Anotace

- Definice vlastních datových typů (výčtové datové typy)
- Konstrukce `case ... of ...`
- Ordinalni typy - typ char, funkce ord, chr, succ, prev, inc, dec, Motivace: Máme dlouhé číslo (nebo číslo ve stringu).
- Zapis cisla v pozicni soustave, jeho vyhodnoceni Hornerovym schematem,
- Evaluace polynomu Hornerovym schematem,
- Fronta a zásobník,
- Základní třídící algoritmy
- Direktivy překladače
- Soubory (textové)

Další využití tvorby vlastních typů

Chceme počítat dny v týdnu. Jak to uděláme?

- Očíslujeme si dny takto: Pondělí=1, Úterý=2,...

Další využití tvorby vlastních typů

Chceme počítat dny v týdnu. Jak to uděláme?

- Očíslujeme si dny takto: Pondělí=1, Úterý=2,...
- Jenže já to přečísluju: Pondělí=0, Úterý=1,...

Další využití tvorby vlastních typů

Chceme počítat dny v týdnu. Jak to uděláme?

- Očíslujeme si dny takto: Pondělí=1, Úterý=2,...
- Jenže já to přečísluju: Pondělí=0, Úterý=1,...
- Přijde američan a očísluje: Neděle=1, Pondělí=2,...

Další využití tvorby vlastních typů

Chceme počítat dny v týdnu. Jak to uděláme?

- Očíslujeme si dny takto: Pondělí=1, Úterý=2,...
- Jenže já to přečísluju: Pondělí=0, Úterý=1,...
- Přijde američan a očísluje: Neděle=1, Pondělí=2,...
- ... anebo Neděle=0, Pondělí=1,...

Další využití tvorby vlastních typů

Chceme počítat dny v týdnu. Jak to uděláme?

- Očíslujeme si dny takto: Pondělí=1, Úterý=2,...
- Jenže já to přečísluju: Pondělí=0, Úterý=1,...
- Přijde američan a očísluje: Neděle=1, Pondělí=2,...
- ... anebo Neděle=0, Pondělí=1,...
- Proto raději uděláme zvláštní typ indexovaný dny v týdnu a čísla necháme na překladači.

Výčtový datový typ

- Definujeme v sekci `type`,
- jednotlivé hodnoty klademe do závorek a oddělujeme čárkou.
- Příklad: `type dnyvtydnu=(pondeli,utery,streda, ctvrtek,patek, sobota,nedele);`
- Anebo definujeme přímo proměnnou tohoto typu: `var kal:(pondeli,utery,streda,ctvrtek,patek, sobota,nedele);`

Příklad

- Chceme vyrobit jednoduchý "kalendář" na rok 2012, tedy vypsat datum a údaj o dni v týdnu.
- Pro jednoduchost předpokládejme, že každý měsíc má 30 dnů...
- Zdrojový kód je na webu (kam.mff.cuni.cz/ perm/programovani/enum.pas).
- V příkladu vidíme, že funkce `write` neumí vypsat příslušné názvy, bylo by proto pěkné v závislosti na čísle dne v týdnu vypsat příslušný text. Nápady?
- Buďto mnoho klauzulí `if`, nebo: `case proměnná of ...`

Konstrukce `case ... of ...`

- Umožňuje vytvořit mnoho větví programu v závislosti na obsahu jedné proměnné.
- Syntax:

```
case jméno proměnné of
    hodnota1: příkaz nebo blok
    hodnota2: příkaz nebo blok
    else příkaz nebo blok
end;
```

- Provede se jen větev označená aktuální hodnotou proměnné, `else`-větev je pro ostatní (explicitně neuvedené) případy.
- Klauzule `else` nemusí být přítomna!
- Je-li poslední klauzule blok, jde `end` dvakrát po sobě (první uzavře blok posledních příkazů, druhý uzavře blok `case`).

Příklad – kalendář s jmény dnů

je na adrese
`kam.mff.cuni.cz/ perm/programovani/case_of.pas`.

Ordinální datové typy

- Typy, jejichž hodnoty tvoří lineárně uspořádanou množinu (tedy pro každou dvojici je jasné, který prvek je větší).
- Z pascalských typů jsou to: `integer`, `longint`, `byte`, `char`, `word`, `boolean` a `shortint` (a další definované uživatelem, zejména výčtový typ a interval).
- Pro ordinální typy jsou definovány funkce "Ord", "Pred" a "Succ".

Funkce pro ordinální typy

- Funkce `Ord` vrátí ordinální hodnotu (tedy hodnotu pro část typů, konkrétně pro `integer`, `longint`, `byte`, `word` a `shortint`).
- Pro typ `char` vrátí ASCII hodnotu příslušného znaku. Umíme tedy zjistit ASCII-hodnotu jednotlivých znaků.
- Co naopak? Použijeme funkci `Chr`, která vrátí znak s příslušným číslem.
- Úloha: Jak převedeme číslo uložené v integeru na řetězec znaků?
- Řetězec je reprezentován jako pole znaků (až na drobná omezení).

