Лекция 4

ОБРАБОТКА ИСКЛЮЧИТЕЛЬНЫХ СИТУАЦИЙ

Лекция 4. Обработка исключительных ситуаций

План

1. Общее понятие об исключительных ситуациях.
2. Перехват и обработка исключительных ситуаций в C++.
3. Создание пользовательских исключений.
4. Стандартные исключения.

1. Общее понятие об исключительных ситуациях

Исключительная ситуация (ИС) – это некоторое событие, которое привело к сбою в работе программы. В результате возникновения ИС программа не может корректно продолжить свое выполнение. Иначе говоря, ИС – это **нарушение естественного хода выполнения алгоритма**.

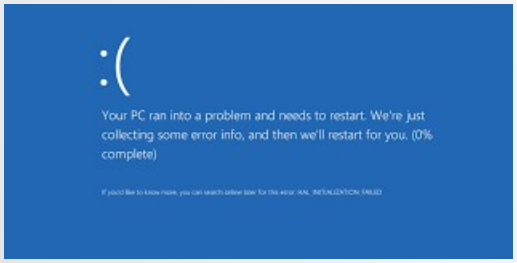
Существуют два основных типа ИС:

- **аппаратные** (генерируются процессором);
- **программные** (генерируются операционной системой или прикладными программами).

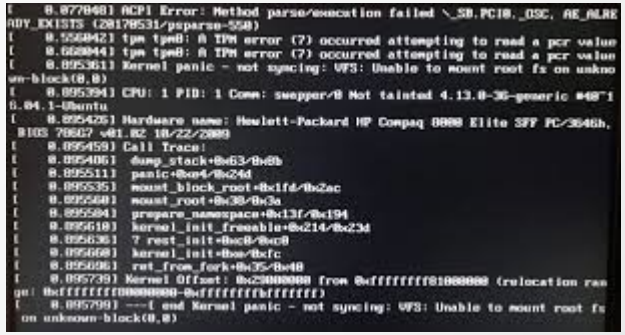
1. Общий понятие об исключительных ситуациях

Аппаратные ИС возникают, как правило, при различных сбоях оборудования, таких, например, как:

- деление на ноль;
- переполнение;
- обращение к неправильному адресу памяти;
- разрыв сетевого соединения;
- отключении питания и т. п.



BSOD – «синий экран смерти» в Windows

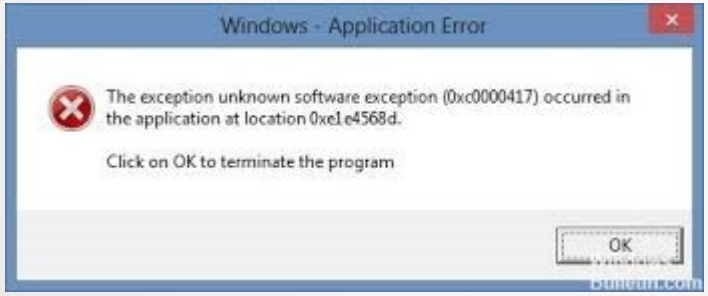


Kernel panic в Linux

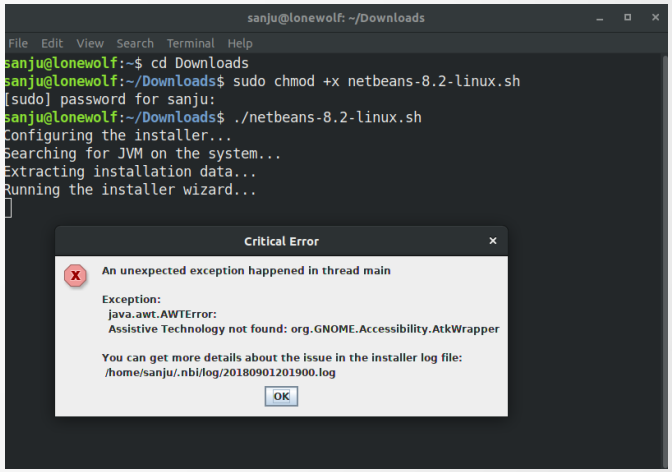
1. Общий понятие об исключительных ситуациях

Программные ИС явно **генерируются** (**возбуждаются**) ОС или прикладной программой, когда они обнаруживают аномальную ситуацию, возникшую в процессе их работы. Например:

- нехватка оперативной памяти;
- выход за пределы массива;
- извлечение корня из отрицательного числа;
- некорректные исходные данные и т. п.



