Тема 8

Статични данни

- Ключова дума static

Статична ф-я – използване, за да ограничим до една компилационна 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 ↗ не изп. оп. ::
(за целта можем да я енкапсулираме в анонимен namespace) → можем да слагаме още неща в него

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; → • извиква се деф. к-р тук (ако не кажем друго в Y.cpp)
namespace за Y играе ролята на obj

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

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

```
class X {

    int x;
    static int y;
  public:
    g();
    static f();

}
```

* Ако в f() създаваме 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 catch (...) – хваща всичко като параметър хвърления обект)

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 ⇒ stack unwinding (изпълнението на програмата прескача до началото на съответстващия блок за обработка); ако не ⇒ std::terminate

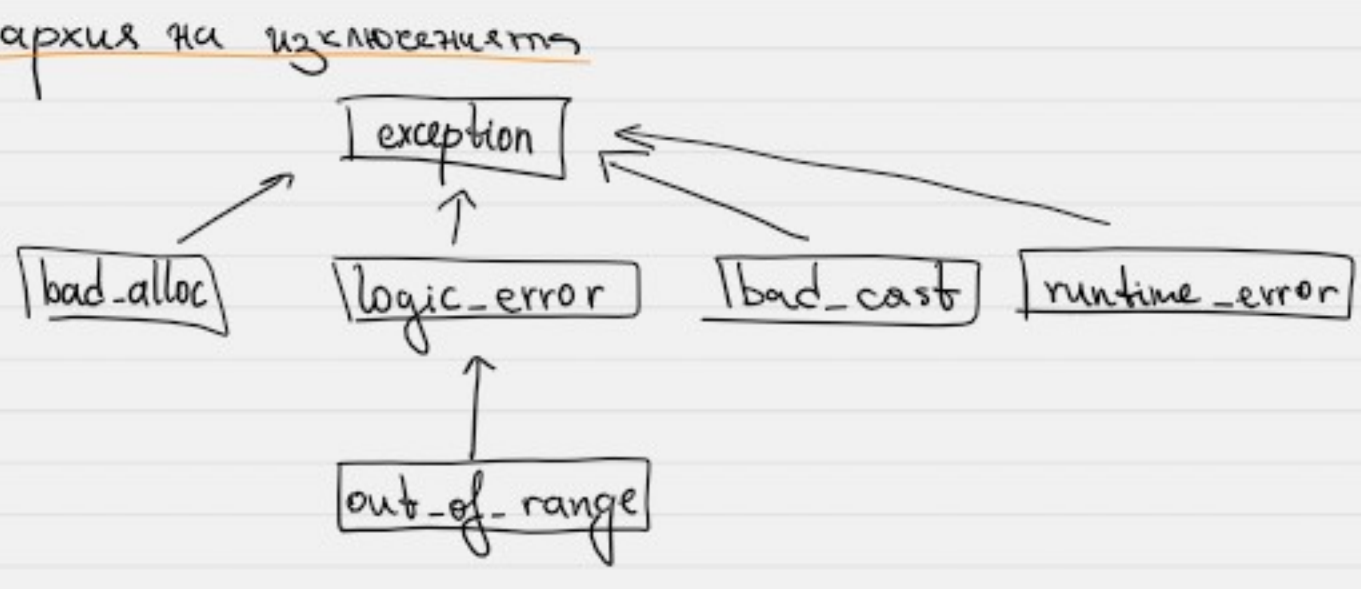
Stack unwinding – премахване на ф-ии от стека, докато не се достигне тази, която може да обработи грешката (всички успешно създадени променливи до момента се унищожават)

Синтаксис

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

int main()

   try {
      Y ob;
       f(-10);  // ще се хване от
      Z ob2;
   }
                                          ~Y()
   catch ( const std:: invalid_argument & e) {

     std:: cout << e.what() << endl;

   }
                     → приемане по const ref
   catch(const std:: logic_error & e) {__}

   catch ( const std:: exception & e) {__}
```

подреждане им от най-конкретния към най-общия

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 конструкторите

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

⇒ не трябва да остават незатворени външни ресурси

```
A {                     A()
  char* str ;            {  str = new char [10];
  char* str 2;
}                           try {
                               str 2 = new char[10];
                            }
                            catch (const std :: bad_alloc & e) {

                                delete[ ] str;
                             throw;
                            }
```

изтриване на вече заделената памет

- при деструкторите

При хвърлена грешка, ако в някой от дестр, които се извикват при stack-unwinding, също се хвърли изключение, то ще бъде второ ⇒ std::terminate

⇒ не се пишат изключения в деструкторите

Нива на сигурност

1. No-throw guarantee – класът / ф-ята не хвърля изкл.
2. Strong exc. safety – може да хвърля изкл., но няма да се промени състоянието му
3. Weak | Basic exc. safete – може да хвърля, но ще остане във валидно състояние
4. No exc. safety – няма никакви граници