Исключения

12 июля 2017 г.

Исключительные ситуации

ситуации, при которых дальнейшее выполнение программы невозможно

Причины:

- ошибки в коде
- неверные данные
- непредусмотренные действия пользователя

и т. п.

Задание 1

Напишите программу, которая получает от пользователя 2 числа и выводит в консоль их частное (делит первое число на второе). После каждого вычисления пользователю предлагается завершить работу или ввести следующие два числа.

Подумайте, как должна вести себя Ваша программа, если пользователь в качестве второго числа ввел 0.

Исключение (exception)

- реакция на исключительную ситуацию, возникшую в программе
- используются блоки try – catch
  - try – защищенный блок (вносится код, при выполнении которого может возникнуть исключение)
  - catch – блок, перехватывающий исключение

```
try { // операторы защищенного блока }
catch (…) { // обработка исключения }
```

Блок try

- включает код, в котором может быть выброшено исключение
- если выброшено исключение, дальнейшие операторы из блока try не выполняются, а управление передается дальше

```
try {
    if (max_vec_size < 0) {
            throw -1;
    }
    std::vector<int> vec;
    vec.reserve(max_vec_size);
}
```

Блок catch

- включает код для обработки исключений
- должен идти сразу за блоком try
- бывает нескольких видов:

catch(тип_исключения) { }

catch(тип_исключения имя) { }

catch(...) { }

- блоков catch может быть несколько

```
catch (char) { std::cout << "Exception" << std::endl; }
catch (int ex) { std::cout << "Exception №" << ex << std::endl; }
catch (...) { std::cout << "Unknown exception" << std::endl; }
```

throw

- когда возникает исключительная ситуация, выбрасывается исключение и передается управление блоку catch с таким же типом
- в одном блоке try может быть несколько операторов throw

```
try {
    if (max_vec_size < 0)
            throw -1;
    // …
}
catch(int x) { }  // goto
catch (…) { }
```

Пример 1

```
#include <iostream>

void f(){
    int x = 0;
    try {
        if (x == 0) { throw 1; }  // выброс исключения
        std::cout << 2/x;         // дальнейшее не выполнится
        int y = 15 + x;
        std::cout << y << std::endl;
    }
    catch (...) {                 // goto сюда
        std::cout << "Division by zero" << std::endl;
    }
}
```

Пример 2

```
#include <iostream>

void f(){
try {
        std::cout << "BEGIN TRY" << std::endl;
        throw 1;
        std::cout << "END TRY" << std::endl;
    }
    catch (int x) {
        std::cout << "Exception " << x << std::endl;
    }
    catch (char c) {
        std::cout << "Exception with " << c << std::endl;
    }
}
```

Пример 3

```
void f(){
  try {
     std::cout << "BEGIN TRY" << std::endl;
     std::srand(std::time(NULL));
     int value = std::rand();
     if (value % 2 == 0){
        throw 1;
     }
     else { throw 'z'; }
     std::cout << "END TRY" << std::endl;
  }
  catch (int x) { std::cout << "Exception " << x << '\n'; }
  catch (char c) { std::cout << "Exception with " << c << '\n'; }
}
```

catch(…)

- универсальный catch (ловит исключения с типом, который не соответствует типам в других блоках catch)
- всегда должен идти последним по счету

```
try {
    if (max_vec_size < 0)
            throw "Error";
    // …
}
catch(int x) { }
catch(char c) { }
catch (…) { }     //  const char*
```

Исключения и функции

- исключение можно генерировать в функции

```
void f(){
	std::cout << "BEGIN TRY" << std::endl;
	std::srand(std::time(NULL));
	int value = std::rand();
	if (value % 2 == 0) { throw 1; }
	else { throw 'z'; }
	std::cout << "END TRY" << std::endl;
}

void main(){
	try { f(); }
	catch (int x) { std::cout << "Exception with " << x << '\n'; }
	catch (char c) { std::cout << "Exception with " << c << '\n'; }
}
```

Ключевое слово noexcept (C++11)

- показывает, что функция не генерирует исключений
- пишется после объявления функции

```
void message() noexcept {
    std::cout << "invalid value "  << cur_value <<  std::endl;
}
```

