Anotace

- Ukazatele a jejich aplikace:
- Spojové seznamy,
- fronta, zásobník,
- uspořádaný spojový seznam,
- samoopravující seznamy,
- hashování,
- stromové datové struktury.

Pointery alias ukazatele

- Paměť je organizována lineárně v podobě jednotlivých adres.
- Na těchto adresách jsou ukládány hodnoty (kód, data).
- Paměť zpravidla adresujeme přirozenými čísly, která zapisujeme v šestnáctkové soustavě.
- Na data v paměti si tedy můžeme ukazovat.
- Pod těmito ukazateli můžeme mít údaje libovolných typů, z čehož vyplývá jisté nebezpečí.
- Z toho vzplývá jisté nebezpečí a nutnost být obezřetný.

Pointery – syntax a sémantika

- S pointery pracujeme zpravidla tak, že definujeme datový typ *ukazatel na něco.*
- *Ukazatel na* řekneme pomocí operátoru stříšky:
- `type pint=^integer; {ukazatel na integer}`
- Následně definujeme proměnnou příslušného typu:
  `var ukaz:pint;`
- Pod pointer "koukneme" opět pomocí operátoru stříšky:
  `writeln(ukaz^);`
- Jenže ono to zdaleka není tak snadné!

Organizace paměti s ohledem na program

- Program sestává z kódu, tzv. statických dat, prostoru zásobníku a oblasti haldy.
- Kam ukazuje pointer, který si lehkomyslně vyrobíme?
- Pokud s ním budeme zacházet rozumně, bude ukazovat někam do oblasti haldy, k tomu ale máme ještě daleko.
- Kam tedy pointer ukazuje?
- V lepším případě na adresu 0, v horším na náhodně vybraný kus paměti.
- Pořádek v paměti hlídá allokátor, který ví, které části paměti jsou použité a které ne a může nám přidělit prostor.

Allokátor a jeho použití

- Abychom mohli pod pointer kouknout, musíme ho někam nasměrovat. A to buďto na už existující pointer (`var a,b:pint;... naalokuj b;... a:=b;`),
- anebo "urafnutím" nové paměti: `new(a);`
- Funkci `new` předáváme jako parametr proměnnou typu ukazatel.
- Funkce `new` najde v paměti vhodné místo a nasměruje na něj příslušný pointer.
- Cvičení zimního učiva: Bere `new` parametr hodnotou nebo referencí?
- Od chvíle, kdy máme naalokováno, můžeme pod pointer koukat i zapisovat:
- `new(a); a^:=5; writeln(a^);` Ale pozor na přesměrování ukazatele!

Pointery a práce s nimi

- `var a,b:pint;`
- `new(a);` – naallokuje místo pro proměnnou typu integer
- `a^:=5;` – zapíše pod pointer hodnotu 5.
- `b^:=a^;` – okopíruje pod pointer `b` hodnotu, na kterou ukazuje pointer `a`.
- `b:=a;` – okopíruje pointer (tedy `a` a `b` ukazují na stejné místo).
- Co v kontextu uvedeného kódu udělá `b^:=10; writeln(a^);`?
- Pozor, více pointery si můžeme ukazovat na to samé místo!

Deallokace

- Nepotřebujeme-li už příslušné místo, musíme to říci allokátoru voláním funkce `dispose`:
- `dispose(a);`
- Tím prohlásíme, že pod dotyčný pointer už nepotřebujeme.
- Pozor pokud si na totéž místo ukazujeme víckrát, nesmíme provést vícenásobné odallokování!
- Pokud pointer přesměrujeme bez odallokování (ničím si na dotyčné místo neukazujeme), vzniká tzv. garbage, o které má allokátor za to, že je používaná, my však už nemáme, jak se k dotyčnému kusu paměti dostat (tak ho nemůžeme ani odallokovat).
- Některé jazyky (Java, C#) takto odallokovávají (přestaneme si na kus paměti ukazovat). Říká se tomu garbage collector, je to nevýchovné a v Pascalu není implementován.

Typičtější použití pointerů

- V zimě jsme se učili o strukturách (records).
- Struktury nám umožňovaly udržovat údaje různých typů spolu souvisejících. Například dvě souřadnice bodu v rovině, údaje o lidech, údaje o knihách...
  Do recordu se přistupuje operátorem tečky.
- Typicky se dělají pointery na struktury:
- Příklad:

```
type tbod=record
x,y:integer;
end;
pbod=^tbod;
```

- Stále ještě neřešíme původní problém, co když máme bodů víc a nevíme předem kolik?

