

## 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:

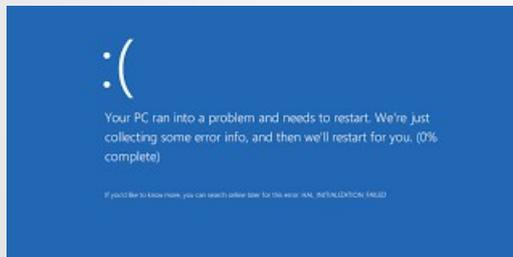
- **аппаратные** (генерируются процессором);
- **программные** (генерируются операционной системой или прикладными программами).



# 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.



```
to your computer.
SYSTEM_SERVICE_EXCEPTION

If this is the first time you've seen this Stop error screen,
restart your computer. If this screen appears again, follow
these steps:

Check to make sure any new hardware or software is properly
installed. If this is a new installation, ask your hardware or
software manufacturer for any Windows updates you might
need.

If problems continue, disable or remove any newly installed
hardware or software. Disable BIOS memory options such as
caching or shadowing. If you need to use Safe Mode to
remove or disable components, restart your computer,
press F8 to select Advanced startup options, and then
select Safe Mode.

Technical information:
*** STOP: 0x0000003B (0x0000000000000005, 0xfffff96002a324a, 0xfffff8c004a1e00, 0x0000000000000000)
*** win32k.sys - Address FFFFF8002A024A, base at FFFFF80000F0000, DateStamp
4a1bc560

collecting data for crash dump ...
initializing disk for crash dump ...
beginning dump of physical memory ...
dumping physical memory to disk: 100
physical memory dump complete.
contact your system admin or technical support group for further assistance.
```

**BSOD – «синий экран смерти» в Windows**

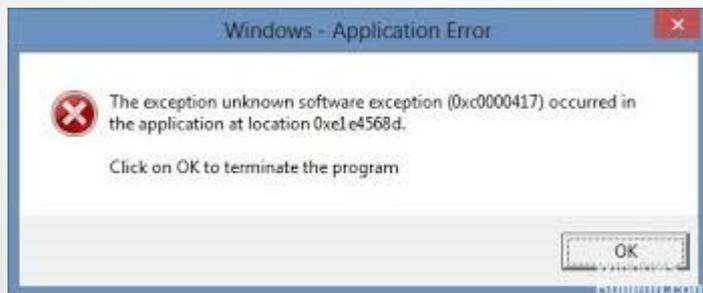
```
[ 0.877948] ACPI Error: Method parse/execution failed \_SB_.PCI0._OSC, AE_MRE
ADV_EXISTS (28178531/psparse-558)
[ 0.558942] tpm tpm0: 0 TPM error (7) occurred attempting to read a pcr value
[ 0.828944] tpm tpm0: 0 TPM error (7) occurred attempting to read a pcr value
[ 0.895361] Kernel panic - not syncing: VFS: Unable to mount root fs on unkno
wn-block(0,0)
[ 0.895394] CPU: 1 PID: 1 Comm: swapper/0 Not tainted 4.13.0-36-generic #48~1
6.04.1-Ubuntu
[ 0.895426] Hardware name: Hewlett-Packard HP Compaq 8888 Elite SFF PC/3646b,
BIOS 78607 v01.02 10/22/2009
[ 0.895459] Call Trace:
[ 0.895466] dump_stack+0x63/0x8b
[ 0.895511] panic+0x4/0x24d
[ 0.895535] mount_block_root+0xc1d/0b2ac
[ 0.895568] mount_root+0x38/0x3a
[ 0.895594] prepare_namespace+0xc13f/0bc194
[ 0.895618] kernel_init_from_memory+0xc214/0bc234
[ 0.895634] ? rest_init+0xc8/0xc0
[ 0.895668] kernel_init+0xa/0xcfc
[ 0.895686] ret_from_fork+0x35/0x48
[ 0.895739] Kernel Offset: 0xc2000000 from 0xffffffff10000000 (relocation ran
ge: 0xffffffff10000000-0xffffffffffffffff)
[ 0.895793] ---[ end Kernel panic - not syncing: VFS: Unable to mount root fs
on unknown-block(0,0)
```

