zpravodaj atari klubu

přerov

x/88

Vydává 487. ZO Svazarmu —
ATARI KLUB v Praze 4.
Šéfredaktor a vedoucí redakční rady JUDR. Jan Hlaváček.
Zástupce šéfredaktora ing. Stanislav Borský.
Obálku navrhl RNDr. J. Tamchyna.
Adresa redakce:
487. ZO Svazarmu - ATARI KLUB Praha
REDAKCE
poštovní přihrádka 51
100 00 Praha 10
Řídí redakční rada: V. Bílek, ing. J. Biskup, RNDr. J. Bok, CSc., ing. S. Borský, ing. V. Friedrich, ing. O. Hanuš, RNDr. L. Hejna, CSc., Z. Lazar, prom. fyz., CSc., F. Tvrdek, ing. M. Vavrda.
Technická redakce Otilie Strnadová.
Otisk povolen se souhlasem redakce při zachování autorských práv a s uvedením pramene. Rukopisy nevyžádané redakcí se nevracejí. Za původnost a věcnou správnost ručí autor.
Vychází šestkrát ročně. Neprodejné. Členům klubu distribuováno zdarma. Nepravidelné přílohy na objednávku jsou kompenzovány zvláštním klubovým příspěvkem.

TISK SNV zak. č. 60/89

Vydávání schváleno OV Svazarmu Praha 4 a OŠK ONV Praha 4.
Evidenční číslo ÚVTEI 86 042.

příloha X

KYAN PASCAL

Miroslav Ondříšek

Recenze
ing. Václav Friedrich

KYAN PASCAL - OBSAH

1. Úvod

Historie Pascalu, dnes jednoho z nejoblíbenějších jazyků, začíná v roce 1960, kdy Niklaus Wirth, profesor počítačových věd, vyvinul nový jazyk, který byl určen pro výuku programování. V roce 1971 byla stanovena oficiální norma jazyka Pascal. Výhody a velká obliba tohoto jazyka vedla firmu KYAN SOFTWARE k tomu, aby ho implementovala také na počítače ATARI řady XL/XE. Tento jazyk je určen pro práci s disketovou jednotkou. Díky odborně fundovaným lidem v Československu se dnes z tohoto jazyka mohou těšit též majitelé počítačů ATARI 130 XE, kteří vlastní mimo počítače jen kazetový magnetofon vybavený systémem TURBO 2000(příloha pražského ATARI Zpravodaje II/1987). To byl také jeden z důvodů, které mě vedly k napsání této příručky, která však nemá být učebnicí Pascalu, ale podrobným návodem k programu a základním popisem jazyka s příklady použití.

V době dokončení této publikace jsem neměl důvěryhodné zprávy o implementaci tohoto jazyka na počítače ATARI 800 XL/XE, které nejsou vybaveny disketovou jednotkou.

A ještě jedno upozornění, než začnete se studiem této publikace. Protože textový editor, ve kterém jsem psal tuto publikaci,má omezenou znakovou sadu, která neobsahuje dva znaky používané v Pascalu, byl jsem nucen tyto znaky nahradit jinými. Proto prosím čtenáře, aby si v uvedených výpisech programů dosadil místo znaku uvozovek (") znak pro apostrof (SHIFT - 7) a místo znaku lomítko (/) střídavě levou a pravou hranatou závorku (SHIFT - , a SHIFT - .). Děkuji čtenáři za pochopení a na závěr bych chtěl ještě poděkovat ing. Václavu Friedrichovi z pražského ATARI klubu, který tuto práci redigoval po odborné stránce.

Autor

2. Funkce editoru a překladače

Kyan PASCAL se skládá ze dvou programů: editoru <ED> a překladače / assembleru <PC>.Při zapnutí počítače na krátkou dobu podržíte klávesu <OPTION>. Při správném nahrání se objeví následující hlavička:

```
KYAN PASCAL VERSION 1.0
COPYRIGHT 1985 BY KYAN SOFTWARE
1850 UNION STREET, SUITE 183
SAN FRANCISCO, CA 94123
```

2.1. editor textu

Použitý editor je obměnou rozšířeného textového procesoru Word Star. Pokud jste zvyklí s tímto editorem pracovat, nebude vám činit přechod na editor Kyan Pascalu problémy.

Editor načtete do paměti tak, že zadáte: >D1:ED a <RETURN>
Všechny příkazy, které operují s diskem, musí mít na svém začátku "D1:" (popř. "D:"), pokud jde o disk číslo 1. Pokud vlastníte více diskových jednotek, musíte použít prefix D2:, D3: atd. Symbol ">" znamená, že počítač čeká na Váš příkaz, který musíte ukončit stisknutím klávesy <RETURN>.

2.1.1. Vytváření souborů

Jestliže máte načtený editor, můžete přistoupit k vytvoření svého prvního programu, např. s názvem TRIAL. Zadáte tedy: D1:TRIAL. Protože tento soubor na disku ještě neexistuje, objeví se hlášení: "FILE NOT FOUND, PRESS ANY KEY".<Soubor nenalezen, stiskni libovolnou klávesu>. Když stisknete jakoukoliv klávesu, obrazovka se vymaže. Vše, co od tohoto okamžiku napíšete, bude chápáno již jako část Vašeho programu (TRIAL).
Vložte např. řetězec šesti znaků X:

XXXXXX

Nyní ukončíte editaci a soubor zapíšete, jak je popsáno dále.

2.1.2. Ukončení editace

Nejprve stiskněte <ESC> a objeví se menu se zvláštními editačními funkcemi (klávesa <ESC> odpovídá kombinaci kláves <CONTROL> a <K> u editoru Word Star). Zpět do Vašeho programu se dostanete opakovaným stiskem klávesy <ESC>.
Jsou tři možnosti, jak ukončit editaci:

1 - <ESC> S (Save and Resume)
2 - <ESC> Q (Quit without Save)
3 - <ESC> X (Save and Exit)

Ad 1 - Tento příkaz zapíše Váš program na disk a vrátí Vás

opět do editoru.
Ad 2 - Tímto způsobem opustíte editor aniž byste zapsali vytvářený soubor.
Ad 3 - Takto zapíšete svůj program na disk a potom opustíte editor - objeví se (>) - a program čeká na další příkazy. Pokud se chcete opět dostat do editoru, stačí zadat "!" a <RETURN>.

2.1.3. Soubory a jména souborů

Programy jsou uchovávány jako textové soubory. Každý program uchovaný na disku musí mít své vlastní jméno. Jméno se skládá z prefixu D1: a následuje max. 8 znaků + 3 znaky rozšiřující (např. D1:XXXXXXXX.YYY), mezery nejsou akceptovány. Pascalská jména, tj. jména proměnných, typů atd., se neshodují se jmény souborů!

Jestliže chcete změnit jméno souboru, který vytváříte, stiskněte <ESC> P a objeví se dotaz: FILE NAME:
Nyní vložte nové jméno souboru, odešlete klávesou <RETURN> a stiskněte <ESC>:

FILE NAME: D1:NAZEV <RETURN><ESC>

2.1.4. Ovládání kursoru

Všechny operace s kursorem se vytvářejí kombinací klávesy <CONTROL> a další klávesy (Pozn.: Znak pro umocňování "^" - <SHIFT-^> je zkratka pro klávesu <CONTROL>)
Jde o kombinace:

<CTRL> S	kursor zpět o 1 pozici
<CTRL> D	kursor vpřed o 1 pozici
<CTRL> E	kursor o řádek zpět
<CTRL> X	kursor o řádek vpřed
<CTRL> A	kursor zpět o jedno slovo
<CTRL> F	kursor vpřed o jedno slovo
<CTRL> R	kursor o 20 řádků zpět
<CTRL> C	kursor o 20 řádků vpřed
<CTRL> T	kursor na začátek programu
<CTRL> V	kursor na konec programu

K pohybu vpravo, vlevo, nahoru a dolů můžete též použít kursorové šipky. Jestliže klávesu chvíli podržíte, dojde k opakovanému pohybu kursoru.

2.1.5. Mazání v editoru

Jde o kombinace kláves:

<CTRL> G	vymazání znaku za kursorem
<CTRL> Q	vymazání znaku vlevo od kursoru
<CTRL> Y	vymazání řádku na kterém je kursor

Funkce <CTRL> Q je shodná s funkcí klávesy <DELETE BACK SPACE>

2.1.6. Hledání a záměna řetězců

Retězec je libovolná kombinace znaků (písmen, číslic a symbolů) a mezer. Retězec může být jak jedno slovo, tak i skupina slov. Popisované funkce Vám pomohou nalézt nějaký řetězec v programu nebo zaměnit jeden řetězec za jiný:

<ESC> - dostanete se do menu editoru
A - vkládání hledaného řetězce
B - nastavení nového řetězce, kterým nahradíme nalezený řetězec
C - nastavení režimu výměny

Zadáte-li "A", objeví se na obrazovce:

FIND STRING:

Nyní vložíte řetězec, který chcete nalézt. Např. pokud chcete nalézt "otevreny obchod", napíšete "otevreny obchod" a <RETURN>:

FIND STRING:otevreny obchod <RETURN>

Pak stisknete <ESC> a tím opustíte režim se zvláštními funkcemi a vrátíte se do svého programu. Zde se pomocí <CTRL> W (zpět) a <CTRL> Z (vpřed) můžete pohybovat po nalezených řetězcích. Maximální délka hledaného řetězce je 40 znaků.

Abyste mohli vyměnit jeden řetězec za jiný, musíte zadat oba řetězce (A i B). Až oba řetězce nastavíte, stiskněte v modu menu klávesu "C". Objeví se dotaz:

CHANGE ALL STRINGS OR SOME (A/S/Q)?

-S- znamená, že editor se na každém nalezeném řetězci zastaví a zeptá se, zda jej chcete vyměnit (odpovídá se Y=ano, N=ne, Q=konec režimu výměny).
-A- znamená, že chcete vyměnit všechny nalezené řetězce.
-Q- je ukončení tohoto režimu práce.

Chcete-li např. vyměnit všechny řetězce "první" za "1.", provedete tyto kroky:

1. Napsat: <ESC> A
2. Objeví se: FIND STRING:
3. Napsat: první<RETURN>
4. Napsat: B
5. Objeví se: REPLACE STRING:
6. Napsat: 1.<RETURN>
7. Napsat: C
8. Objeví se: CHANGE ALL STRINGS...
9. Napsat: A

2.1.7. Editování na určitém řádku

Jednotlivé řádky se automaticky číslují. Tato čísla však nejsou součástí textu programu. Používá je však překladač při hlášení chyb, jak je uvedeno dále.

Pokud chcete editovat na určitém řádku, použijte

příkaz:

<ESC> G

Objeví se hlášení "LINE NUMBER", za které zadáte číslo řádku, který chcete editovat.

2.1.8. Vkládání souboru

Jestliže chcete vložit nějaký soubor do Vašeho programu, použijte příkaz:

<ESC> I

Objeví se dotaz "FILE NAME OF FILE TO INSERT (BLANK TO QUIT)"
Zde vložíte název existujícího souboru, který chcete vložit (včetně názvu zařízení, ze kterého se bude soubor číst) nebo mezeru, chcete-li tuto instrukci opustit. Specifikovaný soubor se pak načte do textu programu od pozice kursoru, přičemž text za kursorem odsune.

2.1.9. Přesun bloku

Editor Vám umožní přesunout nebo zkopírovat libovolnou část programu (blok) na jiné místo. K zadání bloku:

1) Umístěte kursor na znak nebo mezeru, kterým blok začíná a stiskněte <CTRL> O. Všimněte si, jak se celý blok přepíná do inverze, když pohybujete kursorem.
2) Umístěte kursor na poslední znak bloku a stiskněte <CTRL> O. Zdá se, že celý blok zmizel, ale je uložen v paměti, takže může být umístěn kdekoliv v textu.
3) Umístěte kursor na místo, kam chcete vložit blok a stiskněte <CTRL> P. Takto můžete vytvořit libovolné množství kopií.

2.1.10. Vytvoření programu POKUS

Načtěte editor příkazem D1:ED
Napište D1:POKUS
Nyní vložte následující program:

```
PROGRAM Pokus;
BEGIN
  WRITELN('Mnoho zdaru');
  WRITELN('Vám přeje ATARI')
END.
```

Po stisku <ESC> X zapíšete program a opustíte editor.
V následujících řádcích naleznete postup, jak tento program přeložit a spustit.

2.2. Překlad souboru

Dříve než můžete spustit pascalský program, musí být zkompilován, tj. přeložen do strojového jazyka. Výsledek překladu Vašeho zdrojového pascalského souboru je soubor v symbolickém kódu (assembler). Tento musí být dále přeložen Assemblerem a jeho výsledkem je tzv. soubor typu "object" (tj. cílový soubor).

Po opuštění editoru načteme překladač zadáním D1:PC <RETURN>. Po načtení programu se objeví hlášení :

```
PC>
```

Nyní vložte jméno pascalského programu, který chcete přeložit a případné další volby ve tvaru:

```
PC>D1:JMENO-L-E-O <RETURN>
```

"L" sděluje počítači zda chcete výpis assemblovaného programu a na jaké zařízení: "L" znamená obrazovku a "LP" obrazovku a tiskárnu.
"O" mění jméno hotového souboru a to tak, že:
1) Nezadáte-li nic, bude mít výsledný soubor stejné jméno jako zdrojový program s rozšířením .O .
2) Chcete-li jiné jméno, zadáte např. -O D1:NOVEJMENO.
3) Zadáte-li -O bez jména, nebude se výsledný program generovat na disk. Chcete-li však program spustit, je nezbytné, aby se zapsal na disk.
"E" určuje, kam se zapíše výpis chybových hlášení překladače:
-E:Chybové hlášení na obrazovku,
-EP:Chybové hlášení na obrazovku a tiskárnu.
Po stisknutí <RETURN> se spustí překlad. Jestliže se neobjeví žádná chyba, uloží se výsledný program na specifikované zařízení a objeví se ">". Nyní lze program přímo spustit tím, že napíšete jeho jméno (rozuměj jméno přeloženého programu).
Překlad programu Pokus:

1. obrazovka: >
2. napsat: D1:PC
3. obrazovka: PC>
4. napsat: D1:POKUS-L-E <RETURN>

Překladač lze, stejně jako editor znovu spustit pomocí "!". Pokud chcete nyní spustit program POKUS, napište:

```
>D1:POKUS.O
```

Program se nahraje a automaticky spustí. Na obrazovce se objeví:

```
Mnoho zdaru
Vám přeje ATARI
```

Upozornění: Pokud spouštíte program, musí být na stejném disku program "LIB", který umožní spuštění programu.

