1 Úvod do Turbo Pascalu

1.1 Klávesové zkratky TP

- spuštění TP : tp.exe, bp.exe, tpx.exe apod.
- nápověda:
    - F1 - volá vysvětlení (help)
    - Ctrl + F1 - help podle polohy kurzoru
    - Alt + F1 - vrací předcházející obrazovku helpu
- do hlavního menu: F10
- zpět do vyššího menu: Esc
- ukončení TP: Alt + X
- spuštění programu:
    - Ctrl + F9
    - po kroku F8
- prohlížení výstupu na obrazovce: Alt + F5

1.2 Příkazy TP

Program jméno;	Osobní popis programu, není potřebný pro chod programu.
uses crt;	Určení dané jednotky, máme možnost mazat obrazovku, měnit barvu pozadí a písma, zvukové efekty, prací s kódem kláves apod.
begin a end.	Mezi ně zadáváme příkazy, které program vykonává.
clrscr;	Mazání obrazovky.
write('text');	Vypisuje text na obrazovku bez ENTER.
writeln('text');	Vypisuje text na obrazovku s ENTER.
writeln;	Vynechání řádku.
readln;	Potvrzování klávesou ENTER.
readln(proměnná);	Vkládání informací a následné potvrzení klávesou ENTER.
var	Načítá do paměti proměnné, s kterými potom program může pracovat. My jsme si zvolili proměnnou typu *string*, tedy řetězec, která slouží k práci se znaky. Tzn. můžeme pracovat jak s písmeny, tak s čísly, ale i se zbytkem znaků ASCII tabulky. String má kapacitu znaků od 1 do 255. Proměnná typu byte pracuje s číselnými daty pouze kladnými od 0 do 255. Pokud budete chtít zadávat data záporná, doporučuji nadefinovat proměnnou typu integer. Ten má hodnoty v rozmezí od -32768 do 32767 nebo longint s rozmezím od -2147483648 do 2147483647. Ve všech případech jde pouze o celá čísla. Pro čísla desetinná je nejlépe použít proměnnou real.
gotoxy(x,y);	Zajistí umístění kurzoru na obrazovce v poloze zadané souřadnicemi x (vodorovná poloha = horizontální) a y (svislá poloha = vertikální).
random();	Slouží ke generování náhodných čísel
delay();	Startuje pauzu, pokud je hodnota v závorce 1, trvá 1 milisekundu

(ms).

until keypressed; Viz repeat – until cyklus.

1.3 Cykly

1. While cyklus (cyklus se vstupní podmínkou)
- cyklus zapisuje takto: ***while*** podminka ***do*** telo_cyklu
- ***while*** znamená dokud a ***do*** dělej
- tělem cyklu rozumíme příkazy, které se mají vykonávat v cyklu

2. Repeat-until cyklus (cyklus s výstupní podmínkou)
- umožní opakovat část kódu v té smyčce obsaženou
- ***repeat*** má vlastnost "otevření brány" do doby než bude zavřena příkazem ***until*** …(tedy jde o cyklus s podmínkou na konci)
- za ***until*** můžeme použít např. příkaz ***keypressed***
- příkaz ***until keypressed;*** vykonává zadané příkazy do doby stisku klávesy

3. For cyklus (cyklus s určeným počtem opakování)
- používáme, pokud potřebujeme provést cyklus s přesně stanoveným počtem opakovaní
- tento cyklus má dvě podoby:
  - *Vzestupný:*
    for I:=DolniMez ***to*** HorniMez
    do Prikaz;
  - *Sestupný:*
    for I:=HorniMez ***downto*** DolniMez
    do prikaz;
- za slůvkem ***for*** se proměnné přiřadí počáteční hodnota, následuje slůvko ***to*** resp. ***downto*** a za ním se udá konečná hodnota, pak již následuje ***do*** a příkaz cyklu

1.4 Nastavení barvy textu

- příkazem ***textcolor();*** měníme barvu textu:

- 1 (modrá)
- 2 (zelená)
- 3 (azurová)
- 4 (červená)
- 5 (fialová)
- 6 (okrová)
- 7 (světle šedá)
- 8 (tmavě šedá)
- 9 (světle modrá)
- 10 (světle zelená)
- 11 (světle azurová)
- 12 (světle červená)
- 13 (světle fialová)
- 14 (žlutá)
- 15 (bílá)

1.5 Podmínka if a then

- příkazy ***if*** a ***then*** se překládají jako "pokud" nebo "jestliže" a "pak"
- ***if*** určuje, za jaké podmínky se vykoná kód za příkazem ***then***.

