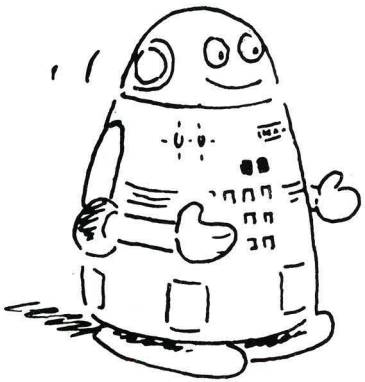


INTEGROVANÝ SYSTÉM PRO PROGRAMOVÁNÍ
V JAZYCE PASCAL PRO MIKROPOČÍTAČ

ONDRA PASCAL '87

-Uživatelská příručka-

Ø.Ø SLOVO NA ÚVOD.

==========

Pascal je vyšší programovací jazyk pro obecné použití. Tento jazyk byl navržen profesorem Niklausem Wirthem z Technical University Zurich. Jazyk byl pojmenován podle známého matematika a filozofa sedmnáctého století Blaise Pascala. Wirthova definice jazyka byla uveřejněna v roce 1971. Pascal byl navržen především pro výuku systematického přístupu k programování a pro úvod do strukturovaného programování. Během krátké doby se začal Pascal používat nejen pro účely výuky, ale také v běžné praxi. V současné době patří Pascal k nejpoužívanějším programovacím jazykům. Díky relativně snadné implementaci se Pascal rozšířil i na mikropočítačích nižší třídy (bez diskové paměti) a tak máte i vy možnost pracovat na mikropočítači ONDRA s překladačem tohoto jazyka.

Tato verze je jednou z minimálních implementací jazyka. Blok standardních podprogramů má velikost 4KB, editor 4KB a překladač 1ØKB. Všechny části jsou realizovány v jazyce symbolických adres mikroprocesoru Z8Ø.

Překladač generuje absolutní, velmi rychlý strojový kód. Příkazem 'T' v editoru (Kapitola 4) lze přeložený program nahrát spolu se standardními podprogramy do souboru na magnetofon a potom ho používat zcela samostatně.

Pokud zjistíte chyby v činnosti překladače, sdělte je prosím na jednu z následujících adres:

Jenne Daniel
U Trojice 23
České Budějovice 37ØØ1

Centrum pro mládež, vědu a techniku ÚV SSM

nám. M. Gorkého 24
Praha 1 11647

Ø.1 ROZSAH PŘÍRUČKY.

Tento návod není učebnicí Pascalu. Jste-li nováček v programováni v jazyce Pascal vyhledejte si nejprve nějakou učebnici Pascalu, např. literaturu uvedenou v příloze F.

Kapitola 1 obsahuje syntaxi a sémantiku jazyka.

Kapitola 2 detailně popisuje různé předdefinované identifikátory

Kapitola 3 obsahuje informace o řízení činnosti překladače

Kapitola 4 ukazuje použití řádkového vestavěného editoru

Příloha A obsahuje chybová hlášení.

Příloha B obsahuje seznam rezervovaných slov.

Příloha C obsahuje seznam předdefinovaných identifikátorů.

Příloha D obsahuje podrobnosti o vnitřní reprezentaci důležité pro programátory, kteří chtějí pracovat i na úrovni strojového kódu.

Příloha E udává některé příklady programů v jazyce Pascal.

Ø.2 ODLIŠNOSTI JAZYKA.

Typ FILE není implementován.

Typ RECORD nemá variabilní část. Identifikátory použité jako položky záznamu, nemají lokální charakter. Nelze tedy použít stejný identifikátor pro položku záznamu a pro normální proměnnou. Dále při přístupu k položkám záznamu není prováděna kontrola zda daný záznam tuto položku obsahuje. Pokud tedy například nadefinujeme:

```
TYPE
A = RECORD
R : REAL
END;
B=  RECORD
IL,IH : INTEGER
END;

VAR
X : A;
```

potom X.R je číslo typu real a X.IL a X.IH jsou typu integer. Toto umožňuje v podstatě využívat možností, které jinak dovoluje tzv. 'nehlídaná' variabilní část záznamu.

Procedury a funkce nehohou být předávány jako parametry.

Příkaz CASE je doplněn částí ELSE (ve FEL Pascalu OTHERWISE).

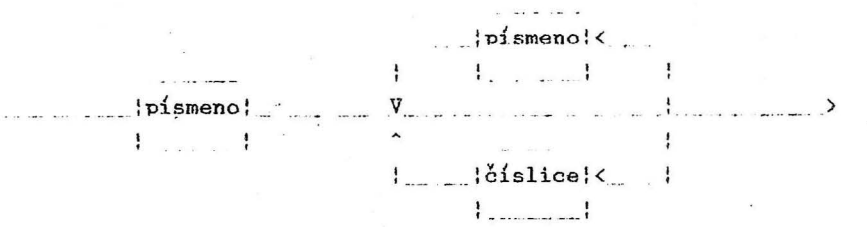
Bázový typ množiny může obsahovat až 256 prvků.

Řídící proměnná cyklu FOR musí být nestrukturovaná a nesmí to být parametr.

Příkazem GOTO lze provádět skoky jen na téže úrovni.

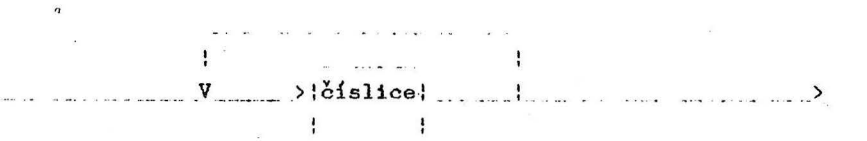
Příkaz PROGRAM nemá parametry.

1.1 IDENTIFIKÁTOR.

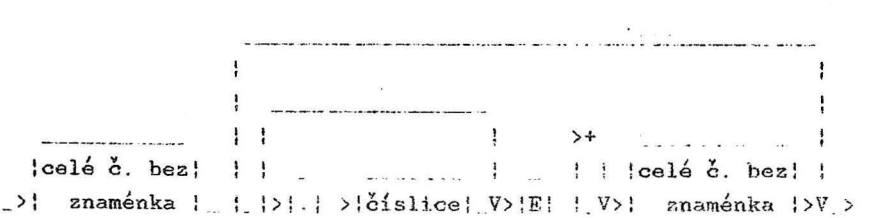


Pouze prvních 10 znaků identifikátoru je významných. Identifikátory mohou být složeny z velkých nebo malých písmen, takže identifikátory HELLO, HELlo a hello jsou odlišné. Rezervovaná slova a předdefinované identifikátory musí být zadávány velkými písmeny!

1.2 CELÉ ČÍSLO BEZ ZNAMÉNKA.



1.3 ČÍSLO BEZ ZNAMÉNKA.

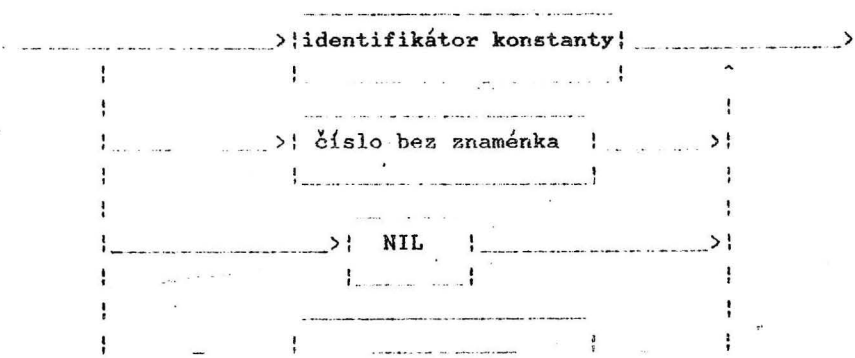


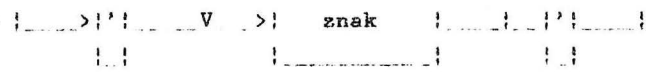
Celá čísla mají absolutní hodnotu menší nebo rovnou 32767. Větší celá čísla jsou zpracovávána jako reálná.
Mantisa reálných čísel je 23 bitová. Přesnost při použití reálných čísel je asi 7 platných míst.

Největší zobrazitelné reálne číslo je 3.4E38, nejmenší 5.9E-39. Nemá smysl pro specifikaci reálných čísel používat více než 7 číslic v mantise, protože další místa jsou ignorována.

Vyžadujete-li přesnost neuvádějte počáteční nuly, protože ty se také počítají jako číslice. Např. Ø.ØØØ123456 je reprezentováno méně přesně, než 1.23456E-4.
Hexadecimální čísla se používají především při práci s pamětí (zápis adres), nebo v návaznosti na programy ve strojovém kódu. Všimněte si, že po '#' musí být alespoň jedna hexadecimální číslice, jinak je hlášena chyba č. 51.

1.4 KONSTANTA BEZ ZNAMÉNKA.

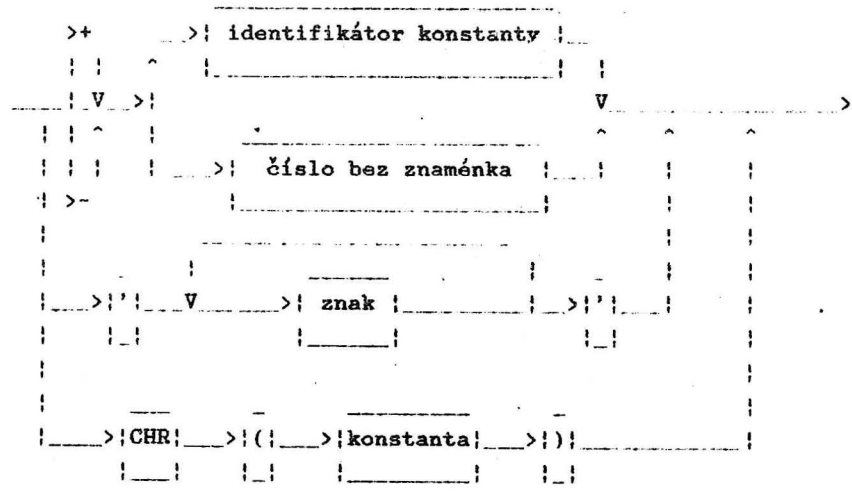




Řetězce nesmí obsahovat více než 255 znaků. Typ řetězu je: ARRAY [1..N] OF CHAR kde N je celé číslo mezi 1 az 255 včetně. Řetězcový literál nesmí obsahovat znak NEW LINE (konec řádky), jinak je vyvolána chyba č. 68.