Přesvědčíte se o tom tak, že napíšete:

```
>DOS
```

Bližší informace pro práci s Vaším operačním systémem DOS viz příručka DOS. Zpět do PASCALu se dostanete z DOSu příkazem "L"<RETURN> (Binary Load) a pak "B"<RETURN>.
Pokud chcete Váš program vypsat na tiskárně, zadejte:

```
>D1:PRINT.O
```

a po dotazu: "FILE NAME?" napište jméno programu, který chcete vytisknout.

2.2.1. Chybová hlášení překladače

Abyste viděli, jak překladač hlásí chyby, uděláte v předešlém programu úmyslně chybu.
Po nahrání programu POKUS umístíte kursor na řádku:

```
WRITELN('Mnoho zdaru');
```

a místo středníku napíšete tečku. Řádka nyní vypadá následovně:

```
WRITELN('Mnoho zdaru').
```

Opustíte editor příkazem <ESC> X, nahrajete překladač (D1:PC) a pokusíte se program POKUS přeložit:

```
PC>D1:POKUS
```

Protože se v programu vyskytla chyba, objeví se na obrazovce výpis chyb:

```
0003    WRITELN('Mnoho zdaru').
                               1
 (1) ';' OR 'END' EXPECTED
```

0003 znamená číslo řádku Vašeho programu, na kterém se chyba vyskytla, jednička pod tečkou označuje místo a číslo chyby na tomto logickém řádku. Pod tím je vysvětlen typ chyby (v tomto případě: Je očekáván ";" nebo slovo "END").

2.2.2. Zastavení běhu programu

Pokud v programu vznikne nekonečná smyčka nebo jej chcete zastavit z jiných důvodů, stiskněte klávesu <BREAK>. Poté se vypíše obsah zásobníku, který obsahuje návratové adresy, pak stavové slovo a obsah AKUMULÁTORU (registr A procesoru).

2.2.3. ATARI RAM Disk

U ATARI 130 XE lze použít 64kB paměti jako tzv. RAM

Disk. Protože se tato oblast paměti jeví operačnímu systému obdobně jako disková jednotka, lze ji použít k uchovávání souborů, které budete používat. Hlavní výhodou použití RAM Disku je velká rychlost překladu. Podobně jako je označena disková jednotka D1 až D4, je RAM Disk označen D8.
K překopírování souboru z diskové jednotky do RAM Disku použijte DOS příkaz C (Copy):

```
C <RETURN>
D1:PC,D8:PC
```

Pozn.: K hladkému průběhu překladu je třeba, aby v RAM disku byl i program LIB a pokud se vejdou, tak i překládaný a přeložený program. V tomto případě nezapomeňte poslední verzi Vašeho programu překopírovat zpět na disketu.

2.3. HELP

Pokud zavoláte program D1:HELP, vypíší se na obrazovku úvodní instrukce, kde jsou uvedena jména editoru (ED), překladače (PC) a tiskového programu (PRINT.O), dále způsob návratu do DOSu (DOS) a spuštění vlastního programu.

3. Základní programové struktury.

V Pascalu je, stejně jako v jiných jazycích, nadefinován určitý počet klíčových slov, pomocí kterých se vytvářejí programové struktury.
V následující části se s většinou z nich seznámíme na příkladech.

3.1. Program Tisk

První program Vám ukáže, jak se dá tisknout na obrazovku:

```
PROGRAM Tisk(Output);
  BEGIN
    WRITELN;
    WRITELN;
    WRITELN('Jmenuji se Jan Novák')
  END.
```

Tento program vypíše sdělení 'Jmenuji se Jan Novák' na obrazovku.

3.2. Výpis programu a klíčová slova

Za klíčovým slovem PROGRAM se uvádí jméno programu, v tomto případě Tisk. Za příkazem musíte napsat středník. Pascal má přesně určený slovník klíčových slov, která nemohou být programátorem použita jako jména v jeho programu. Příkladem těchto slov jsou např. PROGRAM a BEGIN. Tato klíčová slova budou psána velkými písmeny, na rozdíl od jmen proměnných (identifikátorů), která budou psána malými písmeny s počátečním písmenem velkým (překladač pak ovšem nerozlišuje mezi velkými a malými písmeny). Samozřejmě, že komentáře uvedené ve složených závorkách, např. (* PROGRAM *) a texty v řetězcích mezi uvozovkami nejsou tímto slovníkem omezeny.

3.3. Deklarace a tělo programu

Každý program v jazyce Pascal má dvě hlavní části - deklaraci a tělo programu. Uvedený program začíná jménem programu. Většina programů však zahrnuje seznam konstant a proměnných, které dohromady tvoří deklarační část programu. Pak následuje část programu, která obsahuje výpočty, popř. operace vstupu a výstupu. Je označena klíčovým slovem BEGIN a nazývá se tělo programu. Klíčové slovo END následované tečkou označuje konec těla programu.
-Analýza programu Tisk:

První řádek deklaruje jméno programu.
Druhý řádek, BEGIN, označuje začátek těla programu.
Třetí a čtvrtý příkaz (WRITELN, zkratka pro "napiš řádek") vytvoří dva prázdné řádky na obrazovce.
Pátý příkaz vytiskne text mezi apostrofy na obrazovku.

3.4. Konstrukce programu

Druhý program, který vytvoříte, bude vypočítávat cenu za postavení bytu, danou odpracovanými hodinami, mzdou a cenou materiálu.

```
PROGRAM Stavba(Input,Output);
  CONST
    Material=325000
  VAR
    Hodiny,Mzda,MzdaCelk,Celkem:Real;
  BEGIN
    WRITELN("Zadej odpracované hodiny a mzdu");
    READLN(Hodiny,Mzda);
    MzdaCelk:=Hodiny*Mzda;
    Celkem:=MzdaCelk+Material;
    WRITELN("Celková mzda = Kčs",MzdaCelk:8:3,"Náklady =
Kčs",Celkem:8:3)
  END.
```

-Analýza programu Stavba

Cena za postavení bytu je dána celkovou mzdou + náklady na materiál. Celková mzda je dána součinem odpracovaných hodin a mzdou.
V první části programu najdete jméno programu, konstanty a proměnné. Pevná cena materiálu je dána konstantou Material a je 325000. Deklarace proměnných se zapisují po klíčovém slově VAR. Ačkoliv není "Real" klíčové slovo, je předdefinováno a značí, že všechny proměnné, které ho předcházejí jsou reálná čísla.

První příkaz napíše na obrazovku:

"Zadej odpracované hodiny a mzdu"

Další příkaz čte hodnoty pro proměnné Hodiny a Mzda, které uživatel zadá z klávesnice. Jakmile jsou tyto hodnoty zadány, spočítá se třetím a čtvrtým příkazem hodnota celkových mezd a nákladů. Poslední příkaz tyto hodnoty vytiskne.
-- Identifikátory:
Identifikátor je jméno, které může být jménem programu nebo podprogramu nebo jménem nějaké hodnoty. Stejně jako v algebře můžeme definovat konstantu, C=5, v Pascalu:

```
CONST
  C=5;
```

Pravidla pro konstrukci identifikátoru jsou:

1) Musí začínat velkým nebo malým písmenem.
2) Může následovat jakákoliv kombinace písmen anebo čísel.

Ačkoliv můžete použít více než 8 znaků, je rozlišeno pouze prvních 8.

3.5. Základní příkazy vstupu a výstupu

Příkazy WRITE, WRITELN, READ, READLN převádějí informace

z a do počítače. READ a READLN vkládají data z periferního zařízení (klávesnice) do počítače. WRITE a WRITELN posílají data na periferní zařízení (obrazovku).
Slova Input a Output v závorkách za jménem programu znamenají, že se budou přenášet data do a z paměti pouze mezi počítačem, klávesnicí a obrazovkou. Standardní identifikátor Input představuje zařízení "K:", identifikátor Output "F:". Pokud chcete výsledek vytisknout na tiskárně, je nutné podniknout následující kroky:

1.vložit soubor, který obsahuje potřebné procedury, čehož dosáhneme tak, že před příkaz BEGIN napíšeme řádek:

```
    #I  D1:PR.I
```

Pak stačí před příkaz WRITELN, kterým chceme tisknout vložit příkaz PR.ON; a za něj PR.OFF;
-- READLN
Jestliže chceme vkládat data z klávesnice, máme více možností. Je-li příkaz typu: READLN(A,B); vložíme dvě proměnné buď tak, že je oddělíme mezerou nebo klávesou <RETURN>.

3.6. CONST

Použitím konstant se program stává univerzálnějším a lépe opravitelným. Např. kdybyste chtěli v programu Stavba změnit cenu materiálu, stačí změnit pouze číslo v deklaraci konstant. Pokud konstantu nenadeklarujete, musíte měnit všechna čísla v programu.

3.7. Výpočet průměru 2 čísel

```
PROGRAM Prumer(Input,Output);
VAR
  X1,X2,Prumer:Real;
BEGIN
  (* čtení dvou čísel *)
  WRITE('Prvni cislo = ');
  READ(X1);
  WRITE('Druhe cislo = ');
  READ(X2);
  (* vypocet prumeru *)
  Prumer:=(X1+X2)/2;
  (* tisk prumeru *)
  WRITELN('prumer = ',Prumer:9:2)
END.
```

Program běží v těchto krocích:

```
První číslo = >10<
Druhé číslo = >20<
průměr = 15
```

Pozn.: Data, která vkládáte z klávesnice, budeme uzavírat mezi znaky nerovnosti > <.
Rozdíl mezi WRITE a WRITELN spočívá v tom, že po příkazu WRITELN odskočí kursor na začátek dalšího řádku, kdežto po WRITE zůstává na konci vypsaného textu. Příkaz READLN (X1)

vkládá data z klávesnice do počítače. Je přečten celý vstupní řádek do specifikované proměnné. Použití příkazu READ nejlépe ilustruje následující příklad:

```
PROGRAM Read;
VAR
  A,B,C:Integer;
BEGIN
  WRITE('Zadej A,B,C: ');
  READ(A);
  WRITELN('A=',A);
  READ(B);
  WRITELN('B=',B);
  READ(C);
  WRITELN('C=',C)
END.
```

Všechna data zadáme již při prvním příkazu READ a oddělíme je mezerami. Do proměnné A se načte pouze první hodnota a ostatní hodnoty se uloží do vstupní vyrovnávací paměti (bufferu). Odtud se pak vybírají automaticky při použití dalších příkazů READ. Nicméně obsah proměnné závisí na typu proměnné. Jestliže např. napíšeme 123 AHOJ a X1 je typu Char, pak X1="1". Zbývající znaky a <RETURN> se ignorují. Jestliže je X1 typu ARRAY[1..8] OF Char, pak je X1[n] rovno 1,2,3, ,A,H,O,J a <RETURN> je ignorován. A nakonec je-li X1 typu Integer, pak X1 = 123.

3.8. Datové typy Real a Integer

Reálnými čísly se v Pascalu rozumí kladná či záporná čísla s nebo bez desetinné tečky, např. 12.8, 3.456E+5, -2.567E-8, 0. Číslo v desetinném tvaru musí mít nejméně jednu číslici před a jednu za desetinnou tečkou. Rozsah hodnot může být od +-9.9999999999E-99 do +-9.9999999999E+99.
Takže chceme-li nadefinovat proměnnou X typu Real:

```
VAR X:Real;
```

Čísla typu Integer musí být celá čísla od -32767 do +32767.
Jestliže se v matematickém výrazu vyskytují tyto dva typy společně, je výsledek typu Real. Výsledek dělení i dvou čísel typu Integer je vždy typu Real.

3.9. Formátování tisku

Chceme-li číslo vytisknout na určitý počet pozic, použijeme formát, který od proměnné oddělíme dvojtečkou:

```
WRITE(X:9:2);
```

Formát 9:2 rezervuje celkem 9 pozic. Zahrnuje pozici pro znaménko a pozici pro desetinnou tečku a dvě pozice pro dvě čísla za desetinnou tečkou. Jestliže má číslo méně číslic, než je rezervováno, bude zbytek vyplněn mezerami nebo

nevýznamnými nulami. Reálná čísla mohou mít max. 13 platných číslic.

3.10. Trunc, Round, Maxint

Trunc je funkce, která z čísla odebere desetinnou část bez zaokrouhlení:

```
Trunc(5.9) = 5.
```

Round zaokrouhluje číslo, od 0.5 včetně nahoru, do 0.49 dolů. Např.

```
Round(12.2)=12
Round(17.5) = 18
```

Maxint znamená nejvyšší možné číslo typu Integer, v našem případě Maxint = 32767.

