: S+

VOLBA A:

Zajišťuje kontroly umístnění indexů řetezce v mezích daných deklarací ARRAYs.
Je-li zadáno A+ a je-li index řetězce příliš vysoko, nebo nízko, potom je generována zpráva 'Index too hig', nebo 'Index too low' a provádění programu bude zastaveno.
Když je zadáno A-, potom se tyto kontroly neprovádí.
IMPLICITNĚ: A+

VOLBA I:

Při použití 16 bitového dvojkového doplňku při celočíselné aritmetice, dochází k přeplnění při provádění operací >,<,>= nebo <= když se od sebe argumenty odlišují o více než MAXINT (32767). Stane-li se to, bude výsledek porovnávání nesprávný. Za normálních okolností to nezpůsobí těžkosti, ale když by si uživatel přál porovnavat takova čísla, použití I+ zajišťuje správnost výsledku porovnávání. Ekvivalentní situace může nastat u aritmetiky reálnych čísel, kdy na základě přeplnění bude generováno chybové hlášení, pokud se argumenty liší o více než cca 3.4E38; tomu se ale nelze vyhnout.
Pokud se zadá I-, potom nebude prováděna kontrola výsledku shora uvedeného porovnávání.
IMPLICITNĚ: I-

VOLBA P:

Při použití P se mění zařízení, ke kterému je vysílán kompilační listing, t.j. byla-li používána video obrazovka, je použita tiskárna a naopak. Všimněte si, že tato volba není následována '+' nebo '-'.
IMPLICITNĚ: Je použita video obrazovka.

VOLBA F:

Tento znak volby musí následovat mezera a potom osm znaků, zadávajících název souboru. Pokud má název souboru méně než 8 znaků, musí být doplněn mezerami.
Při této volbě dojde k vložení Pascalovského zdrojového textu ze specifikovaného souboru z konce běžného řádku, což je užitečné, přeje-li si programátor vybudovat knihovnu velmi používaných procedur a funkcí na pásce a potom chce tyto procedury a funkce vkládat do jednotlivých programů.
Program by měl být ukládán použitím editorového povelu 'P'. Na největším počtu systémů by měla být použita volba listingu L-, jinak bude kompilátor pracovat pomalu.

Příklad: {$L-,F MATRIX vložit textový soubor MATRIX};
Při psaní velmi rozsáhlých programů se může stát, že nebude v paměti počítače dostatek prostoru pro zdrojový text i cílový kód současně. Je však možné překládat programy uložené po částech na pásku za použití volby 'F'; potom pouze 128 bytes zdrojového textu je v RAM v libovolné době a je ponecháváno mnohem více místa pro cílový kód.
Tato volba není implementovaná ve verzi ZX Spectrum.
(Pozn. překladatele: HP4S verze 1.5 již tuto možnst má, text ale musí být ukládán editorovým příkazem 'W'. Viz úvodní poznámky v této příručce.)

Možnosti volby u překladače mohou být používány výběrově. Tak mohou být odladěné úseky urychlovány a zhušťovány vypnutím relevantních kontrol, zatím co jsou zapnuty na neodladěných místech kódu.

## SEKCE 4 INTEGRÁLNÍ EDITOR

### 4.1 ÚVOD DO EDITORU

Editor je součástí všech verzí Hisoft Pascal 4T. Je to jednoduchý řádkový editor, fungující na všech operačních systémech se Z80, při zachování snadné použitelnosti a schopnosti editovat programy rychle a účinně.
Text je v paměti ukládán ve zhuštěné formě. Je ukládán počet úvodních mezer, jeden znak na začátku řádku a všechna HP4T rezervovaná slova jsou symbolicky zkracována do jednoho znaku. To potom vede k typickému omezení rozsahu textu o 25%.
UPOZORNĚNÍ: Na všech místech v této sekci klávesa DELETE je uváděna místo kontrolního kódu CH, neboť je to přirozenější postup.
Do editoru se vstoupí automaticky po nahrání HP4T z pásky a objeví se zpráva:

Copyright Hisoft 1992
All right reserved

což je následováno editorovou nápovědou '>'.
Jako odpověď na tento znak můžete zadat povelový řádek následujícího formátu:

C N1,N2,S1,S2

následovaný RETURN, kde:

- C je povel, který má být proveden (viz Sekce 4.2 níže).
- N1 je číslo v rozsahu 1 - 32767 včetně.
- N2 je číslo v rozsahu 1 - 32767 včetně.
- S1 je znakový řetězec s maximální délkou 20.
- S2 je znakový řetězec s maximální délkou 20.

čárka se používá k oddělení dvou různých argumentů, (i když to je možné změnit - viz povel 'S') a mezery jsou ignorovány s výjimkou mezer uvnitř řetězců. žádný z argumentů není nezbytný, i když některé z povelů (např. povel 'D'elete) neproběhne, dokud nebude určeno N1 a N2. Editor si pamatuje předchozí čísla a řetězce, které jste do něj vložili a používá tyto původní hodnoty tam, kde je to možné, pokud výslovně nevyznačíte zvláštní argument uvnitř příkazového řádku. Hodnoty N1 a N2 jsou původně nastaveny na 10 a řetězce jsou zpočátku prázdné.

Když zadáte ne přípustný příkazový řádek, jako např. F-1,100,HELLO potom bude tento řádek ignorován a bude zobrazena zpráva 'Pardon?' a musíte potom znovu natypovat správně řádek, tedy zde F1,100,HELLO. Toto chybové hlášení bude také zobrazeno, pokud délka S2 překračuje 20; když délka u S1 je větší než 20, potom jsou všechny přebytečné znaky ignorovány.

