Cvičenie 7

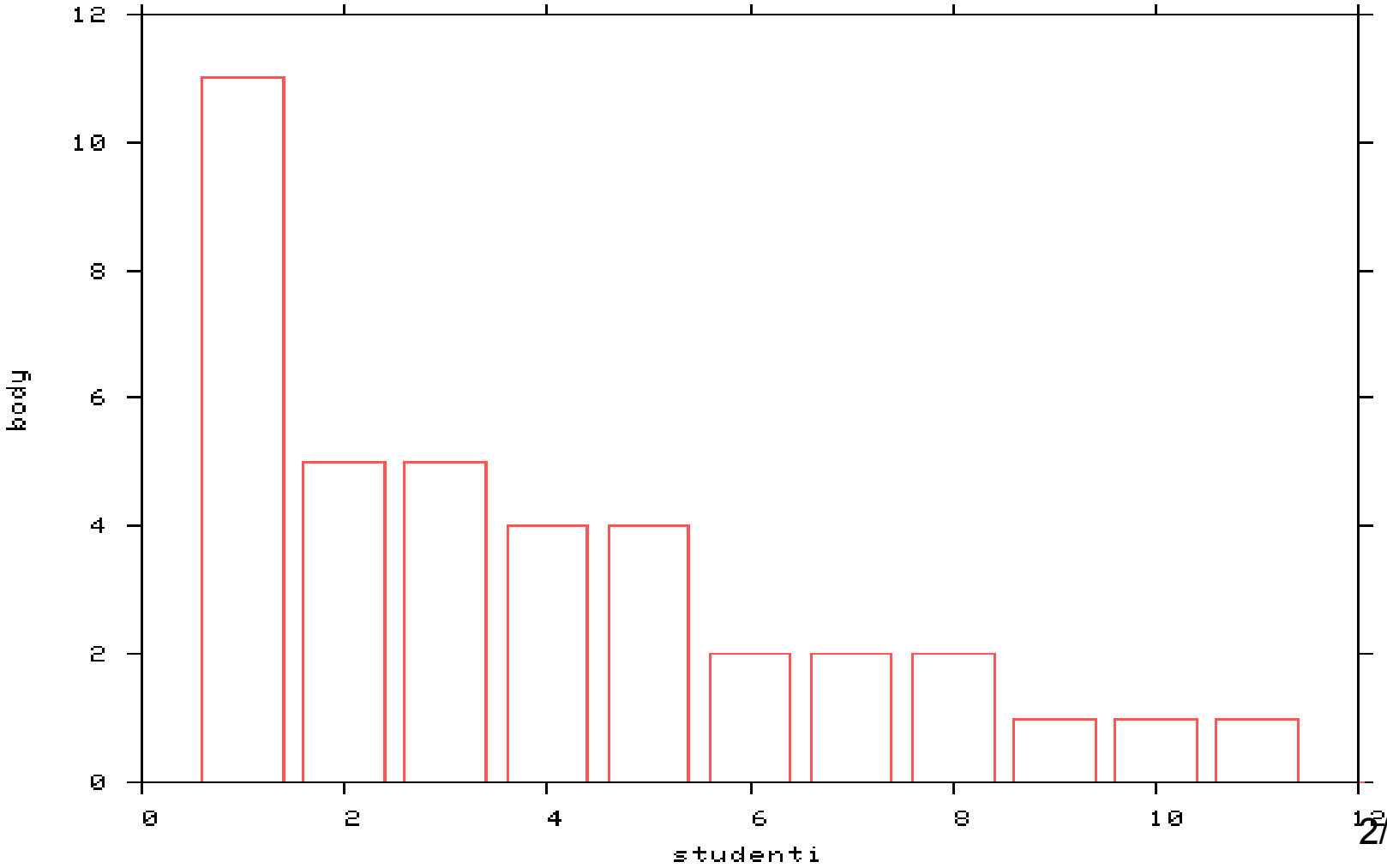
Miroslav Cupák

Obsah

- organizačné záležitosti
  - zhrnutie úloh
  - štatistika
- úloha 6
- úloha 7
- OOP – rekapitulácia
- triedy
- dynamická alokácia pamäte

