OXFORD PASCAL

vyšší programovací jazyk

OBSAH

1. Úvod do Oxford Pascalu

Pascal je výkonný programovací jazyk vyšší úrovně, jehož autorem je Niklaus Wirth z Curichu ze Švýcarska.

Může být efektivně implementován na malých počítačích stejně jako na velkých. Poskytuje četné výhody proti jiným populárním mikropočítačovým jazykům jako je třeba BASIC.

Některé z těchto výhod jsou:

- bloková struktura jako třeba u ALGOLU
- jména proměnných mohou odrážet význam proměnných
- výkonné technické prostředky pro strukturalizaci dat
- uživatelem definované soubory a konstanty
- snadné spojování funkcí a podprogramů
- rekurzivní volání
- průběžné uvolňování již nepotřebného paměť. prostoru
- kontrola chyb během provádění programu
- dynamické přidělování proměnných
- vyšší standardizace
- vysoká prováděcí rychlost
- větší programová logibilita (možnost rozšiřování)

Oxford Pascal je implementace standardního Pascalu, navržená speciálně pro C-64. Poskytuje všechny rysy tohoto mocného jazyka společně s doplňky pro uživatele domácích počítačů.

Oxford Pascal umožňuje dva režimy práce. V jednodušším režimu kooreziduje překladač Pascalu v RAM společně s uživatelským programem. To je ideální při učení se jazyku nebo při vytváření menších programů, které nepotřebují disk. Většina příkazů je v tomto režimu použitelná, kromě těch, které se týkají diskových souborů. Pro složitější programy může být užit diskově založený překladač, který vám poskytne výhody tohoto jazyka v plné míře.

NĚKTERÉ INFORMACE O IMPLEMENTACI

MAXINT = 32767

typ INTEGER = -32768 .. 32767

typ CHAR = ASCII soubor rozšířený o 64 grafických znaků i prvky množiny musí být v rozsahu 0 .. 127 (proto musí být soubor znaků v rozsahu chr(0) .. chr(127))

typ REAL = přesnost 9 číslic, rozsah asi 1E-38 .. 1E38

standardní výstupní formáty:

integer 7 znaků
real 12 znaků
boolean 6 znaků
char 1 znak
string podle délky řetězce

velikost programu a složitost - žádná omezení, kromě překročení celkové kapacity paměti systému (je hlášeno přeplnění sklípku - STACK OVERFLOW)

identifikátory - musí se lišit v prvních 8 znacích

návěští - musí se lišit v prvních 8 číslicích

ROZŠÍŘENÍ STANDARDNÍHO PASCALU

- dynamická specifikace jmen souborů
- vstup řetězců
- hexadecimální aritmetika a hexa I/O
- manipulace s jednotlivými bity
- možnost spolupráce s programy ve stroj. jazyce
- přímý přístup do paměti a paměti displeje (VDU)
- detekce I/O chyb za běhu programu
- generátor náhodných čísel
- řetězení programů
- dostupnost interních hodin C-64
- oddělená kompilace (linkování programových modulů)
- příkazy pro práci v jemné grafice
- příkazy pro práci s tónovými generátory

JAK PASCAL VYUŽÍVÁ RAM

FFFF ----------

Kernal ROM
nebo grafická bitová mapa

E000 ----------

I/O drivery nebo
paměť barvy pro grafiku

D000 ----------

Editor

C000 ----------

****

****

****

horní hranice RAM
Sklípek spouštěcího programu

Dynamické proměnné

Spustitelný p-kód

Editovaný text

0800 ----------

2. ÚVOD DO PASCALU PRO ZAČÁTEČNÍKY

Tato kapitola přístupnou formou seznamuje s některými rysy PASCALU.

2.1. JAK ZAČÍT - ZÁPIS PŘÍKAZU

Zapněte váš počítač.

Měl by vypsat na obrazovku:

```
****COMMODORE 64 BASIC V2 ****
64K RAM SYSTEMS 38911-BASIC BYTES FREE
READY
```

Nyní zapněte disketovou jednotku a založte disketu PASCALU. Pak zadejte:

LOAD"*",8 a odešlete "RETURN".

Počítač by měl odpovědět:

```
SEARCHING FOR *
LOADING
READY
```

Tím dojde k natažení prvního programu z diskety do paměti. Nyní zadejte RUN a opět "RETURN".

Postupně by mělo dojít k vymazání obrazovky a vypsání zprávy:

```
OXFORD PASCAL VX.X
LOADING...
READY
```

Moderní počítače mohou být velice výkonné, ale je třeba jim sdělit, co mají dělat, prostřednictvím programu. Počítače pracují v jazyce čísel nazývaném STROJOVÝ JAZYK, ale všeobecně je pro lidi těžké ovládat programování v něm a navíc jsou verze strojového jazyka široce rozdílné.

Je mnohem jednodušší komunikovat s počítačem ve vyšším jazyce, jako je třeba PASCAL nebo COBOL. Tyto jazyky jsou

vytvořeny na základě jakési angličtiny, která má však svá velice striktní pravidla gramatiky, aby se vyloučila možnost nedorozumění.

Pascal byl vyvinut Niklausem Wirthem v roce 1968. Je ideálním jazykem pro výuku programování. Program je z Pascalu automaticky překládán počítačem do strojového kódu, ve kterém je pak přímo prováděn.

Začneme hned s velmi jednoduchým programem.

2.1.1. PŘÍKLAD 1

Především musíte zadat váš program do paměti počítače, a to se děje prostřednictvím EDITORU. Zadejte první řádek programu, který je uveden níže. Odešlete klávesou "RETURN".

```
10 begin
20 write("AHOJ")
30 end
```

Jakmile zadáte první řádek, editor by měl automaticky vypsat číslo následujícího řádku, které již pak nemusíte zadávat.

Tato čísla nemají v jazyce Pascal žádný význam (na rozdíl od BASICU), jsou zde čistě z důvodu snadného editování, což si ukážeme nyní. Zapamatujte si, že pokud uděláte při zadávání programu chybu, můžete program opravovat pomocí editačních kláves - INST, DEL, CRSR nahoru, CRSR dolů, CRSR doprava, CRSR doleva - zrovna tak, jak jste zvyklí z BASICU. Stisknutí DEL má za následek výmaz posledně zadaného znaku.

Nyní zadejte zbývající dva řádky programu a buďte obzvláště opatrní na interpunkci a správný pravopis a nezapomeňte na tečku za end.

```
write ("AHOJ")
end.
```

Až skončíte, zadejte prázdný řádek (prostě stiskněte "RETURN") a tím odstavíte automatické číslování řádků. Vše, co je nyní třeba udělat pro spuštění programu, je zadat:

```
r "RETURN"
```

Pokud vše proběhne hladce, počítač by měl odpovědět:

```
Compiling
Program 0 0909
0 error(s)
Compilation complete
```

Nedělejte si nyní starosti s detaily, ale vše, co se nyní odehrává, je, že počítač prochází váš program a překládá jej do strojového jazyka. Pokud neobdržíte hlášení: "0 error(s)", pak jste se pravděpodobně dopustili chyby při zadávání. Mohli byste zadat "NEW", čím byste vymazali program z paměti, a začít zadávat od začátku.

Po bezchybné kompilaci by se měl program automaticky spustit a vypsat hlášení:

```
ahoj
```

Jednou zkompilovaný program může být spuštěn kolikrát chcete, opětným zadáním:

```
r "RETURN"
```

Překlad se již neprovádí a počítač by měl vypsat svoje:

```
ahoj
```

Nyní se podívejme na program detailně. Hlavní tělo pascalového programu je vždy uzavřeno mezi slovy BEGIN a END, poslední END musí být vždy zakončeno tečkou. Pascalové programy sestávají z posloupnosti "příkazů", které jsou prováděny v tom pořadí, v jakém byly zadány. Náš příklad 1 má jeden příkaz - WRITE, který říká počítači, aby něco vypsal na obrazovku, v našem případě text "ahoj". Útvar mezi uvozovkami se nazývá řetězec a může obsahovat jakoukoliv sekvenci znaků, kromě "RETURN". Také samy uvozovky mohou být součástí řetězce, ale v tom případě musí být zdvojeny. Proto pascalovský program:

```
begin
write("O""brien""s string")
end.
```

vypíše na obrazovku zprávu:

```
O"brien"s string
```

PŘÍKLAD 2

Kromě řetězců mohou být vypisovány i jiné věci. Vyzkoušejte si následující program. Budeme používat téhož postupu jako v příkladě 1, ale nesmíme zapomenout nejprve vymazat program příkladu 1 z paměti počítače. Zadejte proto:

new "RETURN".

Nyní zadejte do počítače příklad 2.

begin
write (3+4);
WRITE (6-2-1)
end (písmena malé a velké abecedy jsou ekvivalentní)

Zadání ukončete příkazem pro kompilaci a spuštění

r "RETURN"

Po spuštění programu by měl počítač vypsat:

7 3

Příklad 2 obsahuje dva příkazy, které musí být odděleny středníkem. Je to příklad na celočíselnou aritmetiku (INTEGER).

2.1.2. PŘÍKLAD 3 - násobení a dělení

```
begin
write (6*7,18div4,18mod4,-(4+2)*3)
end.
```

Počítač by měl vypsat:

42 4 2 18

V Pascalu "*" znamená násobení

"div" celočíselné dělení (tj. se zaokrouhlením k 0)

"18mod4" dá zbytek při dělení 18:4

Všimněte si, jak se využívá závorek ke změně pořadí provádění operací -4+6 nebo 2. To proto, že počítač provádí násobení a dělení před tím, než provede sčítání a odčítání.

Jediným příkazem "write" může být vypisováno jakékoliv množství položek, jednotlivé položky musí být odděleny čárkami.

2.1.3. PŘÍKLAD 4 - funkce předdefinované v Pascalu

```
begin
write(sqr(4+5),abs(-44),abs(44),odd(3))
end.
```

Počítač by měl vypsat:

```
81 44 44 TRUE
```

SQR, ABS a ODD se nazývají funkce. V Pascalu existuje značné množství předdefinovaných funkcí:

SQR následované číslem v závorkách dává druhou mocninu tohoto čísla.

ABS dává absolutní hodnotu čísla.

ODD(3) je TRUE (pravdivý výrok), neboť 3 je liché číslo. Tato funkce dává logický (nebo říkáme bulský) výsledek, neboť může nabývat dvou hodnot TRUE nebo FALSE. Bulské hodnoty se v Pascalu užívají ve velké míře, a tak se s nimi seznamte blíže.