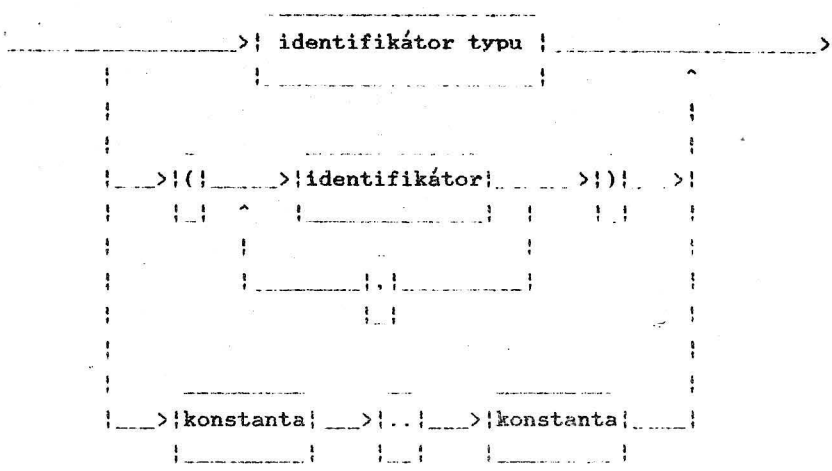
Používané znaky jsou rozšířenou množinou ASCII s 256 prvky. Pro zachování kompatibility se standardním Pascalem není prázdný znak reprezentován jako '' , ale musí se použít zápis CHR(Ø).

1.5 KONSTANTA.



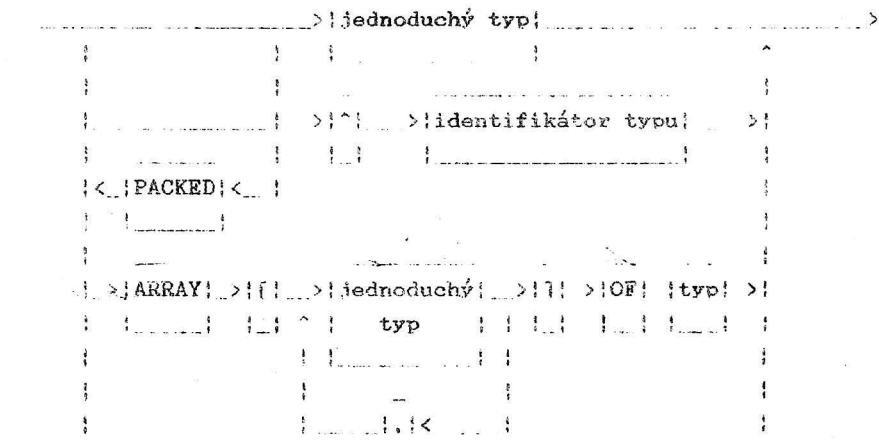
Nestandardní konstrukce CHR je zařazena pro deklaraci kontrolních znaků. Konstanta v závorkách musí být typu INTEGER.
Příklad: CONST bs=CHR(1Ø);
cr=CHR(13);

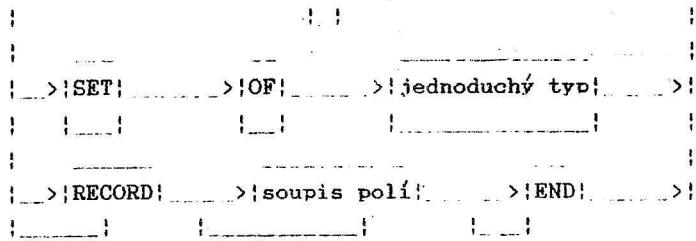
1.6 JEDNODUCHÝ TYP.



Skalární typy (identifikátor, identifikátor.....) nesmějí obsahovat víc než 256 prvků.

1.7 TYP.





Konstrukce s klíčovým slovem PACKED jsou v syntaxi povoleny, ale nemají žádný semantický účinek.

Základní typ množiny smí mít až 256 prvků. (Což například umožňuje definici typu SET OF CHAR).

Jsou přípustné pole polí, pole množin, záznamy množin atd.

Dva typy ARRAY jsou zpracovávány jako ekvivalentní, pokud jejich definice vychází z téhož použití rezervovaného slova ARRAY. T.j. následující typy NEJSOU shodné (ani kompatibilní).
TYPE tablea = ARRAY[1..100] OF INTEGER; tableb = ARRAY[1..100] OF INTEGER;
Proměnná definovaná typem tablea nesmí být přiřazena k proměnné typu tableb.

Zhora uvedené omezení neplatí pro speciální případ pole řetězcového typu (ARRAY OF CHAR).

Ukazatele na typy, které nebyly deklarovány, nejsou dovoleny. Definice typu však může obsahovat proměnnou typu ukazatel na sebe sama.
Např.:

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

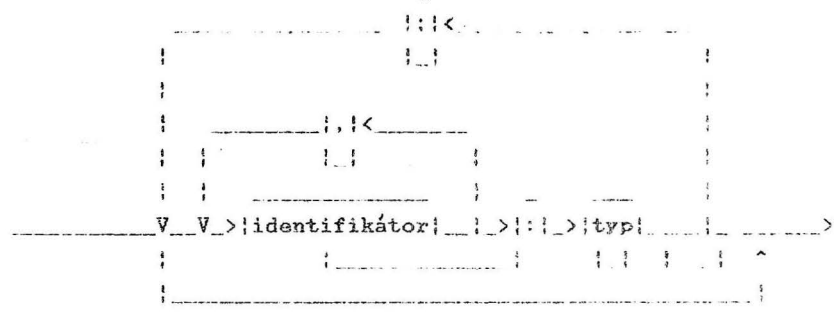
Ukazatele na ukazatele nejsou přípustné.
Ukazatele na ten samý typ jsou shodného typu.
Např.:

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

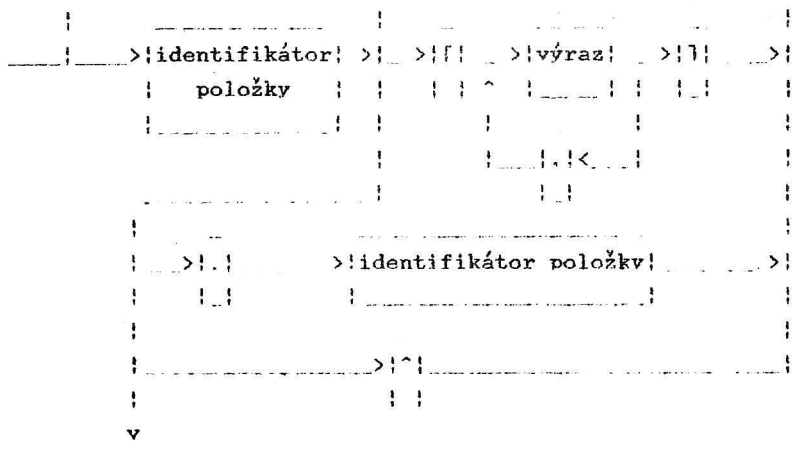
Proměnné first a current jsou shodného typu. Což znamená, že mohou být jedna ke druhé přičleněny, nebo srovnávány.
Variabilní část záznamu není povolena.

Dva typy záznam jsou ekvivalentní pouze tehdy, když jejich deklarace vychází z jednoho použití slova RECORD.

1.8 SEZNAM POLOŽEK.

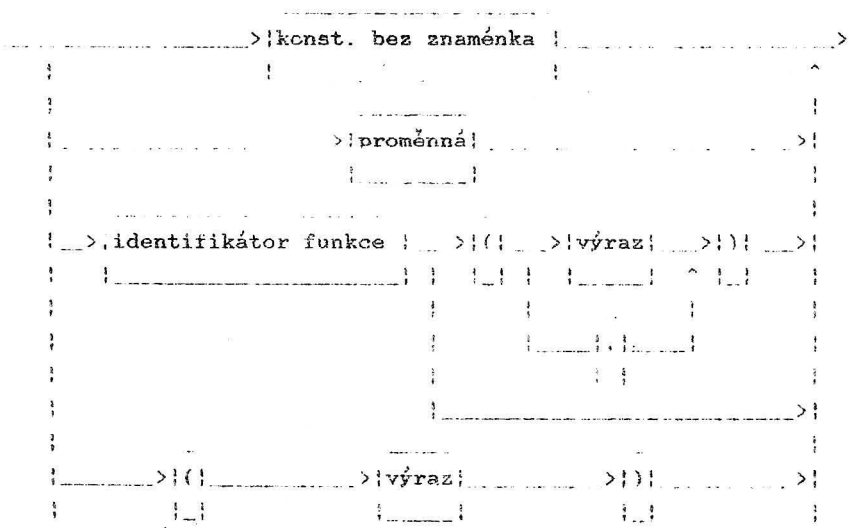


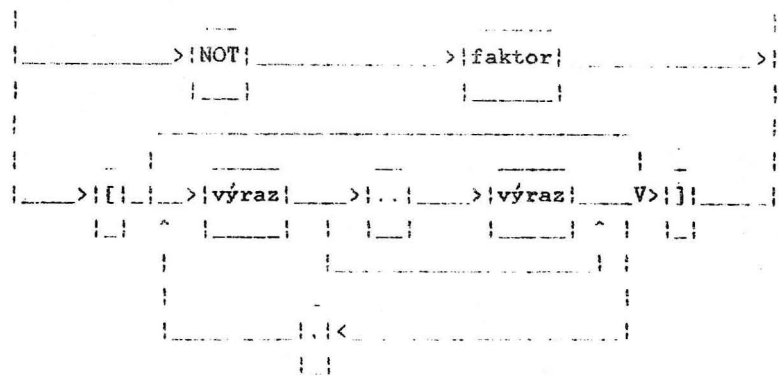
1.9 PROMĚNNÁ.



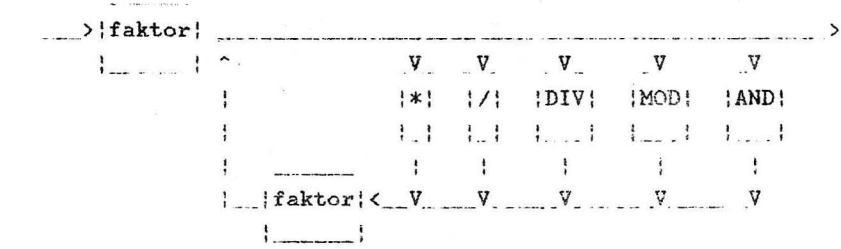
Při specifikaci prvků mnohorozměrných polí není programátor nucen používat tutéž formu indexové specifikace k odkazům jako při deklaraci. Například pokud je proměnná deklarovaná jako ARRAY[1..10] OF ARRAY[1..10] OF INTEGER, pak pro přístup k prvku (1,1) tohoto pole lze použít buď a[1][1] nebo a[1,1].