**Kernel panic в Linux**

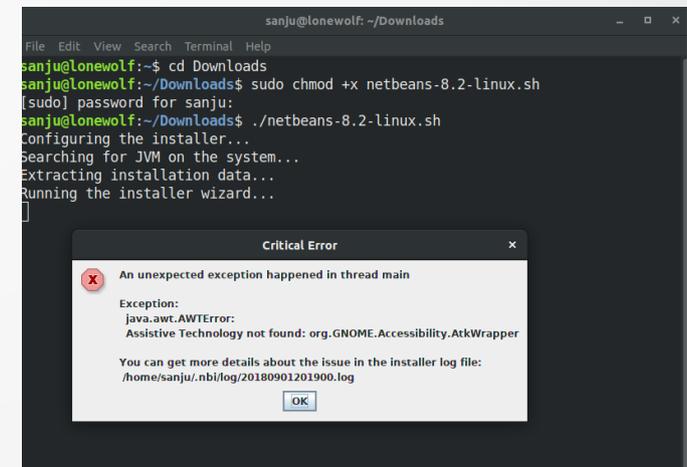
# 1. Общий понятие об исключительных ситуациях

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;
    // ...
}
// ...
```

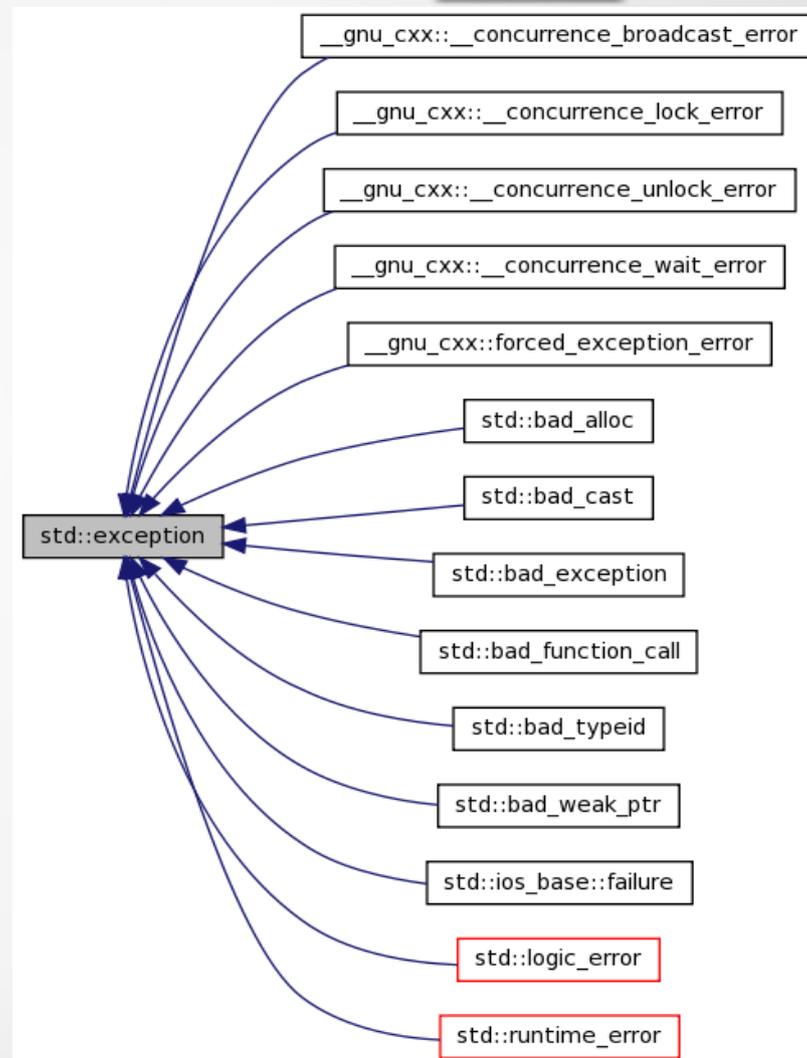
Такой подход к обработке ошибок реализован, например, в языке С. Очевидно, что он является неудобным, т. к. большое количество проверок на ошибки существенно увеличивает размер программного кода и делает его менее читабельным.

## 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++

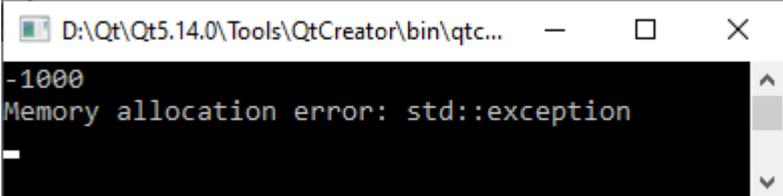
Рассмотрим пример обработки ИС средствами стандартной библиотеки C++.

