Cvičenie 8

Miroslav Cupák

Obsah

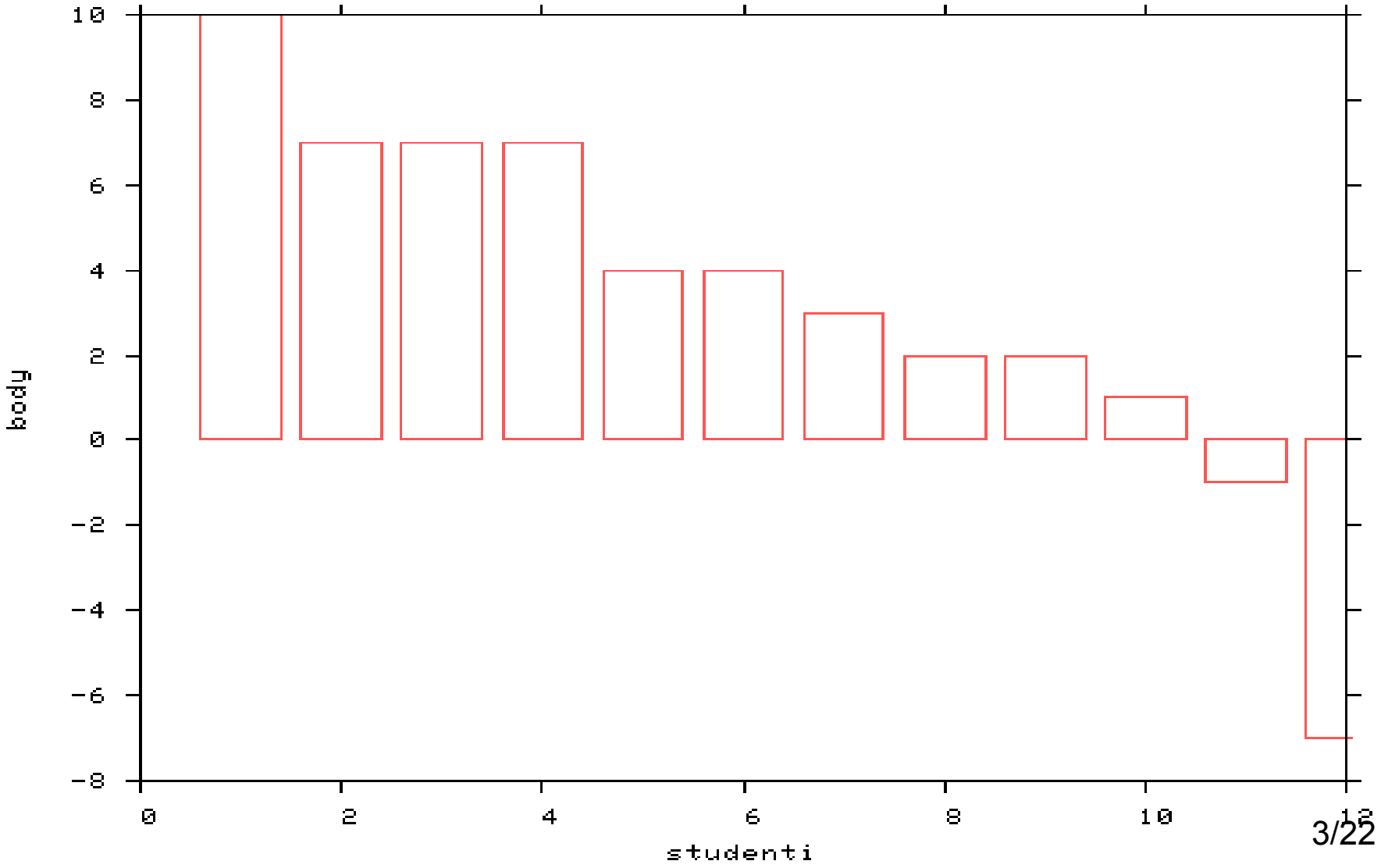
- organizačné záležitosti
- štruktúra C++ programu
- pretypovanie
- namespaces
- úloha 8

Organizačné záležitosti

- úloha 5 (life) uzavretá
  - bodovanie, bonus
- úloha 6 (union-find) do polnoci
  - opravovanie, bonus
- úloha 7 – dobrovoľné vytvorenie vlastnej úlohy (až +6 bodov)

Organizačné záležitosti

- aktuálna štatistika



Štruktúra C++ programu

- typicky [názov]main.cc, [názov].cc, [názov].h
- hlavičkový súbor – ochrana pred viacnásobným include

```
#ifndef _NAZEV_
#define _NAZEV_

#include …
...
// definície tried, prototypy, inline funkcie
...
#endif
```

Pretypovanie

- 3 základné spôsoby (+nejaké ďalšie)
  - ako v C

    (typ)entita
  - pretypovací konštruktor

    typ(entita)
  - C++ operátory

    [static|const|dynamic|reinterpret]_cast<typ>(entita)

static_cast<typ>(entita)

- vhodné na väčšinu prípadov - “klasické” pretypovanie

- demo08_1.cc

const_cast<typ>(entita)

- len na pridávanie a odoberanie *const* a *volatile*
- typ sa musí od entity líšiť len v týchto modifikátoroch!