1.10 FAKTOR.



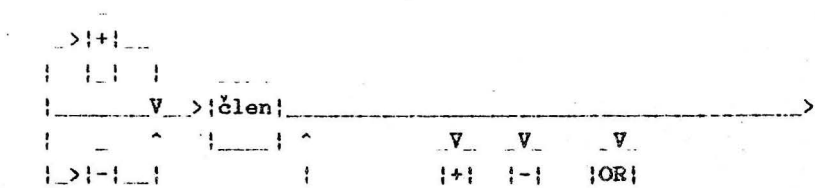


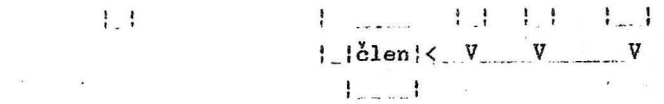
1.11 ČLEN.



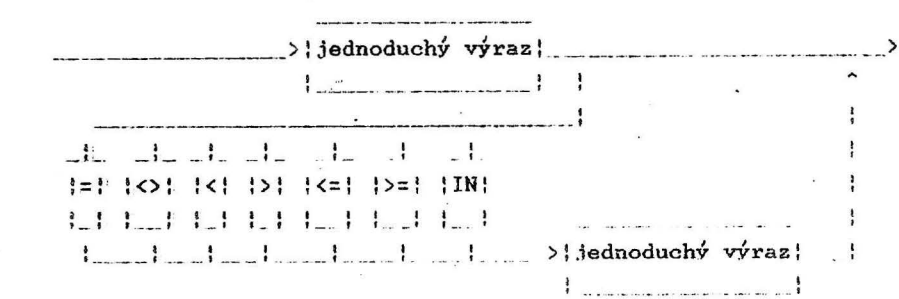
Dolní hranice množiny je vždy Ø a její velikost je rovna maximu základního typu množiny. Např. SET OF CHAR (množina znaků) vždy zaujímá 32 bajtů (je možných 256 prvků - jeden bit pro každý prvek). Podobně je SET OF Ø..1Ø ekvivalentní SET OF Ø..255.

1.12 JEDNODUCHÝ VÝRAZ.





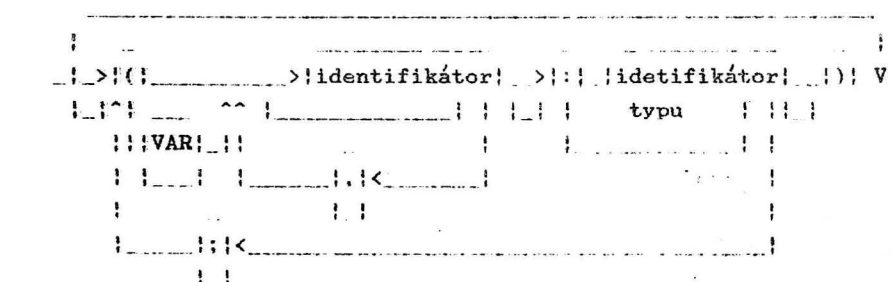
1.13 VÝRAZ.



Používáte-li IN, musí být typ množiny stejný jako typ jednoduchého výrazu. Výjimku tvoří celočíselný argument, pro který je typ omezen na Ø..255.

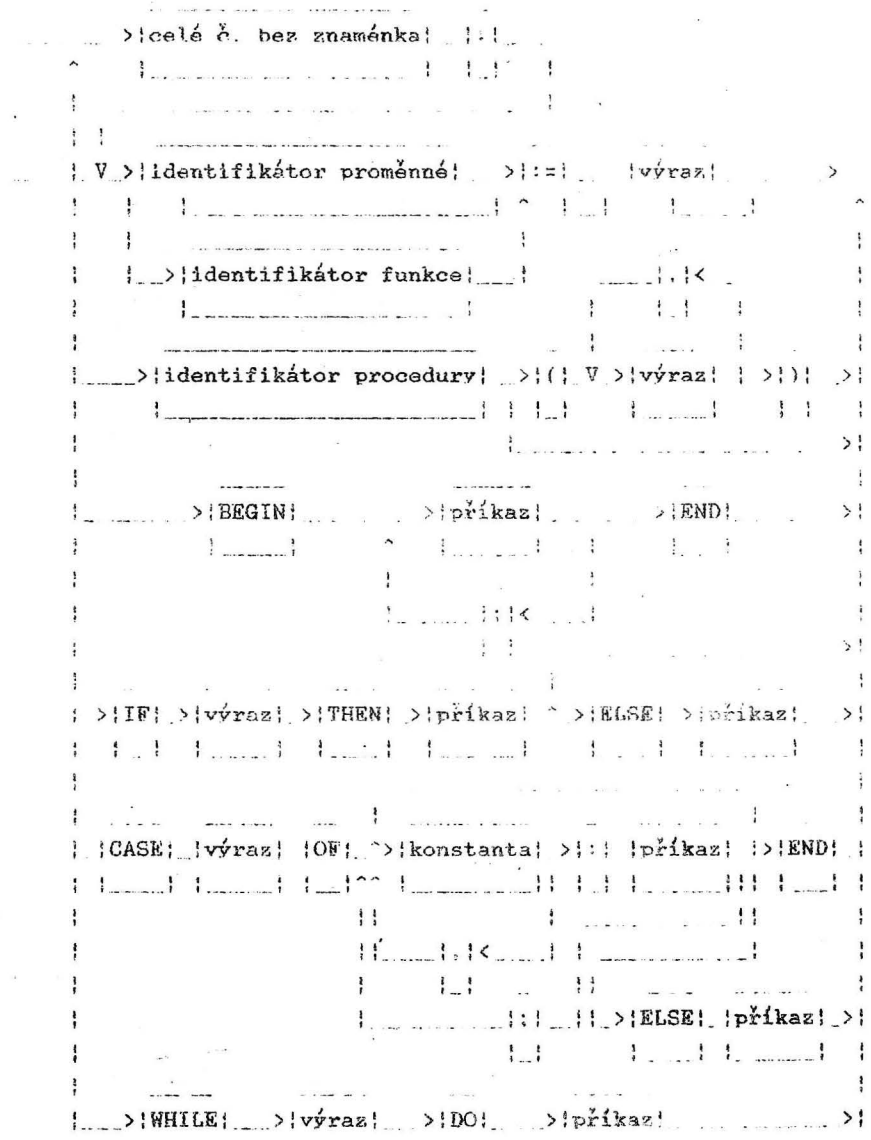
Množiny mohou být srovnávány pouze operátory >=, <=, <>, =, ukazatele mohou být porovnávány pouze operátory = a <>.

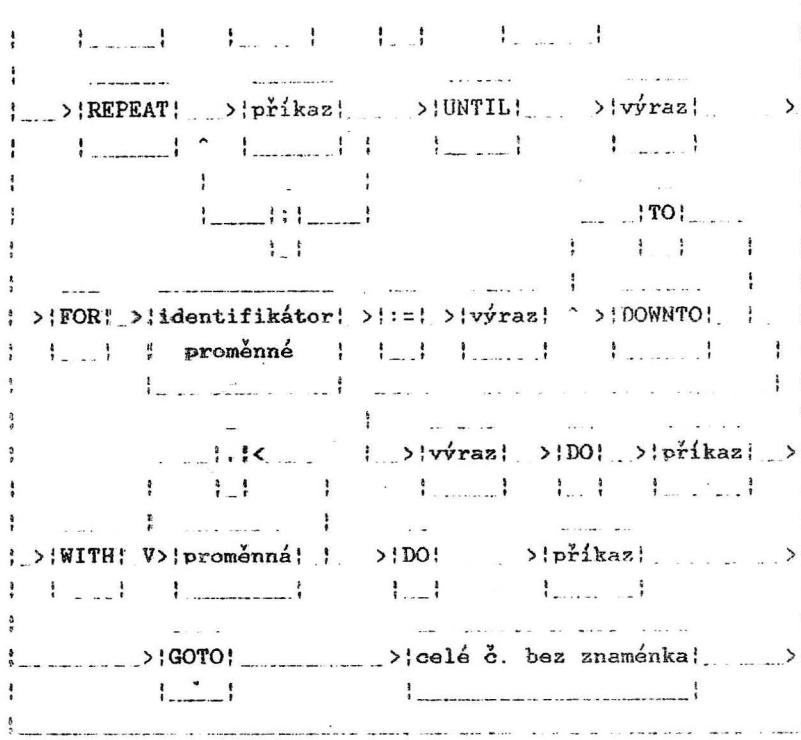
1.14 SEZNAM PARAMETRŮ.



Procedury a funkce nemohou být parametry.

1.15 PŘÍKAZ.



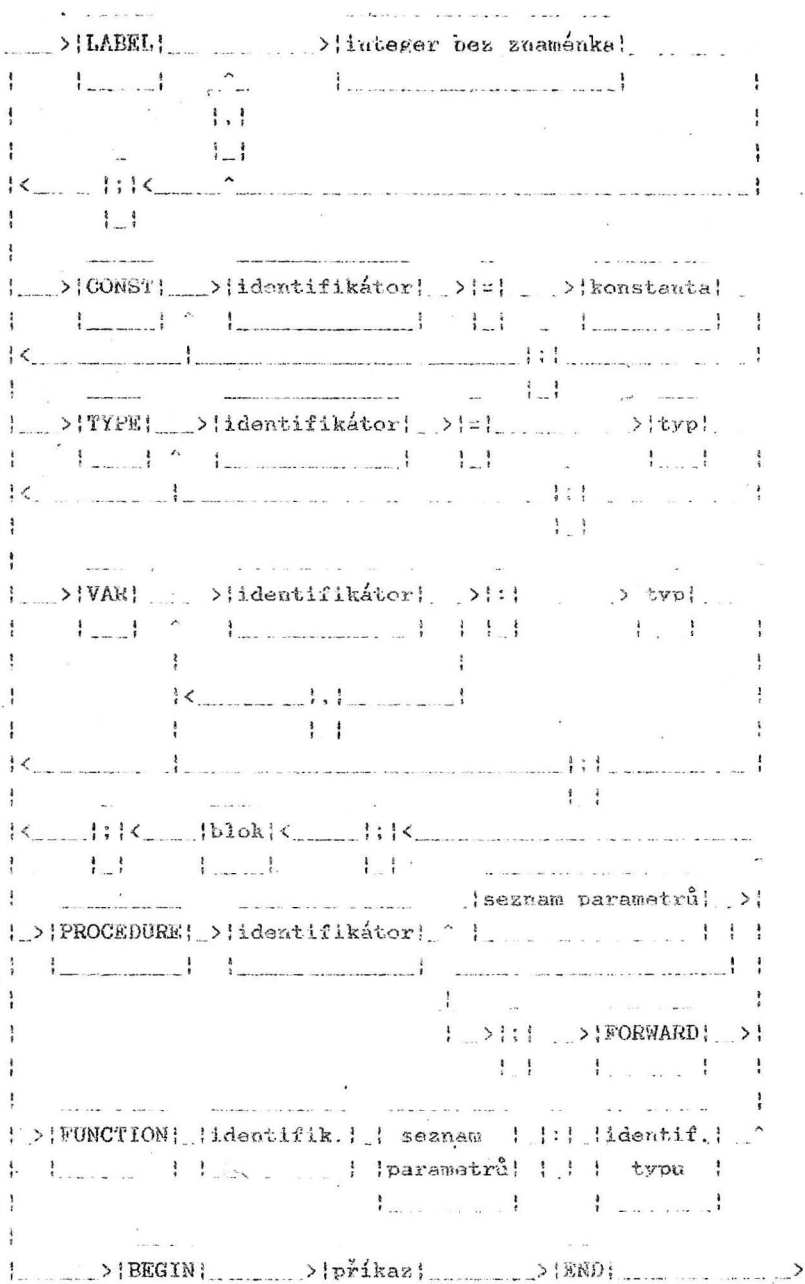


CASE: prázdný příkaz CASE (CASE OF END) není povolen. Příkaz za ELSE je vykonán, když se selektor ('výraz') nenachází v CASE seznamu ('konstanta'). Je-li použit koncový znak END a selektor nebyl nalezen, pak je řízení předáno na příkaz následující po END.