1.6 Procedury

Procedury jsou podprogramy, které jsou volány v různých částech programu. Procedury definujeme mimo vlastní tělo programu, musí být popsány unikátním jménem, mají vlastní begin a end.

```
program Procedures;

uses crt;

procedure Hello;
begin
   Writeln('Hello');
end;


begin

clrscr;

   Hello;

repeat until keypressed;

end.
```

Procedura musí být vždy nad místem, z kterého je volána. Procedura může volat jinou proceduru.

```
program Procedures;

uses crt;

procedure Hello;
begin
   Writeln('Hello');
end;

procedure HelloCall;
begin
   Hello;
end;

begin

clrscr;
   HelloCall;

repeat until keypressed;
end.
```

Procedura může obsahovat i parametry nebo jiné příkazy, které je potřeba použít. Každý parametr musí být jednoznačně identifikován a pojmenována je používán jako jakýkoliv jiná proměnná. Více parametrů oddělujeme středníkem.

```
program Procedures;

uses crt;

procedure Print(s: String; i: Integer);
begin
   Writeln(s);
   Writeln(i);
end;

begin

clrscr;
   Print('Hello',3);

repeat until keypressed;

end.
```

Procedura s proměnnou.

```
program Procedures;

uses crt;

procedure Print(s: String);
var
   i: Integer;
begin
   for i := 1 to 3 do
      Writeln(s);
end;

begin

clrscr;
   Print('Hello');


repeat until keypressed;

end.
```

1.7 Funkce

Funkce jsou téměř jako procedury.

```
program Functions;

uses crt;

function Add(i, j:Integer): Integer;
begin
   Add := i + j;
```

```
end;

begin
clrscr;
   Writeln(Add(1,2));


repeat until keypressed;

end.
```

Proceduru nebo funkci lze ukončit kdykoliv příkazem *Exit.*

```
program Procedures;
uses crt;

procedure GetName;
var
   Name: String;
begin
   Writeln('What is your name?');
   Readln(Name);
   if Name = '' then
      Exit;
   Writeln;
   Writeln('Your name is ',Name,'.');
end;

begin
clrscr;
   GetName;

repeat until keypressed;
end.
```

1.8 Pole

Pole je proměnná proměnných stejného typu mající stejné jméno.

Pole se definují stejně jako obyčejné proměnné, musí však obsahovat informaci kolik obsahují elementů. Počet elementů píšeme do hranatých závorek.

```
program Arrays;
uses crt;

var
   a: array[1..5] of Integer;

begin
clrscr;
   a[1] := 12;
   a[2] := 23;
```

```
   a[3] := 34;
   a[4] := 45;
   a[5] := 56;

   write(a[1],' ');
   write(a[2],' ');
   write(a[3],' ');
   write(a[4],' ');
   write(a[5]);

repeat until keypressed;
end.
```

Je jednodušší načítat proměnné do pole pomocí cyklu.

```
program Arrays;

uses crt;

var
   a: array[1..5] of Integer;
   i: Integer;

begin

clrscr;
   for i := 1 to 5 do
      Readln(a[i]);


   writeln;

   for i := 1 to 5 do

      Write(a[i],' ');



repeat until keypressed;
end.
```

Pole může být i více rozměrné. Takové pole má mimo řádku také sloupec.

```
program Arrays;
uses crt;

var
   a: array [1..3,1..3] of Integer;
   i,j: Integer;

begin
   for i := 1 to 3 do
      for j := 1 to 3 do
         Readln(a[i,j]);

   writeln;
```

```
   for i := 1 to 3 do
      for j := 1 to 3 do
        Writeln(a[i,j],' ');

repeat until keypressed;
end.
```

1.9 Třídění

Vnitřní třídění je takové třídění, kdy předem známe tříděné prvky a máme k dispozici dostatek volné paměti, abychom je uvnitř této paměti mohli setřídit. Opakem je pak třídění vnější, kdy jednotlivé prvky předem neznáme a musíme je ze vstupu postupně načítat v pořadí, v jakém přicházejí. Toto třídění se provádí typicky v situacích, kdy máme operační paměti málo a nejsme schopni v ní všechny prvky uložit. Teoreticky tak můžeme setřídit daleko větší množství dat.

1.9.1 Bubble sort (bublinkové třídění)

Algoritmus postupně "bere" jednotlivé prvky pole (N-1 prvků) a posouvá je dále doprava, dokud jsou větší než prvek před nimi.