Povely mohou být vkládány velkými, nebo malými znaky.
Když vkládáte povelový řádek, mohou být použity všechny řídící znaky uvedené v sekci 0.0. Například CX k odstranění počátku řádku.

Následující sub-sekce rozvádí podrobně různé povely, dosažitelné v editoru. Povšimněte si, že kdykoliv je argument uzavřen symboly '<>', potom zde argument musí být přítomen, aby tento povel mohl proběhnout.
POZOR: V implementaci pro ZX Spectrum odpovídá klávese RETURN klávesa ENTER! Viz Implementační poznámky.

## 4.2 EDITOROVÉ POVELY.

### 4.2.1 VKLÁDÁNÍ TEXTU.

Text může být vkládán do textového souboru buďto natypováním čísla řádku, mezera a potom požadovaný text, nebo pomocí povelu 'I'. Všimněte si, že když zadáte číslo řádku následované RETURN, t.j. bez textu, potom existuje-li, bude tento řádek vymazán z textu. Kdykoliv je právě text vkládán, potom řídící funkce CX (vymaž řádek od začátku), funkce CI (přejdi na následující pozici tabulátoru), funkce CC (návrat do příkazové smyčky) a funkce CP (přepnout tiskárnu) mohou býti použity. Klávesa DELETE (nebo BACKSPACE) vyvolají destruktivní zpětný krok, ale pouze v rozsahu textového řádku. Text je vkládán do vnitřní vyrovnávací paměti v HP4T a když je tento buffer naplněn, nelze do něj vkládat další text. Potom musíte použít DELETE nebo CX k uvolnění prostoru vyrovnávací paměti.

POVEL: I n,m

Použití tohoto povelu umožňuje vkládání v automatickém Insert módu. řádky jsou automaticky číslovány počínaje n a s krokem m. Jakmile se na obrazovce zobrazí číslo řádku, vkládejte požadovaný text a používejte při tom různé řídící znaky podle potřeby a textovou řádku ukončujte RETURN. K zrušení tohoto módu použijte řídící funkci CC (viz sekce 0.0 a také Implementační poznámky).
Když vkládáte řádek s již existujícím číslem, potom bude původní řádek vymazán a nahrazen nově vkládaným řádkem když stisknete RETURN. Když automatické řádkování vytvoří větší číslo řádku než 32767, potom se automaticky zruší Insert mód.
Když se při vkládání textu dostanete až ke konci obrazovkového řádku aniž jste zadal 128 znaků (což je rozsah vyrovnávací paměti), potom se provede scroll obrazovky a můžete pokračovat ve vkládání na dalším řádku a k textu bude přidáno automatické vložení nového obrazovkového řádku tak, že čísla řádků budou efektivně oddělena od textu.

## 4.2.2 LISTING TEXTU.

Text je možné prohlížet použitím povelu 'L'; počet řádků, které jsou ve kterémkoliv okamžiku zobrazovány během provádění tohoto povelu je pevně předem stanoven (viz vaše Implementační poznámky), ale může být měněn použitím povelu 'K'.

POVEL: L n,m

Tento povel zaznamenává běžný text na displej od čísla řádku n až do čísla m včetně. Implicitní hodnota pro n je VŽDY 1 a implicitní hodnota pro m je VŽDY 32767, t.zn. že implicitní hodnoty nejsou brány z předtím vložených argumentů. Pro listing celého textového souboru použijte jednoduše 'L' bez argumentu. Obrazovkové řádky jsou formátovány pomocí levého okraje tak, že číslo řádku je přehledně zobrazováno. Počet zobrazovaných obrazovkových řádků může být řízen použitím povelu 'K'. Po vylistování určitého počtu řádků se listing přeruší; pokud to ještě není na čísle řádku m, kontrolní funkce CC navrací řízení editoru, nebo stiskem libovolné klávesy pokračuje listing).

POVEL: Kn

'K' ukládá počet obrazovkových řádků, které mají být zobrazeny najednou, jak je to shora popsáno u 'L'. Hodnota (n MOD 256) je vypočtena a uložena do zásobníku. Například použijte K5, když si přejete u následujícího listingu vypisovat 5 obrazovkových řádků najednou.

## 4.2.3 EDITAČE TEXTU.

Jakmile již byl vytvořen nějaký text, potom se nevyhnutelně vyskytne potřeba některé řádky editovat. Jsou zde zajištěny různé povely, aby řádky mohly být opravovány, vymazávány, posunovány a nebo přečíslovány.

POVEL: D<n,m>

Všechny řádky od n do m včetně mohou být vymazány z textového souboru. Je-li m<n, nebo jestliže nejsou specifikovány oba argumenty, potom není povel vykonán; je to proto, aby se vyloučily omyly z přehlédnutí. Jednotlivý řádek může být vymazán tak, že se zadá m=n. Může se toho také dosáhnout jednoduchým natypováním čísla řádku následované RETURN.

POVEL: Mn,m

Tento povel umožňuje vymazat jakýkoliv text z řádku m a nahradit jej textem z řádku n. Všimněte si, že text na řádku n je ponechán. Tento povel tedy provádí 'M'ove textového řádku do další polohy v textovém souboru. Pokud číslo řádku n neexistuje, povel se nevykoná.

POVEL: N<n,m>

Použití povelu 'N' umožňuje znovu přečíslovat textový soubor s číslem prvního řádku n s krokem m. Musí být zadáno n i m; pokud by přečíslování vedlo k číslu řádku vyššímu než je 22767, potom zůstane zachováno původní číslování řádků.