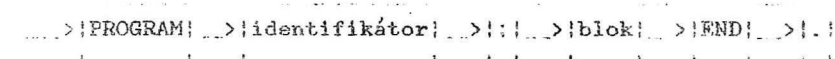
FOR: řídící proměnná příkazu FOR nesmí být strukturovaná a nesmí být parametrem.

GOTO: návěští musí být deklarováno v tomtéž bloku a na stejné úrovni jako GOTO. Návěští se skládají z jedné až čtyř číslic.

1.16 BLOK.


LABEL
integer bez znaménka
CONST
identifikátor
=
konstanta
TYPE
identifikátor
=
typ
VAR
identifikátor
:
typ
blok
PROCEDURE
identifikátor
seznam parametrů
FORWARD
FUNCTION
identifik.
seznam parametrů
identif. typu
BEGIN
příkaz
END

1.17 PROGRAM.



2.1 KONSTANTY.

==========

MAXINT	Největší celé číslo, t.j. 32767.
TRUE, FALSE	Konstanty typu Boolean.

2.2 TYPY.

=========

INTEGER	-MAXINT..MAXINT
REAL	-5.9E-39 až 3.4E38
CHAR	CHR(Ø) až CHR(255)
BOOLEAN	(TRUE, FALSE). Označuje typ logických hodnot

2.3 PROCEDURY A FUNKCE.

======================

VSTUPNÍ A VÝSTUPNÍ PROCEDURY

WRITE

WRITE(a,b,c,d, ...)

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

Parametry P1,P2,.....Pn mohou mít následující tvary:
(e) nebo
(e:m) nebo
(e:m:n) nebo
(e:m:H)
kde e,m a n jsou výrazy a H je literál (konstanta).

1]

e je typu integer: použijeme (e) nebo (e:m). Hodnota integer výrazu e je přeměněna na znakový řetězec doplněný mezerami. Délku řetězce lze zvětšit doplněním mezer zleva použitím m, které specifikuje počet znaků, vyslaných na výstup. Jestliže m nepostačuje pro vypsání e, nebo není použito, je e vypsáno celé s následnou mezerou a m je ignorováno. Všimněte si, že je-li m specifikováno jako délka e bez následné mezery, potom žádná následná mezera na výstup nepřijde.

2]

e je typu integer: použijeme tvaru (e:m:H). V tomto případě je výstup v hexadecimální soustavě. Pro m=1 nebo m=2 bude hodnota (e MOD 16^m) na výstupu v délce m znaků. Pro m=3 nebo m=4 je na výstupu celá hodnota e v hexadecimální soustavě. Pro m>4 jsou vkládány zleva mezery. Tam kde je to možné budou zleva vkládány nuly.

Příklad:

```
WRITE(1Ø25:m:H);
m=1    výstup:  Ø
m=2    výstup:  Ø1
m=3    výstup:  Ø4Ø1
m=4    výstup:  Ø4Ø1
m=5    výstup:   Ø4Ø1
```

3]

e je typu real: použijeme tvaru (e), (e:m) nebo (e:m:n). Hodnota e je přeměněna na znakový řetězec reprezentující reálné číslo. Formát je určen použitím n. Není-li n použito bude číslo zobrazeno ve vědecké notaci s mantisou a exponentem. Je-li číslo záporné bude znaménko '-' vypsáno před mantisou. Číslo je zobrazeno s jedním až pěti desetinnými místy, exponent vždy se znaménkem. Minimální šířka při tomto zobrazení je tedy 8 znaků. Je-li šířka m<8 bude na výstupu plná šířka 12 znaků. Pro m>=8 bude na výstupu jedno, nebo více desetinných míst, až do maxima pěti (pro m=12). Pro m>12 jsou před číslo vkládány mezery.

Příklad:

```
WRITE(-1.23E 1Ø:m);
m=7    dá:  -1.23ØØE+1Ø
```

m=8 dá: -1.2E+1Ø
m=9 dá: -1.23E+1Ø
m=1Ø dá: -1.23ØE+1Ø
m=11 dá: -1.23ØØE+1Ø
m=12 dá: -1.23ØØØE+1Ø
m=13 dá: -1.23ØØØE+1Ø

Je-li použito tvaru (e:m:n) bude e vypsáno ve tvaru s pevnou desetinnou tečkou s n desetinnými místy. Při dostatečné šířce pole budou před číslo doplněny mezery. Pro n=Ø je e zobrazeno jako integer. Je-li e příliš velké pro výstup ve specifikované šířce pole, bude výstup udán ve vědecké notaci s šířkou pole m (viz nahoře).
Příklad: WRITE(1E2:6:2) dá: 1ØØ.ØØ WRITE(1E2:8:2) dá: 1ØØ.ØØ WRITE(23.455:6:1) dá: 23.5 WRITE(23.455:4:2) dá: 2.34550E+Ø1 WRITE(23.455:4:Ø) dá: 23

4]
e je typu CHAR nebo ARRAY OF CHAR. Jsou možné tvary (e) nebo (e:m) a znak, nebo řetězec znaků bude na výstupu v minimální šíři pole = 1 (pro znaky), nebo v délce řetězce (pro pole znaků). Je-li m dostatečně velké, jsou zleva doplněnny mezery.

5]
e je typu BOOLEAN. Jsou možné tvary (e) nebo (e:m), na výstupu bude TRUE nebo FALSE v závislosti na Booleovské hodnotě e, v minimalní šíři vystupního pole 4 nebo 5.

WRITELN

WRITELN(a,b,c,d, ...); WRITELN

Výstupy jsou ukončeny znakem konec řádky.
WRITELN(P1,P2,.....Pn); je ekvivalentní:
BEGIN WRITE(P1,P2,.....Pn);WRITELN END;

CLS

CLS

Procedura CLS smaže obrazovku.

READ

READ(a,b,c,d, ...)

Procedura READ je používána pro vstup dat z klávesnice prostřednictvím vyrovnávací paměti.Ta je zpočátku prázdná (s vyjímkou znaku NEW LINE). Musíme brát v úvahu, že jakýkoliv přístup k této vyrovnávací paměti je prostřednictvím 'textového okna' nad vyrovnávací pamětí- můžeme pozorovat vždy jen jeden znak. Když je toto textové okno umístěno nad znakem NEW LINE, bude před ukončením operace čtení čten do vyrovnávací paměti nový řádek textu z klávesnice.
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. Jsou možné 4 případy:
1)
V je typu CHAR. V tomto případě READ(V) čte znak ze vstupní vyrovnávací paměti a přiřadí ho V. Když je textové okno umístěno na znaku NEW LINE, má funkce EOLN hodnotu TRUE a je čten nový řádek textu z klávesnice. Při následné operaci čtení bude textové okno umístěno na začátek nového řádku.
Důležitá poznámka: Je-li EOLN = TRUE na počátku programu a první READ je typu CHAR, bude přečtena hodnota NEW LINE a následována čtením nového řádku z klávesnice. Následující čtení typu CHAR přijme první znak nového řádku za předpokladu, že není prázdný. Viz také procedura READLN.
2)
V je typu ARRAY OF CHAR. Řetězec znaků může být čten použitím READ jako série znaků do délky vymezené řetězcem nebo než EOLN = TRUE. Není-li řetězec zaplněn čtením (konec řádku byl dosažen před zplněním celého řetězce) bude zbytek řetězce zaplněn znaky

CHR(Ø) - to umožňuje programátorovi vyhodnotit délku přečteného řetězce.
Poznámka pro 1) platí i zde.
3)
V je typu integer. V tomto případě je čtena série znaků, reprezentující celé číslo. Předcházející mezery a znaky NEW LINE jsou přeskakovány.

Čtení celého čísla většího než MAXINT (32767) vyvolá chybové hlášení 'Number too large' a program bude ukončen.
Není-li první znak po mezeře nebo znaku NEW LINE číslice nebo znaménko ('+' nebo '-'), bude vyvoláno chybové hlášení 'Number expected' a program bude přerušen.
4)
V je typu real. V tomto případě je čtena série znaků reprezentující reálné číslo v souhlasu se syntaxí podle 1.3.

Všechny úvodní mezery a znaky NEW LINE jsou přeskočeny a stejně jako pro celá čísla, první znak musí být číslice, nebo znaménko. Je-li čtené číslo příliš velké nebo příliš malé (viz 1.3) bude hlášena chyba 'Overflow'. Není-li 'E' následováno znaménkem nebo číslicí vznikne chyba 'Exponent expected'. Není-li desetinná tečka následována číslicí bude chybové hlášení 'Number expected'.

READLN

READLN(a,b,c,d, ...); READLN
READLN(V1,V2.....Vn); je ekvivalentní :
BEGIN READ(V1,V2,.... ..Vn); READLN END;

READLN čte řádek z klávesnice do vyrovnávací paměti. EOLN nabude hodnoty FALSE po provedení READLN, pokud není následující řádek prázdný.
READLN může být použito k přeskočení prázdného řádku přítomného při spuštění programu - vyvolá tedy čtení do nové řádky. To je užitečné pro čtení složky typu CHAR na začátku programu, ale není

to nutné, při vstupu INTEGER nebo REAL čísla (protože znaky NEW_LINE jsou přeskočeny).

FUNKCE VSTUPU

EOLN

```
EOLN : BOOLEAN
```

Funkce EOLN je booleovskou funkcí, která nabývá hodnoty TRUE když následující znak, který má být čten je znakem NEW LINE (konec řádky). V ostatních případech má hodnotu FALSE.

