

Anotace

- Pointery, dynamické proměnné

K čemu jsou dynamické proměnné?

- K mnoha algoritmům bychom potřebovali pole proměnlivé délky
- nebo aspoň jinou datovou strukturu proměnlivé délky.
- Jak implementovat frontu a zásobník?
- Použijeme pointery.

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 vyplývá 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 lehkomyšlně 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 alokátor, 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:int;`
- `new(a);` – naalokuje 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 alokátoru voláním funkce `dispose`:
- `dispose(a)`;
- Tím prohlásíme, že pod dotyčný pointer už nepotřebujeme.
- Pozor pokud si na totéž místo ukazujeme víckrát, nesmíme provést vícenásobné odalokování!
- Pokud pointer přesměrujeme bez odalokování (ničím si na dotyčné místo neukazujeme), vzniká tzv. garbage, o které má alokátor za to, že je používána, my však už nemáme, jak se k dotyčnému kusu paměti dostat (tak ho nemůžeme ani odalokovat).
- Některé jazyky (Java, C#) takto odalokovávají (přestaneme si na kus paměti ukazovat). Říká se tomu garbage collector, je to nevyhovné 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.