2.1.4. PŘÍKLAD 5 - bulské výrazy

```
begin
writeln(true,false,3=3,3=4);
write( 3<>4,5(6,9)=10 );
end.
```

Počítač by měl vypsat:

```
TRUE FALSE TRUE FALSE
TRUE TRUE FALSE
```

protože 3 se rovná sama sobě
3 se nerovná 4
atd.

= znamená "rovno"
< znamená "menší než"
> znamená "větší než"
>= znamená "větší nebo rovno"

<= znamená "menší nebo rovno"

<> znamená "nerovná se"

WRITELN má týž význam jako WRITE, ale generuje zároveň přechod na nový řádek po vypsání všech položek v závorkách.

PŘÍKLAD 6 - bulské výrazy

Mohou se vám zdát z počátku trochu komplikované, ale počítač je vyhodnocuje na základě striktní logiky.

```
begin
write(3>=nd( 3<5 ),(3=4)or( 3>11);
write(not true,not false,not(1=2));
end
```

Měli bychom obdržet výsledek:

TRUE FALSE FALSE TRUE TRUE

Protože současně (3=3) a (3<5) jsou pravdivé, ani jeden z výrazů (3=4) nebo (3>11) nejsou pravdivé a (1=2) není pravda, takže not(1=2) je pravda.

"x and y" je pravda, když současně x a y jsou pravdivé

"x or y" je pravda, když buď x nebo y (nebo oba zároveň) jsou pravdivé

"not" je pravda, když x je nepravda a naopak

2.2. PASCALOVÉ PŘÍKAZY

Nejprve několik slov o symbolech. Jsou to základní kameny pascalových programů a jsou tři základní druhy:

1. Pascalová klíčová slova. Například BEGIN a END. Jsou v Pascalu rezervovaná a uživatelem nemohou být měněna. Kompletní seznam naleznete v kapitole 3.1.1.
2. Specifické symboly jako . ; = > <.
3. Identifikátory, což jsou jména, která volí uživatel. Mohou to být sekvence písmen nebo číslic, ale musí vždy začínat písmenem.

Např:

Henrythe8th

PI

Upozornění: Identifikátory jsou chápány Pascalem jako různé, liší-li se v prvních osmi znacích, a tak Henrythe8th jsou pro Oxford Pascal týmiž identifikátory. Velká písmena jsou odpovídající jim odpovídajícím písmenům malé abecedy, a tak PI a pi jsou synonyma.

Některé standardní identifikátory jako např. WRITE, WRITELN a podobně jsou předdeklarovány ve všech verzích Pascalu. Mohou být uživatelem předefinovány, avšak musí se lišit od pascalových klíčových slov.

Důležité: Pascalové symboly mohou obsahovat mezery. "Henry the 8th" není však totožný s "Henrythe8th" a "30000" není ekvivalentem čísla 30000 (30,000 je také nevyhovující zápis). Obzvláště si všimněte, že ": =" nemůže být použito jako ":=".

Jinak mohou být mezery, tabulátory a přechod na nový řádek kdekoliv. V pascalovém programu jsou ignorovány.

Nyní se vraťte k několika skutečným pascalovým programům. Grafická úprava vložením některých mezer na začátku některých řádků není povinná, ale napomáhá čitelnosti a přehlednosti programu.

2.2.1. PŘÍKLAD 7 - proměnné a přiřazení

```
var x,y;integer;
begin
x:=3;y:=27;
writeln(x,y);
x:=4;
y:=x+2;
write (x,y,x+y)
end.
```

Mělo by se vypsat: 3 27 4 6 10

Deklarace var předchází begin a informuje kompilátor, že identifikátory x a y jsou proměnné, které mají uchovávat integer hodnoty. Jak vyplývá z názvu, proměnná může měnit svoji hodnotu v důsledku vykonávání programu. Na řádku 3 je proměnné x přiřazena hodnota 3 a proměnné y hodnota 27. Později je proměnné x přiřazena hodnota 4 a proměnné y hodnota x+2 (tj. 4+2=6).

Všimněte si, že můžeme též zapsat y:=y+2 (což by bylo 27+2=29). Proměnné mohou být též deklarovány jako bulské nebo jakéhokoliv typu.

2.2.2. PŘÍKLAD 8 - opakování pomocí smyček "for"

```
var integer;
begin
writeln("směr nahoru");
for i:=1 to 5 do writeln(i);
writeln ("směr dolů");
for i:=5 downto -1 do writeln (i);
end.
```

Mělo by se vypsat:

```
směr nahoru
1
2
3
4
5
směr dolů
5
4
3
2
1
0
-1
```

Příkaz následující po "for...do" (v tomto případě WRITELN) je opakován tak dlouho, dokud i nenabude všech požadovaných hodnot.

2.2.3. PŘÍKLAD 9 - příkaz "if"

```
var i:integer;
begin
for i:=1 to 11 do
begin
write (i);
if odd(i) then writeln ("je liché")
else writeln ("je sudé");
end
end.
```

Výsledek by měl být:

1 je liché
2 je sudé
3 je liché
4 je sudé
5 je liché
6 je sudé
7 je liché
8 je sudé
9 je liché
10 je sudé
11 je liché

"if" příkaz umožňuje počítači výběr, který ze dvou příkazů má udělat v závislosti na hodnotě bulského výrazu. (Připomeňme si, že bulský výraz může nabývat dvou hodnot v závislosti na výsledku logické operace - TRUE nebo FALSE.)

Část "else" podmínkového příkazu je nepovinná, ale co je DŮLEŽITÉ - před "else" nemůže být středník.

Příkazy "write" a "if" jsou v našem případě uzavřeny mezi

BEGIN ... END, aby vytvářely jeden složený příkaz, který má být jako celek prováděn smyčkou "for".

2.2.4. PŘÍKLAD 10 - nalezení aritmetického průměru

Tento příklad seznamuje s realizací vstupu dat z klávesnice a s obecnějším druhem smyček.

```
var :=0;count :=0;
begin
total :=0;count :=0;
write("zadejte libovolná čísla:");
repeat
read(x);
total := total+x;
if x>0 then count := count+1;
until x=0;
writeln("Aritmetický průměr je ",total/count);
end.
```

Po spuštění programu by nás měl počítač vyzvat k zadání série čísel. Sérii ukončete zadáním čísla 0. Zkuste tedy napsat např:

3 47 5 199 0 "RETURN"

Počítač by měl vypsat:

Aritmetický průměr je 6.35000E+01

Příkaz read(x) říká počítači, aby přijal z klávesnice číslo integer a uložil je do proměnné x. Pokud zadáme něco, co počítač neuzná za číslo integer, můžeme obdržet hlášení:

INTEGER READ ERROR line 60

a provádění programu se zastaví.

Operátor "/" provádí dělení v plovoucí desetinné čárce nebo s výsledkem reál (zatímco div dává výsledek integer). Více o reál aritmetice později.

Formát výpisu může být nastaven tak, abychom měli specifikováno celkové množství znaků, které budou vypsány, a také

množství číslic za desetinnou tečkou (zaokrouhlení se provede automaticky). Proto příkaz:

```
writeln ("Aritmetický průměr je:";total/count:10:3)
```

by měl vypsat:

```
Aritmetický průměr je 63.500
```

Číslo je doplněno čtyřmi úvodními mezerami tak, aby bylo dodrženo 10-ti znakové pole, které jsme specifikovali.

Smyčka "repeat ... until" jednoduše provádí příkazy, které jsou již uzavřené, dokud podmínka pro ukončení nenabyde hodnoty TRUE. V našem příkladě je smyčka uzavřena, zadáme-li číslo 0.

2.2.5. PŘÍKLAD 11 - příkaz "case"

Tento vzorový příklad předvádí mnohem rafinovanější způsob výběru mezi několika rozdílnými příkazy (srovnej s if).

```
var verse,i:integer
begin
 for verse :=1 to 4 do
  begin
    writeln;
    for :=verse downto 0 do
      case i of
        3:writeln ("tři muži");
        2:writeln ("dva muži");
        1:writeln ("jeden muž")
        0:writeln ("a jeho pes")
      end
  end
end.
```

Program by měl vypisovat:

```
jeden muž
a jeho pes
dva muži
jeden muž
```

a jeho pes
tři muži
dva muži
jeden muž
a jeho pes
case error line 70

Chybové hlášení bylo způsobeno, protože v poslední verzi nenabylo i hodnoty ani jednoho navěští z příkazu "case" - pro hodnotu i=4 nemůže příkaz case pokračovat. Návěští v příkazu "case" mohou být též skládána. Např.:

4,5,6:writeln ("mnoho mužů");

2.2.6. Poznámka k chybovým hlášením

Hlášení "case error" patří k "běžným" chybovým hlášením, protože bylo způsobeno při skutečném běhu programu. Je několik takových hlášení, se kterými se můžete setkat (viz kap. 8).

Zřejmě jste již začali experimentovat s vlastními programy. (To je pravděpodobně nejlepší způsob, jak zjistit, co je a co není v Pascalu možné.) Pokud tomu tak je, dříve nebo později narazíte na chybové hlášení kompilátoru. To se také mohlo stát, pokud jste udělali chybu při zadávání dřívějších uvedených příkladů.

Jednoduchý program:

```
var x:boolean;
x:integer;
begin
read(x);
write(x)
end.
```

Způsobí při kompilaci následující hlášení:

```
compiling
--- ERROR TYPE 30 LINE 20 NEAR X
program 0 090d
1 error(s)
```

Compilation complete

Poznámka: Číslo řádku může občas překročit jeden řádek nebo i více, v závislosti na tom, jak dlouhou dobu potřebuje kompilátor k detekci chyby.

Zadáte-li povel "L" jako odpověď po ready, začne se na obrazovce vypisovat program při jeho současné kompilaci. Všechna čísla řádku jsou označena. V našem příkladu by se tedy vypsalo:

```
10 var x:boolean;
20 x:ERROR46
20 x:integer;
30 begin
40 read (x);
50 write (x)
60 end.
```

Plná verze řádku 20 je znova vypsána pod chybovou zprávou. Počítač nedovolí spustit program, pokud v něm zbývá nějaká "KOMPILAČNÍ" chyba.