INCH

```
INCH : CHAR
```

Funkce INCH testuje klávesnici a byla-li stisknuta klávesa, obsahuje znak, reprezentovaný stisknutou klávesou. Pokud nebyla stisknuta žádná klávesa obsahuje CHR(0). Výsledek této funkce je typu CHAR.

FUNKCE PŘEVODU

TRUNC

```
TRUNC(X:REAL) : INTEGER
```

Parametr X musí být typu real, nebo integer a hodnota TRUNC je největší celé číslo menší nebo rovné X pro X kladné nebo nejmenší celé číslo větší nebo rovné X pro X záporné.

Příklad: TRUNC(-1.5) = -1 TRUNC(1.9) = 1

ROUND

```
ROUND(X:REAL) : INTEGER
```

X je typu real nebo integer a hodnota ROUND je 'nejbližší' celé číslo k X (podle standardních zaokrouhlovacích pravidel).

Příklad: ROUND(-6.5) = -6 ROUD(11.7) = 12
ROUND(-6.51) = -7 ROUND(23.5) = 24

ENTIER

ENTIER(X:REAL) : INTEGER

X je typu real nebo integer - hodn je celé číslo menší, nebo rovné X pro všechna X.
Příklad:
ENTIER(-6.5) = -7 ENTIER(11.7) = 11

Poznámka: ENTIER není standardní Pascalovskou funkcí, ale je ekvivalentní Basicovske funkci INT.

ORD

ORD(X:skalární typ) : INTEGER

X je skalarního typu s vyjímkou real.Hodnota ORD je celé číslo, reprezentující pořadí hodnoty X v množině definující typ X.
Příklad:
ORD('a') = 97 ORD('@') = 64

CHR

CHR(X:INTEGER) : CHAR

X je typu integer. Hodnotou CHR je znak odpovídající ASCII hodnotě X.
Příklad: CHR(49) = '1' CHR(91) = '['

ARITMETICKÉ FUNKCE

Argument následujících funkcí musí být typu REAL, nebo INTEGER.
ABS

ABS(X:REAL) : REAL ABS(X:INTEGER) : INTEGER

Absolutní hodnota X (např. ABS(-4.5) = 4.5). Výsledek je stejného typu jako X.

SQR

SQR(X:REAL) : REAL SQR(X:INTEGER) : INTEGER

Hodnotou je X * X (druhá mocnina X). Výsledek je stejného typu jako X.

SQRT

SQRT(X:REAL) : REAL SQRT(X:INTEGER) : REAL

Druhá odmocnina X. Výsledek je vždy typu real. Je-li argument X záporný, je generováno chybové hlášení 'Maths Call Error'.

FRAC

FRAC(X:REAL) : REAL FRAC(X:INTEGER) : REAL

Zlomková část X: FRAC(X) = X - ENTIER(X).
Příklad:
FRAC(1.5) = Ø.5 FRAC(-12.56) = Ø.44.

SIN

SIN(X:REAL) : REAL SIN(X:INTEGER) : REAL

Sinus X, kde X je v radiánech. Výsledek je vždy typu real.

COS

COS(X:REAL) : REAL COS(X:INTEGER) : REAL

Cosinus X, kde X je v radiánech. Výsledek je typu real.

TAN

COS(X:REAL) : REAL COS(X:INTEGER) : REAL

Tangens X, kde X je v radiánech. Výsledek vždy typu real.

ARCTAN

ARCTAN(X:REAL) : REAL ARCTAN(X:INTEGER) : REAL

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

EXP

EXP(X:REAL) : REAL EXP(X:INTEGER) : REAL

Hodnota e^X, kde e = 2.71828. Výsledek vždy typu real.

LN

LN(X:REAL) : REAL LN(X:INTEGER) : REAL

Přirozený logaritmus (t.j. se základem e) čísla X. Výsledek je vždy typu real. Je-li X <= Ø je generováno hlášení 'Maths Call Error'.

DALŠÍ PŘEDDEFINOVANÉ PROCEDURY NEW

NEW(p:^typ)

Procedura NEW vytváří prostor pro dynamickou proměnnou. Proměnná p je typu ukazatel; po provedení NEW(p) obsahuje p adresu nově umístěné dynamické proměnné. Typ dynamické proměnné je stejný jako u proměnné p a může být libovolného typu.

K nalezení přístupu k dynamické proměnné se používá p^ - viz příklady použití ukazatele k odkazu na dynamické proměnné v Příloze E.
Pro nové použití prostoru užitého pro dynamické proměnné používejte procedury MARK a RELEASE.

MARK

MARK(p:^typ)

Tato procedura udává adresu posledního obsazeného bytu dynamické proměnné, uložené v proměnné typu ukazatel p. Stav po provedení MARK může být obnoven pomocí procedury RELEASE.
Typ proměnné na kterou p odkazuje nerozhoduje, protože p může být použito pouze ve spojení s MARK a RELEASE, nikdy s NEW. Ukázkový program s použitím MARK a RELEASE v příloze E.

RELEASE

RELEASE(p:^typ)

Tato procedura uvolňuje prostor v paměti pro použití dynamických proměnných. Obsah této části paměti může být znovu obnoven použitím MARK(p) - takto se zruší všechny dynamické proměnné vytvořené procedurou MARK. Tuto proceduru používejte velmi opatrně.

INLINE

INLINE(c1:INTEGER,c2:INTEGER,c3:INTEGER, ...)

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. C1,C2, C3 atd. jsou celočíselné konstanty a může jich být libovolný počet.

USER

USER(v:INTEGER)

USER je procedura s jedním celočíselným argumentem V, která vyvolá program na adrese V. Protože Pascal obsahuje celá čísla ve formě dvojkového doplňku (viz příloha D), je nutno adresy větší nez #7FFF (32767) označit zápornou hodnotou V. Například #C000 je -16384 a tedy USER(-16384); bude volat adresu #C000. Pokud se k odkazu na adresu používá konstanta, je vhodnější používat hexadecimální soustavu.
Volaný program musí končit instrukcí RET (#C9) a musí zachovat registr IX.

HALT

HALT

Tato procedura zastaví běh programu se zprávou 'Halt at PC=XXXX kde XXXX je hexadecimální adresa, na které je povel HALT. Společně s výpisem překladu lze tuto informaci použít k trasování programu.

POKE

POKE(x:INTEGER,v:typ)

POKE uloží výraz V do paměti počítače počínaje adresou X. X je typu integer a v může být libovolného druhu mimo SET.
Příklad: POKE(#6000,'A') umístí #41 na adresu #6000.
POKE(-16384,3.6E3) umístí 00 0B 80 70 (v hex.) od adresy #C000.

TOUT

TOUT(jmeno:ARRAY [1..12] OF CHAR,START:INTEGER,SIZE:INTEGER)

TOUT je procedura určená k uložení jedné proměnné na mgf pásku. První parametr je typu ARRAY[1..n] OF CHAR a je jménem souboru dat, který má být uložen. Velikost pole n musí být v mezích 1..12. K názvu souboru se automaticky připojuje typ .DTA. Od adresy START je uloženo SIZE bytů. Oba tyto parametry jsou typu integer.

Například k uložení proměnné V na pásku pod jménem 'VAR' užijte:

```
TOUT('VAR',ADDR(V),SIZE(V))
```

TIN

```
TIN(jmeno:ARRAY [1..12] OF CHAR,START:INTEGER)
```

Tato procedura se používá ke čtení proměnné z pásky, uložené pomocí TOUT. NAME je stejného typu jako u TOUT a má stejný význam. START je typu integer. Na pásce je hledán soubor NAME, který je vložen od adresy START. Počet vkládaných bytů je již na pásce (nahráno příkazem TOUT).

OUT

```
OUT(bc:INTEGER,a:INTEGER)
```

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

OPENO

```
OPENO(jmeno:ARRAY[1..12] OF CHAR)
```

Otevře výstupní soubor s názvem jmeno a typem .DTA. Do souboru se zapisuje procedurou PUT a PUTBLOCK, zavře se procedurou CLOSEO.

PUT

```
PUT(v:typ)
```

Zapíše do výstupního souboru otevřeného procedurou OPENO proměnnou v ve tvaru vnitřní reprezentace, tak, jak je uložena v paměti. Typ proměnné může být libovolný.

Příklad:
Příkaz PUT('ABC') zapíše do souboru tři bajty - kódy znaků A, B a C.

PUTBLOCK

PUTBLOCK(START,INTEGER,SIZE:INTEGER)

Zapíše do výstupního souboru otevřeného procedurou OPENO blok dat o délce SIZE z paměti od adresy START (obdoba TOUT).

CLOSEO

CLOSEO

Zavření výstupního souboru otevřeného procedurou OPENO.

OPENI

OPENI(jmeno:ARRAY[1..12] OF CHAR)

Otevření vstupního souboru s názvem jmeno. Ze souboru se čte procedurami GET a GETBLOCK, soubor se uzavře procedurou CLOSEI.

GET

GET(v:typ)

Načte ze vstupního souboru otevřeného procedurou OPENI do proměnné v tolik bajtů, kolik odpovídá její vnitřní reprezentaci.

Příklad:
Příkaz GET(v:CHAR) načte ze souboru jeden bajt a uloží ho do proměnné v.

GETBLOCK

GETBLOCK(START:INTEGER,SIZE:INTEGER)

Načte ze vstupního souboru otevřeného procedurou OPENI blok dat o délce SIZE do paměti na adresu START.

CLOSEI

CLOSEI

Zavře vstupní soubor otevřený procedurou OPENI.

DALŠÍ PŘEDDEFINOVANÉ FUNKCE

RANDOM

RANDOM : INTEGER

Tato funkce tvoří pseudonáhodné číslo od Ø a 255 včetně. Ačkoliv je tato funkce velmi rychlá, nedává dobré výsledky při opakovaném použití v cyklech, které neobsahují vstupní operace. Chcete-li lepší výsledky než umožňuje tato funkce, použijte funkci RND.

Poznámka: funkce RANDOM je implementována instrukcí LD A,R.

RND

RND : INTEGER

Funkce RND vytvoří pseudonáhodné číslo s rovnoměrným rozložením na intervalu -32768..32767. Kvalita náhodnosti generovaných čísel odpovídá vlastnostem generátorů využívaných v interpretech jazyka Basic.

SUCC

SUCC(X:ordinální_typ) : ordinální_typ

X může být jakéhokoliv skalárního typu s vyjímkou real. Hodnotou SUCC je následovník X.
Příklad:
SUCC('A') = 'B' SUCC('5') = '6'.

PRED

PRED(X:ordinální_typ) : ordinální_typ

Podobně jako u SUCC je hodnotou PRED předchůdce X.
Příklad:
PRED('j') = 'i' PRED(TRUE) = FALSE

ODD

ODD(X:INTEGER) : BOOLEAN

X je typu integer, hodnotou ODD je TRUE pro X liché a FALSE pro sudé X.