3.11. Aritmetické operátory

Pro operace s daty typu Real a Integer používá Pascal tyto operátory:

```
sčítání   -> +
odčítání  -> -
násobení  -> *
dělení    -> /
```

3.12. Relační operátory

Existuje 6 relačních operátorů, které se používají k větvení programu na dvě části. Jedna část se bude vykonávat, je-li hodnota relačního výrazu True (pravda), druhá část, je-li False (nepravda). Relační operátory jsou typu Boolean (viz dále) a jsou to:

```
=     je rovno
<>    není rovno
<     je menší než
>     je větší než
<=    je menší nebo rovno
>=    je větší nebo rovno
```

3.13. Příkazy IF - THEN

Všimněte si, že v uvedeném programu jsou za příkazem IF dva programové řádky uzavřené mezi příkazy BEGIN a END. Tyto příkazy se vykonají, má-li relace za příkazem IF hodnotu True. Někdy je třeba vložit programové kroky pro případ hodnoty False. Ty se vkládají za klíčové slovo ELSE (tato část není povinná). Za slovem END, které ukončuje složený příkaz za slovem THEN,
se nepíše středník, neboť následuje slovo ELSE, před kterým se středník nepíše. Příkladem nám může sloužit program pro výpočet kvadratické rovnice s reálnými kořeny.

Tedy když je v našem případě diskriminant D větší nebo roven 0, pak se počítají kořeny X1 a X2, je-li menší, vytiskne se sdělení o nereálnosti kořenů:

```
PROGRAM KvadRov(Input,Output);
(*vypocet kvadraticke rovnice s realnymi koreny*)
CONST
  T=4;
  U=2;
VAR
  A,B,C,D,X1,X2:Real;
BEGIN
  WRITE('Zadej koef. A,B,C ');
  READLN(A,B,C);
  D:=B*B-T*A*C;
  IF D>=0 THEN
  BEGIN
    X1:=(-B+SQRT(D))/(U*A);
    X2:=(-B-SQRT(D))/(U*A)
  END (*jestlize ma vyraz D>=0 hodnotu True, tzn.
plati*)
ELSE (*jestlize ma vyraz D>=0 hodnotu False, tzn.
neplati*)
  WRITELN('Tato rovnice nema realne koreny');
(*Tisk vysledku*)
WRITELN('koren X1 = ',X1);
WRITELN('koren X2 = ',X2)
END.
```

3.14. Přiřazovací znaménko

Zde vysvětlím rozdíl mezi znaménkem rovnosti (=) a přiřazení (:=).
Jako příklad vezmeme řádek z programu KvadRov:

```
D:=B*B-T*A*C;
```

Zde pomocí znaménka := vypočítáváme (přiřazujeme) hodnotu D. Znaménko rovnosti = se používá pouze v definici konstant a v příkazu IF jako srovnávací.

3.15. Datový typ Char a řetězce

Dalším předdefinovaným typem, který Pascal obsahuje, je typ Char (Character=znak). Označuje proměnné, konstanty nebo jejich části, které jsou vyjádřeny jediným znakem. Chceme-li tedy nadefinovat proměnnou Číslo typu Char:

```
VAR Číslo:Char;
```

To znamená, že Číslo bude ve formě jediného tisknutelného znaku. Může to být číslo, písmeno nebo symbol. I když bude proměnná Číslo="1" nejde o číslo v pravém smyslu slova. Počítač s ním bude operovat jako se znakem, nikoliv jako s číslem. Nejdou s ním tudíž provádět matematické operace.

Další typ dat se nazývá Řetězec (String). V KYAN Pascalu se Řetězec definuje:

```
String=ARRAY/1..15/OF Char;
```

Protože String si definujete sami, můžete si specifikovat libovolnou délku, v našem případě je to 15 znaků. Jestliže vkládané slovo obsahuje méně než je definovaný počet znaků, READLN si zbytek doplní mezerami. Je-li znaků více, jsou přebytečné znaky ignorovány. Při přímé definici v programu musí mít Retězec přesný počet znaků, pro který je definován:

```
VAR Slovo:String;
  Znak:Char;
BEGIN
  Slovo:="Malo znaku      ";
  Znak:="K";
```

3.16. Smyčka WHILE

WHILE smyčka probíhá tak dlouho, dokud relace, která následuje, má hodnotu True. Jestliže se ve smyčce vykonává více než jeden příkaz, musí být ohraničeny slovy BEGIN a END. Pozor na to, aby nevznikla nekonečná smyčka.

Následující program ilustruje použití dat ve formě znaků a smyčky WHILE. Hledá ze zadaných slov abecedně první a počítá celkový počet zadaných slov.

```
PROGRAM PrvSlovo(Input,Output);
CONST
  Signal="+";
TYPE
  Retezec=ARRAY/1..15/OF Char;
VAR
  Slovo,PrvSlovo:Retezec;
  Smycka:Integer;
BEGIN
  (*Pri kazdem pruchodu smyckou se do promenne PrvSlovo
nacte to slovo, ktere je abecedne drive a citac smycky Smycka
se zvetši o 1 *)
  WRITE('Zadej slovo nebo + : ');
  (*Znakem + ukoncite nacitani slov*)
  READLN(Slovo);
  PrvSlovo:=Slovo;
  Smycka:=0;
  WHILE Slovo/1/<>Signal DO
    BEGIN
      IF Slovo<PrvSlovo THEN
        PrvSlovo:=Slovo;
      Smycka:=Smycka+1;
      WRITE('Zadej slovo nebo + : ');
      READLN(Slovo);
    END; (*Konec WHILE smycky*)
  WRITELN;
  WRITELN;
  WRITELN(Smycka:5,' slov bylo zadano ');
  WRITELN(PrvSlovo,' je abecedne prvni')
END.
```

```
Krok 1: Zadání prvního slova do seznamu.
Krok 2: Inicializace proměnných: PrvSlovo = Slovo, Smycka =
1.
Krok 3: Začátek WHILE smyčky a test na její ukončení
(Slovo="+").
  Krok 3a: Zadání dalšího slova.
  Krok 3b: Zvětšení proměnné Smycka.
  Krok 3c: Porovnání - je-li zadané slovo abec    dříve než
slovo v proměnné PrvSlovo, načte se místo něj do proměnné
PrvSlovo.
Krok 4: Výpis abecedně prvního slova a počtu slov.
```

3.17. Smyčka FOR

Tento program počítá faktoriál ze zadaného celého kladného čísla. Pozor, aby výsledek nebyl větší než 32767 !

```
PROGRAM VypFakt(Input,Output);
VAR
  Cislo,Smycka,Fakt:Integer;
BEGIN
  WRITELN;
  WRITELN;
  WRITELN("Vypocet faktorialu");
  WRITE("Zadej cislo pro vypocet: ");
  READLN(Cislo);
  Fakt:=1;
  FOR Smycka:=1 TO Cislo DO
    BEGIN
      Fakt:=Fakt*Smycka
    END; (*FOR smycka*)
  WRITELN;W;
  WRITELN(" N! = ",Fakt:6)
END.
```

```
-- Algoritmus
Krok 1: Vstup N (Cislo).
Krok 2: Inicializace - Fakt=1 (0!=1).
Krok 3: Začátek smyčky FOR s hodnotou Smycka=1 a následné
zvětšování této hodnoty, dokud nedosáhne hodnoty Cislo.
Při každém průběhu smyčkou se počítá nová hodnota faktoriálu
(Fakt).
Krok 4: Tisk výsledku.
Pro lepší pochopení přeložím řádek:
    FOR Smycka:=1 TO Cislo DO ->
 -> PRO(FOR) proměnnou Smycka od hodnoty 1(:=1) DO(TO)
hodnoty Cislo VYKONEJ(DO)
Smyčka FOR má vždy krok jedna. Pokud byste chtěli použít tuto
smyčku sestupně, např od 5 do 1 použijete místo TO DOWNTO:

    FOR Smycka:=5 DOWNTO 1 DO
```

3.18. Datový typ Boolean

Boolean je další z předdefinovaných typů. Výrazy, proměnné a konstanty typu Boolean nabývají pouze dvou hodnot - buď True nebo False. V uvedeném programu je příkaz IF

vykonán pouze tehdy, má-li správně hodnotu True, tzn., že oba výrazy v závorkách (MOD a DIV) mají hodnotu True.

3.19. Operátory DIV a MOD

Tyto operátory se používají při dělení celých čísel. Použitím operátoru DIV získáme celé číslo, které je výsledkem dělení beze zbytku. Tedy např.:

```
7 DIV 3 = 2
```

Naopak operátorem MOD získáme celé číslo, které je zbytkem po dělení dvou celých čísel, např.:

```
7 MOD 3 = 1
```

3.20. Booleanské (logické) operátory

Na rozdíl od dat typu Real a Integer typ Boolean nepoužívá matematické operátory jako dělení atd., ale operátory logické:

```
NOT - logická negace (doplňek)
OR  - logický součet (disjunkce)
AND - logický součin (konjunkce)
```

Nyní následují pravdivostní tabulky těchto operátorů:

```
NOT:  False = NOT True
      True = NOT False
OR:   True = True OR False
      True = False OR True
      True = True OR True
      False = False OR False
AND:  False = True AND False
      False = False AND True
      True = True AND True
      False = False AND False
```

-- Priorita operátorů
Ve složitějších výrazech se operace provádějí podle priority jejich operátorů:

1. nejvyšší priorita: ()
2. úroveň priority: NOT
3. úroveň priority: *,/,AND,DIV,MOD
4. úroveň priority: +,-,OR
5. úroveň priority: =,<=,>=,>,<,<>

Následující program se Vás nejdříve zeptá na celé číslo, pak na číslo, kterým je první číslo beze zbytku dělitelné a nakonec na výsledek dělení.

```
PROGRAM Deleni(Input,Output);
VAR X,W,Z:Integer;
    Odp:Char;
```

```
        Spravne:Boolean;
    BEGIN
      Odp:="A";
      WHILE Odp="A" DO BEGIN
        WRITE("zadej cele cislo "); Readln(X);
        WRITE("zadej delitele "); Readln(W);
        WRITE(X:3," deleno ",W:3," = "); Readln(Z);
        Spravne:=(X MOD W=0) AND (X DIV W=Z);
        IF Spravne THEN
          BEGIN
            WRITE("Dobre! Dalsi? Zadej A nebo N");
READLN(Odp)
          END (* IF THEN *)
        ELSE
        BEGIN
          WRITE("Spatne! Znovu? Zadej A nebo N");
Readln(Odp)
        END (* IF ELSE *)
      END (* WHILE *)
    END.
```

3.21. Násobné větvení - CASE

Dříve uvedeným příkazem pro větvení IF můžeme program rozvětvit maximálně na dvě části - v jedné je podmínka větvení splněna, ve druhé ne. Někdy je v programu třeba složitějšího větvení než pouze na dvě části a použití IF THEN není optimální. Toto několikanásobné větvení se řeší příkazem CASE OF. Např.:

```
CASE Cislice OF
  "0" :Cislo:=0;
  "1" :Cislo:=1
END;
```

Nahrazuje:

```
IF Cislice="0" THEN
  Cislo:=0 ELSE
IF Cislice="1" THEN
  Cislo:=1;
```

Příkaz CASE nemá klíčové slovo BEGIN, je však zakončen klíčovým slovem END.

3.22. Smyčka REPEAT ... UNTIL

Tato smyčka je velmi podobná smyčce WHILE s tím rozdílem, že smyčka WHILE testuje podmínku pro svůj průběh hned na začátku, kdežto REPEAT až na konci. Z toho vyplývá, že smyčka REPEAT běží alespoň jednou, kdežto WHILE nemusí proběhnout vůbec. Smyčka se znovu vykonává tehdy, je-li hodnota výrazu za UNTIL rovna False.

Následující program převádí čísla ze šestnáctkové soustavy do desítkové a demonstruje větvení na více než 2 části a smyčku REPEAT:

```
PROGRAM HEXDEC(Input,Output);
TYPE
  AnoNe=(Ano,Ne);
VAR
  Cislice,Signal:Char;
  Cislo,StareCis:Integer;
       Odp:AnoNe;
       Pokrac:Boolean;
BEGIN
  StareCis:=0;
  WRITE('Zadej nejvyznamejsi levou cislici ');
  READLN(Cislice);
  REPEAT
    CASE Cislice OF
      '0':Cislo:=0;
      '1':Cislo:=1;
      '2':Cislo:=2;
      '3':Cislo:=3;
      '4':Cislo:=4;
      '5':Cislo:=5;
      '6':Cislo:=6;
      '7':Cislo:=7;
      '8':Cislo:=8;
      '9':Cislo:=9;
      'A':Cislo:=10;
      'B':Cislo:=11;
      'C':Cislo:=12;
      'D':Cislo:=13;
      'E':Cislo:=14;
      'F':Cislo:=15
    END (* CASE *);
    StareCis:=Cislo+StareCis*16;
    WRITELN(' Dalsi cislice - A nebo N ');
    READLN(Signal);
    IF (Signal='A') OR (Signal='a') THEN
      Odp:=Ano
    ELSE
      Odp:=Ne;
    IF Odp=Ano THEN
      BEGIN
        Pokrac:=True;
        WRITE(' Zadej dalsi cislici ');
        READLN(Cislice)
      END (* jestlize Odp = True *)
    ELSE
      Pokrac:=False;
  UNTIL NOT(Pokrac);
  WRITELN;WRITELN;
  WRITELN(' DEC = ',StareCis:6)
END.
```

-- Algoritmus:

1. Inicializace StareCis:=0
2. Vstup první levé platné cifry
3. REPEAT
   3a. Převod Cislice na decimalní Cislo
   3b. StareCis:= Cislo+StareCis*16

```
   3c. Je další číslice?
     3c.a. Jestliže NOT(Pokrac)=False pak vstup další
číslice
4. UNTIL NOT(Pokrac)=True
5. Výstup decimálního čísla (StareCis)
```

3.23. Výčtové typy a Booleovské proměnné

V uvedeném programu je Odp výčtový typ. Tento typ používáme, jestliže proměnná může nabývat pouze několika hodnot, např.:

```
TYPE
  DenVTydnu=(Po,Ut,St,Ct,Pa,So,Ne)
VAR
  Den:DenVTydnu
```

3.24. Typ interval

Typ interval vytvoříme z libovolného typu (s výjimkou Real) tak, že se definuje pouze první a poslední hodnota původního typu. Např.:

```
TYPE
  DenVTydnu=(Po,Ut,St,Ct,Pa,So,Ne)
  PracDen=Po..Pa
  Cisla=1..100
```

V tomto příkladě je DenVTydnu výčtový typ a PracDen typ interval z DenVTydnu, který byl definován předtím. Cisla je také typ interval, ale z předdefinovaného typu Integer.