OPRAVOVÁNÍ CHYB

můžete provádět pomocí editoru, aniž byste museli přepisovat celý program. Abyste opravili výše uvedený program, zadejte nejprve:

"list" a program bude vypsán na obrazovce

```
10 var x:boolean;
20 x:integer;
30 begin
40 read (x)
50 write (x)
60 end.
```

K vymazání druhého řádku stačí zadat 20 následované "RETURN". Opětný výpis by měl vypadat:

```
10 var x:boolean;
20 begin
30 read (x);
40 write (x)
50 end.
```

Řádek 10 je však stále špatný. Chceme číst a vypisovat číslo integer a ne boolean. Proto zadejte:

change /boolean/integer/

Příkaz "list" by měl nyní vypsat správnou verzi programu:

```
10 var x:integer;
30 begin
40 read (x);
50 write (x);
60 end.
```

Program nedělá mnoho, pouze přečte číslo z klávesnice a vypíše je na obrazovce. Ucelenější přehled o tom, jak editor pracuje, najdete v souhrnu editorových příkazů v kapitole 7. Tam je též vysvětleno, jak ukládat vaše pascalové programy na disketu a znova je z ní zavádět do paměti počítače.

2.2.7. PŘÍKAZ "while"

Kromě smyček "repeat" a "for" existuje v Pascalu další druh. Příkaz "while" pracuje podobně jako "repeat" až na to, že test konce smyčky se provádí již na začátku smyčky (a tak smyčka nemusí být vůbec provedena). Zrovna tak jako u smyčky "for" může být opakován jenom jediný příkaz nebo sekvence příkazů uzavřená "begin ... end".

Např:

```
i:=1;
while i<=5 do
begin
writeln (i);
i:=i+1;
end;
```

má stejný efekt jako: foři:=1 to 5 do writeln (i);

2.3. VÍCE O DATOVÝCH TYPECH V PASCALU

2.3.1. PŘÍKLAD 12 - čísla v plovoucí čárce

```
begin
writeln (3.3,33.0,330.0,0.33);
writeln (-3.3E3,3.3E-1,4.5+2.1)
end
```

Počítač by měl vypsat:

```
3.30000E+00 3.30000E+01 3.30000E+02 3.30000E-01
-3.30000E 3.30000E-01 6.60000E+00
```

Přítomnost buď desetinné tečky nebo exponentu ("E") v čísle říká Pascalu, aby s ním zacházel jako s číslem v plovoucí čárce nebo s číslem reál.

Čísla v plovoucí čárce v Pascalu mají přesnost 9-ti číslic a jejich velikost může být v mezích asi 1E-38 až 1E+38. V porovnání s čísly integer byste neměli očekávat, že Pascalová reálná aritmetika bude tak přesná. To znamená, že např. 4.0 může být ve skutečnosti vypsáno jako 3.999999. Také se nemůžete spolehnout při testování reálných čísel na rovnost.

Např:

2.0+2.0=4.0 nemusí být vždy TRUE!

2.3.2. PŘÍKLAD 13 - aritmetika v plovoucí čárce

```
var x,y:real;
begin
x:=9.1;
y:=8.7;
writeln (x+y:7:2,x-y:7:2,x*y:7:2,x/y:7:2);
writeln (sqr(x):7:2,sqrt(x):7:2,abs(x):7:2);
write (trunc(x),trunc(y),round(x),round(y))
end.
```

Mělo by se vypsat:

17,80 0.40 79.17 1.05
82.81 3.02 9.10
9 8 9 9

Již dříve jsme se setkali s +, -, *, /. Užívají se hlavně pro sečítání, odečítání, násobení a dělení čísel reál ("div" znamená celočíselné dělení a spolu s "mod" by měly být používány jen s čísly integer).

2.3.3. SQRT, SIN, ARCTAN, LN, EXP, ROUND, TRUNC

SQR(x) znamená x umocnit na druhou
SQRT(x) znamená druhou mocninu z x
ABS(x) dává absolutní hodnotu z x
TRUNC(x) dává celočíselnou část x
ROUND(x) zaokrouhlí x k nejbližšímu celému číslu
SIN(x) dává sínus x (x je v radiánech)
COS(x) dává cosínus x (x je v radiánech)
ARCTAN(x) dává úhel v radiánech, jehož tangens je x
LN(x) dává přirozený logaritmus (o základu e) čísla x (pro x větší než 0)
EXP(x) dává číslo e povýšené na x-tou

1 radián = 57.29578 stupňů
e = 2.718281

2.3.4. PŘÍKLAD 14 - výstup s formátovanými konstantami

```
program vlny;
const f1=0.5;f2=0.05;amplituda=19;
var x1,x2,y:real;
begin
x1:=0;
x2:=0;
repeat
 x1:=x1+f1;
```

```
    x2:=x2+f2;
    y:=sin(x1)*sin(x2)*amplituda;
    writeln ((x):round(y)+amplituda)
  until false
  end.
```

Program by měl zobrazovat amplitudu modulované sínusové vlny.

Protože smyčka "repeat ... until ... false" v příkladu 14 se bude provádět téměř do nekonečna (přinejmenším dokud se x1 a x2 nestanou příliš velkými!), jediným způsobem, jak jej zastavit, je vypnout počítač. Kdybyste to však udělali, samozřejmě přijdete o program. Lepší způsob je jednoduše stisknout klávesu STOP.

Počítač by měl vypsat:

BREAK AT LINE xxxxx

kde xxxxx je řádek, který byl, když jste stiskli STOP. (Pokud se tak nestalo, zkuste to znova).

Hlavička programu je v Oxford Pascalu nepovinná a v tomto případě slouží pouze k pojmenování programu: vlny. Pro počítač nemá význam, je pouze pomůckou pro dokumentaci.

Jakýkoliv text uzavřený mezi párem symbolů (* text *) je kompilátorem také ignorován. Tato možnost může být využita pro zápis komentářů, které čtenáři pomohou porozumění programu. Konstanty uvedené klíčovým slovem "const" jsou hodnoty, které nemohou uvnitř programu svoji hodnotu měnit. Bude hlášena chyba, použijete-li na levé straně příkazu přiřazení (:=) nebo jako parametr v příkazu "read".

Deklarace konstant jsou užitečné, chceme-li dát jméno určitým konstantám (třeba PI = 3.1415926) a ty pak umožňují jednodušší změnu programu. Zkuste pomocí editoru změnit frekvence f1 a f2 a amplitudu v příkladě 14 a pozorujte, jak se vlna mění.

2.3.5. ZNAKY

Všimněte si, jak program specifikuje šířku výpisového pole (dvojtečkou, následovanou celočíselnou hodnotou), aby sdělil počítači, kolik znaků přiřadit "x", když se vypisuje; pokud je zažádáno o příliš mnoho znaků, je provedeno doplnění potřebným počtem mezer zleva. Pokud není zadán patřičný počet znaků, potřebný pro výpis řetězce, je řetězec zkrácen zprava.

Např:

```
write ("Ahoj":2)
```

by vypsal: Ah

Numerické hodnoty jsou vypsány celé, dokonce i když je specifikováno příliš málo znaků.

2.3.6. PŘÍKLAD 15 - grafika na C-64

```
const psloup =40;
var rad:=1,i:integer;
begin
page;
for rad:=1 to 24 do
  if odd(rad) and odd(i) or
  not odd(rad) and not odd(i) then write (chr(177))
  else write (chr(178))
end.
```

Tento program by měl vytvořit na obrazovce pravidelný vzor. Znaky jsou v Pascalu chápány jako řetězce délky 1.

Např:

```
' x ' ' ? ' ' ' '
```

patří k datovému typu "char", který má v Oxford Pascalu 256 různých hodnot, odpovídající souboru ASCII rozšířenému na C-64.

Funkce ord(z) dává celočíselný ASCII kód (mezi 0 a 255) znaků z.

Funkce chr(x) dává znak odpovídající ASCII celočíselnému kódu x (0-255).

Tak např.:

```
ord("?") = 63
chr("63") = "?"
```

Pozn.: Na C-64 byl datový typ "char" Oxford Pascalu rozšířen do rozsahu (0-255), což umožňuje používat 64 grafických znaků. Dva z výše uvedených znaků byly použity ve výše uvedeném příkladu. Zkuste zapsat programy, které vám vytvoří jiné obrazce, užitím jiných dostupných znaků.

Jednoduchou změnu můžeme udělat, nahradíme-li čísla 178 a 177 čísly 174 a 173. Příkaz "page" jednoduše vymaže obrazovku.

2.3.7. PŘÍKLAD 16 - pole

Představte si, že chcete načíst určitá čísla a vytisknout je v opačném pořadí. Museli byste tato čísla nejprve ukládat, neboť nemůžete začít s tiskem dříve, dokud nebude načteno poslední číslo. Pokud byste věděli, že vstoupí vždy tři hodnoty, mohli byste napsat:

```
var x1, x2, x3:integer;
begin
write ("Zadej tři čísla");
read (x1, x2, x3);
writeln (x3);
writeln (x2);
writeln (x1);
end.
```

Pro 50 hodnot by však takto zapsaný program vypadal trošku neelegantně. Proto použijeme polní proměnnou:

```
const n=3
var x:array[1..n]of integer;
1:integer;
begin
```

```
write ("Zadej",n:1,"čísel:");
for i:=1 to n do read (x[i]);
for i:=1 downto 1 do writeln (x[i])
end.
```

Po spuštění programu a zadání čísel:

463 79 980

byste měli obdržet: 980
79
463

Deklarace pole x zavádí v podstatě n proměnných, na které je možno se odkazovat indexem v hranatých závorkách. Prvky pole x jsou proto x[1], x[2], ..., x[n].

Konstanta n byla použita proto, aby se dal jednoduše měnit počet načtených hodnot pouze změnou jednoho programového řádku.

Prvky pole mohou být jakéhokoliv platného pascalového datového typu, včetně dalšího typu pole. To umožňuje vytváření dvojdimenzionálních i vícedimenzionálních polí.

Kupříkladu šachovnice můžeme nadeklarovat takto:

```
var šachovnice: array [1..8] of array [1..8] of figura
```

kde figura je určitý vhodný datový typ, který si uživatel nadefinuje podle potřeb (o tom více později). Na 5. čtverec ve 3. řadě šachovnice se lze pak odkázat jako na:

```
šachovnice[3][5]
```

Protože deklarace "array of array" se používá velice často, má pro ní Pascal zkratku:

```
šachovnice [3,5]
```

a podobně se to odrazí i v deklaraci:

```
var šachovnice: array [1..8,1..8] of figura;
```

Tento zkrácený zápis může být rozšířen na pole jakékoliv dimenze.