POVEL: Fn,m,f,s

Je prohledáván textový soubor v rozmezí řádků n<x<m a je hledán řetězec f ('find' řetězec). Je-li tento řetězec zjištěn, potom je zobrazen odpovídající textový řádek a přejde se do Edit módu viz dále. Potom můžete použít povely, které jsou přístupné v módu Edit k nalezení následujících výskytů řetězce f v již definovaném rozsahu řádků a nebo k jeho nahrazení řetězcem s ('substitue' string) a potom k hledání dalšího nejbližšího výskytu f; viz další podrobnosti níže.
Všimněte si, že rozsah řádku a dva řetězce již mohly být zadány dříve nějakým jiným povelem, takže potom je pouze zapotřebí zadat 'F' k započetí vyhledávání - viz příklad v sekci 4.3 k objasnění.

POVEL: En

Editovat řádek n. Pokud řádek n neexistuje, potom se příkaz nevykoná; pokud tomu tak není, řádek n se opíše do vyrovnávací paměti a potom je zobrazen s řádkovým číslem. číslo řádku je opětovně zobrazeno dole pod řádkem a přejde se do Edit módu. Veškeré potom následující redakční úpravy jsou prováděny ve vyrovnávací paměti, nikoliv v textu samotném. Takto může být kdykoliv získán původní textový řádek.
V tomto módu je kurzor zobrazen, jako by procházel řádkem (počínaje prvním znakem) a dále jsou umožňovány různé doplňkové povely, umožňující editaci řádku. Doplňkové povely jsou:
' ' (mezera) - posun kurzoru na následující znak v řádku. Nemůžete jej posunout za konec tohoto řádku.
DELETE (nebo BACKSPACE) - dekrementace kurzoru na předchozí znak řádku. Nemůžete jej posunout před začátek tohoto řádku.
CI (kontrolní funkce) - posunutí kurzoru na následující polohu tabulátoru, ale nikoliv za konec řádku.
RETURN - konec redakční úpravy tohoto řádku s ponecháním všech provedených změn.

Q - ukončit redakční úpravu řádku, t.j. učiněné změny se neberou v úvahu a ponechá se původní podoba řádku (před započetím redakční úpravy).
R - znovu naplnění editační vyrovhávací paměti z textu, t.j. zrušení všech provedených redakčních změn a obnovení původní podoby řádku.
L - listing zbytku editovaného řádku, t.j. zbytek řádku, který je mimo rozsah běžné polohy kurzoru. Zůstáváte v Edit modu s kurzorem zpět na začátku řádku.
K - kill (odstranění) znaku na běžné poloze kurzoru.
Z - vymazání všech znaků od běžné polohy kurzoru (včetne) až na konec řádku.

F - hledání nejbližšího výskytu stringu 'find' definovaného dříve (viz povel 'F' nahoře). Tento doplňkový povel automaticky ukončí editaci na běžném řádku (zachovává ale změny) pokud nenalezne další výskyt řetězce 'find' na běžném řádku. Pokud je zjištěn výskyt řetězce 'find' v nejblíže následujícím řádku (v již dříve specifikovaném rozsahu řádků), potom bude zapnut Edit mód na řádek, ve kterém je řetězec nalezen. Všimněte si, že po úspěšném vyhledávání je kurzor vždy umístňován na začátek řádku.
S - nahrazení běžně nalezeného výskytu řetězce 'find' dříve definovaným řetězcem 'substitue' a potom provedení doplňkového povelu 'F', t.j. hledání dalšího nejbližšího výskytu řetězce 'find'. Tak společně se zhorauvedeným doplňkovým povelem 'F' je 'S' používáno ke krokům textovým souborem s možností nahradit výskyt řetězce 'find' řetězcem 'substitue' - příklad viz sekce 4.3.
**Zvláštní poznámka** V běžné verzi HP4T existuje známá skrytá porucha v doplňkové operaci 'S' - tento doplňkový povel by měl být používán bezprostředně po povelu 'F', nebo v doplňkovém povelu 'F' a nebo bezprostředně po doplňkovém povelu 'S'. V praxi by to ale nemělo působit žádné potíže.
I - (insert) vsouvání znaku na běžné poloze kurzoru. V tomto doplňkovém módu zůstanete až do stisknutí RETURN - to vás vrátí do hlavního módu Edit s kurzorem v poloze po vsunutí posledního znaku. Použití DELETE (nebo BACKSPACE) v tomto doplňkovém módu vymaže znak vlevo od kurzoru z vyrovnávací paměti, zatímco použití CI (kontrolní funkce) posune kurzor vpřed k další poloze tabulátoru a vsune mezery.
X - posune kurzor vpřed na konec řádku a automaticky vloží doplňkový mód, popsaný shora.
C - mění doplňkový mód. To vám umožňuje přepsat znak na běžné pozici kurzoru a potom se kurzor posune o jednu pozici vpřed. Vy zůstáváte ve změněném doplňkovém módu, dokud nestisknete RETURN, čímž se vrátíte zpět do redakčního módu s kurzorem v pozici za posledním znakem, který jste změnili. DELETE (nebo BACKSPACE) v tomto doplňkovém módu dekrementuje kurzor o jednu pozici v levo, zatímco povel CY zde nemá žádný účinek.

- 4.2.4 POVELY PRO PÁSKU

Texty mohou být ukládány na pásku, nebo z pásky přehrávány použitím povelu 'P' a 'G'.

POVEL: Pn,m,s

řádky z rozsahu definovaného nerovností n<x<m jsou ukládány na pásku ve formátu HP4T pod souborovým jménem specifikovaným řetězcem s. Nezapomeňte, že tyto argumenty mohou být nastaveny dřívějším povelem. Před vložením tohoto povelu si zajistěte, aby byl váš magnetofon připojen a nastaven na záznam. Zatímco je text nahráván, je na obrazovce zobrazena zpráva 'Busy..'.

