

C/C++ III. Pokročilý

část 2.

IT Academy, 2014

Obsah

1. Priatelia, výnimky
 1. Priatelia
 2. Výnimky
2. Trieda string a štandardná knižnica šablón
 1. Trieda string
3. Triedy a dynamické pridelovanie pamäti

1. Priatel'ia, výnimky

1. Priatel'ia

1. Priatel'ské triedy
2. Priatel'ské členské funkcie
3. Iné priatel'ské vzťahy

2. Výnimky

1. Volanie abort()
2. Vrátenie chybového kódu
3. Mechanizmus výnimiek
4. Objekty ako výnimky
5. Uvoľnenie zásobníku
6. Ďalšie možnosti
7. Trieda exception
8. Výnimky, triedy a dedičnosť
9. Keď výnimky blúdžia
10. Opatrenie pri používaní výnimiek

1. Priatel'ia, výnimky

Odporúčaná literatúra

Začiatok témy - strana 637

Zhrnutie - strana 696

Opakovacie otázky - strana 697

Programovacie cvičenia - strana 698

1.Priatel'ia

Priatel'ská funkcia triedy v C++

- je definovaná mimo scope triedy
- má právo pristupovať ku všetkým súkromným a protected členom triedy
- prototyp priatel'skej funkcie sa síce nachádza v definícii triedy, ale nie je funkciou inštancie
- priateľom môže byť:
 - funkcia a šablóna funkcie
 - funkcia inštancie triedy, trieda, alebo šablóna triedy

1. Priatel'ia

Pre definovanie funkcie ako priateľa triedy:

```
class Box
{
    double width;
public:
    double length;
    friend void printWidth( Box box );
    void setWidth( double wid );
};
```

pre definíciu priateľskej triedy pridáme do definície triedy:

```
friend class ClassTwo;
```

Príklad Friends

1.Priatel'ia

Odporúčania pre priateľské funkcie a triedy:

- pretože môžu porušiť princíp zapúzdrenia, mali by sme ich použitie minimalizovať tak, ako je to možné
- vzťah “je priateľ” nefunguje naopak - ak je A friend s B, potom B nemusí byť priateľom A, pokiaľ sme to tak nedefinovali
- bežný prípad použitia - testovanie
- využiteľné najmä pri veľkých codebases
- využívajú sa pri niektorých Design Patternoch - http://sourcemaking.com/design_patterns/iterator/cpp/1
- aj šablóny môžu mať priateľov - strana 646

1. Priatel'ia

“Here are a couple of guidelines I heard about C++ friends. The last one is particularly memorable.

- Your friends are not your child's friends.
- Your child's friends are not your friends.
- Only friends can touch your private parts.”

Dedia sa priatel'ské vzt'ahy?
Zadefinujme si obojstranné priatel'stvo

1. Výnimky

abort()

- funkcia definovaná v `<cstdlib>` alebo `<stdlib.h>`
- ak je zavolaná, ukončí program so správou **abnormal program termination** do chybového výstupu (presne ako **cerr**)
- vyprázdnenie pamäte je závislé na implementácii
- pamäť korektne vyprázdni funkcia **exit**, ale tá neodošle chybovú správu na chybový výstup
- implicitne ho vyvolávajú nezachytené výnimky

1. Výnimky

Vrátenie chybového kódu/chybová premenná

```
bool hmean(double a, double b, double * ans)
{
    if (a == -b)
    {
        *ans = DBL_MAX;
        return false;
    }
    else
    {
        *ans = 2.0 * a * b / (a + b);
        return true;
    }
}
```

1. Výnimky

- výnimky predstavujú spôsob, ako predať časť riadenia programu inej časti programu (napr v prípade delenia nulou)
- priebeh ošetrovania výnimky sa skladá z troch častí:
 - vyvolanie výnimky - **throw**
 - zachytenie výnimky pomocou handleru - **catch**
 - použitie pokusného bloku **try**
- výnimky vrátené z funkcií - error5.cpp, strana 666

1. Výnimky

Ako to vyzerá všeobecne?

```
1 // exceptions
2 #include <iostream>
3 using namespace std;
4
5 int main () {
6     try
7     {
8         throw 20;
9     }
10    catch (int e)
11    {
12        cout << "An exception occurred. Exception Nr. " << e << '\n';
13    }
14    return 0;
15 }
```

An exception occurred. Exception Nr. 20

1. Výnimky

Viac možností odchytenia výnimky v závislosti na jej type:

```
1 try {  
2     // code here  
3 }  
4 catch (int param) { cout << "int exception"; }  
5 catch (char param) { cout << "char exception"; }  
6 catch (...) { cout << "default exception"; }
```