3.27. Funkce Ord, Pred, Succ a Chr

První tři z těchto funkcí se používají ve spojitosti s libovolným typem s výjimkou Real.

```
TYPE
  DenVTydnu=(Po,Ut,St,Ct,Pa,So,Ne)
```

Funkce Ord nám udává, kolikátý je daný prvek ve výčtu:

```
Ord(Ne)=6
Ord(Po)=0
```

Funkce Pred a Succ udávají, který prvek je před respektive za daným prvkem:

```
Succ(Po)=Ut
Pred(Ne)=So
```

V tomto případě Succ(Ne) a Pred(Po) není definováno.
Funkce Chr k danému ASCII kódu přiřazuje jeho znakovou podobu:

```
Chr(65)="A"
```

U datového typu Boolean je definován:

```
Succ(False)=True
Pred(True)=False
```

Succ(True) a Pred(False) není definováno.
U typu Char se definuje např.:

```
Succ('A')='B'
Pred('B')='A'
```

4.1 Procedury

Následující část této učebnice Vás naučí co nejvíce zkracovat programy až na jednoduché moduly, tzv. procedury. Procedury mohou (ale nemusí) komunikovat jak s hlavním programem, tak s jinými procedurami. Pokud spolupracují, pak je obvykle deklarován soupis parametrů. V následujícím příkladu to jsou X1 a X2:

```
PROCEDURE VymHod(VAR X1,X2:Real);
VAR Y:Real;
BEGIN
  Y:=X1;
  X1:=X2;
  X2:=Y;
END;
```

Tato procedura zamění hodnoty X1 a X2.

4.1.1. Deklarace a vykonávání procedur

Nyní následuje seznam kroků potřebných k tomu, aby mohla být procedura VymHod použita v hlavním programu, který se bude jmenovat Demo. První část je deklarační část programu, druhá část deklaruje proceduru a nakonec třetí část je tělo programu, kde se procedura používá:

1. Deklarace hlavního programu.
   - 1a. Jméno programu
   - 1b. Typy a proměnné
   - 1c. Proměnné pro hlavní program.: A,B
2. Deklarace procedury VymHod.
   - 2a. Jméno procedury a seznam parametrů
   - 2b. Lokální typy a konstanty
   - 2c. Lokální proměnné.: Y
   - 2d. Tělo procedury - zde se nevykonává.
3. Tělo hlavního programu
   - 3a. Vstup dvou čísel: A,B
   - 3b. Výměna hodnot těchto čísel pomocí procedury VymHod
   - 3c. Výstup A a B na obrazovku.

```
PROGRAM Demo(Input,Output);
VAR A,B:Real;
PROCEDURE VymHod(VAR X1,X2:Real);
VAR Y:Real;
BEGIN
  Y:=X1;
  X1:=X2; X2:=Y;
END (*PROCEDURY*);
BEGIN (*Demo*)
  Write('Zadej dve cisla ');
  Readln(A,B);
  VymHod(A,B);
  Writeln('Nyni je prvni: ',A:7:2,' a druhá: ',B:7:2)
END.
```

4.1.2. Seznam parametrů, parametry formální a aktuální

Protože procedura je vlastně program v programu, musí existovat způsob, jak přenášet data mezi procedurou a hlavním programem. V uvedeném programu toto obstarávají proměnné X1,X2 a A,B. Zároveň jsou to tzv. parametry. Parametrem může být proměnná, konstanta nebo dokonce jiná procedura. Parametrům uvedeným v závorkách za jménem procedury v deklarační části - v našem případě X1,X2 - říkáme seznam formálních parametrů.

Parametry uvedené v závorce za jménem procedury v těle programu - v našem případě A,B - se nazývají seznam aktuálních parametrů.
Samozřejmě, že aktuální a jim odpovídající formální parametry musí být stejného typu! Zde bych se chtěl zmínit o kompatibilitě jednotlivých typů. V zásadě platí, že proměnné jednoho typu může být přiřazena hodnota stejného typu. Jedinou výjimku z tohoto pravidla tvoří případ, že proměnné typu Real přiřadíme výraz typu Integer (nikoliv naopak):

```
var
  X:Real;
  I:Integer;
begin
  I:=15;
  X:=I;
...
```

I když v naší ukázce je seznam formálních parametrů na jediném řádku, lze seznam napsat také následovně:

```
   PROCEDURE Vypocet(A,B:Real;VAR X:Real;Y:Integer);
```

odpovídá zápisu:

```
   PROCEDURE Vypocet(
     A,B:Real
     VAR
      X:Real;
      Y:Integer);
```

4.1.3. Parametry volané hodnotou a odkazem

V následující ukázce jsou pouze některé parametry uvedeny slovem VAR.:

```
     PROCEDURE Param(VAR X1,X2:Real;Z:Real;VAR Y:Integer);
```

Parametry definované pomocí VAR - X1,X2,Z - se nazývají parametry volané odkazem a používají se jak pro vstup tak pro výstup z procedury. Parametr Z volaný hodnotou se dá použít pouze pro vstup do procedury a i když během výpočtu může měnit svou hodnotu, nekomunikuje s hlavním programem (nelze jej použít jako výstup). Volání této procedury pak může vypadat např. takto:

```
     Param(A,B,5.0,D); nebo
     Param(C,D,D*C,B);
```

Aktuální paametr volaný hodnotou může být přímo hodnota (5.0), parametr volaný odkazem musí být pouze proměnná (A).

4.1.4. Korespondence mezi aktuálními a formálními parametry

Vždy musí být splněna následující pravidla:

1. Počet aktuálních a formálních parametrů musí souhlasit.
2. Musí souhlasit typy odpovídajících si parametrů.

4.2. Funkce

Funkce jsou podobné procedurám v tom, že používají parametry, ale liší se v tom, že funkce vrací pouze jedinou hodnotu, která je určena jménem funkce.
Pascal má některé běžné funkce již vestavěny:

Abs(X)	=	absolutní hodnota X
Sqr(X)	=	X na druhou
Sqrt(X)	=	druhá odmocnina z X
Sin(X)	=	sinus X, X je v radiánech
Cos(X)	=	cosinus X, X je v radiánech
Arctan(X)	=	arctangens X, výsledek v radiánech
Ln(X)	=	přirozený logaritmus X
Exp(X)	=	e na X

4.2.1. Deklarace funkcí

Uživatelem definovaná funkce je jednoduchá procedura, která používá pouze hodnotové parametry.

```
PROGRAM Ukazka(Input,Output);
VAR E,H1,Write1,Uhel1,UhelX:Real;
  FUNCTION CosVeta(A,B,Theta:Real):Real;
(*vypocitava delku strany c protilehle uhlu Theta*)
  VAR C:Real;
  BEGIN
    C:=A*A+B*B;
    CosVeta:=C-2.0*A*B*Cos(Theta)
  END; (*Procedury*)
BEGIN
  Readln(H1,Write1,Uhel1,UhelX);
  E:=1.0+CosVeta(H1,Write1,Uhel1)*Sin(UhelX);
  Writeln('E = ',E)
END.
```

Z ukázky je vidět způsob deklarování funkce: Jméno funkce CosVeta, seznam formálních parametrů A,B,Theta:Real, výsledný typ (Real), lokální deklaraci VAR C:Real a tělo funkce (BEGIN..END).
Na rozdíl od procedury, která může vracet tolik hodnot, kolik jich je v seznamu proměnných, může funkce vracet pouze jedinou hodnotu.

4.2.2. Funkce Odd

Funkce Odd vrací hodnotu True, je-li parametr funkce lichý, False, je-li sudý. Z toho vyplývá, že parametrem může být pouze číslo typu Integer:

```
Odd(3)=True
```

4.3. Globální a lokální proměnné

Globální proměnnou nazýváme proměnnou, která je definována v hlavním programu a použita v proceduře. Pokud je proměnná deklarována uvnitř funkce nebo procedury, nazývá se lokální proměnná. Parametry nejsou ani lokální ani globální proměnné, třebaže se používají k přenosu hodnot globálních proměnných
do a z procedury.

```
PROGRAM Alfa(Input,Output);
VAR A1:Real;
    A3,A4:Char;
PROCEDURE Ukazka(VAR AA1:Real;AA3:Char);
  VAR BB1:Integer;
  BEGIN
    A4:='Y';   (* A4 je globalni *)
    BB1:=5;    (* BB1 je lokalni *)
    AA1:=15.3
  END;     (* PROCEDURE *)
BEGIN
  Ukazka(A1,A3);    (*A1,A3 jsou parametry *)
  IF A4='Y' THEN
  Writeln('A4 je globalni');
  IF A1=15.3 THEN
  Writeln('AA1 je formalni parametr')
END.     (* PROGRAM *)
```

4.4. Možnosti definování proměnných

1. Mohou být deklarovány v části globální deklarace.
   (VAR A1:Real;A3,A4:Char;)

Proměnná, která již byla deklarována v hlavním programu může být použita ve funkci nebo proceduře jako globální. Příkladem je použití proměnné A4:

```
A4:="Y";
```

Pokaždé, když je volána procedura Ukazka, je A4 nastaveno na hodnotu "Y" a výraz v hlavním programu, A4="Y", má hodnotu True.

2. Může být deklarována v části lokální deklarace.
   (VAR BB1:Integer;)

Takto deklarovaná proměnná může být použita pouze uvnitř procedury.

3. Může být deklarována v seznamu formálních parametrů.
   /Ukazka(VAR AA1:Real;AA3:Char);/

Nedoporučuje se přenos hodnot pomocí globálních proměnných, protože mohou být odstíněny hodnoty vstupních a výstupních dat. Je proto lepší používat aktuální a formální parametry. Následující část rozšiřuje předcházející definice globálních a lokálních proměnných na obecnější příklady, kde jsou proměnné relativně lokální nebo relativně globální. To se

stává v případech, kdy několik funkcí a procedur sdílí stejné proměnné.

4.5. Včleňování funkcí a procedur

Funkce a procedury mohou být včleněny do dalších funkcí nebo procedur. Deklarační část programu je předvedena dále a obsahuje vnořené bloky, které budeme nazývat oblast platnosti proměnné. Nejvíce vnořený blok, Faze1, je v obou dalších polích, jak CosVeta,tak Faze, zatímco CosVeta je pouze v poli hlavního programu, Faze.

Procedury nebo funkce deklarovaná dříve mají větší oblast platnosti proměnné, než ty, které jsou deklarovány později. Identifikátory (proměnných a typů) ve vnějších blocích jsou relativně globální k blokům vnitřním. Identifikátory deklarované v procedurách, které mají větší oblast platnosti proměnné, jsou relativně globální k procedurám v menším poli.
Tedy hodnoty proměnných mohou být přeneseny z procedury ve větším poli do procedury v menším poli jak pomocí parametrů, tak globálními proměnnými procedur ve větším poli.

```
PROGRAM Faze;
VAR Vyska1,Delka1,Uhel1,Uhel2,Rozd:Real;
  FUNCTION CosVeta(A,B,Beta:Real):Real;VAR C:Real;
    PROCEDURE Faze1(H1,Write1,Uh1,UhX:Real;VAR D:Real);
    VAR E:Real;
    BEGIN
    E:=1+CosVeta(H1,Write1,Uh1)*Sin(UhX);
    D:=1.22*C
    END;   (* Faze1 deklarace *)
  BEGIN
   C:=A*A+B*B;
   CosVeta:=C-2*A*B*Cos(Beta)
  END;    (* CosVeta *)
BEGIN
...
END.    (* Faze *)
```

Abychom lépe viděli pole proměnných, vypíšeme pouze deklarační části programů, funkcí, procedur:

```
program:Faze
 deklarované proměnné: Vyska1,Delka1,Uhel1,Uhel2,Rozd
  funkce CosVeta
   deklarované proměnné
    (formální parametry):A,B,Beta
    (lokální proměnná):C
  procedura:Faze1
  deklarované proměnné
   (formální parametry):H1,Write1,Uh1,UhX,D
   (lokální proměnná):E
```

V tomto případě je C globální k Faze1. Nová hodnota pro C se přenese do Faze1 jakmile se vykoná CosVeta. Toto použití globálních proměnných se nedoporučuje. Hodnoty by měly být

přenášeny z a do funkcí a procedur pouze pomocí parametrů.

Porovnejte následující verzi programu Faze s předchozí. Procedura Faze1 není včleněna do CosVety a tedy C není dále relativně globální k Faze1, protože CosVeta nemá větší pole, než Faze1.

```
PROGRAM Faze;
VAR Vyska1,Delka1,Uhel1,Uhel2,Rozd:Real;
  FUNCTION CosVeta(A,B,Beta:Real):Real;VAR C:Real;
  BEGIN
   C:=A*A+B*B;
   CosVeta:=C-2*A*B*Cos(Beta)
  END;   (* CosVeta *)
  PROCEDURE Faze1(H1,Write1,Uh1,UhX:Real;VAR D:Real);
  VAR E:Real;
  BEGIN
  E:=1+CosVeta(H1,Write1,Uh1)*Sin(UhX);
  D:=1.22*C
  END;   (* Faze1 deklarace *)
BEGIN
...
END.   (* Faze *)
```

4.6. Globální a lokální typy

Uživatelem definované typy, např. výčtové typy, mohou být taktéž lokální nebo globální. Platí pro ně stejná pravidla která jsme nyní uvedli.

