Тема 8

Статични член-данни. Изключения. Обработка на изключения. Йерархия на изключенията и примери. Изключения в конструктор и деструктор. Нива на exception safety.

Static – ... Унищожават се при край на програмата и се създават при първото създаване на обекта.

- ф-ции
- член-данни
- член-ф-ции
- данни във ф-ции

I Статични ф-ции

A.cpp

```
static void f(){..}
void g(){..}
```

Отговаря на namespace { f(); }

Ограничена е до една компилационна единица (един .cpp файл)

Edge case

Същото важи и за static променливи

B.cpp

```
f(); X //linker error
g(); V
```

II Статични член-данни

Пример:

```
class X{
    static int z;
    int p;
    void f(){z++;p--;}
    static void g(){z++;p--;X o;}
};
```

Глобален обект скрит в X.

Не е обвързан с конкретна инстанция. Няма

Заделя се веднъж int x::z=0, ако не се задели се

⇒ Compile Time error;

III Член функция

Пример: class A {

static f(){..}

}

В.: статичните ф-ции могат да се използват само други статични ф-ции и ст. променливи. Те не са обвързани с инстанция

A::f(); ✓

Edge case

Не могат да са const/virtual, защото те фактически не са членове.

IV Статични данни във ф-ция

class Obj {

void f(){

static int called=0;

called++

}

};

Създава се 1 път при първоначалното извикване на ф-цията. Всяко следващо извикване брои със същата ф-ция.

main.cpp

Obj x; x.f() // called=1;

•Във ф-ция called може да не е =0; по default има такава

•static не променя големината на обекта

Изключения

- сигнален механизъм, с който програмата известява, че е възникнала грешка

Пример:

```
class A
{
  A() {
    try {
      B b();
      int x=3;
    } catch(int) {...}
}

class B
{
  B() {
    g();
  }
}

g() {
  throw 37;
}
```

Обработка на изключения

try-метод за дефиниране на блок от код, в който може да настъпи грешка.

catch - позволява за дефинирането на блок от код, който да се изпълни при грешка в try блока.

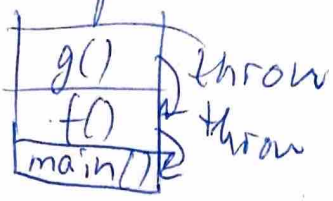
При грешка в код, програмата:

1) проверява дали има "подходящ" catch block, който да хване грешката

Пример:
```
try {
    throw 37;
} catch(char* ex) {...}
```

ако няма програмата се прекратява чрез std::terminate() а ако има:

2) Изпълнява се Stack unwinding
- програмата се връща назад по стека, докато не намери правилното място, където се обработва грешката.