POVEL: G,,s

Na pásce je vyhledáván soubor ve formátu HP4T se jménem s. Během vyhledávání je na obrazovce zobrazena zpráva 'Busy..'. Je-li nalezen platný soubor HP4T, ale má chybné označení souboru, potom je na obrazovce zobrazena zpráva 'Found' (nalezeno) následo-

vaná názvem nalezeného souboru a ve vyhledávání se pokračuje. Je li jednou nalezen soubor se správným jménem, potom se zobrazí 'Found' a soubor je vkládán do paměti. Když je během čtení zjištěna chyba, potom se zobrazí 'Checksum error' a čtení je přerušeno. Pokud se tak stane, musíte převinout pásku zpět, stisknout PLAY a natypovat znovu 'G'.
Pokud je řetězec s prázdný, potom je načten první HP4T soubor z pásky bez ohledu na jeho název.
Během vyhledávání souboru můžete přerušit vkládání podržením CC dolů, to přeruší proces čtení a vrátí hlavní editorový mód.
Všimněte si, že je-li již přítomen nějaký textový soubor, potom z pásky vkládaný textový soubor je přidán k již existujícímu a potom bude celý soubor přečíslován počínaje řádkem číslo 1 s krokem 1.

## 4.2.5 PŘEKLAD A BĚH PROSTŘEDNICTVÍM EDITORU

POVEL: Cn

Spouští překlad textu počínaje řádkem n. Když neuvedete výslovně číslo řádku, potom bude text překládán počínaje prvním existujícím řádkem. Pokud se týče dalších možných podrobností, viz sekci 0.2.

POVEL: R

Předem přeložený cílový kód bude proveden, ale pouze tehdy, pokud zdroj nebyl mezi tím rozšířen. Detaily viz sekce 0.2.

POVEL: Tn

To je povel 'T'ranslate (překládat). Běžný zdroj je překládán od řádku n (nebo od začátku pokud je n vynecháno) a, je-li překládání úspěšné, dostanete připomínkový dotaz 'Ok?': a pokud odpovíte 'Y' na tento dotaz, potom bude cílový kód produkovaný překladem přesunut na konec runtimes (destruuje překladač) a potom budou runtimes a cílový kód vyslány na pásku s názvem souboru odpovídajícím předešle definovanému řetězci "find".
Vy potom můžete později vkládat soubor do paměti za použití HP4T loaderu, načež se bude automaticky provádět cílový program. Všimněte si, že cílový kód je umístněn na konci a posouván ke konci runtimes tak, že po překládání budete muset znovu vložit překladač, nicméně to nebude působit žádné problémy, protože asi budete překládat plně funkční program.
Pokud se rozhodnete nepokračovat nahráním na pásku, potom pouze natypujte jiný znak než 'Y' v odpověď na dotaz 'Ok?'; řízení bude vráceno do editoru, který bude ještě bezvadně fungovat, neboť cílovým kódem nebylo pohnuto z místa.

## 4.2.6 DALŠÍ POVELY

POVEL: B

Tento povel jednoduše vrací řízení do operačního systému. Pro detaily jak se vrátit do překladače nahlédněte do HP4T Poznámky o změnách a do vašich Implementačních poznámek.

POVEL: On,m

Mějte na mysli, že text uchovávaný v paměti je ve formě tokens (symbolické zkrácení) a přitom jsou úvodní mezery zhuštěny do jediného znaku a veškerá rezervovaná slova HP4T jsou omezena na jediný znakový symbol. Máte-li nějak uchován text v paměti, snad od jiného editoru, který nebyl symbolicky zkrácen, potom můžete použít povel 'O' aby vám provedl tokenizaci. Text je přečítán do vyrovnávací paměti v rozšířené formě a potom je vkládán zpět do souboru v tokenizované formě. Provedení ovšem nějakou dobu potrvá. Zvlášť musí být specifikován řádkový rozsah, nebo budou brány v úvahu dříve zadané hodnoty.

POVEL: S,,d

Tento povel vám umožňuje změnit znak pro separátor, který se používá k oddělování argumentů v povelovém řádku. Při vstupu do editoru je brána jako separátor čárka ','. To může být změněno povelem 'S' na první znak specifikovaného řetězce d. Uvědomte si, že pokud jste jednou definovali separátor, musí být používán i v povelu 'S' až do specifikace jiného separátoru.
Všimněte si, že separátorem nesmí být mezera.

## 4.2 PŘÍKLAD POUŽITÍ EDITORU

Předpokládejme, že jste natypovali následující program (s použitím I10,10):

```
 10 PROGRAM BUBLESORT
 20 CONST
 30   Size = 2000;
 40 VAR
 50   Number : ARRAY [1..Size] OF INTEGER:
 60   I, Temp : INTEGER;
 70 BEGIN
 80   FOR I:=1 TO Size DO Number[I] := RANDOM;
 90   REPEAT
100     FOR I:=1 TO Size DO
110     Noswaps := TRUE;
120     IF Number[I] > Number[I+1] THEN
130       BEGIN
140         Temp := Number[I];
150         Number[I] := Number[I+1];
160         Number[I+1] := Temp;
170         Noswaps := FALSE
180       END
190   UNTIL Noswapss
200 END.
```

Tento program má řadu chyb, které jsou dále uvedeny:
řádek 10 chybí středník
řádek 30 nejde o skutečnou chybu, ale dejme tomu, že chcete rozměr 100
řádek 100 rozměr by zde měl být Size-1
řádek 110 tento řádek by měl být na řádku 95 namísto zde
řádek 190 Noswapss má být správně Noswaps

Rovněž proměnná Numbers byla popsána, ale veškeré odkazy jsou na Number. Konečně proměnná Noswaps (typu BOOLEAN) nebyla deklarována.
Aby se to všechno dalo do pořádku, může se postupovat následovně:

| | |
|---|---|
| F60,200,Number,Numbers | a potom opakovaně doplňkový povel 'S' |
| E10 | potom sekvence X;RETURN:RETURN |
| E30 | potom K C 1 RETURN RETURN |
| F100,100,Size,Size-1 | následováno doplňkovým povelem 'S' |
| M110,95,<br>110 | následováno RETURN |
| E190 | potom X DELETE RETURN RETURN |
| 65 Noswaps : BOOLEAN; | |
| N10,10 | přečíslování s krokem 10. |

Velmi důrazně vám doporučujeme, aby jste stále postupovali shora uvedeným postupem při použití editoru. Z počátku vám to může připadat těžkopádné, pokud jste byli zvyklí používat obrazovkový editor, ale přizpůsobení se na to vám nebude dlouho trvat.