2.3.8. DEFINICE VAŠICH VLASTNÍCH DATOVÝCH TYPŮ

Žádný z datových typů, o kterých jsme se již zmínili (integer, real, boolean nebo char), by nebyl skutečně vhodný pro popis figur na šachovnici. Proto vám Pascal umožňuje vytváření vašich vlastních datových typů. To může být provedeno DEKLARAC TYPU.

Např:

```
type figura = (pěšec, střelec, věž, kůň, královna, král);
```

Proměnná typu figura pak může nabývat jednu ze šesti hodnot.

Např.:

```
var mojefigura, tvojefigura:figura
begin
. . mojefigura:=kůň;
tvojefigura:=královna;
```

. . Deklarace typů se musí provést za deklarací konstant a před deklarací proměnných. Identifikátory užité pro nečíselná data, jako je např. figura, musí být jedinečné, tj. nemohou se objevit v jiných nečíselných typech nebo být deklarovány jako proměnné či konstanty. I nečíselné typy mohou být ordinální (seřaditelné), a proto např. naše "figura" - prvky můžeme porovnávat pomocí relačních operátorů =, >, < atd.

Např:

```
král>královna
královna>kůň
```

Jsou pro ně definovány také tři funkce: PRED, SUCC a ORD.

pred(x) dává hodnotu předcházejícího x

succ(x) dává hodnotu následujícího x

ord(x) udává pozici x uvnitř datového typu (počáteční hodnota pěšec=0)

a tak:

```
pred(věž)=střelec
succ(kůň)=královna
```

ale pred(pěšec) a succ(král) nejsou definovány

ord(střelec) = 1
ord(kůň) = 3 atd.

2.3.9. PŘÍKLAD 17 - Eratosthenovo síto

Tento program vypíše všechna prvočísla od 2 do 127.

```
program Eratosthenes;
const n=127;
var sito:set of 2..n;
number,i:integer;
begin
sito:=[2..n];
for number:=2 to n do if number in sito then
 begin
 writeln (number);
 for i:=2 to n div number do
  sito:=sito-[i*number]
 end
end.
```

Prvočísla jsou celá čísla dělitelná jen sama sebou a 1. Naše síto, užité pro vyhledávání, je novým typem proměnné, která se nazývá MNOŽINA.

Množiny jsou v Pascalu soubory předmětů, které uzavřeme do hranatých závorek. Předmět pak buď do množiny patří, nebo nepatří. Proto:

[1,2,3]
[2,1,3]
[1,1,3,3,2] jsou navzájem ekvivalentní množiny.

Zkratka x..y v definici množiny či v přiřazení znamená, že do množiny patří všechny položky mezi x a y, včetně nich.

[1..4,10]=[1,2,3,4,10]

Pomocí operátoru "in" můžeme testovat, zda předmět do množiny patří či nikoliv. Proto "4 in [1..5]" dá bulský výsledek TRUE.

Typ prvků množiny může být jakéhokoliv skalárního typu (tzn. prvky množiny nemohou být pole), kromě typu reál. Hodnoty jsou omezeny rozsahem 0..127 (a tak v "set of char" jsou přípustné hodnoty chr(0)..chr(127)).

Nyní se vraťme k našemu programu. Začíná ve výpočtu od čísla 2 a pokračuje směrem vzhůru. Pokud je číslo ještě v množině sít, pak je prvočíslem. Jednoduše eliminujeme všechny součiny naposledy zjištěného prvočísla s čísly, která ještě zbývají v sítu, neboť ty nejsou prvočísla.

Operace povolené mezi dvěma množinami:

x+y x sjednoceno s y (všechny položky patřící buď do jedné z x nebo y množin nebo do obou)

x-y rozdíl x a y (výsledek je množina položek patřících do x, ale nepatřících do y)

x*y položky přítomné současně v x i v y (průnik x a y)

x=y test, jsou-li obě množiny totožné

x<>y test, nejsou-li dvě množiny totožné (disjunkce)

x<=y test, patří-li všechny prvky x také do y (x je podmnožinou y)

x>=y test, patří-li všechny prvky z množiny y také do množiny x (y je podmnožinou x)

2.4. PROCEDURY A FUNKCE

2.4.1. PŘÍKLAD 18 - procedury

```
var ch:char;
procedure lineof (worsit:char);
var i:integer;
begin
  for i:=1 to 30 do write (worsit);
  writeln
end; (* procedure lineof *)
begin (* začátek hlavního programu *)
```

```
lineof ([?]);
writeln;
for ch:=[a] to [f] do lineof(ch)
end.
```

Komentáře uzavřené v závorkách mezi symboly (* a *) nemusíte opisovat, jsou zde proto, aby vám program vysvětlily.

Počítač by měl vypsat:

```
??????...
aaaaaa...
bbbbbb...
cccccc...
dddddd...
eeeeee...
ffffff...
```

Procedury se používají k dělení hlavního programu buď proto, aby učinily program jasnější tím, že jej rozdělí na samostatné funkční bloky, anebo proto, abychom mohli volat tentýž podprogram z různých částí hlavního programu. Procedura "lineof" má parametr "worsit" typu char. Při volání procedury lineof musíme zadat též odpovídající skutečné parametry v závorkách. Pak lineof jednoduše vypíše řádek znaku, jenž jsme zadali jako parametr.

Pokud procedura nemá žádný parametr, závorky se jednoduše vynechají. Proměnná i v proceduře lineof je lokální - hlavní program o ní neví. Procedura lineof by však mohla, pokud by to bylo potřeba, používat proměnnou ch. Užívání lokálních proměnných pomáhá šetřit paměť, protože tyto proměnné jsou zrušeny, jakmile procedura skončí. Procedury jsou ve své podstatě mini-programy. Mohou mít své vlastní konstanty, datové typy a dokonce i své vlastní procedury.

"worsit" se nazývá hodnotový parametr, neboť je-li procedura zavolaná, je namísto něho dosazena hodnota skutečného parametru. Procedura pak sama může měnit hodnotu worsit, aniž by to mělo vliv na hlavní program.

Oproti tomu existuje parametr PROMĚNNÉ, který je při zavolání procedury nahrazen proměnnou.

2.4.2. PŘÍKLAD 19 - parametr proměnné

```
var x,y:integer;
procedure swap(var a,b:integer);
var temp:integer;
begin
  temp :=a;
  a:=b;
  b:=temp
end;
begin
x:=4;y:=77:
writeln (x,y);
swap (x,y);
writeln (x,y)
end.
```

Po spuštění bychom měli obdržet výsledek:

```
4 77
77 4
```

Všimněte si, že nevadí, kdybyste užili lokální proměnné, konstanty a parametry týchž jmen, jako jsou použity v hlavním programu. Např. v proceduře swap(var x,y:integer). Počítač si je nesplete (ale vy byste mohli). Proměnnové parametry a, b používá swap jako prostředníky k navrácení výsledku hlavního programu. Dalším způsobem, jak navracet hodnoty hlavnímu programu, je definování funkce.

2.4.3. PŘÍKLAD 20 - definování funkce

```
var i:integer;
function cube(x:integer):integer;
```

```
begin
 cube:=x*x*x;
end;
begin
for i:=1 to 20 do writeln ("Třetí
odmocnina",i:=2,"je",cube(i))
end.
```

Program by měl vypsat čísla od 1 do 20 a jejich třetí mocniny. Jak vidíte, funkce pracuje obdobně jako procedura, až na to, že specifikuje návratovou hodnotu.

2.4.4. PŘÍKLAD 21 - rekurze

Rekurzní funkce nebo procedura je taková, která volá sama sebe.

Užití rekurze umožní elegantní řešení takových hlavolamů, jakým jsou třeba "Hanoiské věže". V této dobře známé hře máme dvě hromádky disků. Začínáme s prázdnou 2. a 3. hromádkou a na 1. hromádce je určitý počet disků uložených podle velikosti, nejmenší disk leží na vrcholu. Cílem hry je přemístit disky na třetí hromádku tak, aby bylo zachováno pořadí (nejmenší na vrcholku). Přemísťovat se dá vždy jen jeden disk a nikdy nesmíme položit větší disk na menší.

```
program Hanoi;
var početdisků:integer;
procedure táhni(odkud,kam,náhrada:1..3;n:integer);
begin
if n>1 then táhni(odkud,náhrada,kam,n-1);
writeln ("Táhnu z"odkud:2,"na",kam:2);
if n>1 then táhni(náhrada,kam,odkud,n-1);
end;
begin
write ("Kolik disků ?");
read (počet disků);
```

```
writeln;
táhni (1,2,3,počet disků)
end.
```

Rekurzivní programy nejsou vždy nejefektivnější. Zacházejí nešetrně s pamětí, neboť počítač musí při každém volání ukládat proměnné do sklípku. Pokud zadáváte "počet disků" příliš veliké, pak se může počítač dostat mimo paměť a vypíše STACK OVERFLOW - line xxxx. Totéž hlášení obdržíte, deklarujete-li v programu více proměnných, než máte dostupné paměti, nebo se pokusíte kompilovat příliš velký program.

2.4.5. TEXTOVÉ SOUBORY

Textové soubory jsou speciální znakové proměnné, jejichž datový typ je text a pro něž je podstatné, že jde o řetězce znaků proměnné délky. Dva textové soubory jsou v Pascalu předdefinovány, "input" a "output" jsou normálně přiřazeny klávesnici, resp. VDU displeji.

Standardně je "input" implicitně obsazen v klíčových slovech READ, READLN, EOLN, EOF a "output" je v klíčových slovech WRITE, WRITELN, PAGE. Tak např. EOF je zkratka pro EOF(input). Podobně WRITELN("AHOJ") je ve skutečnosti zkratka pro WRITELN(output,"AHOJ").

Každému text. souboru přísluší "bufferová proměnná" typu CHAR (jméno souboru následované šipkou nahoru), např. input $.

VOLACÍ PROCEDURY

get(input) čte následující znak z klávesnice a uloží jej do proměnné input

put(output) zapíše obsah proměnné output do VDU

Nechť x je proměnná typu char:

read(x) je pak ekvivalentní zápisu X:=input$;get(input)

write(x) - " - output$:=x:put(output)

NEWLINE je speciální znak souboru. Po provedení přiřazení x :=input$ bude do proměnné x přiřazena mezera a funkce EOLN navrátí při dotazu hodnotu TRUE.