Příklad

```
var a,i,j:integer; text,pom:string[255];
begin pom:='xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx';
      readln(a); i:=1; j:=1;
      while a<> 0 do
      begin pom[i]:=chr(a mod 10+48);
            a:=a div 10; i:=i+1;
      end;
      delete(pom,i,255-i); i:=i-1;
      text:=copy(pom,1,length(pom));
      while i>0 do
      begin text[j]:=pom[i];
            i:=i-1; j:=j+1;
      end;
      writeln(text);
end.
```

Hornerovo schéma

- Co když chceme konvertovat obráceně? (řetězec na integer)
- Použijeme tzv. Hornerovo schéma, tedy začneme od začátku řetězce (nejvyšší číslice).
- Zjistíme její hodnotu a postupujeme indukcí:
  Dosavadní výsledek vynásobíme deseti a přičteme číslici na dalším řádu.

číslo $a_n a_{n-1} a_{n-2} ... a_0$ zapsané v desítkovém zápisu je vlastně $a_n 10^n + a_{n-1} 10^{n-1} + ... + a_0$. A platí:

$$a_n 10^n + a_{n-1} 10^{n-1} + ... + a_0 = (...((a_n * 10) + a_{n-1} * 10) + ... + a_1) * 10 + a_0$$

Tímto způsobem můžeme vyhodnocovat čísla zapsaná v různých (pozičních) soustavách (dvojkové, čtyřkové, pětkové, šestkové...).

Příklad

```
program x;
var a:string;
    i,hod:longint;
begin
     readln(a); i:=1; hod:=0;
     while i<=length(a) do
     begin
     hod:=10*hod+ord(a[i])-ord('0');
          i:=i+1;
     end;
     writeln(hod);
end.
```

Evaluace polynomu

- Máme zadán polynom $a_n x^n + a_{n-1} x^{n-1} + \ldots + a_0$.

Evaluace polynomu

- Máme zadán polynom $a_n x^n + a_{n-1} x^{n-1} + ... + a_0$.
- Chceme tento polynom vyhodnotit (evaluovat), tedy určit jeho hodnotu v zadaném bodě x.

Evaluace polynomu

- Máme zadán polynom $a_n x^n + a_{n-1} x^{n-1} + ... + a_0$.
- Chceme tento polynom vyhodnotit (evaluovat), tedy určit jeho hodnotu v zadaném bodě x.
- Možnosti?

Evaluace polynomu

- Máme zadán polynom $a_n x^n + a_{n-1} x^{n-1} + ... + a_0$.
- Chceme tento polynom vyhodnotit (evaluovat), tedy určit jeho hodnotu v zadaném bodě x.
- Možnosti?
- Hrubá síla (počítat $a_n x^n$, $a_{n-1} x^{n-1}$,... a sečíst)

Evaluace polynomu

- Máme zadán polynom $a_n x^n + a_{n-1} x^{n-1} + ... + a_0$.
- Chceme tento polynom vyhodnotit (evaluovat), tedy určit jeho hodnotu v zadaném bodě x.
- Možnosti?
- Hrubá síla (počítat $a_n x^n$, $a_{n-1} x^{n-1}$,... a sečíst)
- nebo Hornerovo schéma

$$\sum_{i=0}^{n} a_i x^i = ((...(a_n x + a_{n-1})x + ... + a_1)x + a_0).$$

Evaluace polynomu Hornerovým schématem

- 1: Načti koeficient u nejvyššího (dosud neprošlého) stupně,
- dosud načtené hodnoty vynásob hodnotou x,
- přičti hodnotu nově načteného koeficientu,
- GOTO 1;

Evaluace polynomu Hornerovým schématem

```
program nic;
var i,a,souc,stupen,x:integer;
{Vyhodnot polynom stupne stupen v bode x, do a
nacitej koeficienty}
begin
      readln(stupen); readln(x);
      souc:=0;
      for i:=0 to stupen do
      begin souc:=souc*x;
            readln(a);
            souc:=souc+a;
      end;
      writeln('Hodnota polynomu je: ',souc);
end.
```

Odbočka – labely a GOTO

- V Pascalu je možno provádět poměrně nekontrolované skoky po programu.
- Za sekcí globálních proměnných (`var`) vytvoříme sekci `label`, kde vyjmenujeme použité labely,
- následně můžeme těmito labely označit vybraná místa programu
- a pomocí `goto label;` udělat nepodmíněný skok.
- GOTO nepoužívejte, používám ho jen já v pseudokódu k pedagogickým účelům.