4.7. Dopředná (neúplná) deklarace

Volání procedur nebo funkcí před jejich nadefinováním se nazývá dopředná deklarace.

```
PROGRAM Ref;
VAR A:Real;
FUNCTION TEST1(seznam formalnich parametru):Real;
FORWARD PROCEDURE TEST2(sez. form.par.);
FUNCTION TEST;
  BEGIN
    TEST2(seznam aktualnich par.)
  END;   (* FUNCTION *)
PROCEDURE TEST2;
  CONST B=4;
  BEGIN
    C:=TEST1(B)*2
  END;   (* PROC *)
BEGIN
  ...
END.     (* PROGRAM *)
```

Tento způsob deklarace se používá tehdy, když není jiná možnost deklarovat proceduru nebo funkci před jejím prvním použitím. Všimněte si, že příkaz pro dopřednou deklaraci obsahuje seznam formálních parametrů; později, když je funkce plně deklarována, se již seznam parametrů neuvádí.

4.8. Nepodmíněný skok: GOTO

Ačkoliv se to běžně nepoužívá, Pascalské příkazy mohou být označeny návěštím (značkou) aby se umožnil nepodmíněný skok.

Návěští, tj. číslo příkazu, je celé číslo následované dvojtečkou a umístěné před příkaz v programu. Max. délka návěští je 4 číslice. Návěští musí být deklarována stejně jako proměnné a konstanty (LABEL):

```
PROGRAM Priklad;
LABEL 22,35;
VAR A:Integer;
BEGIN
     A:=0;
 22: Writeln(' A= ',A:4);
     A:=A+1;
     IF A<5 THEN GOTO 22 ELSE GOTO 35;
     Writeln('radek vynech');
 35: Writeln(' Konec ')
END.
```

Návěští použitá ve funkci nebo procedure musí být deklarována lokálně. Příkaz GOTO může být použit jak ke skoku dopředu, tak dozadu uvnitř funkce, nebo ke skoku z funkce nebo procedury do hlavního programu. Nemůže se použít pro skok z hlavního programu do funkce nebo procedury.

Neexistuje situace, která by si vynutila použití této techniky. Správně strukturovaný program se bez příkazu GOTO umí obejít. Proto se snažíme tento příkaz nepoužívat.

4.9. Pole

Většina typů dat, o kterých jsme hovořili, představuje jedinou hodnotu. (Proměnné typy Integer a Real obsahují jediné číslo, Char jediný znak a Boolean má hodnotu buď False nebo True).

Některá data však lze rozdělit na jednotlivé složky, což je např. vidět na již uvedeném typu "string", který je složen z několika znaků. Toto je charakteristický znak typu pole: pole je vždy souhrn dat jednoho jednodušího datového typu. Jiným příkladem pole je vektor, např. směr vesmírné lodě.

V Pascalu se pole definují následovně:

```
NazevPole=ARRAY/podmnožina typu/OF základní typ
```

1. Typ pole je vždy definován uživatelem.
2. Podmnožina typu specifikuje velikost pole a určuje číslo každého prvku pole.
3. Základní typ musí být buď standardní (Real) nebo definovaný uživatelem. Všechny prvky pole musí být stejného typu.

Množství paměti vyhrazené pro pole závisí na typu pole. U znakových polí je nevyužité místo vyplněno mezerami, u ostatních polí není toto místo definováno. Příklad:

```
Program Graphic;
TYPE
  String=ARRAY[1..15]OF Char;
  Souradnice=(X,Y,Z);
  Vektortyp=ARRAY[Souradnice]OF Real;
VAR
  Vektor:Vektortyp;
  Slovo:String;
BEGIN
  Vektor[X]:=3.0;
  Vektor[Y]:=5.0;
  Vektor[Z]:=4.0;
  Slovo:='Prvni bod      ';
END.
```

První pole, String, může být použito k operacím se slovy nebo větami, které mají maximálně 15 znaků včetně mezer. Celá čísla 1-15 (index) určují polohu prvku v poli. V druhém poli každý vektor obsahuje 3 čísla. Jeho prvky nejsou identifikovány pomocí celých čísel, ale pomocí souřadnic, definovaným uživatelem. Každý skalární typ (výčtový, integer, char, boolean, interval) může indexovat pole.

4.9.1. Pole polí a vícerozměrná pole

Jestliže chceme popsat odstavec, který má 50 slov, můžeme ho definovat jako pole slov, tedy pole polí:

```
TYPE
  Slovo=ARRAY[1..15]OF Char;
  Odstavec=ARRAY[1..50]OF Slovo;
```

Pole Odstavec je příkladem vícerozměrného pole. Dále uvedené pole Matice je též vícerozměrné - je to dvourozměrné pole čísel:

```
TYPE
  Radek=ARRAY[1..3]OF Real;
  Matice=ARRAY[1..3]OF RADEK;
 (* Každý prvek Matice je řádek *)

TYPE
  Matice=ARRAY[1..3,1..3]OF Real;
 (* Indexy jsou čísla řádku a sloupce *)
```

V takovýchto případech je důležité vědět, co který index určuje. Význam ilustruje následující příklad, ve kterém je kopírováno jméno ze seznamu:

```
TYPE
  Slovo=ARRAY[1..14] OF Char;
  TabTyp=ARRAY[1..100]OF Slovo;
VAR
  Tabulka:TabTyp;
  Jmeno:Slovo;
  I:Integer;
BEGIN
```

```
      FOR I:=1 TO 14 DO
      Tabulka/2,I/:=Jmeno/I/
      (* Jmeno je zapsano do druheho radku tabulky *)
    END.
```

Nejlepší způsob, jak si uvědomit, který index je první, je přepsat deklaraci pole následovně:

```
    TYPE
      TabTyp=ARRAY/1..100/ OF ARRAY/1..14/ OF Char;
```

Pak index, který je deklarován dříve je první.

4.9.2. Sčítání dvou vícerozměrných polí

Následující program sečte dvě matice se 3x3 prvky. Nejdříve se načte první matice, která se uloží do paměti. Pak se načítají prvky druhé matice, které se ihned přičítají k prvkům první matice, takže po zadání druhé matice máme obě matice sečteny a výsledek uložen v první matici.

```
    Program ScitaniMatic;
    TYPE
      MatTyp=ARRAY/1..3,1..3/OF Real;
    VAR
      Matice:MatTyp;
      Radek,Sloupec:Integer;
      Plus:Real;
    BEGIN
      FOR Radek:=1 TO 3 DO
        FOR Sloupec:=1 TO 3 DO
        BEGIN
          Write('Prvek ',Radek:3,Sloupec:3,'1. Matice je:
');
        Readln(Matice/Radek,Sloupec/);
        END (* FOR *);
      FOR Radek:=1 TO 3 DO
        FOR Sloupec:=1 TO 3 DO
        BEGIN
          Write('Prvek ',Radek:3,Sloupec:3,'2. Matice je:
');
          Readln(Plus);

Matice/Radek,Sloupec/:=Plus+Matice/Radek,Sloupec/
        END (* FOR *);
      Writeln;
      Writeln('Soucet 2 matic je: ');
      Writeln;
      FOR Radek:=1 TO 3 DO
      BEGIN
        Writeln;
        FOR Sloupec:=1 TO 3 DO
        Write(Matice/Radek,Sloupec/:7:3)
      END; (* FOR *)
    END.
```

4.9.3. Pole jako parametr

V dále uvedené proceduře AAA je pole Sub použito jako proměnný parametr:

```
PROCEDURE AAA(A,B:Integer;VAR Sub:CiselnePole);
```

Kdyby pole Sub přenášelo hodnotu, mohli bychom deklaraci VAR vynechat. Samozrejmé, že se mohou přenášet také jen jednotlivé prvky pole.

4.9.4. Kopírování polí

Když jsou dvé pole stejného typu, pak lze hodnoty jednoho kopírovat do druhého jednoduše - najednou:

```
VAR Matice1,Matice2:ARRAY/1..3,1..3/ OF Real;
BEGIN
  Matice1:=Matice2;
```

4.9.5. Program pro řazení čísel

Tento program seřadí maximálně 150 čísel podle jejich velikosti.

Program vždy vezme dva následující prvky pole a ty porovná. Pokud nejsou seřazeny podle velikosti, přehodí jejich pořadí. Když dojde na konec pole, začne znovu a pokud při průchodu polem nebyla učiněna žádná změna, vypíše seřazené pole (metoda BUBBLE SORT).

```
Program SerazeniCisel;
CONST MaxCis=150;
TYPE CisPole=ARRAY/1..MaxCis/ OF Real
VAR Prvni,Posl,Index:Integer;
    Pole:CisPole;
PROCEDURE VymHod(VAR A,B:Real);
 VAR C:Real;
BEGIN
  C:=A; A:=B;
  B:=C;
END;
PROCEDURE Srov(Prvni,Posl:Integer;VAR Vpole:CisPole);
VAR I:Integer;
    Porov:Boolean;
BEGIN
  REPEAT
    Porov:=False;
    FOR I:=Prvni TO (Posl-1) DO
      IF Vpole/I/>Vpole/I+1/ THEN
      BEGIN
        VymHod(Vpole/I/,Vpole/I+1/);
        Porov:=True
      END;
  UNTIL Porov:=False
END; (* PROCEDURE Srov *)
BEGIN (* PROGRAM *)
  Writeln('Zadej cisla, ktera chces srovnat');
```

```
    Writeln('Po kazdem cisle stiskni <RETURN>');
    Writeln('Po poslednim cisle zadej 0');
    Index:=0;
    REPEAT
      Index:=Index+1;
      Write('Vstupni cislo ',Index:3,' je: ');
      Readln(Pole^[Index]);
    UNTIL Pole^[Index]:=0.0
    Writeln('Mezi kterymi cisly chces ');
    Writeln('seznam srovnat? Prvni je: ');
    Readln(Prvni);
    Writeln('Posledni je: ');
    Readln(Posl);
    Srov(Prvni,Posl,Pole);
    Writeln;
    FOR Index:=Prvni TO Posl DO
      Writeln(Pole^[Index]:7:3,' vstupni cislo ',Index:3)
END.
```

Nyní následuje modifikace předchozího programu, sloužící pro srovnání maximálně šesti 15-ti znakových slov podle abecedy. Je to tedy opět ukázka práce s vícerozměrným polem. Princip je stejný jako u předchozího programu. První index zadává horizontální pozici písmene, druhý vertikální, tedy o kolikáté slovo jde:

```
Program RazeniSlov;
CONST MaxPismen=15;
      MaxSlov=6;
TYPE String=ARRAY[1..MaxPismen]OF Char;
     PoleSlov=ARRAY[1..MaxSlov]OF String;
VAR Slova:PoleSlov;
    Index:Integer;
PROCEDURE VymHod(VAR Slova:PoleSlov;Index:Integer);
VAR C:String;
BEGIN
  C:=Slova^[Index];
  Slova^[Index]:=Slova^[Index+1];
  Slova^[Index+1]:=C
END;(* procedura VymHod *)
PROCEDURE Srov(VAR Slova:PoleSlov);
VAR Index:Integer;
    Zmena:Boolean;
BEGIN
  REPEAT (* Dokud se nesrovnají všechna slova *)
    Zmena:=False; (* Všechna slova jsou srovnána,
jestliže se žádná dvojice nemusí prohazovat *)
    FOR Index:=1 TO Maxslov-1 DO
    IF Slova^[Index]>Slova^[Index+1] THEN BEGIN
      VymHod(Slova,Index);
      Zmena:=True;
    END;(* IF *)
  UNTIL Zmena=False;
END;(* procedura Srov *)
BEGIN (* Hlavní program *)
  Writeln('Vloz 6 slov, kazde maximalne');
  Writeln('15 znaku. Po kazdem slove stiskni');
  Writeln('klavesu <RETURN>');
```

```
    Index:=0;
    REPEAT
      Index:=Index+1;
      Write('Slovo cislo ',Index:3,' je:');
      Readln(Slova[Index]);
    UNTIL Index=MaxSlov;
    Srov(Slova);
    Writeln;
    Writeln('Srovnana slova');
    FOR Index:=1 to MaxSlov DO
    BEGIN
      Writeln;
      Writeln(Slova[Index])
    END (* FOR *)
       (* Program *)
```

4.10. Konec řádku (EOLN)

Řádek vstupních dat vždy ukončujeme klávesou <RETURN>. Tato klávesa generuje kód, který v Pascalu můžeme testovat jako Booleanovskou konstantu EOLN (End of line). Má-li hodnotu true, byla stlačena klávesa <RETURN>. Takže je možno použít příkazy

```
Read(Znak);
IF EOLN THEN ...
```

pro ovládání vstupu z klávesnice, protože příkazy uvedené za slovem THEN se vykonají pouze po stlačení klávesy <RETURN>. Následující ukázka načítá do dvourozměrného pole PoleSlov(číslo slova,poloha znaku ve slově) 4 slova a nakonec vypíše počet znaků v každém slově. Samozřejmě, že před tímto výňatkem z programu je nutno mít všechny zde uvedené proměnné nadeklarovány předem:

```
Writeln('Zadej 4 slova a vstup ukonci');
Writeln('pomoci klavesy <RETURN>');
FOR Slovo:=1 to 4 DO
BEGIN
  Znak:=0;
  WHILE NOT EOLN DO
  BEGIN
    Znak:=Znak+1;
    Read(PoleSlov[Slovo,Znak])
  END; (* WHILE *);
  writeln('Slovo melo ',Znak:3,' znaku');
  Writeln;
END; (* FOR *)
```

4.11. Rekurzivní procedury a funkce

Procedury a funkce, které volají samy sebe, se nazývají rekurzivní. V programu, který srovnával 6 slov podle abecedy, lze přepsat proceduru Srov tak, aby byla rekurzivní. Procedura bude nyní kratší, ale bude naopak delší po překladu.

```
Procedure Srov(Slova:PoleSlov);
VAR Index:Integer;
BEGIN
  FOR Index:=1 TO MaxSlov-1 DO
  IF (Slova^Index^>Slova^Index+1^) THEN BEGIN
    VymHod(Slova,Index);
    Srov(Slova)
  END
END; (* procedura *)
```

Takto přepsaná procedura testuje všechna slova a jestliže je někde narušena abecední posloupnost, přehodí slova v proceduře VymHod a zavolá opět sama sebe.