- "Přivřeme" do struktury ukazatel na další prvek:
- ```
  type pbod=^tbod;
  tbod=record
        x,y:integer;
        next:pbod;
  end;
  ```
- Tomu říkáme spojový seznam.
- Jak poznáme, který ukazatel někam ukazuje?
- Tak, že nepoužitý pointer okamžitě nasměrujeme na nulu, tedy `nil`.
  ```

Spojový seznam čísel – načti a vypiš

Příklad – datové struktury

```
type pint=^sint;
      sint=record
             cislo:integer;
             next:pint;
      end;
var spojak,pom:pint;
```

Spojový seznam čísel – načti a vypiš

Příklad – tělo kódu

```
begin spojak:=nil; pom:=nil;
      while not EOF do
      begin new(pom);
            readln(pom^.cislo);
            pom^.next:=spojak;
            spojak:=pom;
      end;
      while spojak<>nil do
      begin writeln(spojak^.cislo);
            pom=spojak;
            spojak=spojak^.next;
            dispose(pom);
      end;
end.
```

Typologie spojových seznamů

- jednosměrný a obousměrný – prvek ukazuje jen na následníka, nebo na předka i následníka,
- s hlavou, bez hlavy, s ocasem, bez ocasu – hlava je první (prázdný) prvek, ocas je poslední (prázdný prvek). Vždy máme aspoň nějaký prvek seznamu.
- cyklický – poslední prvek si jako na následníka ukazuje na začátek seznamu.

Fronta a zásobník

- Fronta je datová struktura organizující prvky způsobem FIFO.
- Je osazena funkcemi `enqueue` a `dequeue`.
- Zásobník organizuje prvky způsobem LIFO.
- Je vybaven funkcemi `push` a `pop`.
- Jak je implementovat pomocí spojových seznamů?

Zásobník

Implementace

```
type pzas=^zas;
zas=record
      hod:integer;
      next:pzas;
end;
var hlava:pzas;
procedure init;
begin
      hlava:=nil;
end;
```

Zásobník

Implementace

```
type pzas=^zas;
zas=record
      hod:integer;
      next:pzas;
end;
var hlava:pzas;
procedure push(co:integer);
var pom:pzas;
begin
      new(pom);
      pom^.hod:=co;
      pom^.next:=hlava;
      hlava:=pom;
end;
```

Zásobník

Implementace

```
function pop:integer;
var pom:pzas;
begin
      pom:=hlava;
      if hlava<>nil then
      begin pop:=hlava^.hod;
            hlava:=pom^.next;
            dispose(pom);
      end else
      begin writeln("Chyba!");
            pop:=-1;
      end;
end;
```

Fronta

Implementace

```
type pfr=^fronta;
fronta=record
      hod:integer;
      next:pfr;
end;
var hlava,ocas:pfr;
procedure init;
begin
      hlava:=nil;
      ocas:=nil;
end;
```

```
procedure enqueue(co:integer);
var pom:pfr;
begin if hlava=nil then
      begin new(hlava);
            ocas:=hlava;
            hlava^.next:=nil;
            hlava^.hod:=co;
      end else
      begin new(pom);
            pom^.next:=nil;
            pom^.hod:=co;
            hlava^.next:=pom;
            hlava:=pom;
      end;
end;
```

```
function dequeue:integer;
var pom:pfr;
begin if hlava=nil then
      begin dequeue:=-1;
      end else
      begin if hlava=ocas then
            begin dequeue:=ocas^.hod;
                  dispose(ocas);
                  hlava:=nil; ocas:=nil;
            end else
            begin dequeue:=ocas^.hod;
                  pom:=ocas;
                  ocas:=ocas^.next;
                  dispose(pom);
            end;
      end;
end;
```

Prohození prvků ve spojovém seznamu

Prohoď prvek s jeho následníkem v seznamu