Funkce **READLN** je obdobná funkci READ, ale jak sám název označuje, provede před vlastním čtením přeskok na začátek následující řádky. Tuto činnost můžeme jinak popsat:

```
while not eoln do get(input);
get(input);
```

2.4.6. PŘÍKLAD 22 - řetězce

```
program rovwords;
const linesize = 64;
type string = packed array[l..linesize]of char;
var word:string;
  nchars,i:integer;
procedure skibblanks;
begin while (input={})and not eoln do
    get(input) end;
procedure swap(var s:string;i,j:integer);
var temp:char;
begin
temp:=s[i];
s[i]:=s[i];
s[j]:=temp;
end;
begin
repeat if eoln then readln;
  skipblanks;
while not eoln do
begin
 nchars:=0;
 repeat;
   nchars:=nchars+1;
```

```
  read(word[nchars]);
 until input$={};
 for i:=1 to nchars div 2 do
  swap(word,i,nchars-i+1);
 write(word:nchars,{ });
 skipblanks
end;
writeln;
until false
end.
```

Program čte slova a vypisuje je v obráceném písmosledu. Např.:

Mary had a little lamb (zakončené klávesou "RETURN")

způsobí vypsání textu:

yraM dah a elttil bmal

Program zastavíte stisknutím klávesy "STOP".

S řetězci délky n zachází Pascal jako s packed array[1..n] of char. Pakovaná pole (packed array) jsou organizována stejně jako pole obyčejná (array), ale jsou stlačena tak, aby zabírala co nejméně paměti. Prvky pakovaných polí nemohou však být používány přímo jako parametry v deklaracích "var", zatímco pakovaná pole jako celek ano, viz následující příklad.

```
var str:packed array[1..4] of char;
begin
str:="when":
writeln (str)"what"
end.
```

Tento program by měl vypsat TRUE, neboť "when" je větší než "what" (lexikologicky).

2.5. VLASTNOSTI PASCALU PRO POKROČILÉ

2.5.1. PŘÍKLAD 23 - záznamy

```
program clock;
const delay = 1600; (*přibližná konstanta pro C-64 *)
var i:integer;
clock:record
  hrs:0..23;
  mins,secs:0..59;
end;
begin
write ("Zadej čas v hodinách, minutách a vteřinách:");
read (clock.hrs,clock.mins,clock.secs);
with clock do repeat
for i:=1 to delay do (* čekej *); secs:=(secs+1)mod60;
  if secs=0 then
  begin
    mins:=(mins+1)mod60;
    if mins=0 then
      hrs:=(hrs+1)mod24;
  end;
writeln (hrs:1,{:},mins:1,{:},secs:1)
until false
end.
```

Program by měl po spuštění přibližně každou vteřinu vypisovat čas. Zadejte například čas v hodinách-minutách-vteřinách:

1 39 56

Program by měl začít vypisovat:

1:39:57
1:39:58
1:39:59
1:40:00 atd.

Tento příklad je určen pouze k ilustraci použití záznamu v Pascalu a nikoliv k náhradě hodin reálného času, které jsou vestavěny v C-64. Jejich použití viz kapitola 3.9.10. manuálu.

Záznamy tedy dovolují zkombinovat několik mezi sebou závislých proměnných do jedné struktury. Se záznamem pak může být nakládáno jako s celkem nebo můžeme pracovat s jeho složkami individuálně. Struktura WITH...DO říká počítači, aby zacházel s prvky "clock", jako by to byly lokálně definované proměnné v této struktuře. K těmto zpracovaným proměnným pak není třeba přidávat předponu "clock.".

Prvky záznamu mohou být jakéhokoliv typu (např. mohou být opět typu záznam nebo pole).

2.5.2. PŘÍKLAD 24 - ukazovátka

```
var p,q:$integer;
begin
new (p);new (q);
p$:=3;
q$:=4;
write (p$,q$)
end.
```

Program by měl po spuštění vypsat:

```
3 4
```

Proměnné p a q nejsou typu integer, ale "ukazovátka" na proměnné typu integer. Skutečný prostor pro proměnné je v tomto případě vytvořen "dynamicky", (jinými slovy, nevytváří se překladač, ale jsou vytvořeny až po spuštění programu) procedurou NEW. To umožňuje vytvářet programům struktury proměnných v potřebné fyzické podobě. Ukazovátka se většinou používají při zpracování dat ke spojování seznamů.

PŘÍKLAD 25 - reverzace řádky znaků použitím seznamu

```
program revchars;
```

```
type ukazovátko=$položka;
  položka=record
        znak:char;
        další:ukazovátko
  end;
var seznam,u:ukazovátko;
begin
seznam:=nil;
repeat
  new(u);
  read(u$.znak);
  u$.další:=seznam;
  seznam:=u
until eoln;
repeat
  write(u$.znak);
  u:=u$.další
until u=nil
end.
```

Po spuštění napíšeme třeba:

Mary had a little lamb (zakončete klávesou "RETURN")

Výsledkem bude výpis:

bmal elttil a dah yraM

Tento program rekurzivně definuje záznam, obsahující ukazovátko sám v sobě. Spojený seznam umožní pružně zacházet s pamětí, ale musíte pečlivě sledovat, na co právě ukazuje.

Ve standardním Pascalu existuje procedura DISPOSE(P), pomocí které můžete zrušit ukazovátko p, nebude-li již proměnná p$ v programu více potřeba.

V rezidentním režimu Oxford Pascalu nemá použití DISPOSE žádný účinek.

Pascalové klíčové slovo "nil" je taková hodnota ukazovátka, jež zajistí, aby ukazovátko neukazovalo na žádnou položku.

2.5.3. PŘÍKLAD 26 - příkaz "go to"

```
label 294,33;
begin
33:writeln ("Toto by mělo být vypsáno");
  go to 294;
  writeln ("Toto už ne");
294:writeln ("Setrvej ve smyčce");
  go to 33
end.
```

Výsledkem spuštěného programu by mělo být toto vypisování:

```
Toto by mělo být vypsáno
Setrvej ve smyčce
Toto by mělo být vypsáno
Setrvej ve smyčce
```

atd.

"Návěští" užívaná v příkaze "go to" musí být přirozená čísla a měla by být deklarována před konstantami, datovými typy a před proměnnými. Příkazům "go to" byste se měli vyhnout, kde jen to půjde, protože rozbíjejí strukturu programu. Obvykle se používají pro havarijní výběh z vhnízděných procedur. Skok dovnitř smyčky nebo procedury způsobí nepředvídatelný výsledek.

2.5.4. ROZŠÍŘENÍ OPROTI STANDARDNÍMU PASCALU

Jsou popsána v manuálu v kapitole 3.10. Připomeňme si alespoň jednu užitečnou proceduru - VDU.

PŘÍKLAD 27 - plnění obrazovky pomocí VDU

```
var i:integer;
begin page; (* page vymaže VDU obrazovku *)
for i:=1 to 40 do vdu(i mod 4,i,"x") (* VDU(i,j,z) uloží znak z
                        do řady i, sloupce j *)
end.
```

2.6. DISKOVĚ ORIENTOVANÉ OPERACE

Až dosud tento manuál hovořil jen o užívání rezidentního překladače, který je vždy přítomen v RAM. Ačkoliv to jsou ideální podmínky pro učení se Pascalu, nezbytně to omezuje počet použitelných příkazů a paměťový prostor pro uživatelské programy.

Až se důkladně seznámíte s Pascalem, budete chtít zapisovat i delší programy. Užitím diskově založeného kompilátoru a linkeru mohou být spouštěny pascalové programy 6000 a více řádků dlouhé a tento počet lze roszšířit využitím řetězení programů.

Možná budete chtít také psát programy, které budou mít přístup k disketovým datovým souborům. V rezidentním režimu je možné otevření kanálu pro disketovou jednotku, v režimu diskovém můžete používat úplnou pascalovou syntaxi, povolující soubory jakéhokoliv datového typu.

Diskový režim je navozen automaticky zadáním:

```
disk
```

To způsobí vymontování rezidentního kompilátoru z paměti a vytvoření většího prostoru pro editaci.

Je-li váš program již zeditován, může být uložen na disk. Např.:

```
put prog nebo put 0:prog
```

uloží soubor nazvaný "prog" na disketovou jednotku 0. Pro zkompilování programu zadejte:

```
comp prog
```

Kompilátor se vám ozve následujícími hlášeními:

```
Insert your system disk followed by a (RETURN)
(Vložte váš systémový disk a stiskněte (RETURN))
loading compiler...
(zavádění kompilátoru...)
Insert your data disk followed by a (RETURN)
(Vložte váš disk s daty a stiskněte (RETURN))
Pascal compiler vx.x
program 0 0009
```

0 error(s)
compilation complete
(ukončení kompilace)

Existuje množství různých dodatečných možností práce kompilátoru, např. pro současné generování výpisu atd. (viz 2.7.). Pokud jde vše dobře, kompilátor uloží přeložený kód do souboru "prog.obj", který je již spustitelný. Pokud se během kompilace vyskytnou nějaké chyby, pak musí být pascalový zdrojový program opraven pomocí editoru a znova zkompilován. (K načtení pascalového textu zpět do paměti užijte příkazu "get prog".)

Konečně zadejte:

ex prog

Čímž dojde ke spuštění programu (prog. obj).

Při kompilaci je vypisováno jméno každé procedury nebo funkce společně s její statickou úrovní vhnízdění (0 pro hlavní program, 1 pro procedury nebo funkce s nejvýše vnější úrovní vhnízdění atd.). Jsou vypsány též hexadecimální adresy, čímž získáte stručný přehled o relativní poloze programu v paměti.

Následující tabulka shrnuje rozdíly mezi rezidentním a diskovým režimem:

REZIDENTNÍ REŽIM

Kompilátor vždy v RAM
Pascalový zdrojový cílový program vždy v RAM
Rozdíly v implementaci jazyka (viz manuál)
Povoleny pouze textové soubory
DISPOSE je bez účinku (NOP)

DISKOVÝ REŽIM

Kompilátor je v RAM pouze během kompilace
Zdrojový i cílový program je udržován v diskovém souboru
Jsou dovoleny všechny typy souborů
Práce s diskovými soubory je plně programově podporována
Jsou implementovány PACK a UNPACK
Povoleno řetězení programů

2.7. SYSTÉMOVÉ PŘÍKAZY

ČÍSLO ŘÁDKU příkazy

Příkaz začínající číslem je vyhodnocen editorem jako nový programový řádek a je uložen v programovém textu na pozici odpovídající tomuto číslu. Např.:

```
10 end
5 begin (* tento příkaz bude zařazen jako první *)
```

Zadáte-li na řádku pouze číslo, bude to mít za následek vymazání řádku tohoto čísla z programového textu.