ADDR

ADDR(V:typ) : INTEGER

Tato funkce udává adresu proměnné s identifikátorem V.

PEEK

PEEK(X:INTEGER,T:typ) : typ

PEEK vloží obsah paměťového místa s adresou X do proměnné libovolného typu, který udává parametr T. V procedurách PEEK a POKE (opak PEEK) jsou data uváděna ve vnitřní reprezentaci (viz příloha D).
Například:
když paměť obsahuje od adresy #5000 hodnoty 50 61 73 63 61 6C (hex.) potom:
WRITE(PEEK(#5000,ARRAY[1..6] OF CHAR)) dava 'Pascal'
WRITE(PEEK(#5000,CHAR)) dava 'P'
WRITE(PEEK(#5000,INTEGER)) dava 24912
WRITE(PEEK(#5000,REAL) dava 2.46227E+29

SIZE

SIZE(V:typ) : INTEGER

Parametrem funkce je libovolná proměnná. Výsledkem typu integer je velikost paměti v bajtech, kterou zaujíma uvedená proměnná.

INP

INP(X:INTEGER) : CHAR

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

EOF

EOF : BOOLEAN

Funkce EOF nabývá hodnoty FALSE, jestliže byl otevřen vstupní soubor procedurou OPENI a z tohoto souboru ještě nebyly přečteny všechny data. V ostatních případech nabývá hodnoty TRUE.

GRAFICKÉ PROCEDURY A FUNKCE

MODE

```
MODE(mod:INTEGER)
```

Nastaví grafický mód podle parametru mod takto:

```
mod=Ø  => AND    {černě => mazání}
mod=1  => OR     {bíle  => kreslení}
mod=2  => XOR    {inverze}
```

PLOT

```
PLOT(x,y:INTEGER)
```

Vykreslení bodu na souřadnicích [x,y] v nastaveném grafickém módu. Souřadnice [Ø,Ø] je v levém dolním rohu obrazovky.

DRAW

```
DRAW(x1,y1,x2,y2:INTEGER)
```

Vykreslení přímky z bodu [x1,y1] do bodu [x2,y2].

BOX

```
BOX(x1,y1,x2,y2:INTEGER)
```

Vykreslení rámečku s úhlopříčkou určenou body [x1,y1] a [x2,y2].

BOXF

BOXF(x1,y1,x2,y2:INTEGER)

Vykreslení plného obdélníku s úhlopříčkou danou body [x1,y1] a [x2,y2].

CIRCLE

CIRCLE(x,y,r:INTEGER)

Vykreslení kružnice se středem v bodě [x,y] a poloměrem r.

FILL

FILL(x,y:INTEGER)

Vyplnění plochy ohraničené souvislou čarou a obsahující bod [x,y].

POINT

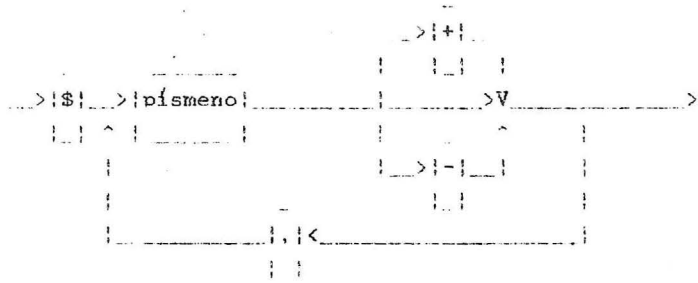
POINT(x,y) : BOOLEAN

Funkce testující rozsvícení bodu: TRUE pokud bod na souřadnicích [x,y] svítí, FALSE když nesvítí.

3.1 KOMENTÁŘE.

Komentář může být mezi dvěma vyhrazenými slovy, čísly, identifikátory nebo speciálními symboly - viz příloha B a C. Komentář začíná znakem '{' nebo dvojicí znaků '(*'. Nenásleduje-li znak '$' jsou všechny znaky ignorovány až do dalšího '}' nebo '*)'. Je-li nalezeno '$' vyhledává překladač sérii znaků volících jeho činnost, ostatní znaky jsou přeskočeny až do '}' nebo '*)'.

3.2 ŘÍZENÍ PŘEKLADU.



L - řízení výpisu

Implicitně: L+

Ovládá řízení výpisu textu programu a adres strojového kódu překladačem.

L+ úplný výpis

L- výpis pouze při nalezení chyby

O - kontrola přetečení

Implicitně: O+

Řídí kontrolu přeplnění. Na přeplnění je vždy kontrolováno násobení a dělení celých čísel a všechny aritmetické operace s reálnými čísly.

O+ kontrola je prováděna i při sčítání a odčítání celých čísel

O- kontrola podle O+ není prováděna

C - test klávesnice

Implicitně: C+

Řídí test klávesnice během provádění strojového kódu.

C+ při stisku STOP se běh programu přeruší se zprávou HALT Test klávesnice se provádí na počátku všech cyklů, procedur a funkcí. Toho lze využít při ladění programu. Není vhodné ho použít chcete-li rychlý běh programu.

C- test klávesnice není prováděn

S - kontrola zásobníku

Implicitně: S+
Řídí provádění kontroly přetečení zásobníku.

S+ na začátku každého volání procedury nebo funkce je kontrolováno, nedojde-li k přeplnění. Dojde-li k přeplnění dynamické proměnné nebo programu je zobrazeno 'Out of RAM at PC=XXXX' a běh programu je přerušen. Používá-li procedura velký rozsah zásobníku může dojít ke zhroucení programu.
S- kontrola zásobníku není prováděna

A - kontrola indexů

Implicitně: A+

Řídí kontrolu mezí indexu polí.

A+ je-li index pole příliš velký nebo malý je generována zpráva 'Index too hig', nebo 'Index too low', běh programu bude zastaven.

A- kontrola se neprovádí.

I - porovnávání čísel bez znaménka

Implicitně: I-

Když se při použití 16 bitového dvojkového doplňku v celočíselné aritmetice liší argumenty o více než MAXINT (32767) dochází u operací >,<,>= a <= k přeplnění,což vede k nesprávnému výsledku porovnávání.

I+ zajistí správný výsledek porovnávání. Stejná situace může nastat u aritmetiky reálných čísel kde na základě přeplnění dojde k chybě liší-li se argumenty o více než cca 3.4E38: tomu se vyhnout nelze.

I- kontrola výsledku porovnávání nebude prováděna

P - výstup na tiskárnu

Implicitně: obrazovka

Přepínač výstupu, na který se vyšle výpis překladu. Byla-li užívána obrazovka je použita tiskárna a naopak. Tato volba není následována '+' nebo '-'.

F - překlad ze vstupního souboru

Překlad textu programu z magnetofonu. Znak F musí následovat název souboru. K názvu souboru se automaticky připojí typ .PAS. Tato funkce je užitečná, přejete-li si vybudovat knihovnu často používaných procedur a funkcí na pásce a potom vkládat do programů. Je vhodné použít k řízení listingu L-, neboť jinak bude kompilátor pracovat pomalu.
Příklad:
{$L-,F MATRIX vložit textový soubor MATRIX.PAS};

Při tvorbě rozsáhlých programů se může stát, že v paměti nebude současně dostatek prostoru pro zdrojový text i strojový kód. Je však možné překládat programy uložené po částech na pásku za použití volby 'F'. Tím je ponecháváno mnohem více místa pro strojový kód.

Kapitola 4. Editor.

Pro psaní a opravy zdrojového textu slouží velice jednoduchý řádkový editor. Pro zápis příkazů tohoto řádkového editoru je využíván obrazovkový editor, který je součástí operačního systému mikropočítače ONDRA.

Název klávesy STOP v dalším textu znamená kombinaci CTRL C.

Příkaz má následující formát:
C N1,N2,S1,S2
C je příkaz editoru (jeden znak)
N1 je číslo v rozsahu 1 - 32767 včetně.
N2 je číslo v rozsahu 1 - 32767 včetně.
S1 je znakový řetězec s maximální délkou 20 znaků.
S2 je znakový řetězec s maximální délkou 20 znaků.

Argumenty se oddělují čárkou (je možno změnit - viz povel 'S'), mezery jsou ignorovány s výjimkou mezer uvnitř řetězců. Žádný

argument není nezbytný, i když některé příkazy (např. 'D'alete) neproběhnou, dokud nebude určeno N1 a N2. Editor si pamatuje čísla a řetězce. Pokud neuvedete argument, dosadí se tyto hodnoty. (Pokud ovšem nejsou argumenty povinné). Hodnoty N1 a N2 jsou po spuštění rovny 1Ø a řetězce prázdné. Zadání nepřípustného příkazového řádku, např. F-1,1ØØ,HELLO bude ignorováno a zobrazí se 'Pardon?'. Řádek je nutno vložit správně, např. F1,1ØØ,HELLO. 'Pardon' bude také zobrazeno, překročí-li S2 délku 2Ø znaků; u řetězce S1 jsou všechny znaky překračující počet 2Ø ignorovány. Příkazy mohou být vkládány velkými nebo malými znaky.

4.2 PŘÍKAZY EDITORU.

Pokud je argument v popisu uzavřen symboly '<>' musí být uveden, jinak příkaz nebude proveden.

VKLÁDÁNÍ TEXTU

Text lze do souboru vložit napsáním čísla řádku, mezery a požadovaného textu, obdobně jako tomu bývá zvykem u jazyka Basic. Zadáním samotného čísla řádku bude řádek tohoto čísla z textu vymazán.

I n,m

Zavádí automatické číslování řádků počínaje n s krokem m. Číslování lze zrušit řídící funkcí STOP. Vložením nového řádku se ruší řádek se stejným číslem, pokud již existoval. Vytvoří-li se číslo řádku větší než 32767 mód I se zruší.

VÝPIS TEXTU.

L n,m

Zobrazuje na obrazovce text od řádku n do m včetně. Implicitní hodnota pro n je 1,pro m 32767, nejsou tedy brány v předtím vložených argumentů. Pro výpis celého textového souboru použijte 'L' bez argumentu. Obrazové řádky jsou formátovány na levém okraji tak, že číslo řádku je zobrazeno přehledně. Počet zobrazovaných řádků může být řízen příkazem 'K'. Po zobrazení určitého počtu řádků se výpis přeruší a pokud není již vypsán řádek m můžete se řídicí funkcí STOP vrátit do editoru nebo pokračovat ve výpisu stiskem libovolné klávesy.

L n

'K' určuje počet najednou zobrazených obrazových řádků. Například použitím K5 se najednou zobrazí 5 obrazových řádků.

EDITACE TEXTU.