ИС, возбуждаемая Windows при некорректной работе программы с памятью



ИС, генерируемая Linux при ошибке установки программы

1. Общий понятие об исключительных ситуациях

Рассмотрим следующий пример.

```
// Функция, реализующая деление двух чисел
double div(double top, double bot)
{
    return top / bot;
}
```

Очевидно, что в таком виде функция div() является небезопасной, т. к. при нулевом значении параметра bot произойдет деление на ноль, что вызовет соответствующее аппаратное прерывание с последующим **аварийным завершением программы**.

Программист, реализующий, например, математическую библиотеку, знает о «узких местах», где возможны возникновения ошибок, но понятия не имеет, как их обрабатывать (это должен делать пользователь библиотеки).

Примечание. Наличие в программе необработанных ИС является не просто дурным тоном, но и плохо характеризует разработчика в глазах заказчика!

1. Общий понятие об исключительных ситуациях

Стандартным подходом к решению этой проблемы является использование некоторой глобальной переменной, которой присваивается код ошибки, в случае ее возникновения. Например.

```
// Коды ошибок
enum { NO_ERR = 0, DIV_ZERO = 1, … };

// Глобальная переменная, содержащая код последней ошибки
int errno = NO_ERR;

// ...
double div(double top, double bot)
{
	errno = NO_ERR;
	if (bot == 0)
	{
		errno = DIV_ZERO;
		return 0;
	}
	return top / bot;
}
```

1. Общий понятие об исключительных ситуациях

Тогда порядок использования функции div() должен быть следующим.

```
// ...
double a, b, c;

cin >> a >> b;
c = div(a, b);
if (errno == DIV_ZERO)
{
      // Обработка ошибки деления на ноль
      cerr << "Devide by zero!" << endl;
      // ...
}
// …
```

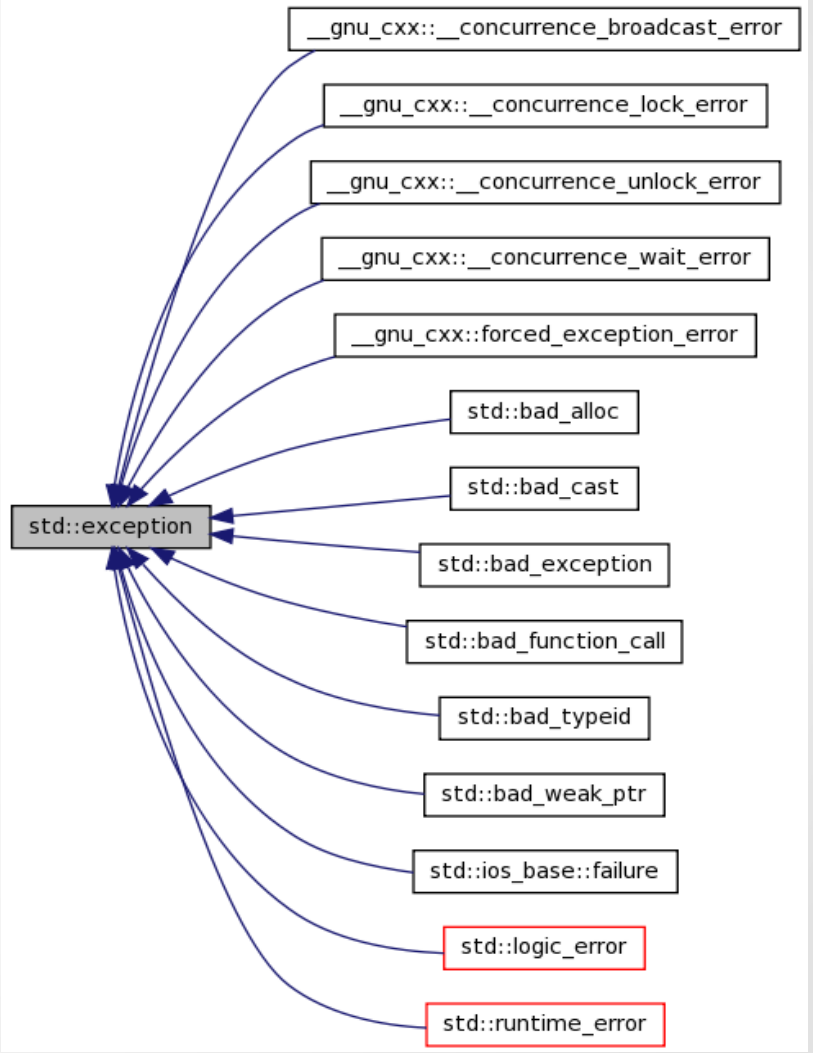
Такой подход к обработке ошибок реализован, например, в языке С. Очевидно, что он является неудобным, т. к. большое количество проверок на ошибки существенно увеличивает размер программного кода и делает его менее читабельным.

