Тема 8

Tuesday, June 25, 2024 11:58 AM

Статични данни

Тема 8

8	Статични член-данни. Изключения. Обработка на изключения. Йерархия на изключенията и примери. Изключения в конструктори и деструктори. Нива на exception safety. Пример с клас, който брои инстанциите си.
9	Масиви от указатели към обекти. Move семантики - попзи. lvalue. rvalue. move конструктор/оп=.

- Ключова дума static

Статична ф-я - използване, за да ограничим до една компилираща единица (.obj)

По този начин не може да бъде използвана от други .cpp файлове и можем да създаваме други ф-ии със същото име в други файлове (за целта можем да я енкапсулираме в анонимен namespace)

↳ или изр. оп. ::

Статична променлива в тялото на ф-я

```
f() {
  static A obj;
}
```

- създава се веднъж - при първото извикване на f()
- общ за всички извиквания на f()
- умира при приключване на програмата

Статична член-променлива на клас - глобална променлива, енкапсулирана в клас

- споделя се от всички обекти на класа
- не се пази в класа (не влияе на големината на обекта)

```
class Y {
public:
  static A obj;
};
```

A Y::obj; → namespace за Y

- извиква се дефолтният к-р на A (ако не кажем друго в Y.cpp)
- Y играе ролята на obj

Статична член-функция - външна ф-я, енкапсулирана в клас

- не ни трябва обект, за да я извикаме
- няма смисъл, защото така можем да я направим private
- класа играе ролята на namespace
- не може да е const или virtual

```
class X {
  int x;
```

* Ако в f() създ. ако X, можем да достъпим член-данните

```
class X {
    int x;
    static int y;
public:
    g();
    static f();
};
```

* Ако в f() създ. ако X, можем да достъпим член-данните

статичен клас - има само статични член-данни

Изключения

def: сигнал, че има проблем

- Обработката на изключения (Exception handling) предоставя механизъм за отделяне на логиката за обработване на грешки от тази на останалия код

синтаксис: throw <object>

Обработка на изключения

- поставяме проблемния код в try блока
- обработваме грешката в catch блок (влизаме в този, който приема като параметър хвърления обект)
  catch(...) - хваща всичко

При хвърлена грешка:

1) текущата ф-я спира изпълнението си
2) програмата проверява дали текущата или някоя от извикващите функции нагоре обработва грешката
3) ако намери съответстващ catch блок ⇒ st ck unwinding (изпълнението на програмата прескача до началото на съответния блок за обработка); ако не ⇒ std::terminate

Stack unwinding - премахване на ф-ии от стека, докато не се достигне тази, която може да обработи грешката
(всички успешно създ. преди променливи до момента се унищожават)

Синтаксис

```
void f(int n)
{   if(n<0)
        throw std::invalid_argument("Number should be positive");
}
```

```
int main()
{
  try {
      Y obj;
      f(-10);
      Z obj2;
  }
```

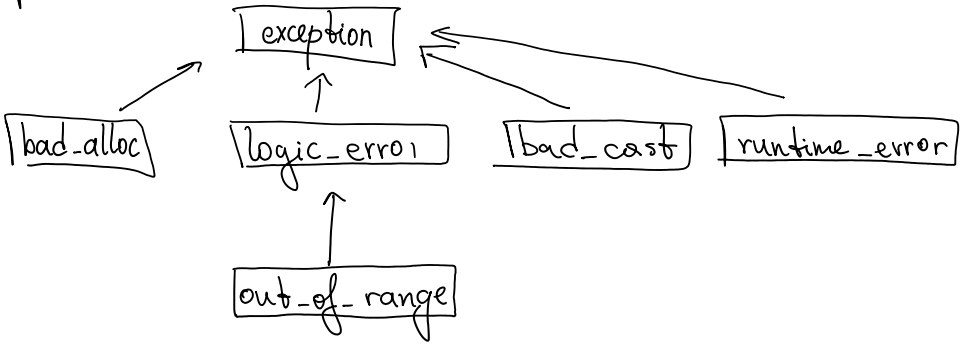
// ще се хване от ~Y()

```
  catch ( const std::invalid_argument & e ) {
    std::cout << e.what() << endl;
  }
  catch ( const std::logic_error & e ) {--}
  catch ( const std::exception & e ) {--}
```

подреждане от най-конкретния към най-общия

приемане по const ref

Йерархия на изключенията



функция .what() - за да разберем къде е станала грешката

```
throw std::exception ( "ABC")

catch ( const std::exception e )
    {
      e.what(); // "ABC"
    }
```

Частни случаи

- при конструктор

При хвърляне на грешка в констр. се извикват деструкторите на всички напълно построени обекти в класа (само прикачените конструктори)

=> не трябва да оставят незатворени важни ресурси

```
A {
  char* str;
```

```
A()
  { str = new char [10];
```

```
A {
  char* str;
  char* str2;
}
```

```
A()
{ str = new char[10];
  try {
    str2 = new char[10];
  }
  catch (const std::bad_alloc& e) {
    delete[] str;
    throw;
  }
```

изтриване на вече заделената памет

- при деструкторите

Ако хвърлена грешка, ако в някой от дестр, които се извикват при stack-unwinding, също се хвърли изключение, то ще бъде викано => std::terminate

⇒ не се пишат изключения в деструкторите

Нива на сигурност

1. No-throw guarantee - касът / ф-ята не хвърля изкл.
2. Strong exc. safety - може да хвърля изкл., но няма да се промени състоянието му
3. Weak / Basic exc. safety - може да хвърля, но ще остане в валидно състояние
4. No exc. safety - няма никакви гаранции