1. Výnimky

Viac možností odchytenia výnimky v závislosti na jej type:

```
1 try {  
2     // code here  
3 }  
4 catch (int param) { cout << "int exception"; }  
5 catch (char param) { cout << "char exception"; }  
6 catch (...) { cout << "default exception"; }
```

1. Výnimky

Try/catch bloky je možné do sebe zanořovat:

```
1 try {  
2     try {  
3         // code here  
4     }  
5     catch (int n) {  
6         throw;  
7     }  
8 }  
9 catch (...) {  
10     cout << "Exception occurred";  
11 }
```

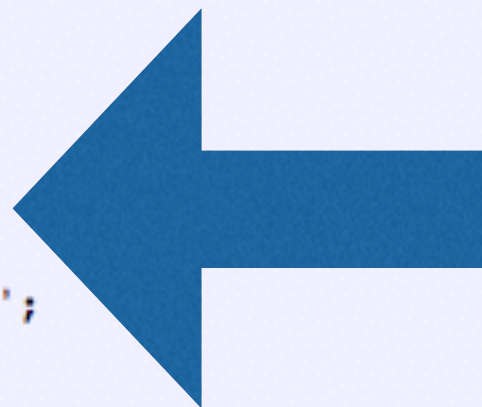

1. Výnimky

Štandardné výnimky:

<exception>

```
1 // using standard exceptions
2 #include <iostream>
3 #include <exception>
4 using namespace std;
5
6 class myexception: public exception
7 {
8     virtual const char* what() const throw()
9     {
10         return "My exception happened";
11     }
12 } myex;
13
14 int main () {
15     try
16     {
17         throw myex;
18     }
19     catch (exception& e)
20     {
21         cout << e.what() << '\n';
22     }
23     return 0;
24 }
```

My exception happened.



1. Výnimky

Niektoré štandardné C++ výnimky:

exception	description
<code>bad_alloc</code>	thrown by <code>new</code> on allocation failure
<code>bad_cast</code>	thrown by <code>dynamic_cast</code> when it fails in a dynamic cast
<code>bad_exception</code>	thrown by certain dynamic exception specifiers
<code>bad_typeid</code>	thrown by <code>typeid</code>
<code>bad_function_call</code>	thrown by empty function objects
<code>bad_weak_ptr</code>	thrown by <code>shared_ptr</code> when passed a bad <code>weak_ptr</code>

Also deriving from `exception`, header `<exception>` defines two generic exception types that can be inherited by custom exceptions to report errors:

exception	description
<code>logic_error</code>	error related to the internal logic of the program
<code>runtime_error</code>	error detected during runtime

1. Výnimky

Deprecated:

```
double myfunction (char param) throw (int);
```

Ak funkcia definuje aj typ výnimky, ktorú vyvoláva, a následne vyvolá výnimku iného typu, vedie to k zavolaniu funkcie **abort()**

1. Výnimky

Vlastná obsluha nezachytených výnimiek:

```
1  #include <iostream>
2  #include <exception>
3
4  using namespace std;
5
6  void myQuit() {
7      cout << "nezachytena vynimka, koncim";
8      exit(5);
9  }
10
11 int main() {
12     // tu mi len zdochne program
13     throw;
14
15     set_terminate(myQuit);
16     |
17     // throw; // tu zavolam myQuit()
18
19     return 0;
20 }
```

1. Výnimky

Vlastná obsluha neznámych výnimiek:

```
1  #include <iostream>
2  #include <exception>
3
4  using namespace std;
5
6  void myUnexpected() {
7      cout << "nezachytena vynimka, koncim";
8      exit(5);
9  }
10
11 int main() {
12     // tu mi len zdochne program
13     // throw "dezo";
14
15     set_unexpected(myUnexpected);
16
17     // throw; // tu zavolam myQuit()
18     // throw "dezo";
19
20     return 0;
21 }
```

2. Trieda string a štandardná knižnica šablón

Trieda string

1. Vytvorenie ret'azca
2. Vstup triedy string
3. Práca s ret'azcami
4. Čo môže trieda string ešte ponúknuť

2. Trieda string a štandardná knižnica šablón

Odporúčaná literatúra

Začiatok témy - strana 699

Zhrnutie - strana 770

Opakovacie otázky - strana 772

Programovacie cvičenia - strana 773

3. Triedy a dynamické pridelovanie pamäti

1. Dynamická pamäť a triedy
2. Príklad na zopakovanie a statické položky tried
3. Implicitné členské funkcie
4. Kedy sa v konštruktoch používa operátor new
5. Používanie ukazovateľov na objekty

3. Triedy a dynamické pridelovanie pamäti

Odporúčaná literatúra

Začiatok témy - strana 451

Zhrnutie - strana 504

Opakovacie otázky - strana 505

Programovacie cvičenia - strana 507

3. Triedy a dynamické pridelovanie pamäte

Typická situácia - potrebujem ako položku triedy zadať priezvisko. Môžem použiť pole o 14 alebo 40 znakov. Ak budem mať inštancie 2, plytvanie pamäťou bude zanedbateľné. Ak budem mať inšancií 2000, už sa môže jednať o problém.