2. Перехват и обработка исключительных ситуаций в C++

В языке программирования C++ для работы с ИС используется понятие исключения.

Исключение – это переменная (объект некоторого класса или значение базового типа), которая описывает конкретную ИС и соответствующим образом обрабатывается.

Программист может как создать свою собственную систему описания ИС (например, квалифицировать разные типы ошибок целочисленными значениями), так и использовать специализированные классы стандартной библиотеки C++ (производные от класса **std::exception**).



2. Перехват и обработка исключительных ситуаций в C++

C++ поддерживает механизмы генерации, перехвата и обработки ИС. Для **перехвата и обработки** ИС в C++ используется конструкция try…catch, имеющая следующую форму:

```
try
{
   // Код, где возможно возникновение ИС
   // ...
}
catch (type1 arg1)
{
  // Код, обрабатывающий ИС типа type1 (данные об ИС находится в arg1)
  // ...
}
...
catch (typeN argN)
{
  // Код, обрабатывающий ИС типа typeN
  // ...
}
```

2. Перехват и обработка исключительных ситуаций в C++

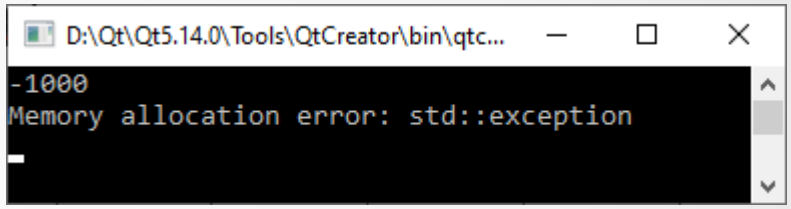
Рассмотрим пример обработки ИС средствами стандартной библиотеки C++.

```
#include <iostream>
#include <exception>

int main()
{
    int size, *array = nullptr;

    std::cin >> size; // Запрос размера массива
    try
    {
        array = new int[size]; // Попытка выделения памяти
    }
    catch (std::exception e)
    {
        // Обработка ИС
        std::cerr << "Memory allocation error: " << e.what() << std::endl;
        return 1;
    }
    delete [] array; // Освобождение памяти
    return 0;
}
```

Пример работы программы



2. Перехват и обработка исключительных ситуаций в C++

Следует отметить, что в вышеприведенном примере будет осуществлен перехват только ИС, тип которой соответствует std::exception. Все другие ошибки будут проигнорированы. В случае, если, например, в блоке try {} произойдет деление на ноль, то такой тип ИС обработан не будет.
Для исправления данной ситуации используется вариант catch(…) {}, в котором перехватываются все типы ИС, правда, без возможности идентифицировать причину их возникновения.

```
try
    {
       // Код, где возможно возникновение ИС
       // ...
    }
    catch (std::exception e)
    {
       // Стандартный обработчик
       // ...
    }
    catch (...)
    {
       // Обработчик ИС по умолчанию
       // ...
    }
```