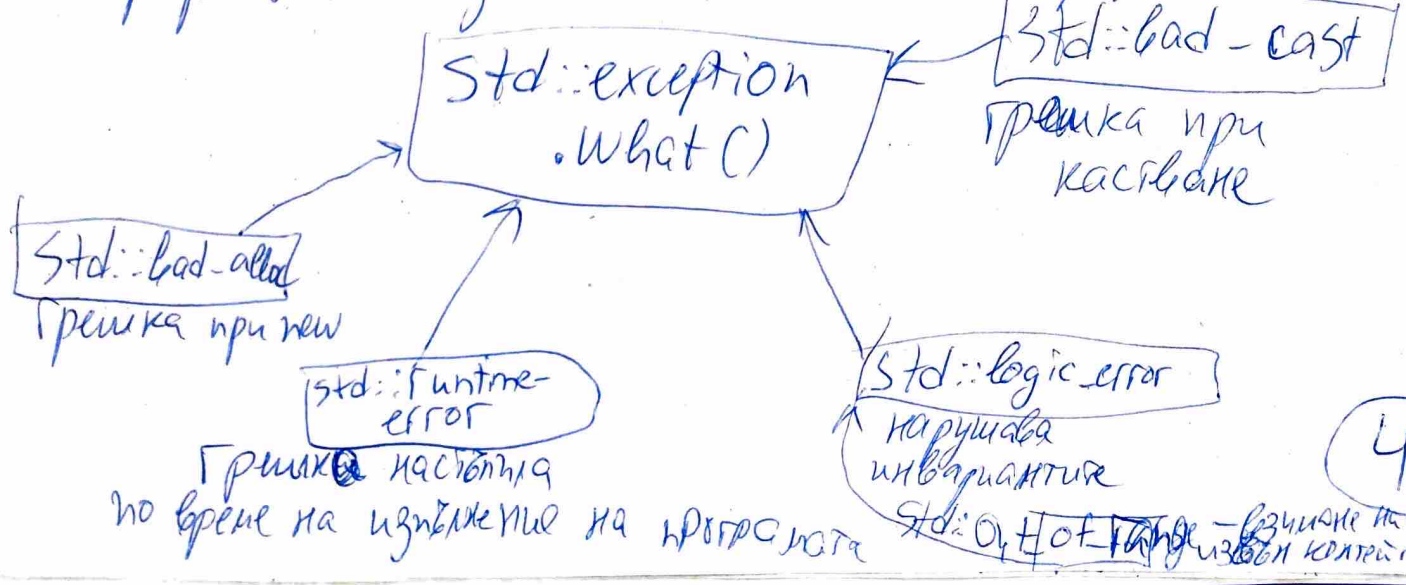


Пример: try {
...
} catch (int a) {
...
} catch (double d) {
...
} catch (const A&) {
...
} catch (std::bad_cast&) {
...
} catch (std::exception&) {
...
} catch (...) {
...

Изреждане на catch блокове от най-конкретния към най-общия

⚠ catch (std::exception&) - хваща само грешки от STL
catch (...) - хваща всякаква грешка → Код този catch не можем да напише други catch блокове

Йерархия на изключения



Изключения в конструктора:
- при хвърляне на изключения в констр. се извиква деструк. на # напълно заделени обекти
- Трябва да се погрижим за # вземени ресурси

1) class A
{
A() {
throw 37; // не се вика.
// ~A()
}

2) A() { char* _data = new char[37];
throw 37; // Трябва да се погрижим за изчистването на _data

3) A: B, C, D
A(): B(), C(), D()
↑ throw
// B(), C(), ~B()

За да решим проблем от 2)., ни трябва следната структура:

A() { char* _data = new char[37];
try {
throw 37;
} catch(int i) {
delete[] _data; throw; // rethrow на общата грешка }

⚠ При хвърлянето на 2 последователни грешки се извиква std::terminate

Пример:

```
A():B() {
    throw 42;
}

~B() {
    throw 420; // std::terminate
}
```

Изключения в деструктора

- след C++11 деструкторе е автоматично генериран с noexcept
- не трябва дестр. да хвърля грешки

nothrow

- начин за избягване на грешка при заделяне на памет

Пример: `char* str = new (std::nothrow) char[5];`

При грешки `str = nullptr;`

- трябва употреба при .clone()

Нива на exception safety:

1. Strong exception guarantee -
   ако n-ия хвърлен грешка, програмата се връща до състоянието си преди настъпване на грешката

Пример: vector(10), push на 11-ти елемент, останалите 10 се запазват

2. Basic exception guarantee -
ако ф-цията хвърли грешка, програмата ще
продължи да работи и е във валидно състояние.
Пример: vector(10) push на 11-тия, не се знае какво
ще е състоянието на останалите, но все още
ще може да се добави.

3. No exception guarantee - няма гаранция за
нищо.

4. No throw guarantee -
Никога няма да се върне грешка.
Пример: move(), mod = и ~()

SelfCounting.h

```
class SelfCounting
{
public:
    SelfCounting();
    SelfCounting(const SelfCounting&);
    ~SelfCounting();

    static unsigned getLive();
    static unsigned getTotal();

private:
    static unsigned live;
    static unsigned total;

};

unsigned SelfCounting::total=0;
unsigned SelfCounting::live=0;
SelfCounting::SelfCounting()
{
    live++;
    total++;
}
SelCounting(const SelfCounting & other)
{ live++;
  total++;
```

```
SelfCounting::~SelfCounting()
 { live--;
 }
unsigned SelfCounting::getLive()
 {
  return live;
}
unsigned SelfCounting::getTotal()
 {return total;
 }
```