AUTO

Zapíná nebo vypíná automatickou generaci čísel řádku. Např.:

auto 20 - zapíná automatické číslování s přírůstkem 20

auto - vypíná automatické číslování řádku

Standardně je po zapnutí nastaveno auto 20.

LIST

Výpis programu z paměti. Např.:

list - výpis celého programu

list 330 - výpis pouze řádku 330

list 100- - výpis od řádky 100 výše

list 100-200 - výpis řádků od 100 do 200

list -200 - výpis řádků až do řádku 200 včetně

Pozn: Když během listingu zmáčknete klávesu STOP, dojde k úplnému zastavení výpisu. Pokud zmáčknete jinou klávesu, dojde k zastavení výpisu až do té doby, než klávesu stisknete podruhé.

UPPER, LOWER

Nastavování zobrazování ve velké nebo malé abecedě (obsah programu v paměti zůstává nezměněn). Standardně je po zapnutí nastaveno lower (malá abeceda). Přepínání mezi malou a velkou abecedou můžete také provádět současným stisknutím kláves SHIFT a COM.

BASIC

Návrat do BASICU se současným přepnutím na zobrazování ve velké abecedě.

DISK

Přechod do režimu diskového kompilátoru.

NEW

Vymaže program z paměti.

RESIDENT

Znovuzavedení rezidentního kompilátoru.

NUMBER

Přečíslování programových řádků v paměti. Např.:

number 1000,2000,30 - přečísluje řádky od 1000 výše, nová čísla řádku začínají od 2000 s krokem 30

Všimněte si, že nové počáteční číslo musí být větší nebo rovno starému počátečnímu číslu.

FIND

Nalezne a vypíše výskyt zadaného řetězce v programu. Např.:

find/function/ - najde čísla všech řádků, na nichž se vyskytuje řetězec "function".

find/function/,100-200 - najde všechny výskyty řetězce "function" na řádcích 100 až 200.

Znakem "/" můžete nahradit jakýkoliv znak vyskytující se v hledaném řetězci. Zastavit či pozastavit hledání lze stejně jako u příkazu list.

CHANGE

Pracuje podobně jako příkaz find, ale ve všech případech výskytu prvního zadaného řetězce provede jeho náhradu řetězcem druhým. Např.:

change /function/procedure/,150 - provede výměnu všech řetězců "function" za "procedure", které se vyskytnou na řádku 150.

Zastavit či pozastavit výměnu lze stejně jako u příkazu list.

DELETE

Vymaže programový řádek (parametry podobně jako v příkazu list). "delete" bez udání parametru je ekvivalentní příkazu "new".

PUT

Uloží program z paměti na disketu. Např.:

put 0:sara - uloží soubor nazvaný "sara" na disketovou jednotku 0 (jednotka musí být inicializována).

put 1:jim - uloží soubor na místo již existujicího souboru "jim" na disketovou jednotku 1 (jednotka musí být (inicializována).

GET

Čte program z diskety a ukládá jej do paměti. Např.:

get sara (prohledá diskety na obou jednotkách, je-li to třeba)

R (nebo RUN)

Spouští program, který je právě v paměti (a provede kompilaci, je-li to třeba).

L (pouze v rezidentním režimu)

Zkompiluje a zobrazí program na displeji.

P (pouze v rezidentním režimu)

Zkompiluje a zobrazí program na tiskárně.

COMP (pouze v diskovém režimu)

Zkompiluje program. Např.:

comp sara - zkompiluje soubor "sara" a vytvoří přemístitelný cílový soubor "0:SARA.OBJ"

comp sara,l - "L" - volíme současný listing na displeji
"P" - listing na tiskárnu
"N" - nevytváří se cílový soubor
"C" - neprovádí se kontrola rozsahu nebo číslování řádek (tím získáme trochu rychlejší a kompaktnější kód)
"1" - cílový soubor je uložen na jednotku 1

EX (pouze v diskovém režimu)

Spuštění programu cílového souboru. Např.:

ex sara - bude provádět cílový soubor "SARA. OBJ".

Pozn. Příkaz "comp" a "ex" vymaže textový buffer. Dojde k nastavení logického čísla tiskárny, typu tiskárny (ASCII/C64), příznaku automatického číslování řádku a je zavedeno "menu".

HEX

Převádí z decimální soustavy do hexadecimální:
hex 32 dá výsledek 0020

DECIMAL

Převádí z hexadecimální soustavy do dekadické:
decimal 7f dá výsledek 127

DUMP

Vylistuje program na tiskárnu. Příkaz "dump" má stejnou syntaxi jako příkaz "list".

COLD

Studený start BASICU C-64.

BASIC příkazy

V editoru může být v přímém módu použit jakýkoliv z následujících BASIC příkazů:

LOAD
PEEK
POKE
PRINT (nebo ?)
PRINTN
OPEN
CLOSE
CMD
SYS
FOR
LET

Např.:

let ti$="120000" - nastav interní hodiny na poledne
? ti$ - vytiskni čas
? fre(0) - vytiskni počet volných bytů
for i=1 to 20:? i,i*i:next

LINK (pouze v diskovém režimu)

U větších programů požadujeme (a často je to fyzicky nezbytné) mít možnost určité formy modularizace. Několik pascalových zdrojových souborů s vnitřními funkcemi a

procedurami mohou být zkompilovány odděleně a jejich cílové soubory mohou být potom spojeny do jednoho souboru. Linker (spojovací program) může být také použit k přípravě přímo spustitelných souborů.

Např.:

link 0:prog=prvprog,druhprog,třetprog

spojí soubory PRVPROG.OBJ, DRUHPROG.OBJ, TRETPROG.OBJ do jednoho cílového souboru PROG.OBJ na disketovou jednotku 0.

Pozn: "link" vymaže textový buffer.

OMEZENÍ:

a) Program určený k linkování musí mít totožné vnější deklarace proměnných.

b) Každá funkce nebo procedura vnější úrovně může být definována pouze v jediném ze souborů.

c) Pokud tuto funkci nebo proceduru potřebuje jiný soubor, měla by být v tomto souboru zahrnuta pod stejnou hlavičkou, přičemž tělo funkce nahradíme jediným klíčovým slovem "extern".

d) Předpokládá se, že první soubor uvedený v seznamu obsahuje hlavní program. Další soubory by měly normálně obsahovat pouze fiktivní hlavní program:

```
            begin
            end.
Např.:      soubor f1:
            program test (input, output);
            var i:integer;
            procedure x;extem; (* x je def. v dalším souboru*)
            procedure y;
            begin
              write (i);
            end;
            begin (* hlavní program *)
             x
            end.
```

```
soubor f2:
program testcast2 (input,output);
var i:integer; (* spol. proměnná musí být identická s f1 *)
procedure y;extern; (* y je def. v předchozím souboru *)
procedure x;
 begin
  i=3;
  write ("tři"); y;
 end;
begin (* fiktivní hlavní program *)
end.
```

Příkazová sekvence pro překlad a spuštění by měla vypadat:

```
comp f1
comp f2
link 0:test=f1,f2
ex test
```

Program by měl vypsat: tři = 3

ZAHRNUTÍ DALŠÍCH SOUBORŮ DO KOMPILACE

(jen v diskovém režimu)

Pokud znak "H" následovaný bezprostředně disketovým jménem souboru umístíme na počátek pascalového zdrojového řádku, dáváme tím kompilátoru najevo, aby byl obsah specifikovaného souboru zahrnut v tomto místě programu.

To je zvláště důležité, mají-li být spojovány programové segmenty jako jsou globální deklarace (které musí být totožné v každém spojovaném segmentu) a které mohou být udržovány ve zvláštním souboru, což usnadňuje budoucí změnu programu.

LOCATE (jen v diskovém módu)

Vytvoří požadovaný soubor, který pak může být zaváděn "pod BASIC" a prováděn pouhým zadáním RUN. Např.:

locate 0:xyz=jane - vytvoří cílový, pod BASICEM spustitelný, soubor xyz na jednotce 0 ze souboru JANE.OBJ

Pozn. "locate" vymaže textový buffer.

2.8. CHYBOVÁ HLÁŠENÍ

?SYNTAX ERROR - editorový příkaz má špatný pravopis nebo nesprávné parametry.

?OUT OF MEMORY ERROR - je nedostatek paměťového prostoru pro činnost, kterou jste specifikovali příkazem. Např. vkládání nového programového řádku nebo při zavádění souboru pomocí "get".

?ILLEGAL QUANTITY ERROR - špatné numerické hodnoty v editorovém příkazu.

?FILE DATA ERROR - jeden ze souboru pascalovské knihovny není na disku přítomen nebo byl znehodnocen.

COMPILER NOT RESIDENT - příkazy L, P a R mohou být užity jen v rezidentním režimu.

NO SOURCE PROGRAM - zadali jste příkaz L nebo R a programový text není přítomen v paměti počítače.

CHYBY VE SPUŠTĚNÍ PROGRAMU

1. **STACK OVERFLOW** (během kompilace) - program je příliš dlouhý, (během provádění programu) - program potřebuje příliš mnoho prostoru pro proměnné nebo používá mnoho úrovní rekurze.
2. **INTEGER READ ERROR** - bylo očekáváno zadání čísla z klávesnice.
3. **INTEGER OVERFLOW** - přetečení při násobení dvou čísel integer nebo DIV či MOD (dělení) nulou nebo TRUNC či ROUND příliš velkého čísla.
4. **ARRAY INDEX ERROR** - výraz užitý jako index pole je mimo deklarovaný rozsah.