2. Перехват и обработка исключительных ситуаций в C++

Для **генерации** (**возбуждения**) ИС в C++ используется оператор **throw**. В общем виде он имеет следующую форму:

throw [исключение];

Здесь необязательный параметр **исключение** содержит некоторое выражение или переменную (в т. ч. это может быть объект класса), идентифицирующие ИС. В результате выполнения throw генерируется ИС, тип которого определяется значением параметра «исключение» и которое должно быть обработано в блоке catch {}.

```
// …
try
{
	// …
	throw 1; // Генерация ИС (в catch будет передано целочисленное значение 1)
}
catch (int err)
{
	// Обработка...
}
```

Примечание. Вызов throw без параметров вне блока catch {} аварийно завершает работу программы.

2. Перехват и обработка исключительных ситуаций в C++

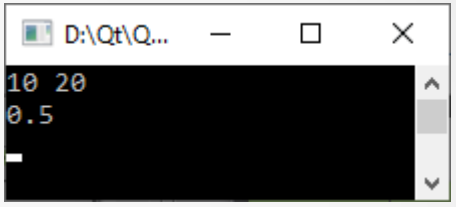
Рассмотрим еще один простой пример.

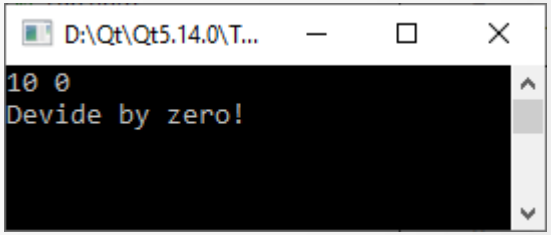
```
#include <iostream>

int main()
{
    double a, b, c;

    std::cin >> a >> b;
    try
    {
        if (b == 0)
            throw 1; // Генерация ИС
        c = a / b;
    }
    catch (...)
    {
        std::cerr << "Devide by zero!" << std::endl;
        return 1;
    }
    std::cout << c << std::endl;
    return 0;
}
```

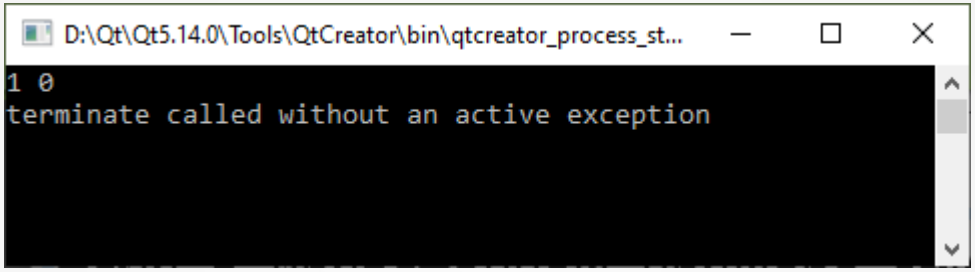
Результаты работы программы





2. Перехват и обработка исключительных ситуаций в C++

Если в приведенном примере заменить «throw 1;» на «throw;», то вывод программы в случае ошибки изменится.



Таким образом, вызов throw без параметров в блоке try {} приводит к аварийному завершению программы.

Как уже отмечалось, throw без параметров можно вызывать в блоке catch {}. В этом случае ИС будет передано для обработки вышестоящему обработчику.

