Cvičenie 11

Miroslav Cupák

Obsah

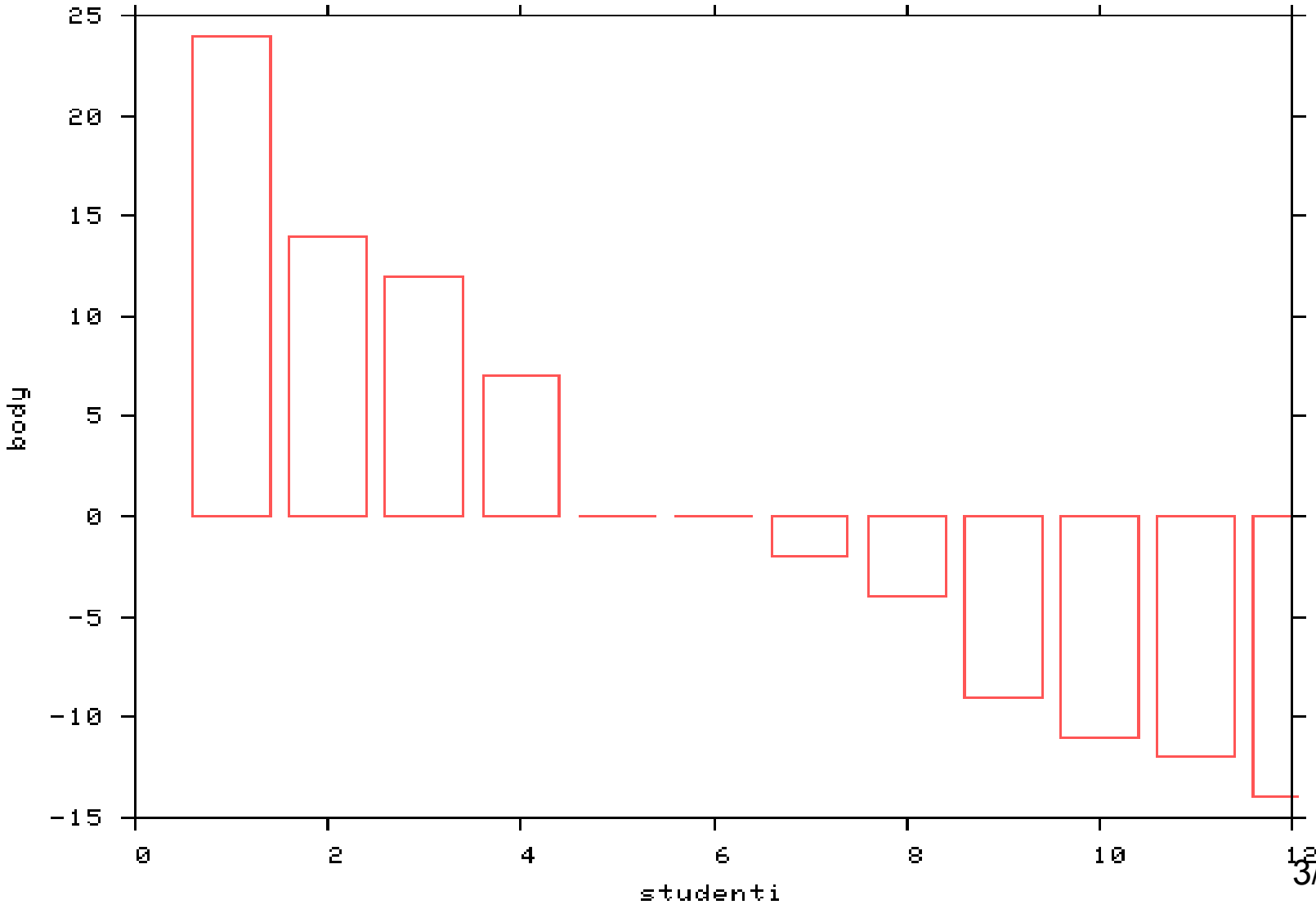
- organizačné záležitosti
- výnimky
- úloha 11

Organizačné záležitosti

- úloha 8 (agenti) opravená
  - pri hodnotení dôraz na objektový návrh
- úloha 9 (120) – deadline dnes
- úloha 10 (textové ovládacie prvky) – deadline o týždeň
  - nenechávať na poslednú chvíľu

Organizačné záležitosti

- aktuálna štatistika



Výnimky

- testovať je nutné všetko
  - vstup (fuzz testing, quick testing...)
  - návratové hodnoty funkcií
  - argumenty funkcií a operátorov (delenie 0)
- snaha vyhnúť sa chybám nie je vždy úspešná
- každý program má chyby → chce to prípravu
  - výnimky!