```
procedure prohod(var hlava:seznam;co:seznam);
var pom:seznam;
begin pom:=hlava;
      if hlava=co then
      begin hlava:=hlava^.next;
            pom^.next:=hlava^.next;
            hlava^.next:=pom;
      end else
      begin while(pom^.next<>co) do
                  pom:=pom^.next;
            pom^.next:=co^.next;
            co^.next:=pom^.next^.next;
            pom^.next^.next:=co;
end; end;
```

Dynamické datové struktury

- V příkladu je kvůli místu zanedbáno ošetření singulárních případů (prázdný seznam, prvek chybějící v seznamu, jednoprvkový seznam, prohazování posledního prvku a podobně). Vše je ale jen testování pointerů na `nil`.
- Cvičení na prohazování prvků ve spojovém seznamu: Bubblesort
- Organizace setříděného spojového seznamu (funkce insert, delete a member, které vkládají do setříděného spojového seznamu, mažou z něj a zjišťují, zda je hodnota ve spojovém seznamu.

Uspořádaný spojový seznam

- Spojový seznam, ve kterém jsou prvky uspořádány v (BÚNO) neklesajícím pořadí.
- Nad spojovými seznamy typicky implementujeme funkce:
  - `member` – zjistí, zda je prvek s příslušným klíčem přítomen,
  - `insert` – přidá prvek s příslušným klíčem,
  - `delete` – smaže prvek s příslušným klíčem.
- Příklad viz web, resp. si ho napíšeme.

Další datové struktury

- Samoopravující seznamy:
- Move-front rule, transposition rule:
- Pokud přistupujeme k prvku v seznamu, dotyčný prvek vytáhneme na začátek, resp. prohodíme s předchůdcem (myšlenka: pokud přistupujeme k prvku, zpravidla to děláme vícekrát za sebou).
- Má-li prvek více pointerů, vznikají stromové datové struktury.
- Binární vyhledávací strom (pokud možno co nejlépe vyvážený),
- Vyváženost se těžko vynucuje, proto AVL-stromy či červeno-černé stromy, kde se vyváženost vynucuje pomocí rotací.

Reprezentace stromů

- Strom je (v této chvíli) datová struktura, ve které má každý vrchol nějaký počet synů a nejvýše jednoho rodiče.
- Vrchol bez rodiče je jen jeden a označujeme ho kořen.
- Ve stromě nesmějí vznikat cykly (ani slabé).
- Jak reprezentovat v Pascalu?
- Podobně jako spojový seznam, třeba takto:

```
type  strom:^vrchol;
      vrchol=record
               hod:longint;
               syn1:strom;
               syn2:strom;
               ...
      end;
```

Binární vyhledávací strom

- Jedná se o binární strom, ve kterém pro každý vrchol všechny prvky v levém podstromě mají klíč menší než je klíč tohoto vrcholu a prvky v pravém podstromě mají klíč větší (než současný vrchol).
- V tomto stromě půjde snadno vyhledávat.
  Co výhody a nevýhody?
- Pokud se dobře postaví, je mnohem efektivnější, než spojový seznam.
- Problém je nepostavit ho špatně a při přidávání a ubírání nestrávit příliš času.
- Vyváženost odkazuje k tomu, jak moc se liší počty prvků v jednotlivých podstromech.

Binární vyhledávací strom

- Jak postavit BVS?
- Najít medián, ten zakořenit, menší i větší vyřešit rekurzívně a z rekurze vzešlé podstromy zavěsit jako levého resp. pravého syna.
- Jak přidat prvek?
- Podívat se, zda je přítomen. Pokud ne, najít místo, kde by měl být a tam ho přidat.
- Najdeme tedy vrchol výstupního stupně nejvýš 1 a přidáme mu toho správného syna.

Jak vyhodit prvek?

- Zjistíme, zda prvek má nejvýš jednoho syna. Pokud ano, zruš jeho vrchol (a je hotovo).
- Pokud ne, najdi nejlevější prvek v pravém podstromě (nebo nejpravější prvek v levém podstromě, to je jedno).
- Prohoď klíče těchto dvou prvků (porušíme vlastnost být binárním vyhledávacím stromem).
- Tím má vyhazovaný prvek výstupní stupeň nejvýš 1 $\Rightarrow$ vyhoď ho přímo.

Stromy

- Nevýhoda BVS je, že se těžko udržuje vyvážený.
- Různé modifikace udržující rozumný poměr vyváženosti.
- AVL-stromy používají pojem balance reflektující hloubku podstromů.
- Červeno-černé stromy obarvují vrcholy červeně resp. černě. Červené vrcholy nesmějí jít dva za sebou a černých musí být na všech cestách z kořene do listu stejně.
- Oba stromy spoléhají na "rotaci", tedy prohazování syna a otce (a rekonstrukce s tím související).
- A-B stromy udržují všechny listy na stejné úrovni, v každém vrcholu (mimo kořen) udržují aspoň A prvků a nejvýše B a v případě potřeby vrcholy štěpí resp. slučují.
- O všech lze ukázat (a o některých ukážeme), že mají hloubku logaritmickou vůči počtu prvků.

Konec

Děkuji za pozornost...