## DODATEK 1: CHYBY

### A.1.1 KÓD CHYB, GENEROVANÝCH KOMPILEREM

1. číslo je příliš veliké.
2. Očekává se středník.
3. Nedeklarovaný identifikátor.
4. Je očekáván identifikátor.
5. Použijte '=' místo ':=' v deklaraci konstanty.
6. Očekává se '='.
7. Tento identifikátor nemůže zahájit příkaz.
8. Očekává se ':='.
9. Očekává se ')'.
10. Chybný typ.
11. Očekává se '.'.
12. Očekává se faktor.
13. Očekává se konstanta.
14. Tento identifikátor není konstanta.
15. Očekává se 'THEN'.
16. Očekává se 'DO'.
17. Očekává se 'TO', nebo 'DOWNTO'.
18. Očekává se '('.
19. Nelze psát tento typ výrazu.
20. Očekává se 'OF'.
21. Očekává se ','.
22. Očekává se ':'.

23. Očekává se 'PROGRAM'.
24. Očekává se proměnná, neboť parametrem je proměnná parametru
25. Očekává se 'BEGIN'.
26. Očekává se proměnná při vyvolání READ.
27. Nelze porovnávat výrazy tohoto typu.
28. Může být buďto typu INTEGER, nebo REAL.
29. Nelze číst tento typ proměnné.
30. Tento identifikátor není typem.
31. Očekává se exponent v reálném čísle.
32. Očekává se skalární (nikoliv numerický) výraz.
33. Nejsou povoleny nulové řetězce (použijte CHR(0)).
34. Očekává se '['.
35. Očekává se ']'.
36. Index u Array musí být typu scalar.
37. Očekává se '..'.
38. Očekává se ']', nebo ',' v deklaraci ARRAY.
39. Dolní mez je větší než horní mez.
40. Soustava je příliš veliká (více než 256 možných prvků).
41. Funkční výsledek musí být identifikátor typu.
42. V sestavě se očekává ',', nebo ']'.
43. V sestavě se očekává '..', nebo ']'.
44. Typ parametru musí být typu identifikátor.
45. Nulová sestava nemůže být prvním faktorem v nepřiřazovacím příkazu.
46. Očekává se skalár (včetně real).
47. Očekává se skalár (nikoliv real).
48. Sestavy nejsou kompatibilní.
49. Nemůže být použito '<' a '>' k porovnávání sestav.
50. Očekává se 'FORWARD', 'LABEL', 'CONST', 'VAR', 'TYPE', nebo 'BEGIN'.
51. Očekává se hexadecimální číslice.
52. Není možné POKE sestavy.
53. Array je příliš veliké (> 64 K).
54. V definici RECORD se očekává 'END', nebo ';'.
55. Očekává se identifikátor pole.
56. Očekává se proměnná po 'WITH'.
57. Proměnná ve 'WITH' musí být typu RECORD.
58. Identifikátor pole nebyl přiřazen příkazu WITH.
59. Po 'LABEL' je očekáváno celé číslo bez znaménka.
60. Po 'GOTO' je očekáváno celé číslo bez znaménka.
61. Toto návěstí je na nesprávné úrovni.
62. Nedeklarované návěstí.
63. Parametr u SIZE by měla být proměnná.
64. Může být použit pouze test ekvivalence pro ukazatele.
67. Jediný záznamový parametr pro celá čísla sé dvěma ':' je e:m:H.
68. řetězce nesmí obsahovat znaky 'end of line'
69. Parametr u NEW, MARK, nebo RELEASE by měla být proměnná typu pointer.
70. Parametr u ADDR by měla být proměnná.

## A.1.2 CHYBOVÁ HLÁŠENÍ U RUNTIME.

Když je detekován omyl v runtime, potom je zobrazana jedna z následujících zpráv, následovaná 'at PC=XXXX', kde XXXX je adresa paměti v místě, kde došlo k chybě.

často bude zdroj chyby zcela zřejmý, pokud ne, obraťte se s dotazem na kompilační listing, abyste zjistili skutečný výskyt chyby v programu a použijte k tomu křížovou referenci XXXX. Příležitostně to ale nedává správný výsledek.

1. Halt
2. Overflow (přeplnění)
3. Out of RAM (mimo rozsah RAM)
4. / by zero (dělení nulou) - také vyvoláváno DIV.
5. Index too low (index příliš nízký)
6. Index too high (index příliš vysoký)
7. Maths Call Error (vyvolána matematická chyba)
8. Number too large (číslo příliš vysoké)
9. Number expected (očekáváno číslo)
10. Line too long (řádek příliš dlouhý)
11. Exponent expected (očekáván exponent)

Chyby, které se vyskytnou během runtime způsobí zastavení chodu programu.