Způsob třídění:

Nechť máme nesetříděné pole pěti čísel, vezmeme prvek na 1. pozici (zde 9) a postupně ho porovnáváme s ostatními. Pokud je prvek z 1. pozice větší, dojde k přehození prvků, pokud ne, prvek zůstane před větším prvkem a pokračujeme dále. Takto pokračujeme do úplného setřídění pole. Zde si můžete prohlédnout schéma třídění (**x** prvek, který budeme porovnávat s ostatními, x konečná pozice prvku v daném kroku):

Nesetříděné pole: 9 5 3 8 1

1. krok: **9** 5 3 8 1 => 5 3 8 1 9
2. krok: **5** 3 8 1 9 => 3 5 **8** 1 9 => 3 5 1 8 9
3. krok: **3** 5 1 8 9 => 3 **5** 1 8 9 => 3 1 5 8 9
4. krok: **3** 1 5 8 9 => 1 3 5 8 9

Setříděné pole: 1 3 5 8 9

```
procedure BubbleSort(var pole:TPole; N:word);
var i,j, pom: integer;
begin
  for j:=1 to N-1 do {probublavani provadime pro n-1 prvku}
    for i:=1 to N-1 do {postupne pro vsechny prvky pred poslednim}
      if pole[i]>pole[i+1] then {pokud je prvek vetsi nez nasledujici}
      begin {prehod prvky => probublavani vetsiho prvku polem}
        pom:=pole[i+1]; {prvky prohodime pomoci pomocne promenne pom}
        pole[i+1]:=pole[i];
        pole[i]:=pom
```

```
      end

end.
```

1.9.2 Třídění přímým výběrem minima

Algoritmus postupně prochází pole a na dané místo hledá minimum ze zbývajících prvků.

Způsob třídění:

Nechť máme nesetříděné pole pěti čísel, vezmeme prvek na 1. pozici (zde 9) a ve zbytku čísel na pravé straně hledáme minimum. Po nalezení minima tyto dvě hodnoty porovnáme. Pokud je prvek z 1. pozice větší, dojde k přehození tohoto prvku s minimem, pokud ne, prvek zůstane na své pozici. V dalším kroku vezmeme prvek na 2. pozici a porovnáváme stejným způsobem jako v prvním kroku. Takto pokračujeme do úplného setřídění pole. Zde si můžete prohlédnout schéma třídění (**x** prvek, který budeme porovnávat s ostatními, x_m nalezené minimum, x konečná pozice prvku v daném kroku):

Nesetříděné pole: 9 5 3 8 1

1. krok: **9** 5 3 8 1_m => 1 5 3 8 9
2. krok: 1 **5** 3_m 8 9 => 1 3 5 8 9
3. krok: 1 3 **5** 8_m 9 => 1 3 5 8 9
4. krok: 1 3 5 **8** 9_m => 1 3 5 8 9

Setříděné pole: 1 3 5 8 9

```
 procedure PrimyVyber(var pole:TPole; N:word);
var i,j, min, pom: integer;
begin
  for i:=1 to N-1 do {na tyto pozice budeme vybirat minimum}
  begin
    min:=i; {nastaveni pozice minima}
    for j:=i+1 to N do  {vyhledani mozneho minima v dalsich prvcich}
      if pole[j]<pole[min] then min:=j;
    if pole[min]<pole[i] then {pokud byl nalezen mensi prvek}
    begin                     {-> provedeni vymeny prvku}
      pom:=pole[i];
      pole[i]:=pole[min];
      pole[min]:=pom
    end
  end

end.
```

1.9.3 Třídění přímým vkládáním

Algoritmus prvky z pravé části pole přímo zatřiďuje do levé části, kde se utváří seřazená posloupnost.

Způsob třídění:

Nechť máme nesetříděné pole pěti čísel, vezmeme prvek na 2. pozici (zde 5) a porovnáváme ho s prvkem na 1. pozici. Pokud je prvek z 1. pozice větší, dojde k přehození prvků, pokud ne, prvek zůstane na své pozici. V dalším kroku vezmeme prvek na 3. pozici a porovnáváme ho s prvky nacházející se na 2. a 1. pozici stejným způsobem jako v prvním kroku. Takto pokračujeme do úplného setřídění pole. Zde si můžete prohlédnout schéma třídění (**x** prvek, který budeme porovnávat s ostatními, x konečná pozice prvku v daném kroku):

Nesetříděné pole: 9 5 3 8 1

1. krok: 9 **5** 3 8 1 => 5 9 3 8 1
2. krok: 5 9 **3** 8 1 => 3 5 9 8 1
3. krok: 3 5 9 **8** 1 => 3 5 8 9 1
4. krok: 3 5 8 9 **1** => 1 3 5 8 9