Toto byl první příklad použití rekurze. Rekurze se také používá tehdy, chceme-li počítat funkci ve formě nějaké posloupnosti, jako je třeba faktoriál: N!=N*(N-1)*(N-2)*...*1/N-(N-1)/.:

```
Function Faktorial(N:Integer):Integer;
BEGIN
  If N=0 Then Faktorial=1
  ELSE
  Faktorial=N*Faktorial(N-1)
END;
```

4.12. Typ RECORD (záznam)

Datový typ záznam nám umožňuje smísit do jedné proměnné několik různých datových typů. Např. v datumu - třeba 15.leden 1987 - je obsažen typ String ("leden") a dvakrát typ Integer (15,1987). Tyto typy můžeme v Pascalu definovat následovně:

```
TYPE
  TypDatum=RECORD
    Mesic:ARRAY[1..10]OF Char;
    Den:Integer;
    Rok:Integer
  End; (* TypDatum *)
VAR
  Datum:TypDatum;
...
```

Uvedený záznam má 3 položky (Mesic,Den,Rok) a Datum je identifikátor proměnné typu TypDatum. Poslední položka není od klíčového slova END oddělena středníkem. Pokud chceme vypsat jen jednu položku z proměnné Datum, například rok, dosáhneme toho takto:

```
Writeln(Datum.Rok:4:0)
```

Abychom však nemuseli vždy vypisovat jméno proměnné - v tomto případě Datum - můžeme použít příkazu WITH:

```
WITH Datum DO
  BEGIN
    Readln(Mesic);
    Readln(Den);
```

```
      Readln(Rok);
    END; (* WITH *)
```

Tímto postupem načteme do proměnné Datum všechny její položky.

4.12.1. Kopírování recordu

Jestliže jsou dva záznamy stejného typu, tj. všechny jejich položky jsou shodného typu, můžeme prostým přiřazením převést hodnoty z jednoho do druhého:

```
VAR Data1,Data2:DataTyp;
 BEGIN
   Data1:=Data2;
```

Tento postup zkopíruje všechny položky proměnné Data2 do Data1 a výraz Data1=Data2 má hodnotu True.

Abychom si přiblížili možnosti využití typu záznam, uvedeme příklad výpočtu absolutní hodnoty komplexního čísla. Tento vzorec se neliší od vzorce pro výpočet vzdálenosti bodu od počátku:

```
Program Absol;
TYPE
  KomplexTyp=RECORD
    RealCast:Real;
    ImagCast:Real;
  END;
VAR
  KomCis:Komple)Typ;
  Abs:Real;
BEGIN
  WITH KomCis DO
  BEGIN
    Write('Realna cast = ');
    Readln(RealCast);
    Write('Imaginarni cast = ');
    Readln(ImagCast);
    Abs:=Sqrt(Sqr(RealCast)+Sqr(ImagCast));
    Writeln;Writeln('Absolutni hodnota = ',Abs:10:2)
  END
END.
```

Následující program vypočítává přibližný počet dní od 1.1.1980 do zadaného data:

```
Program VypocetCasu;
CONST
  StartDen=1;
  StartMes=1;
  StartRok=1980;
TYPE
  DatumTyp=RECOD
    Den:1..31;
    Mes:1..12;
    Rok:Integer
  END;
```

```
      VAR
        B:Integer;
        Datum:DatumTyp;
        VstMes:ARRAY[1..3]OF Char;
      BEGIN
        Write('Mesic (prvni tri znaky,
cerven=CEN,cervenec=CEC) = ');
        Readln(VstMes);
        WITH Datum DO
        BEGIN
          Write('Den = ');Readln(Den);
          Write('Rok = ');Readln(Rok);
        END; (* WITH *)
        Datum.Mes:=0;
        IF VstMes='LED' THEN Datum.Mes:=1;
        IF VstMes='UNO' THEN Datum.Mes:=2;
        IF VstMes='BRE' THEN Datum.Mes:=3;
        IF VstMes='DUB' THEN Datum.Mes:=4;
        IF VstMes='KVE' THEN Datum.Mes:=5;
        IF VstMes='CEN' THEN Datum.Mes:=6;
        IF VstMes='CEC' THEN Datum.Mes:=7;
        IF VstMes='SRP' THEN Datum.Mes:=8;
        IF VstMes='ZAR' THEN Datum.Mes:=9;
        IF VstMes='RIJ' THEN Datum.Mes:=10;
        IF VstMes='LIS' THEN Datum.Mes:=11;
        IF VstMes='PRO' THEN Datum.Mes:=12;

B:=(Datum.Den-StartDen)+30*(Datum.Mes-StartMes)+365*(Datum.Ro
-StartRok);
        IF Datum.Mes = 0 THEN Writeln('Spatna zkratka
mesice')
        ELSE Writeln('Dni od 1.1.1988: ',B:8)
      END.
```

Tento program počítá s tím, že každý měsíc má 30 dní a každý rok 365 dní. Příkaz IF na konci programu testuje, zda není číslo měsíce rovno nule, což by znamenalo, že nedošlo v předchozích podmínkách k přiřazení jiné hodnoty, a že tudíž došlo k chybnému zadání měsíce. Takováto ošetření, ačkoliv se někdy zdají zbytečná, patří k profesionálně napsaným programům.

4.12.2. Pole záznamů

Stejně jako existují pole jednoduchých proměnných, existují i pole záznamů, se kterými se pracuje naprosto stejně jako s jinými typy:

```
      TYPE
        DataTyp=RECORD
          C1:Integer;
          C2:Real;
          C3:Char
        END;
        PoleTyp=ARRAY[1..20]OF DataTyp;
      VAR
        Pole:PoleTyp;
        Index:Integer;
      BEGIN
```

Jednotlivé prvky pole pak vypisujeme následovně:

```
Writeln(Pole/Index/.C1);
```

kde "Pole" je název proměnné, "Index" je pořadí prvku v poli a "C1" je jeden ze základních typů recordu.

4.12.3. Záznamy s variantní částí

Může se stát, že budeme v programu potřebovat záznam, u něhož se bude jedna nebo více složek složek měnit v nějaké závislosti (např. na hodnotě určité proměnné). Nyní existují dvě cesty: 1 - napsat 2 záznamy s různou spornou položkou:

```
TYPE
  Rec1=RECORD
    A,B:Integer;
    C:Real;
    D:Char
  END;
  Rec2=RECORD
    A,B:Integer;
    C:Real;
    X:Boolean
  END;
```

Ovšem tento způsob je jak náročný na práci programátora, tak na paměť počítače (hlavně jde-li o pole). Problém jde řešit velmi elegantně právě pomocí záznamů s variantní částí:

```
TYPE
  Rec1=RECORD
    A,B:Integer;
    C:Real;
    CASE Podm:Integer OF
      1:(D:Char);
      2:(X:Boolean)
  END;
```

Nyní závisí už jen na proměnné Podm. Bude-li rovna 1, pak se bude poslední složka jmenovat "D" a bude typu Char, bude-li rovna 2, bude se složka jmenovat "X" a bude typu Boolean.

4.13. Množiny

Množina je další definovatelný typ v Pascalu. Množiny mohou mít maximálně 256 prvků a deklarují se následovně:

```
Identif=SET OF ZaklTyp
```

Základní typ, ze kterého tvoříme množinu musí být skalár, ale ne typu Real (přijatelná jsou např. čísla od 10 do 25, prvočísla ...). Následující program Vám zjistí, zda Vámi

zadané číslo je prvkem zadané množiny čísel a je-li prvočíslo:

```
Program Mnoz;
TYPE
  MnozTyp=SET OF 10..25;
VAR
  PrvCis,NePrv:MnozTyp;
  N:Integer;
BEGIN
  PrvCis:=/11,13,17,19,23/;
  NePrv:=/10,12,14,15,16,18,20,21,22,24,25/;
  Write('Zadej cislo od 10 do 25 ');
  Readln(N);
  IF N IN PrvCis THEN
    Writeln('Toto je prvocislo ')
  ELSE
    IF N IN NePrv THEN
      Writeln('Toto neni prvocislo ')
    ELSE
      Writeln('Toto neni cislo ze zadaneho rozmezi ')
END.
```

Příkazy typu "IF N IN PrvCis " zjišťují, zda je N prvkem dané množiny PrvCis. Je-li prvkem, má uvedený příkaz hodnotu True. (Pozn.: Před ELSE se nepíše středník.)

Jistě jste si povšimli podobnosti mezi množinovým typem a již dříve uvedeným typem výčtovým. Rozdílů je několik:

- prvky ve výčtovém typu jsou vždy uspořádány
- proměnná výčtového typu může mít v daném okamžiku pouze jedinou hodnotu, kdežto množina může zahrnovat až 256 hodnot
- s množinami se dají provádět dále uvedené operace

4.14. Operace s množinami

S množinami lze provádět 3 základní operace: sjednocení, průnik a rozdíl.

```
Program Oper;
TYPE
  Cisla=SET OF 1..9;
VAR
  PrvCis,LicCis,Test:Cisla;
BEGIN
  PrvCis:=/1,2,3,5,7/;
  LicCis:=/1,3,5,7,9/;
  Test:=PrvCis+LicCis; (* Sjednoceni = /1,2,3,5,7,9/
*)
  Test:=PrvCis*LicCis; (* Průnik = /1,3,5,7/ *)
  Test:=PrvCis-LicCis; (* Rozdíl = /2/ *)
END.
```

K těmto třem základním operacím existuje ještě 7 relačních operací. Jejich výsledkem je hodnota True nebo False a jsou srovnatelné s aritmetickými relačními operátory, o kterých jsme diskutovali dříve. Jsou to:

```
Rovnost       Mn1  =  Mn2
Nerovnost     Mn1  <> Mn2
Podmnožina    Mn1  <= Mn2
Nadmnožina    Mn1  >= Mn2
Prvek         Mn1  IN Mn2
```

Poslední uvedená relační operace IN znamená, že prvky množiny Mn2 jsou opět množiny a jedna z nich je množina Mn1.

4.15. Datové soubory

Datové soubory jsou množiny informací (čísla, znaky atd.), které vstupují do počítače nebo z něj vystupují. S některými příkazy, které vstup a výstup dat umožňují, jste se již setkali. Jsou to příkazy Read a Readln pro vstup a Write a Writeln pro výstup. Když je informace vkládána do počítače pomocí klávesnice, tvoří soubor Input. Pokud informace opouští počítač a je zobrazována na obrazovce, tvoří soubor Output.

Data však mohou být ukládána nebo čtena i z jiných zařízení než je klávesnice a obrazovka. Jde především o kazetový magnetofon a disketovou jednotku. Zde se mohou data uchovávat libovolně dlouhou dobu bez nároku na energii na rozdíl od paměti počítače nebo monitoru.

Programy využívající výměny dat mezi pamětí a vnějšími médii nejsou tak náročné na velikost uživatelské paměti, ale pomalejší přístup do dat se může u programů běžících v reálném čase odrazit ve výrazném zpomalení programu (i při využití disketové jednotky). Stále musíte mít na paměti, že data se u ATARI přenášejí do (z) počítače sekvenčně, tj. postupně, což je proti paralelnímu přenosu podstatně pomalejší způsob.

4.15.1. Deklarace souborů

Dříve než soubor použijete v programu, musí být deklarován. Prvním krokem v deklaraci souboru je specifikace jména souboru v závorce za jménem programu:

```
Program Uschova(Input, Output, List);
```

Tento zápis pro počítač znamená, že některá data budou získána z jiného souboru než Input, nebo budou vystupovat na jiné zařízení než na monitor (soubor Output).
Pozn.: Specifikace souborů Input a Output není povinná.
V dalším kroku se soubor specifikuje v bloku proměnných:

```
VAR List : FILE OF Integer;
```

což znamená, že prvky souboru List jsou celá čísla. Obecně však soubor může obsahovat i proměnné typu Real, Char a nakonec i uživatelem definované typy - pole, množiny, záznamy.

4.15.2. Zápis do souboru

Abychom mohli uchovat data, nejdříve musíte otevřít soubor pro zápis. K tomu se používá příkaz "Rewrite", který popřípadě také vymaže soubor, který se jmenuje stejně, pokud existuje:

```
Rewrite(List);
```

Pokud pracujete s disketovou jednotkou, je třeba navíc specifikovat jméno, pod kterým bude soubor na disketě zapsán:

```
Rewrite(List,"D1:LST.DAT");
```

nebo může být použita proměnná typu řetězec:

```
Rewrite(List,a);
```

kde a="D1:LST.DAT".

Pokud chcete ukládat data do souboru (musí být již otevřen), jsou k tomu potřebné další dva příkazy:

```
List^ :=J;
Put(List);
```

List^ je jméno proměnné označující buffer (vyrovnávající proměnná) souboru. Před zápisem do souboru je nutné informaci kterou chcete přenést, uložit právě do této proměnné. Prvním krokem je tedy uložení přenášené informace do uvedené proměnné. Příkaz Put potom zapíše hodnotu J uloženou v bufferu souboru do daného souboru. První ukládaný prvek se zapíše na první pozici v souboru, druhý na druhou atd. Je nemožné psát (číst) do souboru, aniž bychom nezačali od první pozice.

Pokud je soubor List souborem typu Integer, pak také proměnná List^ je typu Integer. Následující program uchová celá čísla od 11 do 45 v souboru List:

```
Program Uschova(Input,Output,List);
VAR List: FILE OF Integer;
        J: Integer;
BEGIN
  Rewrite(List,"D1:LST.DAT");
  FOR J:=11 TO 45 DO
  BEGIN
    List^ :=J;
    Put(List)
  END (* FOR *)
END.
```

Pro ty uživatele, kteří vlastní kazetovou verzi bude 5. řádek vypadat následovně:

```
Rewrite(List,"C:");
```

4.15.3. Čtení ze souboru

Stejně jako v případě zápisu musí být soubor nejdříve otevřen, nyní však pro čtení. Toho dosáhnete příkazem "Reset":

```
Reset(List);
```

Pokud chcete číst soubor zapsaný na disketě, budete modifikovat příkaz na:

```
Reset(List,"D1:LST.DAT");
```

nebo použijete proměnnou, jak bylo uvedeno výše.