Po vytvoření textu se vyskytne potřeba některé řádky opravit. K dispozici jsou příkazy k opravám, vymazávání, posunům a přečíslování řádků.

D <n,m>

Vymaže všechny řádky textového souboru od n do m. Je-li m<n nebo argumenty nejsou uvedeny není příkaz proveden. Pro smazání jednoho řádku zadejte m=m.

M n,m

Umožňuje nahradit text řádku m textem z řádku n. Text na řádku n je zachován. Příkaz tedy provádí přesun ('M'ove) textového řádku do další polohy v textovém souboru. Pro neexistující n se příkaz nevykoná.

N <n,m>

Přečísluje textový soubor s číslem prvního řádku n a s krokem m. Musí být zadáno n i m; pokud by přečíslování vedlo k číslu řádku vyššímu než je 32767 zůstane zachováno původní číslování.

F n,m,f,s

Hledá řetězec f ('find') v rozmezí řádků n<x<m. Je-li řetězec nalezen, je zobrazen odpovídající textový řádek a přejde se do Edit módu (viz příkaz 'E'). Potom můžete použít příkazy Edit módu k nalezení následujících výskytů řetězce f v již definovaném rozsahu řádků nebo k jeho nahrazení řetězcem s ('substitue') a k hledání dalšího nejbližšího výskytu f. Rozsah řádků a oba řetězce již mohly být zadány dříve jiným povelem, takže je možno zadat pouze 'F'.

E n

Editace řádku n. Řádek n se přenese do vyrovnávací paměti a je i s číslem zobrazen. Číslo řádku je zobrazeno také pod řádkem a přejde se do Edit módu. Veškeré následující úpravy jsou prováděny ve vyrovnávací paměti a proto může být kdykoliv opakován původní textový řádek.

' ' (mezera) - posun kurzoru na následující znak. Nelze posunout za konec řádku.
BACKSPACE - posun kurzoru na předchozí znak řádku. Nelze posunout před začátek řádku.
TAB - posun kurzoru na následující pozici tabulátoru, ale ne za konec řádku.
NEW LINE - konec úpravy tohoto řádku s ponecháním všech provedených změn.
Q - zrušení úpravy řádku, zachová se původní podoba řádku (před započetím úpravy).
R - opětné naplnění editační vyrovnávací paměti z textu, t.j. zrušení všech provedených změn a obnovení původní podoby řádků.

L - výpis zbytku editovaného řádku. Edit mód zůstává, kurzor na začátku řádku.

K - odstranění znaku na pozici kurzoru.

Z - vymazání všech znaků od pozice kurzoru (včetně) až na konec řádku.

F - hledání nejbližšího výskytu řetězce definovaného dříve (viz povel 'F'). Tento doplňkový povel automaticky ukončí editaci na řádku (zachovává změny) pokud nenalezne další výskyt řetězce na řádku. Pokud je řetězec zjištěn v nejblíže následujícím řádku (v již dříve specifikovaném rozsahu řádků), potom je do Edit módu vložen řádek, ve kterém je řetězec nalezen. Po úspěšném vyhledání je kurzor umístěn na začátek řádku.

S - nahrazení nalezeného výskytu řetězce dříve definovaným řetězcem 'substitue' a potom provedení doplňkového povelu 'F'.

Obě posledně jmenované příkazy by měly být používány bezprostředně po povelu 'F'.

I - vkládání znaků od pozice kurzoru. Tento doplňkový mód se zruší stiskem NEW LINE (návrat do hlavního módu Edit s kurzorem v poloze po vsunutí posledního znaku). Použití BACKSPACE v tomto doplňkovém módu vymaže znak vlevo od kurzoru z vyrovnávací paměti, zatím co použití TAB posune kurzor vpřed k další poloze tabulátoru a vsune mezery.

X - posune kurzor na konec řádku a automaticky přepne do doplňkového módu I.

C - změna znaků. Tento doplňkový mód zůstává do stisku NEW LINE. BACKSPACE v tomto doplňkovém módu posune kurzor o jednu pozici vlevo.

PŘÍKAZY PRO PRÁCI SE SOUBORY

P n,m,s

Řádky v intervalu n<x<m jsou uloženy na pásku pod jménem s. Tyto argumenty mohou být zastaveny již dříve. K názvu souboru se automaticky připojí typ .PAS.
Program v Pascalu je do souboru zapisován jako standardní textový soubor, obsahuje tedy pouze grafické znaky a kód CR (ØDH).

G.,s

Načte soubor s názvem s a typem .PAS do paměti.

Je-li již v paměti vložen textový soubor bude nově načtený soubor k němu připojen a celý soubor bude přečíslován počínaje řádkem číslo 1 s krokem 1.

PŘEKLAD A SPUŠTĚNÍ PROGRAMU

C n

Spustí překlad textu počínaje řádkem n. Není-li n uvedeno bude text překládán počínaje prvním existujícím řádkem.
Překladač po spuštění generuje výpis v tomto tvaru:
xxxx nnnn text zdrojového řádku

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

Výpis lze směřovat na tiskárnu příkazem 'P' (viz kapitola 3).
Výpis lze kdykoliv zastavit stiskem MEZERY. Stiskem STOP se vrátíte do editoru, stiskem jiné klávesy pokračujete ve výpisu.

Při zjištění chyby během překladu se na obrazovce vypíše hlášení '*ERROR*' následované '^' ukazující za (!) symbol, který chybu způsobil a číslem chyby (viz příloha A). Výpis se zastaví. Pokud stisknete 'E' vrátíte se do editoru k opravě řádku zobrazeného na obrazovce. Pokud stisknete 'P' vrátíte se do editoru k opravě

předešlého řádku pokud existuje nebo jakoukoliv jinou klávesu k pokračování překladu. Končí-li program chybně (např. chybí 'END'), je zobrazena zpráva 'No more text' (není další text) a řízení se vrátí do editoru. Je-li tabulka symbolů pro překladač malá objeví se hlášení 'No Table Space' (není místo v tabulce) a řízení se vrátí editoru. Jestliže překlad obsahoval chyby, bude zobrazen počet chyb.

Po úspěšném překladu se zobrazí dotaz 'Run?'. Pokud si přejete ihned spustit program, odpovězte 'Y', jinak bude řízení navráceno editoru. Stiskem STOP předčasně ukončíte chod.

R

Předem přeložený program bude spuštěn, ale pouze tehdy, pokud nebyl text programu mezitím změněn.

T n

Zdrojový program je překládán od řádku n (nebo od začátku pokud je n vynecháno) a je-li přeložen úspěšně, objeví se dotaz 'Ok?': a pokud odpovíte 'Y' bude cílový kód produkovaný překladem přesunut na konec standardních procedur (zničí překladač) a standardní procedury a cílový kód budou vysláný na pásku s názvem souboru odpovídajícím předešlé definici řetězce. Přeložený program je pak možné používat samostatně, bez přítomnosti překladače.
Rozhodnete-li se program na pásku nenahrávat odpovězte na dotaz 'Ok?' jinou klávesou než 'Y'; řízení bude vráceno do editoru, který bude stále fungovat, neboť cílovým kódem nebylo pohnuto z místa.

DALŠÍ PŘÍKAZY.

B

Vrací řízení do operačního systému.

O n,m

Provádí zkrácení programu v paměti (rezervovaná slova na jeden symbol, zhuštění mezer). Text je čten do vyrovnávací paměti v rozšířené formě a vkládán zpět do souboru ve zkrácené.

S,,d

Změna znaku pro oddělovač používaný k oddělování argumentů v příkazovém řádku. Implicitně je separátorem čárka ','. Po použití 'S' je separátorem první znak řetězce d. Pokud iste jednou definovali separátor, musí být používán i v povelu 'S' až do specifikace jiného separátoru. Separátorem nesmí být mezera.

A.1 KÓDY CHYB, GENEROVANÉ PŘEKLADAČEM.

1. Číslo je příliš veliké.
2. Je očekáván středník.
3. Nedeklarovaný identifikátor.
4. Je očekáván identifikátor.
5. Užijte '=' místo ':=' v deklaraci konstanty.
6. Je očekáváno '='.
7. Tímto identifikátorem nemůže začínat příkaz.
8. Je očekáváno ':='.
9. Je očekáváno ')'.
10. Chybný typ.
11. Je očekáváno '.'.
12. Je očekáván faktor.
13. Je očekávána konstanta.
14. Tento identifikátor není konstanta.
15. Je očekáváno 'THEN'.
16. Je očekáváno 'DO'.
17. Je očekáváno 'TO' nebo 'DOWNTO'.
18. Je očekáváno '('.
19. Nelze psát tento typ výrazu.
20. Je očekáváno 'OF'.
21. Je očekávána ','.
22. Je očekávána ':'.
23. Je očekáváno 'PROGRAM'.
24. Je očekávána proměnná, neboť parametrem je proměnný parametr.
25. Je očekáváno 'BEGIN'.
26. Je očekávána proměnná při volání READ.
27. Výrazy tohoto typu nelze porovnávat.
28. Možné typy INTEGER nebo REAL.
29. Proměnnou tohoto typu nelze číst
30. Tento identifikátor není typem.
31. Je očekáván exponent u reálného čísla.

32. Je očekáván skalární (nikoliv numerický) výraz.
33. Prázdné řetězce nejsou povoleny (použijte CHR(Ø)).
34. Je očekávana '('.
35. Je očekávána ')'.
36. Index u ARRAY musí být skalárního typu.
37. Je očekáváno '..'.
38. Je očekáváno ')', nebo ',' v deklaraci ARRAY.
39. Dolní mez je větší než horní mez.
4Ø. Množina je příliš velká (více než 256 prvků).
41. Výsledek funkce musí být identifikátor typu.
42. V množině je očekáváno ',' nebo ')'.
43. V množině je očekáváno '..' nebo ')'.
44. Typ parametru musí být typu identifikátoru.
45. Prázdná množina nesmí být prvním faktorem v přiřazovacím příkazu.
46. Je očekáván skalár (včetně real).
47. Je očekáván skalár (nikoliv real).
48. Množiny nejsou slučitelné.
49. K porovnání množin nelze použít '<' a '>'.
5Ø. Je očekáváno 'FORWARD', 'LABEL', 'CONST', 'VAR', 'TYPE' nebo 'BEGIN'.
51. Je očekávána hexadecimální číslice.
52. Není možné POKE množin.
53. Array je příliš velké (> 64 K).
54. V definici RECORD je očekáváno 'END' nebo ';'.
55. Je očekáván identifikátor pole.
56. Po 'WITH' je očekávána proměnná.
57. Proměnná ve 'WITH' musí být typu RECORD.
58. Identifikátor pole nebyl přiřazen příkazu WITH.
59. Po 'LABEL' je očekáváno celé číslo bez znaménka.
6Ø. Po 'GOTO' je očekáváno celé číslo bez znaménka.
61. Toto návěští je na nesprávné úrovni.
62. Nedeklarované návěští.
63. Parametrem v SIZE musí být proměnná.
64. Pro ukazatele může být použit pouze test ekvivalence.
67. Jediný parametr pro tisk celých čísel se dvěma ':' je e:m:H
68. Řetězce nesmí obsahovat kód "konec řádky".
69. Parametr u NEW, MARK, nebo RELEASE by měla být proměnná typu ukazatel.