Setříděné pole: 1 3 5 8 9

```
procedure PrimeVkladani(var pole:TPole; N:word);
{prvky postupne zarazujeme do leve casti pole - vytvarime setridene pole}
var i,j, pom: integer;
begin
  for i:=2 to N do {prochazime nesetridenou cast pole}
  begin
    pom:=pole[i]; {ulozime si zatrizovany prvek}
    for j:=i-1 downto 1 do {tento i-ty prvek zarazujeme do leve setridene casti pole}
      if pole[j]>pole[j+1] then begin {jestlize je prvek vlevo vetsi, prvky prehodime}
        pole[j+1]:=pole[j];
        pole[j]:=pom
      end
  end

end.
```

1.9.4 QuickSort (rekurzivní)

QuickSort vychází z rekurzivního přístupu: Pokud máme třídit elementární pole o dvou prvcích, prvky jednoduše přehodíme na tu stranu, kam patří (porovnat je můžeme s pomyslnou hodnotou mezi nimi - pivotním prvkem). Pokud máme třídit pole větší, roztřídíme takto podle zvoleného pivota i dva velké úseky pole, které potom dále

necháme třídit rekurzivně stejnou procedurou, až budou nakonec uspořádány všechny části pole.

Způsob třídění:

Nechť máme nesetříděné pole pěti čísel, vezmeme prvek (tzv. pivota) na prostřední pozici (zde 3), který nám pole rozdělí na 2 části. V levé části nalezneme maximum, v pravé minimum a porovnáme je s pivotním prvkem, jestliže je maximum z levé strany vetší a minimum z pravé strany menší, oba prvky (maximum a minimum) vzájemně prohodíme. Takto pak paralelně pokračujeme v každé části pole zvlášť (opět vybereme pivotní prvek, a hledáme maximum a minimum v levé a pravé části pole) až do úplného setřídění pole. Zde si můžete prohlédnout schéma třídění (**x** pivotní prvek, x^m nalezené maximum, x_m nalezené minimum, x konečná pozice maxima a minima v daném kroku, | rozdělení pole):

Nesetříděné pole: 9 5 3 8 1

1. krok: 9^m 5 **3** | 8 1_m => 1 5 3 | 8 9
2. krok: 1 **5**^{m} | 3_m => 1 3 5
8^m | 9_m => 8 9

Setříděné pole: 1 3 5 8 9

```
procedure QuickSort(var pole:TPole; Zac,Kon:integer);
{procedura setridi v poli usek od indexu Zac do indexu Kon}
var P: integer; {hodnota pro rozdeleni pole na useky - pivot}
    pom: integer; {pomocna promenna pro vymenu prvku}
    i,j: integer; {pracovni indexy pro vymezeni casti pole}
begin
  i:=Zac; {na zacatku zabiraji mezni indexy cele pole}
  j:=Kon;
  P:=pole[(Zac+Kon) div 2];
{urcime si pivotni prvek - vezmeme prostredni prvek pole}
{idealni pivot by byl median - prostredni z hodnot v poli}
  repeat
{nalevo od pivota budeme radit mensi prvky, napravo vetsi prvky nez
pivot}
    while pole[i]<P do inc(i);
{posouv me levě index, dokud neni na prvku vetsim nez pivot}
    while pole[j]>P do dec(j);
{posouvame pravy index, dokud neni na prvku mensim nez pivot}
    if i<j then {pokud se indexy neprekrizily a nejsou shodne}
    begin
      pom:=pole[i]; {vymenime nalezene prvky}
      pole[i]:=pole[j];
      pole[j]:=pom;
```

```
      inc(i); dec(j); {a posuneme indexy na dalsi prvky}
    end
    else if i=j then {pokud se indexy sesly (ukazuji na pivota)}
    begin
      inc(i);
{posunume je jeste dale, aby vymezovaly roztridene poloviny pole}
      dec(j);
{a doslo k prekrizeni, coz vede k ukonceni cyklu}
    end
  until i>j;
{opakujeme, dokud nejsou obeti casti pole roztrideny podle pivota}
{Pole [Zac,Kon] je rozdeleno na 2 useky [Zac,j] a [i,Kon], ktere
zpracujeme rekurzivne (opet touto procedurou)}
  if Zac<j then QuickSort(pole,Zac,j);
{ma smysl tridit nejmene dva prvky}
  if i<Kon then QuickSort(pole,i,Kon);
end.
```