## DODATEK 2 REZERVOVANÁ SLOVA A PŘEDDEFINOVANÉ IDENTIFIKÁTORY.

### A 2.1 REZERVOVANÁ SLOVA

| | | | | | | |
|---|---|---|---|---|---|---|
| AND | ARRAY | BEGIN | CASE | CONST | DIV | DO |
| DOWNTO | ELSE | END | FORWARD | FUNCTION | GOTO | IF |
| IN | LABEL | MOD | NIL | NOT | OF | OR |
| PACKED | PROCEDURE | PROGRAM | RECORD | REPEAT | SET | THEN |
| TO | TYPE | UNTIL | VAR | WHILE | WITH | |

### A 2.2 SPECIÁLNÍ SYMBOLY

Následující symboly jsou používány v Hisoft Pascal 4 s vyhraženým významem:

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

### A 2.3 PŘEDDEFINOVANÉ IDENTIFIKÁTORY

Následující entity mohou být použity v deklaracích v celém programovém bloku a jsou k dispozici v celém programu dokud nejsou opětovně deklarovány programátorem v rozsahu vnitřního bloku. Pro další informaci viz sekce 2.

```
CONST     MAXINT=32767;

TYPE      BOOLEAN=(FALSE,TRUE);
          CHAR {Rozšířený soubor znaků ASCII};
          INTEGER=-MAXINT..MAXINT;
          REAL {Podmnožina reálných čísel. Viz sekce 1.3};
```

PROCEDURE WRITE;WRITELN;READ;READLN;PAGE;HALT;USER;POKE;
INLINE;OUT;NEW;MARK;RELEASE;TIN;TOUT;

FUNCTION ABS;SQR;ODD;RANDOM;ORD;SUCC;PRED;INCH;EOLN;PEEK;CHR;
SQRT;ENTIER;ROUND;TRUNC;FRAC;SIN;COS;TAN;ARCTAN;EXP;
LN;ADDR;SIZE;INP;

## DODATEK 3 REPREZENTACE A UKLÁDÁNÍ DAT.

### A 3.1 REPREZENTACE DAT.

V následujícím je dopodrobna diskutována vnitřní reprezentace dat v Hisoft Pascal 4.

Informace o způsobu ukládání je potřebná v každém případě a měla by být užitečná pro většinu programátorů (může být používána funkce SIZE - viz sekce 2.3.6.7); další podrobnosti mohou být užitečné pro ty, kteří se snaží sloučit programy v Pascalu se strojovými programy.

#### A 3.1.1 CELÁ ČÍSLA.

Celá čísla jsou ukládána každé ve 2 bytes, ve formě dvojkového doplňku. Příklady:

```
   1  ...  `0001
 256  ...  `0100
-256  ...  `FF00
```

K podržení celých čísel se standardně používá Z80 registrový HL pár.

#### A 3.1.2 ZNAKY, BOOLEAN A DALŠÍ SKALÁRY.

Tyto při ukládání zabírají každý jeden byte, v čistém pojetí, dvojkově bez znaménka.
Znaky: 8 bitů, je užito rozšířeného ASCII.

```
'E'  ...  `45
'[' ...  `5B
```

Booleans:
ORD(TRUE)=1 tedy TRUE je reprezentováno 1.
ORD(FALSE)=0 tedy FALSE je reprezentováno 0.

Kompiler používá pro shorauvedené standardně Z80 registr A.

#### A 3.1.3 REÁLNÁ ČÍSLA.

Forma (mantisy, exponentu) je užívána podobně jako u standardní vědecké notace, pouze se používá dvojkové, místo desítkové soustavy. Příklady:

```
2  ...  2 * 10^0  nebo  1.0B * 2^1
1  ...  1 * 10^0  nebo  1.0B * 2^0
```

```
-12.5  ...  -1.25 * 10^1  nebo  -25 * 2^-1
                                ...  -11001B * 2^-1
                                ...  -1.1001B * 2^3 normalizovaně

0.1  ...  1.0 * 10^-1  nebo  1/10  ... 1/1010B  ...  0.1B/101B
          takže nyní musíme provádět dlouhé binární dělení...
                                 0.0001100
                               ___________________
                        101 ! 0.1000000000000000
                                101
                                -----
                                 110
                                 101
                                 -----
                                   1000
                                    101  v tomto okamžiku je
                                    ---  již vidět, že se
                                         zlomek periodicky
                                         opakuje.
                 =   0.1B/101B = 0.00011000B

                                1.10011000B * 2^-4
                              ---------------------
```

Jak tedy použijeme hořejších výsledků ke znázornění těchto čísel v počítači? Nuže, nejprve rezervujeme 4 bytes pro uložení každého reálného čísla v následujícím formátu:

```
ZNAMÉNKO    NORMALIZOVANÁ MANTISA    EXPONENT    data

   23            22            0      7     0    bit
!_______________________!!_____________________________!
        H    L                   E    D        registr
```

znaménko: znaménko mantisy; 1=záporné, 0=kladné.
normalizovaná mantisa: mantisa je normalizovaná do tvaru 1.xxxxxx s horním bitem (bit 22) je vždy 1, s vyjímkou zobrazení nuly (HL=0 DE=0).
exponent: exponent v binárním doplňkovém tvaru.