Organizačné záležitosti

- ešte neopravené úlohy
- “aktuálna” štatistika



Organizačné záležitosti

- zhrnutie úloh
  - úloha 3 – hádanie
  - úloha 4 – pamatovak
  - úloha 5 – life
    - deadline dnes

Úloha 6 - Union-Find

- zadaná minulý týždeň, deadline o týždeň
- bonus +2 body za efektivitu
- implementácia dátovej štruktúry pre reprezentáciu disjunktných množín v grafových algoritmoch

- predpísané metódy, štruktúra triedy, možnosť spustenia Kruskalovho algoritmu (netreba implementovať), konkrétny spôsob ukladania dát na vás

http://www.fi.muni.cz/usr/jkucera/pb161/union_find.htm

Úloha 7

- nie je :)
- možnosť vymyslieť a vypracovať vlastnú úlohu
  - nepovinne
  - až za +6 bodov
  - žiadne záporné body
- hodnotená nielen implementácia, ale aj originalita, formulácia zadania, kvalita testovacích dát a vzorového riešenia, použiteľnosť v predmete...

- podmienka: zmysluplné využitie dedičnosti tried

OOP - rekapitulácia

- čo je to trieda?
  - analógia štruktúry
  - atributy + metody
  - vzor, “šablona”
- dedičnosť tried
  - ISA
  - pozor na viacnásobnú dedičnosť!
  - prekrývanie položiek
  - zápis
    - class třída:specifikátory předek,
      specifikátory předek { ... } ... ;
- prístupové práva

Metódy tried

- normálne
- konštantné
- statické
- spriatelené
- virtuálne

Normálne metódy

- prístup - trieda::identifikátor
- deklarácia
  - prototyp v triede, definícia inde (s kvalifikáciou)
  - celá deklarácia v triede (inline)
- this
  - príklad – lokálna premenná v metóde a premenná v triede

Konštantné metódy (const)

- normálne metódy menia atributy – nemôžme ich volať na konštantných objektoch
- môžme volať na konštantných objektoch
- deklarácia toho, že metóda nemení nestatické atributy

```
void konstantna_metoda(void) const;
```

- s tým súvisí modifikátor *mutable*
  - nestatická položka môže meniť hodnotu, aj keď je objekt konštantný
  - využitie?
    - atribúty neprispievajúce k stavu objektu
    - cache

Konštantné metódy (const)

- demo07_1.cc

- pozn. o modifikátore *volatile*:
  - metódy nestálych objektov (môže meniť niekto iný)
  - môžme použiť naraz *const* a *volatile?*

Konštantné metódy (const)

- demo07_1.cc

- pozn. o modifikátore *volatile*:
  - metódy nestálych objektov (môže meniť niekto iný)
  - môžme použiť naraz *const* a *volatile?*
    - ÁNO
    - dostaneme metódy pre prácu s konštantnými, ale nestálymi objektami
    - náš program meniť nemôže, ale niekto iný áno (napr. OS)

Statické metódy (static)

- podobne ako statické atributy nepatria inštancii, ale typu (spoločné pre inštancie)
- operujú len so statickými atribútmi
- nemajú *this*
- kvalifikujeme menom triedy (zriedka inštancie)

Spriatelené metódy (friend)

- bežne majú externé entity len prístup k *public* položkám
- spriatelené entity prístup k *private* položkám
- typicky používané v súvislosti s operátormi

"C++ : Where friends have access to your private members." — Gavin Russell Baker.