5. **VARIABLE OUT OF RANGE** - proměnné nebo parametru procedury byla přiřazena hodnota mimo povolený rozsah daného typu dat.
6. **CASE ERROR** - neexistuje návěští odpovídající hodnotě selekčního výrazu v příkazu "case".
7. **BAD PCODE** - váš program byl znehodnocen nebo jde (doufáme že ne) o systémovou chybu. Objevuje-li se toto hlášení častěji náhodně, může to indikovat závadu paměti.
8. **SET VALUE ERROR** - množinový prvek je mimo rozsah.
9. **FLOATING POINT OVERFLOW** - může se objevit, pokud výsledek +, -, *, /, SQR nebo EXP je příliš velký.
10. **FLOATING POINT READ ERROR** - bylo očekáváno zadání čísla reál z klávesnice.
11. **UNDEFINED GO TO** - příkaz GO TO se odkazuje na neexistující návěští.
12. **COMPLEX LOG OR SQUARE ROOT** - pokus získat logaritmus nebo druhou odmocninu ze záporného čísla nebo logaritmus nuly.
13. **FILE NOT OPEN FOR READING** - "read" nebo "get" bez předchozího "reset".
14. **FILE NOT OPEN FOR WRITING** - "write" nebo "put" bez předchozího "rewrite".
15. **END OF FILE** - pokus o čtení souboru, jehož EOF je true.
16. **NO FREE I/O CHANNELS** - operační systém C-64 dovoluje mít současně otevřeno maximálně 10 souborů.
17. **DEVICE READ ERROR** - přišel špatný stavový byte v době čtení z IEEE sběrnice.
20. **až 72. DISK ERROR** - chybový stav byl detegován na disketové jednotce. Je navrácen typ chyby a pokud je to možné i jméno závadného souboru.

SOUHRN CHYBOVÝCH HLÁŠENÍ CBM FLOPPY JEDNOTKY

0 OK žádná chybová podmínka
1 odpověď, kolik souborů bylo vyškrtnuto z directory
2-19 nevyužitá čísla chybových hlášení
20 nebyla nalezena hlavička bloku
21 nenalezena synchronizační značka
22 požadovaný blok dat není přítomen
23 chyba kontrolního součtu v bloku dat
24 chyba v dekódování bytu dat
25 chyba při zápisu nebo verifikaci
26 pokus o zápis na disketu s ochranou proti přepisu
27 chyba kontrolního součtu hlavičky
28 dlouhý blok dat - data přesahují do dalšího bloku
29 odlišné diskové id
30 obecná chyba syntaxe - nesprávný příkaz
31 příkaz nezačíná na první pozici
32 příkaz má více než 58 znaků
33 nesprávné zadané jméno souboru
34 nebylo zadáno jméno souboru nebo je DOS nerozeznal
39 příkaz odeslaný do příkazového kanálu DOS nerozeznal
50 záznam není přítomen
51 přetečení záznamu
52 soubor je příliš velký
60 soubor je již pro zápis otevřen
61 soubor nebyl otevřen
62 soubor nebyl nalezen
63 soubor tohoto jména již existuje
64 zmatek v typech souboru
65 blok není volný
66 nepovolené číslo stopy nebo sektoru
67 nedovolená stopa nebo sektor
70 není již dostupný zadaný kanál
71 chyba directory

72 disk je plný
73 smíšení diskových operačních systémů
74 jednotka nepřipravena

3. ROZŠÍŘENÍ STANDARDNÍHO PASCALU

Záležitosti popsané v této sekci jsou specifické pro Oxford Pascal a nebudou proto implementovány ve stejném rozsahu na jiných systémech.

3.1. HEXADECIMÁLNÍ KONSTANTY

Jsou uvedeny symbolem $ (pro konstanty typu integer nebo pro znakové konstanty). Těžiště jejich použití je zvláště pro programy ve strojovém jazyce a pro práci s I/O zařízeními.

Příklady:

```
const portA = $e84f;
   linefeed = "a";
var chardata:char;
begin
 :
 :
 chardata := linefeed; (* linefeed je konstanta typu
CHAR *)
 poke (portA,$3f)
```

(zapíše data 3f hex do imagin. portu "A" na adrese e84f hex)

3.2. PAMĚŤ VDU A PŘÍSTUP K PORTŮM

Standardní funkce/procedury PEEK, POKE, ORIGIN, GETKEY a VDU slouží k následujícím účelům:

PEEK (x:integer):0..255

je funkce, která navrací obsah z fyzické paměťové lokace x, zatím co procedura

POKE (x:integer;y:0..255)

se používá ke změně obsahu na lokaci x bytem y. Poke by se mělo, samozřejmě, používat velice opatrně, abyste se vyhnuli znehodnocení vašeho programu.

ORIGIN (x:jakéhokoliv typu;y:integer)

nastavuje ukazovátko x na fyzickou lokaci v paměti y. x může být jakéhokoliv "pointer type". Také origin byste měli používat velice opatrně.

VDU (x,y:integer;c:char)

uloží znak c do paměti VDU (video unit), na řádek x, do sloupce y.

GETKEY :char

navrací znak, který přečte přímo z klávesnicového portu. Pokud nebyla stisknuta žádná klávesa, je navráceno chr(0).

Příklady:

```
var x:0..255;
begin poke ($014c,$33); (* uloží byte 33 hex na adresu
                           14c hex *)
  x:= peek(47); (* uloží do proměnné x obsah paměťové
                   lokace 47 dek *)
  page;
  vdu (0,3,"?"); (* vypíše na obrazovce v řadě 0, sloupci
                    3 znak otazník *)
    while getkey = chr(0) do; (* čekání na stisknutí
                                 klávesnice *)
```

3.3. PŘÍDAVNÉ PŘÍKAZY OXFORD PASCALU V. 1.0

ZVUKOVÉ PŘÍKAZY

V Oxford Pascalu jsou předdeklarovány tři zvukové procedury. Umožňují používání všech tří hlasů a generátoru obálky.

ENVEL(V,A,D,S,R):

parametry		typ	rozsah
V -	číslo hlasu	integer	1 - 3
A -	poměr náběhu	integer	1 - 15
D -	poměr sestupu	integer	1 - 15
S -	výdrže	integer	1 - 15
R -	doběhu	integer	1 - 15

VOICE(V,F,W,O):

parametry		typ	rozsah
V -	číslo hlasu	integer	1 - 3
F -	frekvence	integer	0 - 65535
W -	tvar vlny	integer	0 - 3
0 -	délka zvuku	integer	0 - 65535

Tvar vlny může být trojúhelník, pila, obdélník a šum.

Šířka obdélníkového signálu je přednastavena tak, že obě půlvlny jsou stejně dlouhé. Frekvenci určíme podle příručky Commodore 64 podle vzorce F(out) = (F* 0.059604645) Hz.

Výdrž je hodnota užitá ve zpožďovací smyčce a vztahuje se k době, kdy signál dosáhne své doby výdrže až k počátku doběhového cyklu. Realistických efektů dosáhneme, učiníme-li výdrž závislou na poměru délky násobené určitou konstantou.

Příkaz ENVEL musí předcházet příkazu VOICE, neboť VOICE způsobuje spouštění hlasu nastaveného předchozím ENVEL.

VOLUME(L):

parametr		typ	rozsah
L -	úroveň hlasitosti	integer	0 - 15

Příkaz VOLUME ovládá celkovou hlasitost SID čipu C-64. Může být nastaven kdykoliv na jakoukoliv velikost.

GRAFICKÉ PŘÍKAZY

Oxford Pascal dovoluje pomocí různých předdeklarovaných procedur a funkcí užívat grafiku C-64. Příkazy se dělí na příkazy pro normální užívání obrazovky a na příkazy pro práci v jemné grafice.

Příkazy normální grafiky:

BORDER(C): nastav barvu rámečku
SCREEN(C): nastav barvu obrazovky
PEN(C): nastav barvu pro výpis textu

C je číslo INTEGER v rozsahu 0 - 15 a udává barvu podle příručky C-64.

Příkazy pro práci v jemné grafice:

PAPER(C): nastav barvu pozadí
INK(C): nastav barvu bodu
HIRES(C): vypínač jemné grafiky (0-vypnuto, 1- zapnuto)
P:=EXAMINE(X,Y): testuj výskyt bodu (0-nevyskytuje se, 1-vyskytuje se)
PLOT(F,X,Y,X1,Y1): víceúčelový příkaz, všechny jeho parametry jsou typu integer

F=1 nastav pozadí na barvu danou PAPER
F=2 vymaž všechny body na obrazovce jemné grafiky
F=2 vykresli čáru z X,Y do X1,Y1
F=3 vymaž čáru z X,Y do X1,Y1
F=4 vybarvi plochu kolem bodu X,Y až k nejbližší hranici
F=5 vymaž plochu kolem bodu X,Y až k nejbližší hranici

Při kreslení nebo mazání bodu nastavujte X1,Y1=X,Y.

WINDOW(U): vytvoří textové okénko na obrazovce jemné grafiky, které začíná zezhora obrazovky a končí na řádku U.

3.4. HEXADECIMÁLNÍ VSTUP A VÝSTUP

WRHEX(f:text;x:integer) zapíše x jako 4 hexadecimální číslice do textového souboru f
WRHEX2(f:text;x:0..255) zapíše byte x jako 2 hexadecimální číslice do textového souboru f

Příklady:

```
rewrite(printer,4,0);
wrhex(printer,-1);wrhex2(output,3)
```

Natiskne FFFF na tiskárnu a na obrazovce napíše 03.

RDHEX(f:text):integer přečte 16-tibitovou hodnotu ze souboru f, přeskakujíce při tom jakékoliv úvodní mezery a znaky, jež nejsou hexadecimálními číslicemi. Dojde k přečtení posledních čtyř hexa číslic.

3.5. MANIPULACE S BITY

ANDB, ORB, XORB, NOTB, SHL jsou funkce s INTEGER parametrem, pracují však v podstatě s 16-tibitovými daty. Uvedené první čtyři funkce provádějí bitové AND, inkluzivní OR a vytvářejí 1. doplněk.

SHL(x,y) provádí logický shift (posun) x doleva o y-bitů (zprava vstupují nuly)

SHR(x,y) provádí logický shift x doprava o y-bitů

Pozn. shr(x,y) je ekvivalentní shl(x,-y)

Příklady:

```
andb($fff0,$00ff) = $00f0
orb($ff00,$000f) = $ff0f
xorb($ff00,$0ff0) = $f0f0
notb($f0f0) = $0f0f
shl(4,4) = $40
shl(4,-1) = $40000
shl(4,0)=shr(4,0) = $4
shr($4444,4) = $444
```

3.6. ZACHYTÁVÁNÍ I/O CHYB

Občas je nezbytné, aby byl program ošetřen proti výskytu neočekávaných zakončení vlivem nepřípustných vstupních údajů.

Volání procedury:

IOTRAP(false)

Vypne pascalová chybová hlášení pro real a integer vstupní operace a diskové I/O chyby.

IOTRAP(true)

Zapíná opět kontrolu.