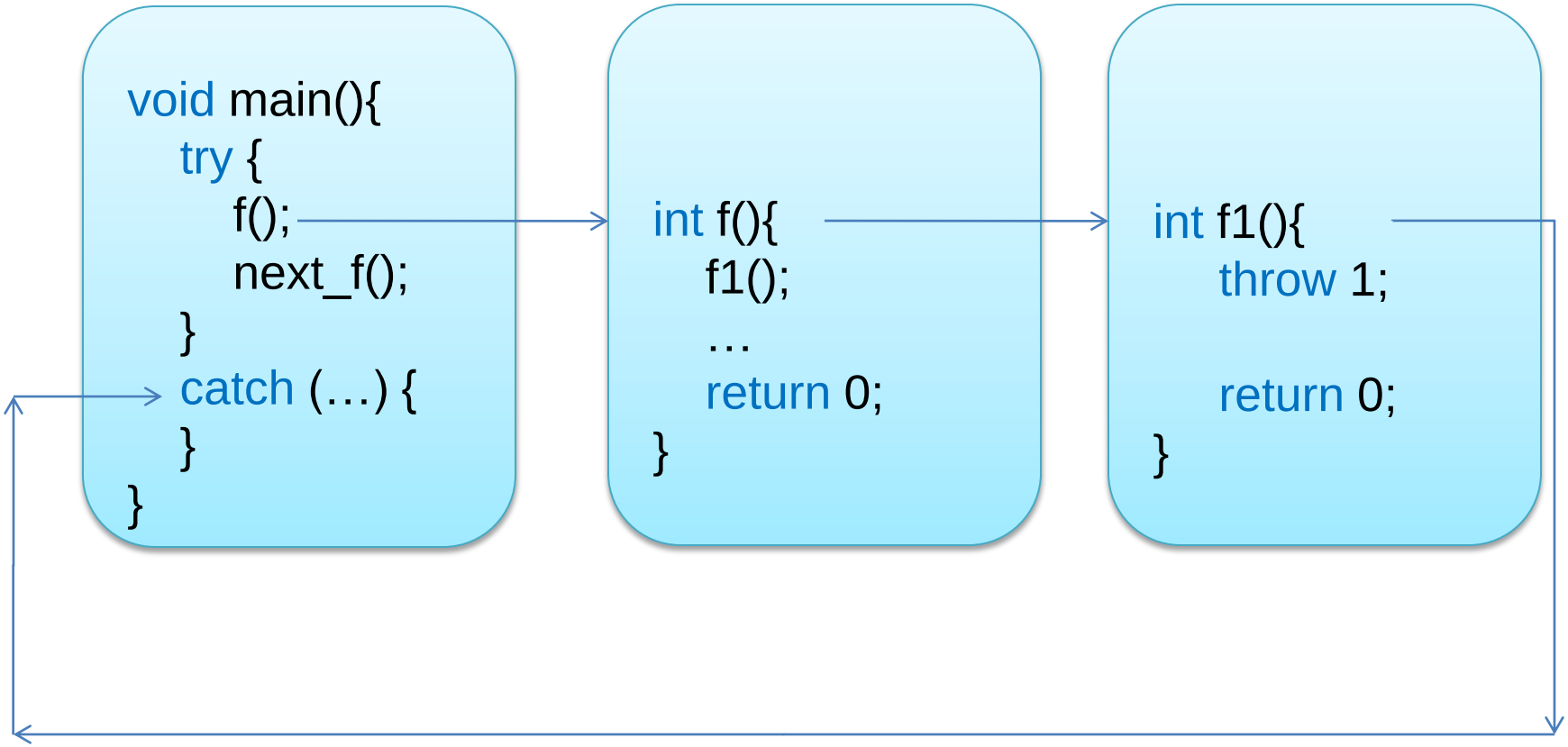
Что происходит внутри…

генерируется исключение

стек просматривается и ищется блок кода, откуда была вызвана функция, сгенерировавшая исключение

в найденном блоке кода ищется подходящий блок catch и выполняется обработчик исключения

Раскрутка стека



Задание 2

Создайте четыре функции:

void f1(), void f2(), void f3(), void f4().

Функция с номером i (1 <= i <= 3) выводит информацию о начале работы, вызывает функцию с номером i + 1 и печатает информацию о том, что отработала. В функции f4 с консоли считывается число. Если это число кратно 3 или 5, то генерируется исключение "fizz"; если число кратно 15, то генерируется исключение "buzz".

Функция f1 вызывается из main'а. Там же перехватываются исключения.