Výnimky

- cieľ – vysporiadať sa s chybovými stavmi bez rozsiahleho testovania všetkých možností
- hlavne na zachytenie chýb, ktoré nespôsobil náš program
- umožňujú ošetriť chybu niekde “vyššie”
- fungujú na príncípe vyhodenia (*throw*) “udalosti” - výnimky - a ich chytania (*catch*) - handler

Výnimky

- schéma použitia

```
try {
        // príkazy
        throw výraz; // vyvolanie výnimky
}
catch(typ_výnimky deklarátor) // handler 1
{ /* ošetrenie výnimky */ }
catch(...) // handler 2 (aj viac), chytá všetko
{ /* ošetrenie výnimky */ }
```

Výnimky

- ak nastane výnimka
  - to, čo je za *throw* sa v prípade ignoruje
  - ignorujú sa aj *catch* bloky, ktoré nie sú potrebné
  - po throw sa skočí na 1. vhodný catch blok
  - ak sa vhodný handler nenájde, prebubláva vyššie
  - pozn.: ak nepotrebujem v handleri hodnotu výnimky, netreba za catch uvádzať identifikátor
- ak nenastane výnimka
  - príkazy *try* bloku sa dokončia
  - handlery sa preskočia
  - pokračuje sa kódom za handlermi

Výnimky

- organizované v triednej hierarchii
- výnimka je ošetrená 1. handlerom, ktorý je pre daný typ
  - pozor na uvedenie všeobecnejšieho handleru najprv!
- ak žiadny handler nezachytí, výnimka prebubláva do nadradeného bloku (funkce)
- ak výnimka prebublá do main a nie je zachytená, *terminate*
- výnimku môžem poslať vyššie aj z handleru
  - throw bez operandu

Výnimky

```
try { throw E; }
catch(H){ /* ošetrenie */}
```

- kedy sa vstúpi do bloku catch?
  - H je rovnakého typu ako E
  - H je jednoznačným prístupným predkom E
  - H, E sú ukazatele na typ splňujúci podmienku 1/2
  - H je odkaz na typ splňujúci podmienku 1/2

Výnimky

- výnimka sa považuje za ošetrenú, ak vstúpime do handleru
- pozn.: handler má vždy max. 1 parameter bez implicitnej hodnoty
- v C++ len podpora synchrónnych výnimiek
  - delenie 0 - áno
  - stlačenie CTRL+C – nie
- hádzať a chytať môžem takmer čokoľvek
  - demo11_1.cc

Výnimky

- môžme si definovať vlastné výnimky, ale v STL už nejaké sú (môžem od nich odvodiť nové)
- sú to objekty odvodené od zákl. triedy *exception* (<stdexcept>)
  - konštruktory, deštruktory, preťažené =, *what()*
- viac na
  http://www.fi.muni.cz/usr/jkucera/pb161/sl10.htm

Výnimky

- hierarchia štandardných výnimiek
- exception
  - logic_error
    - length_error
    - domain_error
    - out_of_range
    - invalid_argument
  - bad_alloc
  - bad_exception
  - bad_cast
  - bad_typeid
  - runtime_error
    - range_error
    - overflow_error
    - underflow_error

Výnimky

- ak môže výnimka uniknúť z funkcie, spomenieme to v prototype

  ```
  typ názov(parametre) throw (zoznam typov);
  ```

- pozn. o throw v prototype
  - neuvedenie znamená, že sa môže šíriť ľubovoľná výnimka
  - throw() znamená, že sa nemôže rozšíriť žiadna výnimka

Výnimky

- komplexné demo na štandardné výnimky, únik výnimky z funkcie, použitie viacerých handlerov a what()
  - demo11_2.cc // delenie bez výnimiek
  - demo11_3.cc // delenie s výnimkami

Výnimky v konštruktoroch/deštruktoroch

- konštruktory
  - výnimka sa z neho šíriť môže, ale nie vhodné
    - v prípade globálneho objektu ju nikto nezachytí a program končí
  - ak voláme konštruktor, z ktorého sa šíri výnimka v inicializačnej časti konštruktoru inej triedy, problém
    - výnimka sa vyvolá pred vstupom do tela volajúceho konštruktoru → nemôžem ju ošetriť

Výnimky v konštruktoroch/deštruktoroch

- deštruktory
  - pri prebublávaní výnimky dochádza k volaniu deštruktorov objektov s pamäťovou triedou auto
  - v C++ nemôžem mať viac výnimiek na rovnakej úrovni → nesmiem vyvolať výnimku počas hľadania vhodného handleru inej výnimky
    - deštruktory teda písať tak, aby sa z nich výnimky nešírili

Výnimky v handleroch, vlastné výnimky

- výnimky v handleroch

```
try {  // blok 1
        try { /* blok 2 */ }
        catch { /* handler na blok 2 */ }
} catch { /* handler na blok 1 */ }
```

- vlastné výnimky
  - demo11_4.cc

Výnimky - záver

- výhody
  - reakcia na miestach, kde sme chybu nečakali
  - nemusíme všetko testovať
  - čistejšie
  - rýchlejšie
- nevýhody
  - pomerne zložitý aparát
  - náročné na zdroje
- rady
  - výnimky používať, ale nie za každú cenu
  - výnimky vždy chytať

Úloha 11 – Bezpečné dynamické pole

- implementácia dynamického poľa z podpornými vlastnosťami

- nepovinná úloha
- rozsiahla a náročná, ale za veľa bodov
- rozsah a bodové hodnotenie si určujete sami
- využíva šablony, budeme spomínať nabudúce
- zadanie:

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

Úloha 11 – Bezpečné dynamické pole

- bodovanie
  - žiadne body za odovzdanie do nedele
  - +2 za data wrapper, pole so zákl. operáciami
  - +2 za ochrany (rozsah indexov, kopírovanie prázdneho poľa)
  - +2 za zmysluplnú konverziu medzi poľami s rôznymi ochranami
  - +2 optimálne shallow copy
  - +3 podpora dátových typov bez bezparam. konšt.

Dotazy?