IOERROR

Po každé integer, real nebo hex vstupní čtecí operaci lze funkcí IOERROR zjišťovat číslo chyby (integer číslo):

0 žádná chyba
2 integer read chyba
10 real read chyba
atd. (viz kapitola 2.8., kde je úplný seznam I/O chyb)

3.7. PŘERUŠENÍ OD KLÁVESNICE

Volání:

restore(true)
restore(false)

Povoluje a odstavuje funkci klávesy "restore". Standardně je nastaveno restore (true).

3.8. GENERÁTOR NÁHODNÝCH ČÍSEL

Funkce random:0..255 navrací náhodná čísla mezi 0 až 255. Je užito pseudonáhodné generační sekvence, ale náhodnost podporuje i to, že je tento generátor inicializován časováním vstupu z klávesnice a při spuštění pascalového interpretru.

Konstrukce:

random +(random mod 128) * 256 vám bude generovat náhodná čísla v rozmezí 0 až MAXINT.

random mod n+1 vám bude generovat (téměř) náhodná čísla z rozmezí 1..n, pokud nejsou příliš velká.

3.9. ZNAK "-"

Znak "-" (CBM klávesa spolu s ? klávesou) je povolen jako znak v identifikátorech, čímž se zvýší přehlednost (náhrada mezery, která není povolená jako znak identifikátoru).

3.10. INTERNÍ HODINY C 64

Reálný čas může být zjišťován pomocí tří funkcí:

```
hours:integer
minutes:integer
seconds:integer
```

a může být nastavován procedurou:

```
settime(h,m,s:integer)
```

Příklad:

```
settime(12,47,00); (* nastavení hodin na 47 minut po poledni *)
writeln(hours,{:},minutes,{:},seconds); by měl vypsat: 12:47:00
```

3.11. VSTUP ŘETĚZCOVÝCH PROMĚNNÝCH

Řetězcové proměnné, tj. "packed array[1..n] of char", mohou být čteny z textových souborů jako znaky, čísla real nebo integer. Jakékoliv úvodní mezery a přechody na nový řádek jsou nejprve přeskočeny a pak je celý řádek načten ze souboru do řetězcové proměnné. Pokud je řetězec příliš dlouhý, je odříznut zprava, pokud je krátký, je doplněn mezerami. Hlavní aplikací je zadávání jmen z klávesnice.

3.12. ŘETĚZENÍ PROGRAMŮ

(pouze v diskovém režimu)

Příkaz Oxford Pascalu:

chain(jméno souboru)

Zastaví provádění spuštěného programu a vyvolá program zadaného jména. Hodnota globálních proměnných bude zachována pouze tehdy, jsou-li deklarace identické ve starém i v novém programu. Všechny soubory jsou přitom uzavřeny.

Jako "jméno souboru" je možné použít buď řetězec nebo řetězcovou proměnnou (pokud je použita řetězcová proměnná, musí být jako zakončovač použitá přinejmenším jedna mezera).

Pokud je zřetězení použito pod příkazem EX, je v názvu souboru dodatek ".obj" doplněn automaticky.

Příklad:

"prog1" (cílový program v souboru "prog1.obj"):

```
begin
  writeln("První program");
  chain("prog2")
end.
```

"prog2" (cílový program v souboru "prog2.obj"):

```
begin
  writeln("Druhý program");
  chain("prog1")
end.
```

Příkaz: ex prog1 způsobí výpis:

```
První program
Druhý program
První program
Druhý program
:
:
```

až do té doby, než zmáčknete klávesu "STOP".

Řetězení programů je užitečnou technikou pro rozdělení programů na menší celky nebo pro aplikace se zaváděním programů podle menu.

4. INTERFACE PŘÍRUČKA OXFORD PASCALU

Posláním této kapitoly je poskytnout nezbytné informace, jak zapisovat programy ve strojovém jazyce mikroprocesoru 6502 pro Oxford Pascalové programy.

4.1. FORMÁT PRO SPOLUPRÁCI SE STROJOVÝM JAZYKEM

Podprogramy ve strojovém jazyce jsou deklarovány jako pascalovské funkce nebo procedury, ale jejich tělo je nahrazeno slovem "extern", následovaným integer konstantou (adresou rutiny). Všechny parametry jsou předávány prostřednictvím sklípku a měly by být převzaty rutinou ve strojovém jazyce. Rutina by pak měla do sklípku strčit návratovou hodnotu, je-li deklarována jako funkce. Nejlepší pro pochopení bude následující příklad jednoduché funkce, která sečítá dvě čísla integer:

```
program test;
function addxy(x,y:integer):integer;
extern $C000;
begin
write(addxy(3,4))
end.
```

Výsledkem by mělo být vypsání:

```
7
```

Sečtení provedla rutina ve strojovém jazyce, přemístěná na adrese $C000.

```
sptr =$2A ; ukazovátko sklípku pascalu
```

```
* =$C000
 addxy clc
    ldy MO
    lda (sptr),y spodní byte y
    ldy M2
    adc (sptr),y přičtení spodního bytu x
    sta (sptr),y spodní byte výsledku
    dey
    lda (sptr),y horní byte y
    ldy M3
    adc (sptr),y přičtení horního bytu x
    sta (sptr),y horní byte výsledku
    clc
    lda sptr
    adc M2 vystrč y, ponechej výsledek
    sta sptr
    bcc addrts
    inc stpr+1
 addrts rts
```

Pozn.: Ukazovátko sklípku se nachází na $2A, $2B. Ukazovátko konce paměti (top-of-memory) na $37, $38. Mělo by být nastaveno na $C000 nebo níže, aby se zabránilo Pascalu přepsání těchto lokací.

4.2. FORMÁT ULOŽENÍ

Všechny skaláry a jejich podtypy s omezením rozsahu (kromě typu real) a ukazovátka jsou uložena jako šestnáctibitová slova v obvyklé formě (spodní, dolní byte).

Reálná čísla jsou uložena v šesti bytech, v basicovém C-64 formátu.

loc n+5 : nevyužity
loc n+4 : LS mantisa

loc n+3 : .
loc n+2 : .
loc n+1 : MS mantisa
loc n : exponent

Pole jsou uložena řádek po řádku (na rozdíl od FORTRANU), nejnižší prvek má nejnižší adresu. Pole jsou pakovaná, pokud jejich prvky jsou skaláry menší než 0..255 (např. char) a bylo zadáno "packed". V tomto případě je velikost zaokrouhlena vždy nahoru k sudému počtu byte. Záznamy jsou uloženy s položkami v opačném pořadí (nejvýše deklarované mají nejvyšší adresu).

Množiny jsou uloženy ve 128-bitové mapě a "1" indikuje, že daný prvek do množiny patří. Liché a sudé byty jsou reverzovány:

loc n+15 bit 15....bit 8
loc n+14 bit 7.....bit 0
:
:
loc n+1 bit 127...bit 120
loc n bit 119...bit 112

Důležité: Ukazovátka ukazují vždy na lokaci nad nejvyšší byte užitý pro skutečná data. To se týká též parametru VAR, které jsou chápány jako adresy.

Příklad:

```
const VDUSIZE = 1000; (* velikost paměti pro displej je
                         25 řad po 40 znacích *)
type screen = packed array[1..VDUSIZE] of char;
var vduptr: screen
begin
origin (vduptr, $0400+VDUSIZE)
:
:
```

Tento program deklaruje pole začínající na adrese 0400 hex (což je obrazovková paměť C-64). V ní je potom vduptr[1] první lokace VDU displeje.

DOPLNĚK

Directory můžete nahrát do paměti podobně jako v BASICU:

LOAD"$0",8 pro jednotku 0 (nebo LOAD"$",8)

LOAD"$1",8 pro jednotku 1

Upozornění: Zavedením directory do paměti si přepíšete váš program.

Příkaz LOCATE

Dejte si pozor, abyste nedali souborům spustitelným v BASICU též jméno, jaké má váš zdrojový soubor (nebo jiný pascalový soubor na témže disku), aby je Commodore nepřepsal na jiný, s rozdílným typem.

Chyby při kompilaci:

Chybová hlášení v rezidentním módu obsahují nejen číslo chyby, ale i příslušné chybové hlášení. Buďte si vědomi toho, že se budete muset podívat na příčinu výskytu chyby někdy i několik řádků zpět, neboť její potvrzení může nastat až hlouběji v programovém textu. Může se také stát, že jedna chyba může způsobit řadu souběžných chybových hlášení, která zmizí současně s odstraněním chyby.

Jemná grafika - zvláštní poznámky:

1. Ohraničení pracovní plochy je ve směru souřadnice x 0..255, ve směru souřadnice y 0..200. Počátek 0,0 je ve spodním levém rohu obrazovky. Jakýkoliv bod zadaný mimo toto ohraničení nebude zobrazen.
2. HIRES(1) standardně obsadí celou obrazovku a je tedy ekvivalentní příkazu WINDOW (25).
3. Doporučuje se, aby byl příkaz HIRES(0) používán před ukončením programu, který užívá jemnou grafiku, pokud není vyloženě požadováno nechat okénko, zobrazující část displeje. Pokud tento příkaz chybí nebo je program zastaven dříve než mohlo dojít k provedení tohoto příkazu, může být stejná akce provedena z editoru příkazem KILL (je-li to třeba, zadejte jej

naslepo). KILL vypíná jemnou grafiku a odstavuje "grafický programový klín" z interrupční rutiny.

4. K uvolnění a vymazání displeje v jemné grafice by měly být použity obě funkce 0 a 1 příkazu PLOT.
5. Oxford Pascal používá pro jemnou grafiku standardní mód bitové mapy, a proto, i když máme možnost volby 16-ti barev, v jedné pozici znaku může být definována jen barva pozadí (PAPER) a barva popředí (INK). Jedna znaková pozice má velikost 8 x 8 bodů. Toto omezení poznáte, když dojde ke zkřížení dvou linií různých barev a v tomto místě dojde ke změně barvy křížené linie.

Užitečné paměťové lokace:

HEX DEC UŽITÍ

C000 49152 logické číslo tiskárny v edit. režimu (standardně 4)

C001 49153 typ tiskárny - standardně 1 (0 = 64 typ, 1 = ASCII)

C002 49154 příznak řádkování standardně

0 - automatické řádkování povoleno

1 - vypnuto

COMMPAS
vydavatelství, služby a distribuce
Dr. Ivan Pavlíček, Pavel Škvrně
p. o. box 80
140 00 Praha 4