Задание 3

Создайте класс Test. В классе должны быть следующие методы и поля:

- статическое поле, которое подсчитывает, сколько раз вызван конструктор класса;
- конструктор, в котором печатается информация о вызове конструктора и порядковый номер вызова
- деструктор, в котором печатается информация о вызове деструктора и порядковый номер вызова.

В каждой из функций из задания 2 создайте по объекту класса Test. Проанализируйте раскрутку стека.

Исключения и объекты

- создается специальный класс для типа исключений
- при генерации исключения вызывается конструктор

```
class bad_range{
        int cur_value;
public:
        bad_range(int value) : cur_value(value) { }
        void message() {
            std::cout << "invalid value "  << cur_value <<  std::endl;
        }
};
```

Пример

```
class bad_range{
        int cur_value;
public:
        bad_range(int value) : cur_value(value) { }
        void message() {
            std::cout << "invalid value "  << cur_value <<  std::endl;
        }
};

void createArray(){
        int x;
        std::cin >> x;
        if (x > 100) { throw bad_range(x); }
        // …
}
```

```
try {
        createArray();
}
catch(bad_range &x) {
        x.message();
        // …
}
```

Особенности исключений

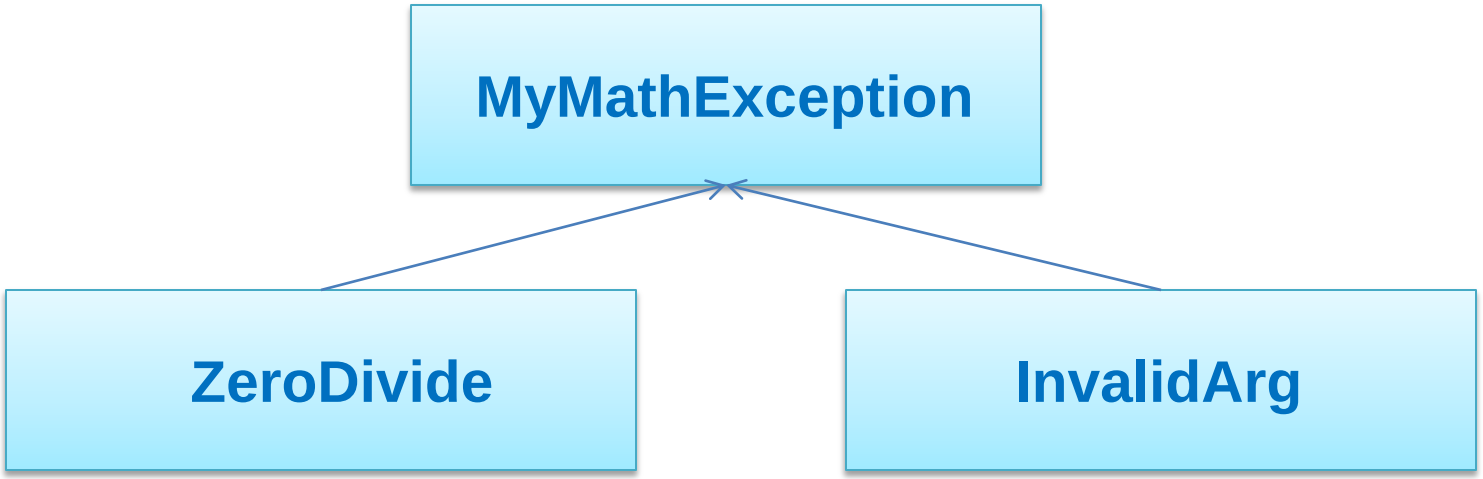
- оператор throw передает выполнение в ту внешнюю функцию, в которой есть подходящий по типу исключения блок try – catch
- при генерации исключения всегда создается копия (даже если ловится будто бы ссылка)

```
class  MyException { … };

void f () { if (1 == 1) throw MyException(); }
void g() {
   try {   f();   }
   catch(MyException& exc) { … }       // копия объекта
}
```

Иерархия исключений

- классы исключений можно наследовать друг от друга
- действуют стандартные правила наследования



Иерархия исключений (1)

```
class  MyMathExc{
public:
    virtual void print_info() { std::cout << "Math error\n"; }
};

class  ZeroDivide : public MyMathExc {
public:
    void print_info() { std::cout << "Divide by zero\n"; }
};

class  InvalidArg : public MyMathExc {
    int x;
public:
    InvalidArg(int tmp) : x(tmp) { }
    void print_info() { std::cout << "Invalid value " << x << "\n"; }
};
```