Jak jste si již řekli, číst soubor můžete pouze sekvenčně, tedy od 1. prvku do posledního. Čtení však můžete kdykoliv ukončit. Postup čtení je inverzní k postupu při zápisu - nejdříve do naší proměnné načtete z bufferu souboru hodnotu proměnné:

```
J:=List^;
```

a poté připravíte do bufferu další hodnotu ze souboru příkazem Get:

```
Get(List);
```

Pozn.: První hodnotu načte do bufferu již příkaz Reset.

Budeme-li číst celý soubor, pak existují 2 možnosti. Buď víte, kolik má prvků a použijete smyčku FOR, nebo tento údaj neznáte a pak bude řešení následující:

```
Reset(List,"D1:LST.DAT");
WHILE NOT EOF(List) DO
BEGIN
  J:=List^;
  Writeln(J);
  Get(List)
END;
```

V ukázce jste použili předdefinovanou funkci EOF (Konec souboru - End of File), která nabývá hodnoty True, pokud se pokusíte číst po posledním prvku souboru. Jinak má hodnotu False. Někdy nám stačí nalézt jen jeden určitý prvek souboru:

```
Reset(List,"D1:LST.DAT");
WHILE NOT EOF(List) DO
BEGIN
  IF 77=List^ THEN ...
  Get(List)
END;
```

Je-li prvek roven 77, pak se vykonají příkazy za slovem THEN, není-li, načte se další až do přečtení celého souboru.

4.15.4. Textové soubory

Protože se často používají soubory složené ze znaků, existuje v Pascalu standardní typ souboru - Text, který je předdefinován jako "Text = FILE OF Char;". Soubor se deklaruje obdobně jako předchozí soubory:

```
Program TextProc(Jmeno);
VAR Jmeno:Text;
```

Ačkoliv se může vstup a výstup realizovat stejně jako u předchozích souborů, může se použít následujícího zjednodušení:

```
Read(Jmeno,Identifikator);
```

se může použít místo

```
Identifikator:=Jmeno^; Get(Jmeno);
```

Také můžete použít

```
Write(Jmeno,Identifikator);
```

místo

```
Jmeno^:=Identifikator; Put(Jmeno);
```

Jestliže v příkazech Read a Write není uvedeno jméno souboru (Jmeno), pak se chápou jako soubory Input a Output.

4.15.5. Soubory záznamů

Většina souborů má položky typu záznam. Na následujících příkladech ukáži, jak pracovat s takovými soubory. První program vytvoří na disketě soubor, který bude obsahovat jména (do 15 znaků) a příslušné roky narození. Druhý program pak po dotazu na jméno vypíše příslušný rok narození. Je tedy jasné, že první program je ukázkou výstupu z počítače, druhý ukázkou vstupu.

Program č. 1 (výstup):

```
Program VystupDat(Out);
Type
  String=Array/1..15/OF Char;
  KartTyp=Record
    Jmeno:String;
    Rok:Integer
  END;
  OutSoubor=FILE OF KartTyp;
VAR
  Out:OutSoubor;
  A:String;
BEGIN
  Writeln('Vkladani ukonci slovem KONEC ');
  A:="KONEC          ";
  Rewrite(Out,"D1:TESTOUT.DAT");
```

```
    REPEAT
      Write('Jmeno: ');
      Readln(Out^.Jmeno);
      IF Out^.Jmeno<>A THEN BEGIN
        Write('Rok narozeni: ');
        Readln(Out^.Rok);
        Put(Out)
      END;
    UNTIL Out^.Jmeno=A
  END.
```

Program č. 2 (vstup):

```
  Program VstupDat(Inp);
  Type
    String=Array[1..15]OF Char;
    KartTyp=Record
      Jmeno:String;
      Rok:Integer
    END;
    InpSoubor=FILE OF KartTyp;
  VAR
    Inp:InpSoubor;
    Kart:KartTyp;
    A:String;
  BEGIN
    Writeln('Program ukonci slovem KONEC ');
    REPEAT
      Write('Jmeno: ');
      Readln(A);
      IF A<>"KONEC          " THEN BEGIN
        Reset(Inp,"D1:TESTOUT.DAT");
        REPEAT
          Kart:=Inp^;
          IF Kart.Jmeno=A THEN Writeln(Kart.Rok);
          Get(Inp);
        UNTIL EOF(Inp) OR (Kart.Jmeno=A)
      END;
    UNTIL A="KONEC.          "
  END.
```

4.15.6. Soubory s přímým přístupem

Standardní Pascal nezahrnuje možnost souborů s přímým přístupem (ve smyslu dostupnosti určitého prvku souboru - např. 22.). Nicméně se často dostanete do takové situace, že potřebujete pouze určitou položku nebo část datového souboru. Pokud je tato část více vzdálena od začátku souboru, program se významně zpomaluje. Většinu souborů lze v Kyan Pascalu změnit ze sekvenčního na relativní a tím umožnit přístup k libovolnému prvku souboru. K tomu slouží funkce Seek.

Pozn.: Soubory typu Char nebo Boolean však zůstanou zapsány sekvenčně .

```
  Seek(F,N); (* Umisti buffer souboru F na N-tý prvek *)
  Put(F); (* Zapíše obsah bufferu souboru F na N-tou
pozici *)
```

```
Get(F); (* Načte obsah N-tého prvku souboru F a uloží ho do
bufferu *)
```

Pozn.: První prvek souboru má číslo 0.

```
      Program DemonstraceSeek(F);
      TYPE
        String=ARRAY[1..32] OF Char
      VAR
        F:FILE OF String;
        C:Char;
      PROCEDURE CTENI;
      VAR I:Integer;
      BEGIN
        Write('Jakou pozici chcete? ');
        Readln(I);
        Seek(F,I);
        Get(F);
        IF NOT EOF(F) THEN Writeln(F^); (* EOF má hodnotu
true, je-li prvek prazdny *)
      END;
      PROCEDURE ZAPIS;
      VAR I:Integer;
      BEGIN
        Write('Na kterou pozici? ');
        Readln(I);
        Write('Data? ');
        Readln(F^);
        Seek(F,I);
        Put(F)
      END;
      BEGIN
        Reset(F,"D1:DATA");
        REPEAT
          Writeln('C-Cteni, Z-Zapis, K-Konec");
          Readln(C);
          IF C="C" THEN CTENI;
          IF C="Z" THEN ZAPIS;
        UNTIL C="K"
      END.
```

Aby byl uvedený program funkční, je nejprve nutné vytvořit na disketě prázdný soubor DATA. V Pascalu je možno připsat příslušnou proceduru:

```
      PROCEDURE NEW(F);
      BEGIN
        Rewrite(F,"D1:DATA");
      END;
```

kterou spustíte ještě před příkazem Reset.

4.16. Proměnná typu ukazatel

Předpokládejme tuto část programu:

```
      VAR
```

```
  Cislo:Integer;
BEGIN
  Cislo:=54;
```

Kdybychom testovali paměť počítače, zjistili bychom, že předchozí příkazy uloží číslo 54 do zvláštních paměťových buněk. Pro tento příklad předpokládejme, že se číslo 54 uloží na adresy 12156 a 12157:

```
Cislo = 5 -> 12156
        4 -> 12157
```

Jiná, efektivnější cesta, jak dostat určité číslo do paměti, je použití ukazatelů:

```
VAR
  Cislo:^Integer;
BEGIN
  New(Cislo);
  Cislo^:=54;
```

Jestliže bychom nyní testovali paměť počítače, nalezli bychom opět číslo 54 ve zvláštních paměťových buňkách, např. 11343 a 11344. Ale nalezli bychom také uchovanou hodnotu 11343 v paměti:

```
   Cislo = 1 -> 11338
           1 -> 11339
           3 -> 11340
           4 -> 11341
           3 -> 11342
  Cislo^ = 5 -> 11343
           4 -> 11344
```

"Cislo" je skupina paměťových buňek, které ukazují na tu část paměti, kde je uchováno číslo 54. V prvním uvedeném příkladě žádný takovýto ukazatel neexistuje.

"Cislo" je proměná typu ukazatel, zatímco Cislo^ označuje vlastní data. Symbol pro ukazatel (^) se při deklaraci objeví na levé straně názvu proměnné typu ukazatel v deklarační části (Cislo:^Integer), ale na pravé straně identifikátoru pro uložená data (Cislo^:=54). Je také možno definovat proměnnou typu ukazatel následovně:

```
TYPE
  TypCisla = ^Integer;
```

4.16.1. New

New je standardní procedura v Pascalu, která vyhrazuje oblast paměti pro proměnnou typu ukazatel. Pokaždé, když se vykoná procedura New, je proměnné přiřazena nová oblast paměti. Pokud bychom v uvedeném příkladě vynechali tuto proceduru, mohlo by se stát, že by číslo 54 bylo umístěno do

již obsazené části paměti, což by se mohlo později projevit jako chyba.

4.16.2. DPEEK a DPOKE

Záměrně zde používám názvy příkazů známých z některých interpretů jazyka Basic, abych osvětlil rozdíl mezi tím, co jsem vyložil o proměnné typu ukazatel a tím, co bude následovat, i když v Pascalu žádná takováto standardní funkce (PEEK) ani procedura (POKE) není obsažena (je nutno si ji nenadefinovat).

Pokud totiž použijete příkaz New, pak nemůžete ovlivnit někdy velice podstatnou věc, tj. kam se mají data do paměti uložit. Týká se to především operací s těmi částmi paměti, které umožňují řešit problémy, které se pomocí příkazů vyšších programovacích jazyků nedají na počítači ATARI řešit. Jde o operace s video pamětí a Display listem, ovládání barev, zvuku, PM grafiky atd.

Je však nutné mít stále na zřeteli, že dále uvedené řešení pracuje s 16-ti bitovými slovy. Nemůžete tedy např. na adresu 709 uložit číslo, aniž bychom neovlivnili adresu následující (710).

Operace (PEEK a POKE) s 8-bitovými informacemi budou popsány dále. Proto budu nyní používat označení DPOKE a DPEEK.

Operace DPEEK bude tedy vypadat takto:

```
Program TestMem;
VAR
  Cislo:^Integer;
  A:Integer;
BEGIN
  Write("Kterou adresu ? ");
  Readln(A);
  Assign(Cislo,A);
  Write(Cislo^)
END.
```

Zeptáme-li se tedy nyní na obsah adresy 88 (začátek video RAM), obdržíte výsledek podle algoritmu:

Obsah adresy 88 + 256*obsah adresy 89

tedy 16-ti bitové číslo, které v tomto případě přímo udává pozici levého horního rohu obrazovky v paměti.
Pozn.: Protože proměnná A může být pouze typu Integer, což znamená v rozsahu -32767 až +32767, ale paměť se čísluje od 0 do 65535, zobrazí se vyšší čísla než 32767 jako záporná podle vzorce:

Ekviv. paměť. buňka = Paměť. buňka - 65535.

Operace DPOKE vypadá následovně:

```
Program PsaniDoPameti;
VAR
  Cislo:^Integer;
  A,B:Integer;
```

```
BEGIN
  Write("Na kterou adresu ? ");
  Readln(A);
  Write("Co na tuto adresu ulozit ? ");
  Readln(B);
  Assign(Cislo,A);
  Cislo^:=B;
END.
```

Vkládáte opět 16-bitovou informaci (B), takže ji pro vyšší čísla než 32767 přepočítáte podle výše uvedeného vzorce.

Nyní následuje ukázka využití - výpis abecedy na 1. řádek obrazovky:

```
Program Abeceda;
TYPE
  Obrazovka=ARRAY[0..959] OF Char;
VAR
  Znaky:^Obrazovka;
  VideoRAM:^Integer;
  I:Integer;
BEGIN
  Writeln(chr(125));
  Assign(VideoRAM,88);
  Assign(Znaky,VideoRAM^);
  FOR I:=0 TO 25 DO
    Znaky^[I]:=Chr(I+Ord("A")-32);
END.
```

Příkaz Assign(VideoRAM,88); načte do proměnné VideoRAM^ adresu začátku video paměti - je tedy použit jako DPEEK. Druhý příkaz Assign ve spojení s přiřazením Znaky^[I]:=... je pak použit jako DPOKE. Posunutí o 32 v předposledním řádku je způsobeno tím, že kód znaků je ve video paměti posunut proti ATASCII o 32.

V tomto případě si nemusíte lámat hlavu tím, že nejde o přenos 8-mi, ale 16-ti bitové informace - ukládáte totiž vždy dva znaky. Ale díky tomu, že informace je vždy nižší než 256, je druhým znakem mezera (kód 0) a to na výsledek nemá vliv.