Bude lepšie rozhodovať o množstve alokovanej pamäte až počas behu programu - C++ má pre tento prípad operátory **new** a **delete**.

V C++ sa neodporúča použitie malloc()

3. Triedy a dynamické pridelovanie pamäte

Statické položky tried majú iba jednu hodnotu pre celú triedu.

Statické členské funkcie:

- ak je deklarovaná vo verejnej časti, je možné ju zavolať pomocou názvu triedy
- môže používať iba statické členské premenné (tie, ktoré patria triede, nie objektu)
- vlastne ani nemá pointer **this**

3. Triedy a dynamické pridelovanie pamäte

Implicitné členské funkcie sa automaticky zadefinujú, pokiaľ ich nepredefinujeme manuálne:

- konštruktor
- kopírovací konštruktor
- operátor priradenia
- operátor získania adresy (vracia hodnotu this)

3. Triedy a dynamické pridelovanie pamäte

Implicitný konštruktor

- je prázdny
`Test::Test() {}`
- zavolá sa pri vytvorení objektu
Test test;
- môžeme ním nastavovať default hodnoty
- môže preberať parametre, ale musia mať zadefinované implicitné hodnoty
- môže byť iba jeden

3. Triedy a dynamické pridelovanie pamäte

Kopírovací konštruktor

- jeho prototyp je:
`Class_name(const Class_name &)`
- používa sa pre inicializáciu objektu pomocou iného objektu
- ak nie je zadefinovaný, tak implicitný kop. konštruktor začne kopírovať nestatické členy inštancie rad za radom
- ak rátame inštancie triedy pomocou statickej premennej, mali by sme ho zadefinovať

3. Triedy a dynamické pridelovanie pamäte

Kopírovací konštruktor

```
String ditto(motto);           // volá String(const String &)  
String metoo = motto;          // volá String(const String &)  
String also = String(motto);    // volá String(const String &)  
String * pstring = new String(motto); // volá String(const String &)
```

3. Triedy a dynamické pridelovanie pamäte

Implicitný operátor priradenia

- podobne ako kopírovací konštruktor kopíruje položky jednu po druhej

```
string metoo = ditto; // použije se kopírovací konstruktor
```

```
string motto("Home Sweet Home");  
string ditto;  
ditto = motto; // použije přetížený operátor přiřazení
```

3. Triedy a dynamické pridelovanie pamäte

Kedy sa v konštruktoze používa operátor **new**

- keď nezabudneme v deštruktoze deklarovať **delete**
- **volania new a delete musia byť kompatibilné (ak new [], tak delete [])**
- všetky konšuktory musia byť definované kompatibilne s deštruktorom

Pointre na objekty >>

- ◆ Deklarujete ukazatel na objekt obvyklou notací:

```
String * glamour;
```

- ◆ Ukazatel můžete inicializovat pomocí již existujícího objektu:

```
String * first = &sayings[0];
```

- ◆ Ukazatel můžete inicializovat pomocí operátoru new. Tímto způsobem se vytvoří nový objekt:

```
String * favorite = new String(sayings[choice]);
```

- ◆ Použití operátoru new společně s třídou vyvolá odpovídající konstruktor třídy a inicializuje nově vytvořený objekt:

```
// vyvolá implicitní konstruktor
```

```
String * gleep = new String;
```

```
// vyvolá konstruktor String(const char *)
```

```
String * glop = new String("my my my");
```

```
// vyvolá konstruktor String(const String &)
```

```
String * favorite = new String(sayings[choice]);
```

- ◆ Pomocí operátoru -> získáte přístup k metodě třídy přes ukazatel:

```
if (sayings[i].length() < shortest->length())
```

- ◆ Pomocí operátoru dereference (*) u ukazatele na objekt získáte hodnotu objektu:

```
if (sayings[i] < *first) // porovná hodnoty objektů
```

```
    first = &sayings[i]; // přiřadí adresu objektu
```