Иерархия исключений (2)

```
double div(){
    int x, y;
    std::cin >> x >> y;
    if (y == 0) throw ZeroDivide();
    if (y < 0) throw InvalidArg(y);
    if ( x > 10000000 ) throw MyMathExc();
    return static_cast<double>(x) / y;
}

void test() {
    try { std::cout << div() << std::endl; }
    catch(MyMathExc mm) { mm.print_info();  }   // Всегда здесь!
    catch(ZeroDivide zd) { zd.print_info(); }
    catch(InvalidArg ia) { ia.print_info(); }
}
```

Иерархия исключений (3)

```
double div(){
    int x, y;
    std::cin >> x >> y;
    if (y == 0) throw ZeroDivide();
    if (y < 0) throw InvalidArg(y);
    if ( x > 10000000 ) throw MyMathExc();
    return static_cast<double>(x) / y;
}

void test() {
    try { std::cout << div() << std::endl; }
    catch(ZeroDivide zd) { zd.print_info(); }
    catch(InvalidArg ia) { ia.print_info(); }
    catch(MyMathExc mm) { mm.print_info(); }          // OK
}
```

Иерархия исключений (4)

```
double div(){
    int x, y;
    std::cin >> x >> y;
    if (y == 0) throw ZeroDivide();
    if (y < 0) throw InvalidArg(y);
    if ( x > 10000000 ) throw MyMathExc();
    return static_cast<double>(x) / y;
}

void test() {
    try { std::cout << div() << std::endl; }
    catch(MyMathExc& mm) { mm.print_info(); }    // OK
    catch(…) { std::cout << "Unknown exception\n"; }
}
```

Выводы

- нужно располагать блоки catch в правильном порядке:
  от самого последнего в иерархии до базового
- при перехвате исключения удобно пользоваться указателем или ссылкой на базовый класс, а в самих классах иметь виртуальные методы
- если перехватывать объекты-исключения по значению, а ловить объект базового класса, то будет вызываться метод базового класса

Класс exception

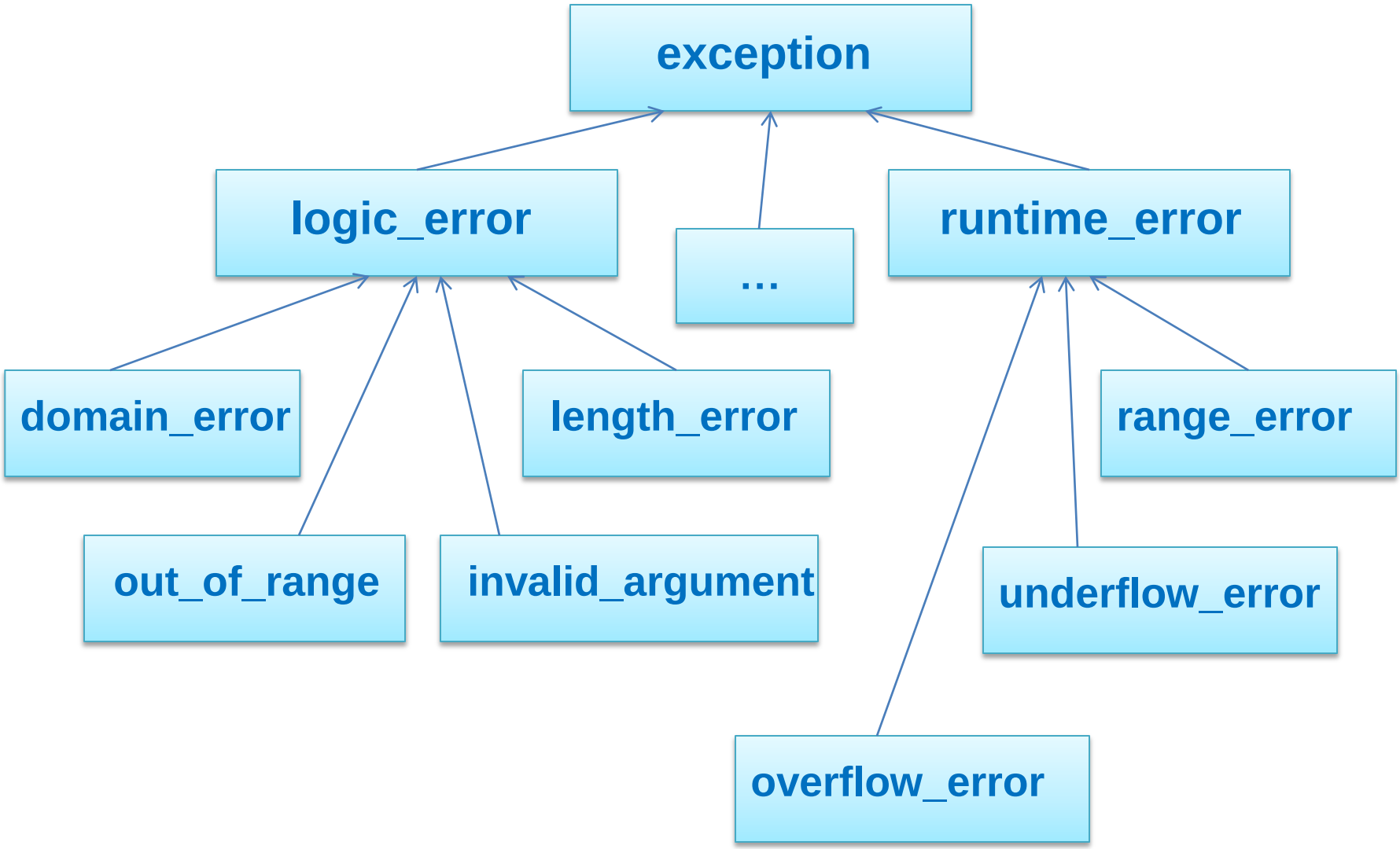
- нужно подключить заголовочный файл <exception>
- класс exception – базовый класс для остальных классов исключений
- виртуальный метод what() выводит строку с сообщением про тип исключения
- от классов стандартных исключений можно наследовать свой класс