Fronta a zásobník

- Fronta je datová struktura osazená operacemi `zařaď` a `vyřaď`,
- `zařaď` zařadí daný prvek na konec fronty,
- `vyřaď` vyřadí daný prvek ze začátku fronty.
- Zásobník je datová struktura osazená operacemi `push` a `pull`.
- `push` přidá na konec zásobníku, `pull` vytáhne z konce zásobníku.

Figurky na šachovnici anebo prohledávání grafu - algoritmus vlny

- Na děravé šachovnici je umístěn král. Určete jak rychle se dostane na určené (cílové) pole?

Figurky na šachovnici anebo prohledávání grafu - algoritmus vlny

- Na děravé šachovnici je umístěn král. Určete jak rychle se dostane na určené (cílové) pole?
- Vlastně jde o úlohu prohledávání grafu.

Figurky na šachovnici anebo prohledávání grafu - algoritmus vlny

- Na děravé šachovnici je umístěn král. Určete jak rychle se dostane na určené (cílové) pole?
- Vlastně jde o úlohu prohledávání grafu.
- Podobné: Thesseus hledá Mínótaura.

Figurky na šachovnici anebo prohledávání grafu - algoritmus vlny

- Na děravé šachovnici je umístěn král. Určete jak rychle se dostane na určené (cílové) pole?
- Vlastně jde o úlohu prohledávání grafu.
- Podobné: Thesseus hledá Mínótaura.
- Rozdíl: Tehdy nám nešlo o nejkratší cestu.

Algoritmus vlny (myšlenky)

- Na šachovnici vysadíme na specifikované políčko mravence,
- mravenci polezou rovnoměrně všemi dostupnými směry,
- mravenci zkusí všechny cesty, rychleji než první mravenec se tam dostat nelze.
- Jiná intuice: Na příslušné poličko vychrstneme vodu,
- voda teče všemi dostupnými cestami, až doteče na cílové políčko.

Algoritmus vlny (implementace)

- Vytvoř prázdnou frontu f.
- Nastav vzdálenost do všech políček kromě startovního na nekonečno, vzdálenost do startovního nastav na 0.
- Přidej do f startovní políčko.
- Dokud není fronta f prázdná, opakuj:
  - a:=vyřad'(f);
  - Pro všechny sousedy z pole a zkus:
  - Pokud vzdálenost do z je větší než vzdálenost do a +1, nastav políčku z vzdálenost $a+1$ a zařad'(z, f).

Problém třídění – motivace

- Máme načtena data (například čísla),
- chceme je zpracovat například v rostoucím pořadí.
- Jak to udělat? Setřídíme, zpracujeme.
- Předpokládejme, že data jsou v poli.

Problém třídění – jednoduché třídící algoritmy

- Bublinkové třídění (BubbleSort),
- zatřiďování alias třídění přímým vkládáním (InsertSort),
- třídění vyběrem (SelectSort),
- QuickSort.

Bublinkové třídění

- Geometrická interpretace:
  Bublinky v kapalině jdou zpravidla vzhůru
- Myšlenka: Porovnáváme po sobě jdoucí čísla (ve smyslu sousední v zadaném poli) od prvního k poslednímu, jsou-li v nesprávném pořadí, prohodíme je.
- Prvky "probublávají""správným"směrem.
- Opakujeme bublání, dokud se prohazuje.

Bubblesort v pseudokódu

```
prohazovalose:=true;
while prohazovalose:=true do
begin
    for i:=1 to pocet - 1 do
    begin
        prohazovalose:=false;
        if pole[i]>pole[i+1] then
        begin prohod(pole[i],pole[i+1]);
              prohazovalose:=true;
        end;
    end;
end;
```

Složitost bubble-sortu

- Kolikrát se provede vnější `while`-cyklus?

Složitost bubble-sortu

- Kolikrát se provede vnější `while`-cyklus?
- V i-tém kroku dojede i-tý největší na své místo!

Složitost bubble-sortu

- Kolikrát se provede vnější `while`-cyklus?
- V i-tém kroku dojede i-tý největší na své místo!
- Stačí tedy n-krát probublat, jedno probublání porovná po sobě jdoucí dvojice, tedy má složitost lineární.

Složitost bubble-sortu

- Kolikrát se provede vnější `while`-cyklus?
- V i-tém kroku dojede i-tý největší na své místo!
- Stačí tedy n-krát probublat, jedno probublání porovná po sobě jdoucí dvojice, tedy má složitost lineární.
- Složitost `BubbleSortu` je tedy $O(n^2)$.

Složitost bubble-sortu

- Kolikrát se provede vnější `while`-cyklus?
- V i-tém kroku dojede i-tý největší na své místo!
- Stačí tedy n-krát probublat, jedno probublání porovná po sobě jdoucí dvojice, tedy má složitost lineární.
- Složitost `BubbleSortu` je tedy $O(n^2)$.
- Implementaci, kdy se střídavě bublá z jedné strany na druhou a z druhé na první se říká `ShakeSort` a funguje pro něj stejný odhad složitosti.