Takto:

```
  2  ... 0 1000000 00000000 00000000 00000001 (`40,`00,`00,`01)
  1  ... 0 1000000 00000000 00000000 00000000 (`40,`00,`00,`00)
-12.5 .. 1 1100100 00000000 00000000 00000011 (`E4,`00,`00,`03)
  0.1 .. 0 1100110 01100110 01100110 11111100 (`66,`66,`66,`FC)
```

Tedy, máme-li na paměti, že HL a DE jsou používány k uchovávání reálných čísel, potom bychom měli do registrů vkládat následovně abychom reprezentovali každé shorauvedené číslo:

```
2  ...  LD  HL,`4000
        LD  DE,`0100

1  ...  LD  HL,`4000
        LD  DE,`0000
```

```
-12.5  ...  LD  HL,`E400
            LD  DE,`0300

  0.1  ...  LD  HL,`6666
            LD  DE,`FC66
```

Poslední příklad nám ukazuje, proč výpočty s dvojkovými zlomky mohou být nepřesné; 0.1 nemůže být přesně reprezentováno jako dvojkový zlomek, k finitnímu číslu s desetinnými místy.
Všimněte si dobře, že reálná čísla jsou ukládána do paměti v pořadí ED LH.

A 3.1.4 ZÁZNAMY A ŘETĚZCE.

Záznamy potřebují stejný úložný prostor jako je celkový součet pamětového prostoru jejich složek.
Uspořádání: je-li n = počtu prvků v uspořádání a
s = rozměru každého prvku,
potom počet bytes obsazených uspořádáním je n*s.
Př.:
ARRAY[1..10] OF INTEGER vyžaduje 10*2 = 20 bytes
ARRAY[2..12,1..10] OF CHAR má 11*10 = 110 prvků a vyžaduje tedy 110 bytes.

A 3.1.5 SOUSTAVY.

Soustavy jsou ukládány jako řetězce bitů, tak jestliže má základní typ má n prvků, potom počet použitých bytes je:
(n-1)DIV 8+1. Příklady:
SET OF CHAR vyžaduje (256-1)DIV 8+1 = 32 bytes
SET OF (modrá,zelená,žlutá) vyžaduje (3-1)DIV 8+1 = 1 byte.

A 3.1.6 UKAZATELÉ.

Ukazatelé zabírají 2 bytes, které obsahují adresu (ve formátu Intel, t.j. spodní byte je první) proměnné na kterou ukazují.

A 3.2 UKLÁDÁNÍ PROMĚNNÉ BĚHEM CHODU.

Existují 3 případy, o kterých uživatel potřebuje informaci jak jsou proměnné ukládány při chodu programu.
a. Globální proměnné - deklarované v hlavním programovém bloku.
b. Lokální proměnné - deklarované ve vnitřním bloku.
c. Parametry a hodnoty vracené funkcemi - jsou předávány do a z procedur a funkcí.
Tyto jednotlivé případy jsou podrobně rozebrány dále a příklad použití těchto informací jsou v Dodatku 4.

GLOBÁLNÍ PROMĚNNÉ
Globální proměnné jsou uloženy od vrcholu zásobníku proměnných dolů, například:

```
VAR   i: INTEGER;
      ch: CHAR;
      x: REAL;
```

potom:
i (které zabírá 2 bytes - viz předchozí sekce) bude uloženo na adresách B000-2 a `B000-1, t.j. na adr. `AFFE a `AFFF.

ch (1 byte) bude uloženo na adrese 'AFFE-1, t.j. na 'AFFD.
x (4 bytes) bude uloženo na adresách 'AFF9, 'AFFA, 'AFFB a 'AFFC.

LOKÁLNÍ PROMĚNNÉ

Lokální proměnné nejsou tak snadno přístupné přes zásobník, místo toho je registr IX nastaven na počátku každého vnitřního bloku tak, že (IX-4) ukazuje na počáteční adresu bloku lokálních proměnných. Například:

```
PROCEDURE    test;
VAR          i,j: INTEGER;
```

potom:
i (integer, tedy zabírá 2 bytes) bude umístněno na adresách IX-4-2 a IX-4-1, tedy IX-6 a IX-5.
j bude umístněno na adresách IX-8 a IX-7.

PARAMETRY A HODNOTY VRÁCENÉ FUNKCEMI

Hodnoty parametrů jsou zpracovávány jako s lokálními proměnnými a jako ty, čím dříve je parametr popsán, tím vyšší adresu má v paměti. Na rozdíl od proměnných je ale pevně stanovena nejnižší (nikoliv nejvyšší) adresa jako (IX+2). Například:

```
PROCEDURE  test(i: REAL; j: INTEGER);
```

potom:
j (umístněno první) je na adrese IX+2 a IX+3.
i je na adrese IX+4, IX+5, IX+6 a IX+7.

Proměnné parametry jsou zpracovávány přesně tak, jako hodnotové parametry, s výjimkou toho, že jsou vždy ukládány jako bytes a tyto 2 bytes obsahují adresu proměnné. Například:

```
PROCEDURE test(i: INTEGER; VAR x: REAL);
```

potom
odkaz na x je umístněn na adrese IX+2 a IX+3. Zde je adresa, kde je uloženo x. Hodnota i je uložena na adrese IX+4 a IX+5.

Funkční hodnoty jsou ukládány nad prvním parametrem v paměti. Například:

```
FUNCTION test(i: INTEGER): REAL;
```

potom
i je na adresách IX+2 a IX+3 a mezera je rezervována pro vracenou funkční hodnotu na adresách IX+4, IX+5, IX+6 a IX+7.

DODATEK 4 - NĚKOLIK PŘÍKLADŮ HP4T PROGRAMU.