```
#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



The screenshot shows a console window titled "D:\Qt\Qt5.14.0\Tools\QtCreator\bin\qtc...". The output of the program is as follows:

```
-1000
Memory allocation error: std::exception
```

## 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++

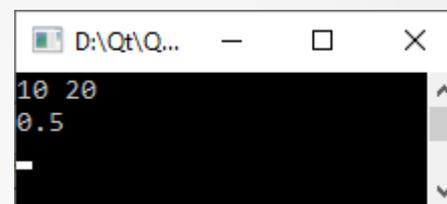
Рассмотрим еще один простой пример.

```
#include <iostream>

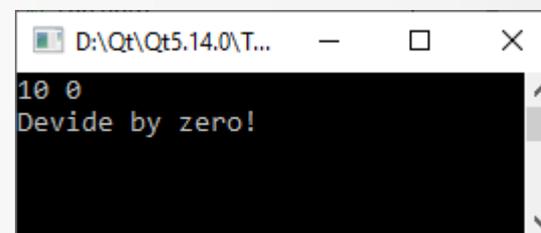
int main()
{
    double a, b, c;

    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



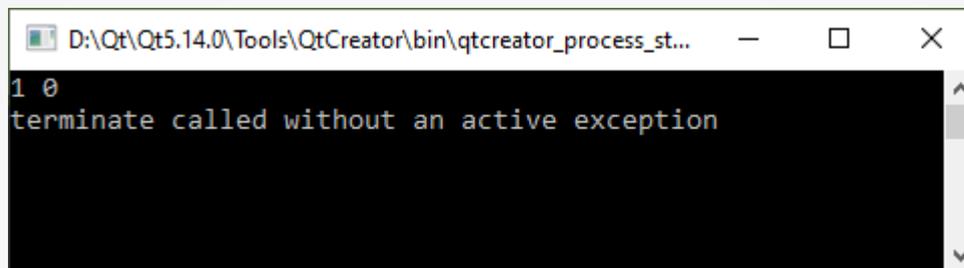
```
D:\Qt\Q...
10 20
0.5
```



```
D:\Qt\Qt5.14.0\T...
10 0
Devide by zero!
```

## 2. Перехват и обработка исключительных ситуаций в C++

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



```
D:\Qt\Qt5.14.0\Tools\QtCreator\bin\qtcreator_process_st...  
1 0  
terminate called without an active exception
```

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>

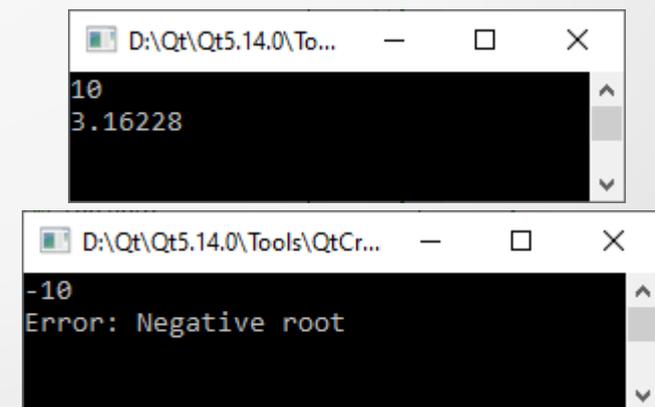
// Заголовочный файл,
// в котором описан класс Error
#include "error.h"

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



```
D:\Qt\Qt5.14.0\To...
10
3.16228

D:\Qt\Qt5.14.0\Tools\QtCr...
-10
Error: Negative root
```

## 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 оператора `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();
}
```

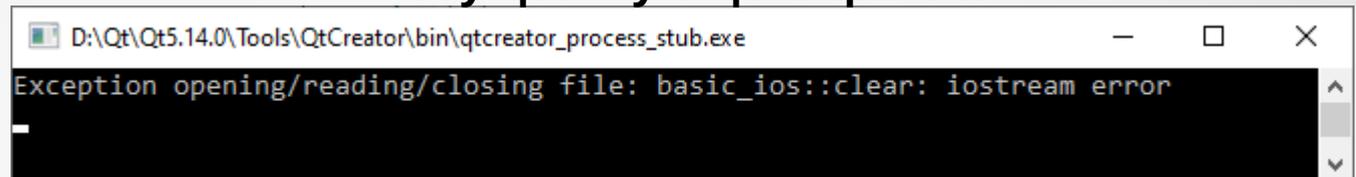
```
catch (std::ifstream::failure e)
```

```
{
    std::cerr << "Exception opening/reading/closing file: " << e.what() << std::endl;
    return 1;
}
```

```
return 0;
```

```
}
```

### Вывод программы при попытке открытия несуществующего файла

A screenshot of a terminal window titled "D:\Qt\Qt5.14.0\Tools\QtCreator\bin\qtcreator\_process\_stub.exe". The terminal displays the following error message: "Exception opening/reading/closing file: basic\_ios::clear: iostream error". The message is printed in a light blue font on a black background. The terminal window has standard Windows window controls (minimize, maximize, close) in the top right corner.

```
D:\Qt\Qt5.14.0\Tools\QtCreator\bin\qtcreator_process_stub.exe
Exception opening/reading/closing file: basic_ios::clear: iostream error
```