Třídění přímým výběrem a zatřiďováním

Přímý výběr:

- Opakuj, dokud není tříděné pole prázdné:
- Najdi v poli minimum a přesuň ho na konec setříděného pole.

Zatřiďování:

- Opakuj, dokud není tříděné pole prázdné:
- Vyjmi z něj první prvek a zatřiď do cílového pole, tedy: najdi pozici, kam prvek patří, přidej ho tam a zbytek setříděného pole posuň (o jedna dál).

Analýza složitosti: $n$krát opakujeme proces, který trvá nejvýše n kroků, tedy také $O(n^2)$.

Quicksort – třídění za pomoci rekurze – idea:

- Pokud třídíme jedno číslo, nic nedělej (posloupnost je setříděna),
  tedy vrať posloupnost tak, jak jsme ji dostali.
- V poli POLE vyber jeden prvek (dále pivot).
- Rozděl POLE na pole A obsahující prvky menší než pivot
- a na pole B obsahující prvky větší nebo rovné pivotu.
- Pomocí sebe sama setřiď pole A,
- pomocí sebe sama setřiď pole B,
- Vypiš: pole A, pivot, pole B.

Direktivy překladače

- Překladač kontroluje plno věcí, například:
- zda nekoukáme za konec pole,
- zda nám nepřetekl zásobník,
- anebo zda nenastala chyba na vstupu/výstupu...
- Většinou je užitečné mít kontroly zapnuté, někdy však "víme, co děláme".
- V tom případě můžeme na nezbytnou dobu chování překladače změnit pomocí tzv. *direktiv překladače.*
- Direktivy vypadají jako komentář, tedy jsou ve složených závorkách, ovšem začínají znakem string ($), jméno je zpravidla 1znakové a následuje přepínač $+/-$.

Direktivy překladače:

- Příklad: $\{\$R-\}$ – vypni *range-checking*.
- Nejdůležitější:
  - $\$Q$ – overflow-checking,
  - $\$R$ – range-checking,
  - $\$I$ – test vstupu a výstupu,
  - Úplný seznam najdete v helpu (některé direktivy se liší podle překladačů).

Soubory a práce s nimi

- V tomto semestru budou pouze soubory textové. Ačkoliv existují i soubory binární, ty budou předmětem výuky až v létě!
- Textový soubor ovládáme pomocí proměnné typu `Text`.
- Příslušnou proměnnou "napojíme" na daný soubor pomocí funkce `Assign`,
- otevřeme pomocí funkce `Reset`, `Rewrite` nebo `Append`,
- soubor čteme pomocí funkce `Read` (resp. `Readln`), které jako první argument předáme příslušnou proměnnou typu `Text`, zapisujeme analogicky pomocí funkcí `Write` a `Writeln`.
- Nakonec soubor uzavřeme pomocí funkce `Close`.

Práce se souborem – syntax (1)

- `var f:Text;`
- `Assign(f,'soubor.txt');` – asociuj proměnnou `f` se souborem `soubor.txt`.
- `Reset(f);` – otevři soubor `f` (pro čtení).
- `Rewrite(f);` – otevři soubor `f` a jeho dosavadní obsah znič.
- `Append(f);` – otevři soubor `f` pro zápis za jeho dosavadní konec.

Práce se souborem – syntax (2)

- `Writeln(f,'Zapiseme text do souboru');` – zapiš do souboru příslušný text.
- `Read(f,a);` – načti ze souboru proměnnou `a`.
- `Close(f);` – uzavři soubor (už s ním nebudeme pracovat).
- `eof(f);` – funkce, která sdělí, zda jsme na konci souboru.
- `eof;` – funkce oznamující konec standardního vstupu (z klávesnice).
- Existuje mnoho dalších funkcí jako `Rename`, `Erase`, ...

Potíže se soubory

- Často se stane, že otevíraný soubor neexistuje.
- Tato událost vyvolá input/output error.
- Nechceme-li při zapnuté této direktivě překladače soubor zničit (pomocí `Rewrite`, které sice neexistující soubor založí, ale existující přemaže), použijeme direktivu překladače a zda nastala chyba zjistíme pomocí funkce `IOResult`.

Příklad

```
Assign(f,'soubor.txt');
{$I-} {Vypni test na vstupně-výstupní chyby}
Reset(f);
{$I+} {Zapni test vstupně-výstupních chyb}
if IOResult<>0 then
begin writeln('Chyba!'); halt;
end;
while not eof(f) do begin
      readln(f,s);
      writeln(s);
end;
```

Pozor, `IOResult` je funkce a po zavolání ztratí hodnotu, nelze ji tedy číst opakovaně a její výsledek je případně třeba uložit do proměnné!