Virtuálne metódy (virtual)

- umožňujú nám prepnutie defaultnej “časnej” väzby (v dobre prekladu) na “pozdní” väzbu (v dobe behu, pomalé)

Virtuálne metódy (virtual)

```
obrazec o[4];
bod b(...);          // potomok triedy obrazec
kruznica k(...);     // potomok triedy obrazec
o[1]=k;

b.Vykresli(); // použije sa bod::Vykresli()
k.Vykresli(); // použije sa kruznica::Vykresli()
o[1].Vykresli(); // použije sa ???
```

Virtuálne metódy (virtual)

- čiste virtuálna metódy

```
virtual void metoda() = 0;
```

- princíp abstraktnej triedy
  - nemôžme robiť inštancie
- vynucuje implementáciu v potomkovi

- pozn.: kľúčové slovo *virtual* sa používa aj pri dedičnosti v trochu inom význame
  - potomok neobsahuje celého predka, len odkaz
  - vyhneme sa tým nejednoznačnosti pri viacnásobnom dedení

Operátory

- deklarované podobne ako obyčajné metódy, len s kľúčovým slovom *operator*
- môžme len preťažovať už existujúce operátory
  - okrem *?: :: . .* sizeof typeid ..._cast*
- rôzna arita

Konštruktory a deštruktory

- špeciálne metódy volané pri vytváraní (zániku) inštancie
- omedzenia
  - bez návratovej hodnoty
  - meno zhodné s menom triedy (~)
  - nededia sa, ale sú volané z predkov
    - konštruktory v poradí od predkov k potomkom, deštruktory naopak
  - ak nedefinujeme, urobí to prekladač
  - typicky používané na získanie (uvoľnenie) zdrojov
  - nemôžu byť statické

Konštruktory

- časté použitie konštruktorov – inicializácia atribútov triedy (inštancie) podľa argumentov
- na inicializáciu vhodné použiť inicializačnú časť (prevencia pred dvojitou inicializáciou)

```
NazovTriedy() : inicializátory {}
```

```
class pole
{ int *p, delka;
public:
  pole(/* tu môžu byť parametre */):p(0),delka(0){}
...}
```

Konštruktory

- špeciálny význam má tzv. kopírovací konštruktor
  - parametrický
  - inicializácia z už existujúcej inštancie
  - typicky robí kópiu objektu, ale nemusí

- demo na virtuálne dedenie a poradie volania konštruktorov a deštruktorov
  - demo07_2.cc

Metódy tried - príklad

```
class kniha
{ private:
    string autor;
    string nazev;
    int rok_vydani;
  public:                    // veřejně přístupné:
    kniha();                 // prázdný konstruktor
    kniha(string aut, string naz, int rok);          // inic. konstruktor
    kniha(const kniha& vzor); // kopír. konstruktor
    ~kniha();                // destruktor
    void Vypis(ostream& vystup) const; // metoda
  private: // následující metodu mohou používat
           // jen jiné metody této třídy
    void Zaznam(string aut, string naz, int rok);
                           // spřátelené operátory:
    friend istream& operator>>(istream& vstup,
                           kniha& k);
    friend ostream& operator<<(ostream& vystup,
                           kniha k);
}/* zde mohou následovat přímo deklarace instancí */;
```

Dynamická alokácia pamäte

- podľa C
  - malloc (calloc) – alokuje danú časť pamäte
  - realloc – zmena veľkosti alokovanej pamäte
  - free – uvoľnenie pamäte

  - nepracujú s objektami (konštruktory, deštruktory), ale inak môžme používať aj v C++
  - <cstdlib>

Dynamická alokácia pamäte

- C++
  - new – alokuje potrebnú pamäť pre objekt
  - new[] - alokuje potrebnú pamäť pre jednorozmerné pole
  - delete, delete[] - uvoľnenie pamäte
    - ale nemení obsah (bezpečnosť)
  - variantu pre *realloc* nemáme (ale máme kontajnery!)
  - pracujú s objektami (konštruktory, deštruktory)

Dynamická alokácia pamäte

- podstatné je nemiešať prístup C a C++!
  - pokiaľ alokujeme cez malloc, uvoľníme cez free
  - pokiaľ alokujeme cez new, uvoľníme cez delete

Dotazy?