Класс exception (пример)

```
double div(){
    double x, y;
    std::cin >> x >> y;
    if (x / y < 0) throw std::exception();
    std::cout << x / y << std::endl;
}

void test() {
    try { std::cout << div() << std::endl; }
    catch(std::exception& e) {
        std::cout << e.what() << std::endl;
    }
}
```

Стандартные исключения



Семейство logic_error

- класс logic_error – описание логических ошибок
- во всех производных классах реализован конструктор с параметром (текстовая строка для описания исключения)

```
class logic_error : public exception {
public:
  explicit logic_error(const std::string& what_arg);
};

class domain_error : public logic_error {
public:
  explicit domain_error(const std::string& what_arg);
};
```

Семейство runtime_error

- класс runtime_error – описание ошибок, которые могут возникнуть при исполнении программы
- реализован конструктор с параметром (текстовая строка для описания исключения)
- класс range_error – результат вычислений лежит вне допустимого диапазона
- класс underflow_error – потеря точности
- класс overfow_error – переполнение (результат превышает допустимый максимум)

Основные стандартные исключения

Оператор / функция	Исключение	Когда возникает
dynamic_cast	bad_cast	не может выполнить приведение типов
new	bad_alloc	не может выделить память
string	out_of_range length_error	выход за границы диапазона недостаточно памяти для append() – результат длиннее максимума
vector	out_of_range	выход за границы диапазона
typeid	bad_typeid	не может вернуть объект типа type_info
regex	regex_error	проблемы с регулярными выражениями