3. Создание пользовательских исключений

Рассмотрим пример создания класса, реализующего пользовательские исключения.

```
// Коды ошибок
enum { NO_ERR = 0, DIV_ZERO, NEG_ROOT };

// Реализация пользовательских исключений
class Error
{
private:
    int err_code = NO_ERR; // Код ошибки
public:
    Error(int e) : err_code(e) {} // Конструктор
    ~Error(void) {} // Деструктор
    void setError(int e) // Присвоение кода ошибки
    {
        err_code = e;
    }
    int getError(void) const // Возврат кода ошибки
    {
        return err_code;
    }
```

```
    // Возврат информации об ошибке
    std::string sayError(void)
    {
        std::string ret;

        switch (err_code)
        {
            case NO_ERR:
                ret = "OK";
                break;
            case DIV_ZERO:
                ret = "Divide by zero";
                break;
            case NEG_ROOT:
                ret = "Negative root";
                break;
            default:
                ret = "Unknown error";
        }
        return ret;
    }
};
```

3. Создание пользовательских исключений

Использовать вышеприведенный класс можно, например, таким образом.

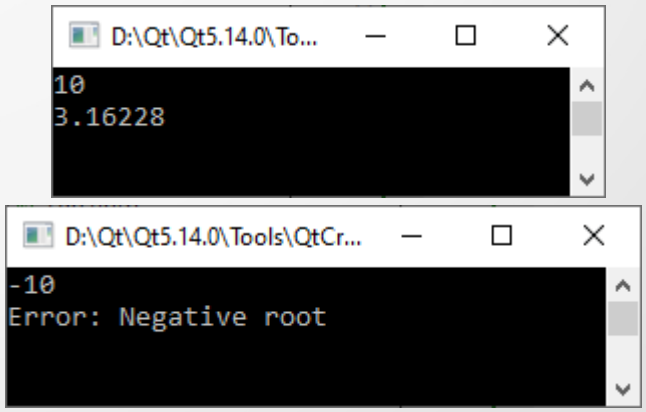
```
#include <iostream>
#include <cmath>

// Заголовочный файл,
// в котором описан класс Error
#include "error.h"

int main()
{
    double val,
           res;

    std::cin >> val;
    try
    {
        if (val < 0)
            // Генерация пользовательского исключения
            throw Error(NEG_ROOT);
        res = sqrt(val);
    }
    catch (Error err)
    {
        // Вывод сообщения об ошибке
        std::cerr << "Error: " <<
                     err.sayError() << std::endl;
        return 1;
    }
    std::cout << res << std::endl;
    return 0;
}
```

Результаты работы программы



4. Стандартные исключения

В стандартной библиотеке языка C++ реализовано множество классов для обработки ИС. Все они являются производными от класса **std::exception**, подключить описание которого к программе можно следующим образом:

```
#include <exception>
```

Класс std::exception помимо конструктора, деструктора и оператора присваивания содержит виртуальный метод **what()**, возвращающий строку-пояснение о возникшей ошибке.

Класс std::exception предоставляет единый интерфейс для обработки ошибок посредством опеаратора throw. Все исключения, генерируемые стандартной библиотекой, наследуются от std::exception.

Пример его использования приведен на Слайде № 11.

4. Стандартные исключения

Наследниками exception является множество классов, среди которых можно выделить следующие:

- **logic_error** – сообщает об ошибках, которые являются следствием неправильной логики в рамках программы;
- **runtime_error** – сообщает об ошибках, выходящих за рамки программы и трудно предсказуемых (переполнение, исчезновение точности и т. п.);
- **bad_function_call** – информирует о некорректных вызовах функций (например, при неправильной адресации к ним и т. п.);
- **bad_alloc** – генерируется в случае, если произошла ошибка выделения памяти;
- **bad_cast** – сообщает об ошибках, возникающих при преобразования типов данных;
- **failure** – генерируется при ошибках ввода-вывод и и. д.

4. Стандартные исключения

Пример обработки ИС, возникающих при файловом вводе-выводе.

```
#include <iostream>
#include <fstream>

int main ()
{
    std::ifstream file;

    // Настройка параметров перехватываемых ИС
    file.exceptions(std::ifstream::failbit | std::ifstream::badbit);
    try
    {
        file.open("test.txt");
        while (!file.eof())
            file.get();
        file.close();
    }
    catch (std::ifstream::failure e)
    {
        std::cerr << "Exception opening/reading/closing file: " << e.what() << std::endl;
        return 1;
    }
    return 0;
}
```

Вывод программы при попытке открытия несуществующего файла