4.16.3. Spojování seznamů a identifikátor NIL

Ukazatele nám též umožňují vytvořit zřetězené seznamy. NIL je konstanta, která představuje prázdný ukazatel. V následujícím příkladě je deklarován typ ukazatel a záznam, který v sobě zahrnuje proměnnou typu ukazatel - proměnnou Link, která spojí všechny záznamy a umožní jejich následné třídění:

```
Program Ukazatel(Input,Output);
TYPE
  String=ARRAY[1..15] OF Char;
  Appointer=^AppointRec;
  AppointRec=Record
    Link:Appointer;
    Jmeno:String;
  END;
```

```
VAR
  Appointm,Pt:Appointer;
BEGIN
  Pt:=NIL;
  New(Appointm);
  Appointm^.Jmeno:="Ernie          ";
  Appointm^.Link:=Pt;
  Pt:=Appointm;
  New(Appointm);
  Appointm^.Jmeno:="Bob            ";
  Appointm^.Link:=Pt;
  Pt:=Appointm;
  New(Appointm);
  Appointm^.Jmeno:="Gina           ";
  Appointm^.Link:=Pt;
  Pt:=Appointm;
  WHILE Appointm<>NIL DO BEGIN
    Writeln(Appointm^.Jmeno);
    Pt:=Appointm^.Link;
    Appointm:=Pt
  END
END.
```

Konstanta NIL se používá k indikaci posledního prvku seznamu. Seznam se totiž čte pozpátku systémem first in = last out, což znamená, že prvky jsou v seznamu uloženy jako v zásobníku - první ukládaný prvek je tedy až na dně, vyjmeme ho tedy jako poslední. Výpis začíná v našem programu příkazem WHILE. Protože ukazatel na první prvek má hodotu NIL (na začátku programu byla přiřazena), máme zajištěnu kontrolu toho, že jsme na konci (resp. na prvním uloženém prvku) seznamu. Po spuštění tohoto programu se zadaná jména vypíší v opačném pořadí než v jakém byla vkládána - tedy Gina, Bob, Ernie.

4.16.4. Dispose

Tato funkce vymaže obsah paměťových buňek pro ukazatel Appointm^. Jestliže chceme smazat celý seznam, musí být použito pro každý prvek seznamu:

```
Dispose(Appointm);
```

4.17. Externí procedury a funkce - INCLUDE

Kyan Pascal umožňuje uživateli připojit k programu své vlastní knihovny programů a podprogramů, aniž by je do programu musel vždy vypisovat. Každá procedura nebo funkce, která se často využívá v programech, může být nahrána na disketě nebo kazetě zvlášť (pochopitelně jen zdrojový text, nelze kompilovat části programů, třeba proceduru). A pak už jen na tom místě programu, kde má být tato část programu umístěna, napíšeme např. následující:

```
#I D1:JmenoSbr.Ext
```

Znak "#" se musí bezpodmínečně objevit v 1. sloupci

sloupci editoru (tedy úplně vlevo) a "I" ve sloupci druhém. Samozřejmě, že při kompilaci musí být na disketě se zdrojovým programem i všechny ostatní programy, které jsme v programu uvedli.
Například si na disk vytvoříme proceduru Halo pod jménem D1:Halo.I:

```
Procedure Halo;
BEGIN
  Writeln("Ahoj kamaradi")
END;
```

Po nahrátí této procedury na disk vymažeme editor a napíšeme hlavní program:

```
Program Ukazka;
#I D1:Halo.I
BEGIN
  Halo
END.
```

Pozn.: Na konci řádku, který obsahuje příkaz #I se nepíše středník a bývá dobrým zvykem pojmenovat soubor na disku stejně jako se jmenuje daná procedura nebo funkce.
Nemusí se vkládat jen procedury nebo funkce, ale také např. několik řádek programu. Pak příslušný #I vložíme do potřebného místa programu.

5. Assembler v Pascalu

Kyan Pascal umožňuje přímé programování v assembleru, tj. v instrukcích strojového kódu. Díky této výhodě můžeme vytvářet mnoho podprogramů, procedur a funkcí, které se jen s použitím Pascalu dají programovat jen těžko nebo vůbec ne. Tyto rutiny nejsou ani v nejmenším omezeny strukturou programu. Jediný požadavek, který musíme splnit je uzavření rutiny
v assembleru mezi slova BEGIN a END.

Do režimu programování v assembleru se dostaneme tak, že do prvního sloupce napíšeme znak "#" a za něj znak "a". Tento režim ukončíme zapsáním znaku "#" do prvního sloupce. Jako příklad nám poslouží procedura na zdržení (pauzu):

```
PROCEDURE Pauza;
BEGIN
#a
          LDY #100
SMYCKA    DEY
          BNE SMYCKA
#
END;
```

Na uvedeném příkladě si ukážeme hned 2 zvláštnosti:
- pouze návěští začínají v prvním sloupci
- za příkazy se nedělá středník

Pozn.: Jméno návěští nesmí začínat na "L", protože tato návěští používá překladač a mohlo by dojít k chybě.

5.1. Pseudo instrukce v assembleru

Tyto příkazy může programátor použít, ale nejsou součástí kódu mikroprocesoru. V Kyan Pascalu je takovýchto příkazů 6 a nesmí začínat v prvním sloupci, aby nebyly zaměněny s návěštím:

```
ORG - udává adresu počátku programu
EQU - definice konstanty
DB  - definice byte
DW  - definice slova (2 byte)
<   - významější byte (Hi)
>   - méně významný byte (Lo), tedy:
>$FF01 = $0001
<$FF01 = $00FF
```

5.2. Použití pascalských proměnných v assembleru

V předchozí ukázce uvedená procedura vždy při svém průchodu zdržela program o určitou hodnotu. Tato hodnota byla napevno dána a nelze měnit pomocí proměnných z Pascalu. Abychom mohli ovlivnit proměnnou v Pascalu, musíme vědět, kde ji najít. Umístění těchto proměnných není nikdy absolutní, ale vždy relativní k ukazateli překladače, zvaném LOCAL. Oblasti pascalských proměnných mohou být též spočítány vzhledem ke stack pointru (SP).

V následující ukázce se hodnota proměnné Ce přiřadí

akumulátoru A:

```
PROCEDURE CeDoA(Al,Be,Ce:Integer);
VAR M,N:Integer;
BEGIN
#a
     LDY #7
     LDA (SP),Y
#
END;
```

V prvním řádku se registru Y přiřadí hodnota 7. V dalším řádku se akumulátoru A přiřadí hodnota prvního byte z proměnné Ce. Proč zrovna 7?
Proměnné se do akumulátoru ukládají postupně a na vrchu zásobníku zůstávají vždy 3 nulové byte:

Pozn.: Registr X používá překladač jako ukazatel zásobníku. Chceme-li tento registr použít ve vlastním programu, musíme ho nejdříve uschovat a po použití tuto hodnotu do X registru opět vrátit.
Následující tabulka ukazuje, kolik je třeba na daný typ v Pascalu vyhradit místa:

Real	8 bytes
Integer	2 bytes
Char	1 byte
Boolean	1 byte
Ukazatel	2 bytes
ARRAY/1..n/ OF Char	n bytes
ARRAY/1..n/ OF Boolean	n bytes
ARRAY/1..n/ OF Integer	2*n bytes
Hodnotový par. (HP)(Real)..	8 bytes
HP (Integer)	2 bytes

```
HP (Integer) ..............  2 bytes
HP (Char,Boolean) .........  1 byte
HP ARRAY/n/OF Char, Boolean n bytes
HP ARRAY/n/OF Integer .....  2* bytes
Všechny parametry
        volané odkazem ....  2 bytes
```

Parametry volané odkazem jsou parametry v závorkách při deklaraci procedur a funkcí. Liší se od hodnotových parametrů tím, že se nenaplňuje paměť hodnotou proměnné, ale zapisuje se pouze její adresa, na kterou je třeba 2 byte.
V Pascalských programech vždy deklarace předcházejí tělo hlavního programu, procedury a funkce a tedy jsou oblasti proměnných snadněji spočitatelné. Je nutné vždy počítat oblast proměnných relativně k začátku procedury, funkce nebo hlavního programu, ve kterém se vyskytuje.

5.3. Předdefinovaná návěští

Následující tabulka ukazuje absolutní umístění SP, LOCAL, a T. SP a LOCAL obsahují adresy spodku a vrcholu Pascalského zásobníku proměnných. T je začátek oblasti přechodných registrů, kterých může být 16 - od T do T+15

```
SP          EQU 4
LOCAL       EQU 2
T           EQU 16
```

5.4. Chain

Složitější a delší programy, které by se nevešly do paměti nebo by už nezbývalo místo na pole apod., se dají rozdělit
na samostatně fungující části, které se volají z diskety (popř. z kazety) postupně. Samozřejmě, že se při tomto procesu dají přenášet všechny proměnné ze starého programu do nového.
Mějme dva programy (ALFA a BETA):

```
Program BETA;
VAR
  D,E,F:Integer;
  Y:Real;
  L:Char;
BEGIN
...
END.
```

```
Program ALFA;
VAR A,B,C:Integer;
        X:Boolean;
        P:Char;
BEGIN
...
  Chain("D1:BETA.O");
...
```

Nyní přeložíme program BETA a přeloženému programu ponecháme jméno BETA.O. Pak přeložíme program ALFA. Když spustíme program ALFA.O, běží, dokud nenarazí na příkaz Chain. Nyní podle deklarací převede proměnné z programu ALFA do programu BETA takto: hodnoty proměnných A,B a C převede do proměnných D,E a F. Hodnotu proměnné X samozřejmě nemůže převést do proměnné Y, protože to nejsou proměnné stejných typů. A zrovna tak nepřevede již žádnou další proměnnou. Pak se program ALFA smaže a dál poběží program BETA, do kterého jsme z programu ALFA převedli proměnné A,B a C. Další proměnné (Y a L) zůstanou prázdné.

5.5. Mapa paměti

* FMS je uložen od $800 do $2000 a od $9400 do $BF00

** Stack začíná na $93FF

*** Uživatelská paměť začíná na $2000

**** Knihovna je uložena od $9400 do $BBFF. Video Ram a Display List jsou od $BC00 do $BFFF

6. Některé užitečné procedury a funkce

Nyní následuje popis procedur a funkcí, které jsou na disku s Kyan Pascalem již hotové.
Začneme programy pro práci s řetězci:
Protože STRING není předdefinovaný Pascalský typ, je nutno ho deklarovat:

```
CONST
  Maxstring=10 (* 10 je příklad *)
TYPE
  String=ARRAY/1..Maxstring/OF Char;
```

Nyní můžeme připsat pomocí INCLUDE název funkcí a procedury, které máme pro manipulaci s řetězci k dispozici. Jsou to funkce Substrin, Length, Index a procedura Concat. Tedy např. za uvedenou řádku TYPE napíšeme:

```
#I   D1:Substrin.I
#I   D1:Index.I
BEGIN
...
```

-- Length :
String končí první mezerou nebo posledním znakem v poli. Funkce Length vrací délku Stringu. Např.:

```
Program Priklad;
CONST
  Maxstring=10 (* 10 je příklad *)
TYPE
  String=ARRAY/1..Maxstring/OF Char;
VAR
  S:String;
#i   D1:Length.I
BEGIN
  S:="abcd      ";
  IF (Length(S)=4) THEN Writeln("Je to pravda")
END.
```

Funkce Length může být použita i v příkazu writeln pro formátování výstupu :

```
Writeln(S:Length(S));
```

-- Concat
Tato procedura spojuje dva řetězce do třetího. Jestliže máme 3 řetězce S1,S2 a S3 a S1="kdo   ", S2="koliv   ", pak po vykonání procedury:

```
Concat(S1,S2,S3);
```

bude S3="kdokoliv "

-- Index
Tato funkce vypočítává pozici jednoho řetězce ve druhém. Hledáme-li řetězec S1="d   " v řetězci S2="kdokoliv ",

pak následující výraz má hodnotu True:

```
Index(S1,S2)=2;
```

Pokud funkce nenalezne řetězec S1 v řetězci S2, pak vrací hodnotu 0.

-- Substrin
Tato funkce nám z daného řetězce vyjme specifikovanou část. Je-li S1="abcdef " a chceme z něj vyjmout 2.-4. znak, vypadá funkce následovně:

```
S2=Substrin(S1,2,3)
```

(Od druhého znaku počínaje vyjmi 3 znaky, tedy S2="bcd ").

Dále jsou na disketě rutiny pro grafiku, zvuk a generování náhodných čísel. Jmenují se stejně jako příkazy v jazyce BASIC, pracují podobně, podobné jsou i parametry:

Procedura	Parametry
Graphics	Integer
Setcolor	Registr, Barva, Odstín
Plot	Horiz., Vert., Barva
Drawto	Horiz., Vert., Barva
Position	Horiz., Vert.
Locate	Horiz., Vert., Data
Random	bez parametrů
Sound	kanál, výška, zkreslení, hlasitost

Nyní následuje několik procedur a funkcí v assembleru, které na disketě s Pascalem nenajdete.
Funkce Peek a procedura Poke již byly popsány, ale jen jako 16-ti bitové operace s pamětí. Následující funkce Peek a procedura Poke pracují jen s 8-mi bitovou informací:

```
function Peek(Loc:Integer):Integer;
BEGIN
#A
     LDY #5; LOC
     LDA (SP),Y
     STA T
     INY
     LDA (SP),Y
     STA T+1

     LDY #0; PEEK
     LDA (T),Y
     LDY #3
     STA (SP),Y
     INY
     LDA #0
     STA (SP),Y
#
END;
```

```
Procedure Poke(Loc,Val:Integer);
BEGIN
#A
   LDY #5; Loc
   LDA (SP),Y
   STA T
   INY
   LDA (SP),Y
   STA T+1
;
   LDY #3; Val
   LDA (SP),Y
;
   LDY #0; Poke
   STA (T),Y
#
END;
```

Následující funkci vytvořil člen pražského ATARI klubu Petr Jaroš:

```
Function Consol:Integer;
BEGIN
#a
     LDA 53769
     LDY #3
     STA (SP),Y
     LDA #0
     INY
     STA (SP),Y
#
END;
```

Tato funkce vrací hodnotu, která udává stav konzole, tedy např. 7 v klidovém stavu, 6 při stisknuté klávese start apod. Pokud zaměníme v prvním řádku číslo 53769 za 624, budeme testovat Paddle(0), nebo 632 - test Stick(0), 644 - Trigger(0) apod.

Byli bychom rádi, aby uživatelé Pascalu všechny své připomínky, náměty a nové zkušenosti předali svým kolegům prostřednictvím Zpravodaje ATARI klubu.

Publikované zo súhlasom - viď Prohlášení představitelů AK Praha.

Igi/2021