70. Parametr v ADDR musí být proměnná.

A.2 CHYBOVÁ HLÁŠENÍ PŘI BĚHU PROGRAMU.

Je-li zjištěna chyba je zobrazena jedna z následujících zpráv doplněná řetězcem 'at PC=XXXX', kde XXXX je adresa paměti v místě, kde došlo k chybě. Často bude zdroj chyby zcela zřejmý, pokud ne, obraťte se k výpisu překladu kde zjistíte skutečný výskyt chyby v programu (použijte k porovnání adresu uvedenou ve výpisu před následným během programu).

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

Tyto chyby způsobí zastavení chodu programu.

Příloha B. Rezervovaná slova.

B.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	

B.2 SPECIÁLNÍ SYMBOLY.

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

---------- Příloha C. Předdefinované identifikátory.

```
CONST       MAXINT=32767;

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

PROCEDURE   WRITE; WRITELN; READ; READLN; PAGE; HALT; USER; POKE;
            INLINE; OUT; NEW; MARK; RELEASE; TIN; TOUT; OPENI;
            GET; GETBLOCK; CLOSEI; OPENO; PUT; PUTBLOCK; CLOSEO;
            MODE; PLOT; DRAW; BOX; BOXF; FILL; CIRCLE;

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

---------- Příloha D. Reprezentace a uložení dat.

D.1 REPREZENTACE DAT.

Znalost způsobu ukládání je potřebná a užitečná pro většinu programátorů, kteří se snaží sloučit programy v Pascalu se strojovými programy.

Celá čísla

Každé celé číslo je uloženo ve 2 bytech ve formě dvojkového doplňku.
Příklady:
1 ... 'ØØØ1
256 ... 'Ø1ØØ
-256 ... 'FFØØ
K uchování celých čísel se standardně používá registrový pár HL.

Skalární typy

Zabírají pouze jeden byte bez znaménka.
Znaky: 8 bitů, je užito rozšířeného ASCII.
'E' ... #45
'[' ... #5B

Boolean:
ORD(TRUE)=1 tedy TRUE je reprezentováno 1.
ORD(FALSE)=Ø tedy FALSE je reprezentováno Ø.

Překladač používá standartně registr A.

Reálná čísla

Je užito podobného tvaru (mantisy, exponentu jako u standardní vědecké notace, která je použita ve dvojkové soustavě.

Příklad:

2 ... 2 * 1Ø^Ø nebo 1.ØB * 2^1
1 ... 1 * 1Ø^Ø nebo 1.ØB * 2^Ø

-12.5 ... -1.25 * 1Ø^1 nebo -25 * 2^-1

```
                         ... -11ØØ1B * 2^-1
                         ... -1.1ØØ1B * 2^3 normalizováno
Ø.1 ... 1.Ø * 1Ø^-1  nebo  1/1Ø ... 1/1Ø1ØB ... Ø.1B/1Ø1B
              nyní musíme provádět  binární dělení...
                          Ø.ØØØ11ØØ
                     ---------------------
                   1Ø1 | Ø.1ØØØØØØØØØØØØØØ
                           1Ø1
                           ----
                           11Ø
                           1Ø1
                           ----
                            1ØØØ
                             1Ø1   v tomto okamžiku je
                                   již vidět, že se
                                   zlomek periodicky
                                   opakuje.
         =  Ø.1B/1Ø1B = Ø.ØØØ11ØØB
                           ----
                      1.1ØØ11ØØB * 2^-4
                     ---------------------
```

Jak tedy použít těchto výsledků ke znázornění čísel v počítači? Nejprve rezervujeme 4 bajty pro uložení každého reálného čísla v následujícím formátu:

```
ZNAMÉNKO   NORMALIZOVANÁ MANTISA    EXPONENT    data
  23           22          Ø        7     Ø     bit
!...........................!!.......................!
        H    L                      E     D     registr
```

znaménko: znaménko mantisy; 1=záporné, Ø=kladné.

normalizovaná mantisa: mantisa je normalizovaná do tvaru 1.xxxxxx s horním bitem (bit 22) vždy 1 s vyjímkou zobrazení nuly (HL=Ø,DE=Ø).

exponent: exponent v binarním doplňkovém tvaru.

Takto:

```
  2    = Ø 1ØØØØØØ ØØØØØØØØ ØØØØØØØØ ØØØØØØØ1 #4Ø,#ØØ,#ØØ,#Ø1
  1    = Ø 1ØØØØØØ ØØØØØØØØ ØØØØØØØØ ØØØØØØØØ #4Ø,#ØØ,#ØØ,#ØØ
-12.5  = 1 1ØØ1ØØ ØØØØØØØØ ØØØØØØØØ ØØØØØØ11 #E4,#ØØ,#ØØ,#Ø3
  Ø.1  = Ø 1ØØ11Ø Ø11ØØ11Ø Ø11ØØ11Ø 11111ØØ #66,#66,#66,#FC
```

Uvědomíme-li si, že HL a DE jsou používány k uložení reálných čísel, musíme číslo do registrů vkládat následovně:

```
    2  ...  LD  HL,#4ØØØ
            LD  DE,#Ø1ØØ
    1  ...  LD  HL,#4ØØØ
            LD  DE,#ØØØØ
-12.5  ...  LD  HL,#E4ØØ
            LD  DE,#Ø3ØØ
  Ø.1  ...  LD  HL,#6666
            LD  DE,#FC66
```

Poslední příklad nám ukazuje, proč výpočty s dvojkovými zlomky mohou být nepřesné; Ø.1 nemůže být přesně vyjádřeno dvojkovým zlomkem s omezeným počtem desetinných míst. Reálná čísla jsou ukládána do paměti v pořadí ED LH.

Záznamy a řetězce

Záznamy potřebují stejný prostor jako je celkový součet paměťového prostoru jejich složek.
Uspořádání: je-li n = počet prvků pole s = rozměr každého prvku, potom počet bajtů obsazených polem je n*s.

Př.:
ARRAY[1..1Ø] OF INTEGER vyžaduje 1Ø*2 = 2Ø bajtů
ARRAY[2..12,1..1Ø] OF CHAR má 11*1Ø = 11Ø prvků
a vyžaduje tedy 11Ø bajtů.

Množiny

Množiny jsou ukládány jako řetězce bitů; má-li základní typ n prvků obsadí množina (n-1)DIV 8+1 bajtů.

Příklady:
SET OF CHAR obsadí (256-1)DIV 8+1 = 32 bajtů
SET OF (modra,zelena,zluta) obsadí (3-1)DIV 8+1 = 1 bajt.

Ukazatelé

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

D.2 UKLÁDÁNÍ PROMĚNNÝCH PŘI BĚHU PROGRAMU.

Existují 3 možnosti jak jsou proměnné uloženy při chodu programu.

a. Globální proměnné - deklarované v hlavním programovém bloku.
b. Lokální proměnné - deklarované ve vnitřním bloku.
c. Parametry a hodnoty funkcí - jsou předávány do a z procedur a funkcí.

Globální proměnné

Globální proměnné jsou uloženy od vrcholu zásobníku proměnných dolů, například když je zásobník na adrese #B000 proměnné jsou:

```
VAR   i: INTEGER;
```

```
ch: CHAR;
x: REAL;
potom:
```

i (které zabíra 2 bajty) bude uloženo na adresách #B000-2 a #B000-1, t.j. na adr. #AFFE a #AFFF.

ch (1 bajt) bude uloženo na adrese #AFFE-1, t.j. na #AFFD.

x (4 bajty) 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, 2 bajty) bude umístěno na adresách IX-4-2 a IX-4-1, tedy IX-6 a IX-5.

j bude umístěno na adresách IX-8 a IX-7.

Argumenty a hodnoty funkcí

Hodnoty parametrů jsou zpracovávány jako lokální proměnné tedy 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 (umístěno první) je na adrese IX+2 a IX+3.
i je na adrese IX+4, IX+5, IX+6 a IX+7.

Výstupní parametry (popsány jako VAR) jsou zpracovávány přesně tak, jako hodnotové parametry, s vyjímkou toho, že jsou vždy ukládány do 2 byte a tyto 2 byte obsahují adresu proměnné.

Například:
PROCEDURE test(i: INTEGER; VAR x: REAL);
potom

odkaz na x je umístěn 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 prostor pro funkční hodnotu je rezervován na adresách IX+4, IX+5, IX+6 a IX+7.

```
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('Prosím jméno');
310       READLN;
320       READ(Name);
330       WRITELN;
340       WRITE('Číslo prosím');
350       READLN;
360       READ(Number);
370       WRITELN
380     END
390   END;
400
```

```
410 {K ulcžení 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 zpracovavat adresář podle přání....}
500 END.
```

```
 10 {Program pro výpis znaků řádku v obráceném pořadí.
 20 Ukazuje použití ukazatelů, záznamu, MARK a RELEASE.}
 30
 40 PROGRAM Reverseline;
 50
 60 TYPE elem=RECORD              {Tvoří strukturu řádků výpisu}
 70            next: ^elem;
     8    0
ch: CHAR
 90           END;
100       link=^elem;
110
120 VAR prev,cur,heap: link;     {všechny ukazatele na 'elem'}
130
140 BEGIN
150  REPEAT                      {provést několikrát}
160    MARK(heap);               {přiřadit vrchol k 'hesp'.}
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 STOP k přerušení}
400 END.
```

```
10 {Program ukazuje použití rekurze}
20
30 PROGRAM FACTOR;
40
50 {Tento program počítá faktoriál čísla, zadaného
60 z 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.
```

```
 10 {Program pro výpis textového souboru na obrazovku}
 20 PROGRAM vypis;
 30
 40 VAR znak:CHAR;
 50
 60 BEGIN
 70 OPENI('SOUBOR');      {otevření vstupního souboru SOUBOR.DTA}
 80 WHILE NOT EOF DO
 90 BEGIN
100    GET(znak);         {načtení znaku ze souboru}
110    WRITE(znak)        {výpis znaku na obrazovku}
120 END;
130 WRITELN;              {na nový řádek}
140 WRITELN('Konec souboru');
150 END.
```

[1] Muller, K. Programovací jazyky. Ediční středisko ČVUT.

[2] Wirth, N. Algorithmus+Data Structures=Programs. Prentice Hall, Englewood Cliffs, New York.

[3] Wirth, N. Systematické programovanie. Alfa.