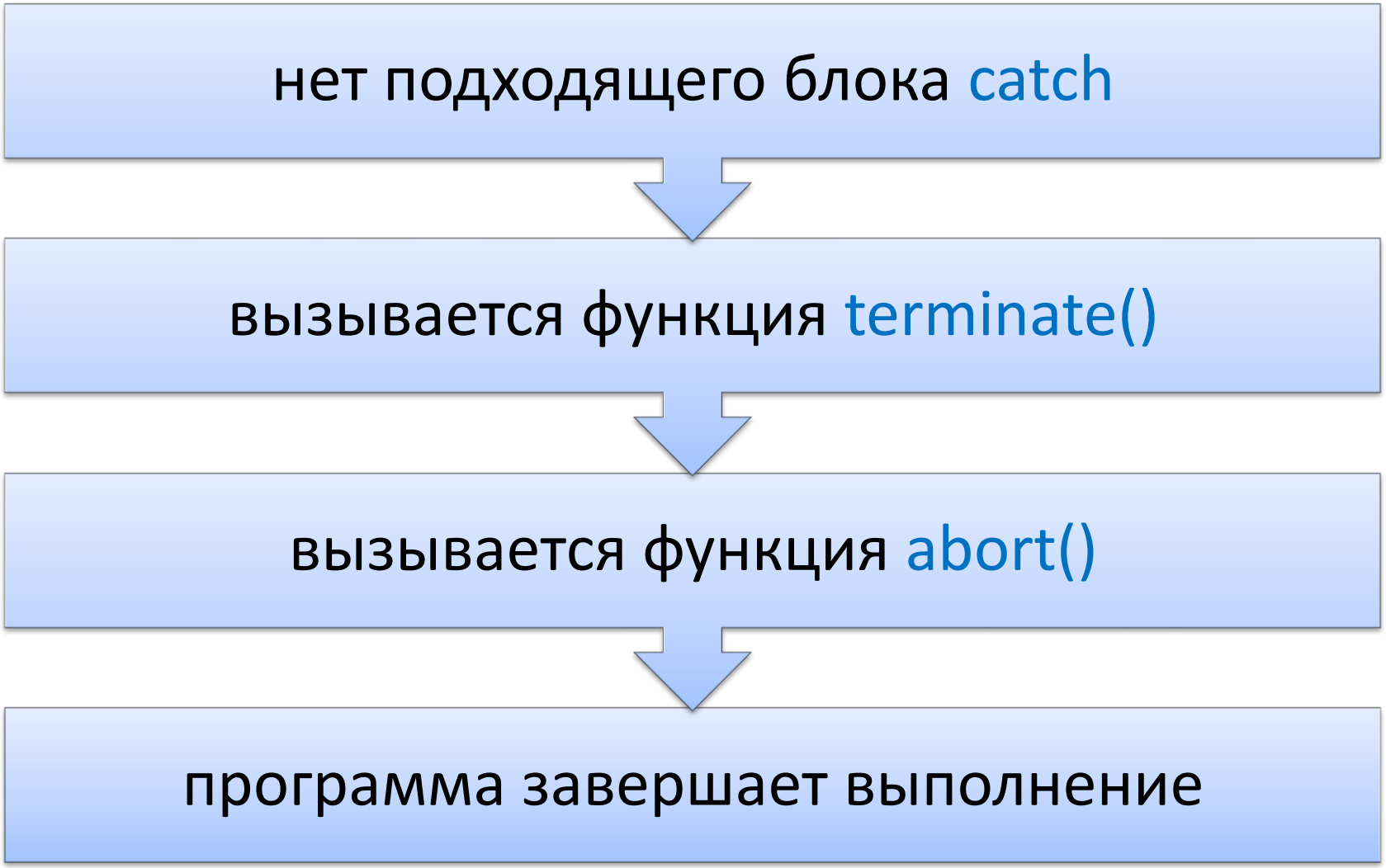
Лучше перехватывать исключение родительского класса (*exception*)

Исключения и выделение памяти

- оператор new может генерировать исключение bad_alloc, если у него нет запрашиваемого количества свободной памяти

```
void f {
    MyClass* obj;
    try { obj = new MyClass[100]; }
    catch (std::bad_alloc& ba){
        std::cout  << "Exception: " << ba.what() << std::endl;
        exit(1);
    }
    delete [] obj;
}
```

Неперехваченные исключения



Изменение поведения

- замена вызова функции abort() на свою функцию
- своя функция должна быть типа void без аргументов
- замена делается через set_terminate()

```
#include <exception>

void myExit() {
   exit(1);
}
set_terminame(myExit);
```

Советы

«Мы настоятельно рекомендуем не генерировать исключения встроенных типов, например числа типа int или С-строки. Вместо этого следует генерировать объекты типов, специально разработанных для использования в качестве исключений. Для этого можно использовать класс, производный от стандартного класса exception».

Бьерн Страуструп

«Функцию what() можно использовать для получения строки, предназначенной для указания информации об ошибке, вызвавшей исключение».

Бьерн Страуструп

Задание

Напишите функцию, которая конвертирует запись десятичного числа в виде строки (std::string) в целое число типа int.

Предусмотрите случай выхода за границы диапазона, определяемого типом int.

Обязательно используйте механизм исключений.

Пользоваться стандартными функциями для приведения типов нельзя.

Вопросы?