Následující programy byste si měli pečlivě prostudovat, pokud máte nějaké nejasnosti při programování v Hisoft Pascal 4T.

```
 10 {Program pro ilustraci použití TIN a TOUT.
 20  Program vytváří velmi jednoduchý telefonní
 30  seznam na pásku a potom jej zpětně čte.
 40  Musíte napsat nějaký požadavek na vyhledání.}
 50
 60 PROGRAM TAPE
 80 CONST
```

```
 90   Size=10;    {Pozor, 'Size' je psáno velkými
100               a malými písmeny, nikoliv 'SIZE' !}
120 TYPE
130   Entry = RECORD
140             Name : ARRAY [1..10] OF CHAR;
150             Number : ARRAY [1..10] OF CHAR;
160           END;
180 VAR
190   Directory : ARRAY [1..Size] OF CHAR;
200   I : INTEGER;
210
220 BEGIN
240 {Vytvoření seznamu..}
250
260   FOR I:= 1 TO Size DO
270   BEGIN
280     WITH Directory[I] DO
290     BEGIN
300       WRITE('Prosim jmeno');
310       READLN;
320       READ(Name);
330       WRITELN;
340       WRITE('Cislo prosim');
350       READLN;
360       READ(Number);
370       WRITELN
380     END
390   END;
400
410 {K uložení seznamu na pásek se použije..}
430   TOUT('Seznam',ADDR(Directory),SIZE(Directory))
440
450 {Nové čtení pole zpět následovně..}
460
470   TIN('Seznam',ADDR(Directory))
480
490 {A nyní již můžete zpracovávat adresář podle přání....}
500 END
```

```
 10 {Program pro výpis řádků souboru v obráceném pořadí.
 20 Ukazuje použití ukazatelů, záznamů, MARK a RELEASE.}
 30 .
 40 PROGRAM Reverseline;
 50
 60 TYPE elem=RECORD            {Tvoří strukturu řádků výpisu}
 70            next: ^elem;
 80            ch: CHAR
 90           END;
100      link=^elem;
110
120 VAR prev,cur,hep: link;    {všechny ukazatele na 'elem'}
130
140 BEGIN
150  REPEAT                    {provést několikrát}
160    MARK(heap)              {přiřadit vrchol k 'heap'.}
170    prev:=NIL;          {neukazuje zatím na žádnou proměnnou.}
180    WHILE NOT EOLN DO
```

```
190        BEGIN
200          NEW(cur);              {vytvořit nový dynamický záznam}
210          READ(cur^.ch);         {a přiřadit jeho pole jednomu
220                                  znaku ze souboru.}
230          cur^.next:=prev;       {tento ukazatel pole adresuje}
240          prev:=cur              {předchozí záznam.}
250        END;
260
270 {Vypíše řádek odzadu prohlížením záznamů
280 vytvořených pozpátku.}
290
300    cur:=prev;
310    WHILE cur <> NIL DO        {NIL je první}
320      BEGIN
330          WRITE(cur^.ch);      {Vypsat toto pole, t.j. znak}
340          cur:=cur^.next       {Adresovat předchozí pole.}
350      END;
360    WRITELN;
370    RELEASE(heap);        {Uvolnit prostor dynamické proměnné.}
380    READLN                    {Počkat na další řádek.}
390  UNTIL FALSE                {Použijte CC k výstupu z programu}
400 END.
```

```
 10 {Program ukazuje použití rekurze}
 20
 30 PROGRAM FACTOR;
 40
 50 {Tento program počítá faktoriál čísla, zadaného z
 60 klávesnice 1) použitím rekurze a 2) použitím iterace.}
 70
 80 TYPE
 90   POSINT = 0..MAXINT;
100
110 VAR
120   METHOD : CHAR;
130   NUMBER : POSINT;
140
150 {Rekurzivní algoritmus.}
160
170 FUNCTION RFAC(N : POSINT) : INTEGER;
180
190  VAR F : POSINT;
200
210  BEGIN
220    IF N>1 THEN F:= N * RFAC(N-1)     {RFAC vyvoláno N krát.}
230           ELSE F:= 1;
240    RFAC := F
250  END;
260
270 {Iterační řešení.}
280
290 FUNCTION IFAC(N : POSINT) : INTEGER;
300
310   VAR I,F: POSINT;
320   BEGIN
330    F := 1;
340    FOR I := 2 TO N DO F := F*I;      {Jednoduchá smyčka.}
```

```
350     IFAC:=F
360    END;
370
380 BEGIN
390    REPEAT
400      WRITE('Zadejte metodu (I nebo R) a číslo   ');
410      READLN;
420      READ(METHOD,NUMBER);
430      IF METHOD = 'R'
440           THEN WRITELN(NUMBER,'! = ',RFAC(NUMBER))
450           ELSE WRITELN(NUMBER,'! = ',IFAC(NUMBER))
460    UNTIL NUMBER=0
470 END.
```

```
 10 {Program pro demonstraci modifikace
 20  Pascalovských proměnných použitím strojového kódu.
 30  Ukazuje PEEK, POKE, ADDR a INLINE.}
 40
 50 PROGRAM divmult2;
 60
 70 VAR r:REAL;
 80
 90 FUNCTION divby2(x:REAL):REAL;  {Funkce dělení 2 ..
100                                 .. rychle.}
110 VAR i:INTEGER;
120 BEGIN
130  i:=ADDR(x)+1;                 {Ukazuje na exponent u x}
140  POKE(i,PRED(PEEK(i,CHAR)));   {Dekrementuje exponent u x.
150                                 viz Dodatek 3.1.3.}
160  divby2:=x
170 END;
180
190 FUNCTION multby2(x:REAL):REAL; {Funkce násobení 2 ..
200                                 .. rychle.}
210 BEGIN
220  INLINE(`DD,`34,3);            {INC (IX+3) - exponent u x
230                                 - viz Dodatek 3.2.}
240  multby2:=x
250 END;
260
270 BEGIN
280  REPEAT
290   WRITE('Zadej číslo r ');
300   READ(r);                     {Není zapotřebí READLN, viz
310                                 Sekce 2.3.1. }
320
330   WRITELN('r děleno dvěma je ',divby2(r):7:2);
340   WRITELN('r násobeno dvěma je ',multby2(r):7:2);
350  UNTIL r=0
360 END.
```