- demo08_2.cc

reinterpret_cast<typ>(entita)

- prevod nesúvisejúcich typov, najčastejšie snáď ukazateľ → číslo a naopak
- používať veľmi opatrne!
- konverzie môžu byť platformovo závislé a neprenositeľné

- demo08_3.cc

dynamic_cast<typ>(entita)

- typicky pretypovanie ukazateľa na objektový typ na ukazateľ na iný objektový typ v tej istej hierarchii
- analogicky aj odkazy
- vie pretypovať aj na virtuálneho potomka
- v prípade neúspechu vracia 0 (NULL), preto úspech môžem zistiť pomocou *if(pretypovanie)*

- demo08_4.cc

RTTI

- RunTime Type Information
- zistenie typu v dobe behu programu
- pracuje len s triedymi a virtuálnymi metódami
- využíva operátor *typeid([názov triedy])*, ktorý vracia odkaz na objekt *type_info*
- <typeinfo>
- preťažené operátory == a !=

- demo08_5.cc

Iný pohľad na konverzie

- implicitné (automatické)

```
double a;
int b=10;
a=b;
```

- explicitné

```
double a=3.14;
int b;
b=int(a);   // oreže desatinnú časť
```

Konverzný konštruktor

- konštruktor s 1 parametrom prekladač automaticky používa na konverziu (parametru na našu triedu)
- ak chceme zabrániť automatickej konverzii, deklarujeme konštruktor ako explicitný (*explicit*)

Konverzné funkcie

- slúžia na automatické pretypovanie našej triedy na niečo iné
- aj viacnásobná konverzia
- deklarované *operator typ_T(void);*
- musia byť metódou našej triedy
- nemajú parametre
- nemajú návratový typ, ale vracajú hodnotu daného typu!

- demo08_6.cc

Namespaces

- priestory mien
- používané v knižniciach a veľkých projektoch
- umožňujú správu veľkého množstva identifikátorov
- umožňujú rozdelenie identifikátorov do skupín, a teda používať bezkonfliktne identifikátory s rovnakými názvami

Namespaces

- prístup – kvalifikáciou pomocou *::*
- stretli ste sa už s *namespace std;*
- môžme deklarovať vlastné namespaces

  namespace nazov ns {deklaracia}
- deklarácia nie vo funkcii, len globálne!

- demo08_7.cc

Namespaces - zanáranie

- menné priestory môžem zanárať (podpriestory), kvalifikovaným menom identifikátoru je potom

  `::ns1:: … ::ns?::identifikátor`

  – prekrývanie

- demo08_8.cc

Namespaces – postupná deklarácia

- môžem deklarovať postupne (aj vrámci rôznych súborov)

- demo_08_9.cc

Anonymné namespaces, prístup

- nemusia mať meno → anonymný namespace
  - môže byť len 1 vrámci súboru
- sprístupnenie identifikátorov
  - spomínaná kvalifikácia
  - vo vnútri daného namespace bez kvalifikácie
  - pomocou direktivy *using*, následne bez kvalifikácie
    - nemusím celé NS, pomocou :: stačí len niektoré identifikátory
    - tranzitívna (sprístupní všetky NS sprístupnené v danom NS)

Namespaces a aliasy

- kvôli zanáraniu môžu vznikať dlhé mená, hodia sa aliasy
  - riešené obalením do namespace
  - bez aliasu:

```
a::b::c::vypis();
```

  - s aliasom:

```
namespace d = a::b::c;
d::vypis();
```

Namespaces – ďalšie info

- triedy a štruktúry tvoria vlastný NS
- operátory sú hľadané tam, kde sú deklarované operandy, neskôr tam, kde je deklarovaná funkcia, v ktorej ich použijeme

Úloha 8 - Agenti

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

- správanie entít (ľudia, boh, agenti) vo svete
- simuláciu sveta máte dostupnú, implementujete len samotné správanie a interakciu entít (všetky požadované metódy)
- zamerené na dedičnosť, princípy OOP, viacnásobnú dedičnosť, prístupové práva, spriatelené funkcie
- pozor na splnenie všetkých požiadavkov!

Dotazy?