Механизм обработки исключений

> ***Исключение*** – событие при выполнении программы, при котором ее дальнейшее выполнение становится бессмысленным.

1. Обработка исключительных ситуаций (`try/catch/throw`)

Исключение – это аномальное событие, произошедшее во время выполнения программы.

Примеры:

- логические ошибки разработчика: деление на 0, выход индекса за границы массива и т.п.;
- ошибки времени выполнения: отсутствие свободной памяти и т.п.

Исключения нарушают нормальный ход работы программы и на подобные события нужно немедленно отреагировать.

> Механизм обработки исключительных ситуаций в языке C++ основан на операторах `try`, `catch` и `throw` (ключевые слова).

Блок `try` содержит фрагмент программы, подлежащий контролю.

Оператор `throw` генерирует исключение.

Оператор `catch` обрабатывает исключение.

Как это работает.

Если в ходе выполнения программы в блоке `try` возникает исключительная ситуация, то для ее обработки генерируется ошибка с помощью оператора. Синтаксис:

```
throw <тип_генерируемого_исключения>;
```

Затем исключительная ситуация перехватывается блоком `catch` для ее обработки. Выполнение программы переходит от оператора `throw` к блоку:

```
catch (<тип_исключения>){// тело }.
```

С одним блоком `try` может быть связано несколько `catch`. Выбор нужного `catch`-обработчика определяется типом исключительной ситуации.

После завершения обработки исключения выполнение возобновляется с инструкции, следующей за последним `catch`-обработчиком в списке.

Для перехвата ***любых*** исключений используется конструкция вида:

```
catch(...){// тело }
```

Управление к нему передается при исключении любого типа.

Если в программе нет подходящего обработчика `catch`, то вызывается функция `terminate()` из стандартной библиотеки C++. По умолчанию `terminate()` активизирует функцию `abort()`, которая завершает программу.

Исключения и освобождение стека в C++

В механизме исключений C++ управление передается от оператора `throw` в первый оператор `catch`, который может обработать выброшенный тип исключения. При достижении оператора `catch` все автоматические переменные, находящиеся в области между операторами `throw` и `catch`, удаляются. Этот процесс называется ***очистка стека***. Удаление происходит в порядке, обратном созданию локальных переменных. Далее выполнение программы продолжается с инструкции, следующей за последним `catch`-обработчиком.

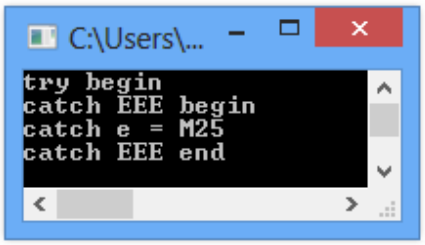
```
struct EEE {int k; char c;};

int _tmain(int argc, _TCHAR* argv[])
{
    try
    {
        std::cout << "try begin"<< std::endl;
        throw (int)77;
        std::cout << "try end"  << std::endl;
    }
    catch (int e)
    {
        std::cout << "catch int begin"<< std::endl;
        std::cout << "catch e = "<<e<<std::endl;
        std::cout << "catch int end"  << std::endl;
    }
    catch(char e)
    {
        std::cout << "catch char begin"<< std::endl;
        std::cout << "catch e = "<<e<<std::endl;
        std::cout << "catch char end"  << std::endl;
    }
    catch(EEE e)
    {
       std::cout << "catch EEE begin"<< std::endl;
       std::cout << "catch e = "<<e.c << e.k<<std::endl;
       std::cout << "catch EEE end"  << std::endl;
    }
```

```
C:\Users\Use...
try begin
catch int begin
catch e = 77
catch int end
```

В приведенном примере блок `try` содержит три оператора. С ним связана инструкция `catch (int i)`, выполняющая обработку целочисленной исключительной ситуации. Внутри блока `try` выполнено только два из трех операторов: первый оператор `cout` и оператор `throw`. Далее управление передается блоку `catch (int i){…}`, а выполнение блока `try` прекращается. Говорят, блок `catch` ***не вызывается***, а программа ***переходит*** к его выполнению (для этого стек программы автоматически обновляется).

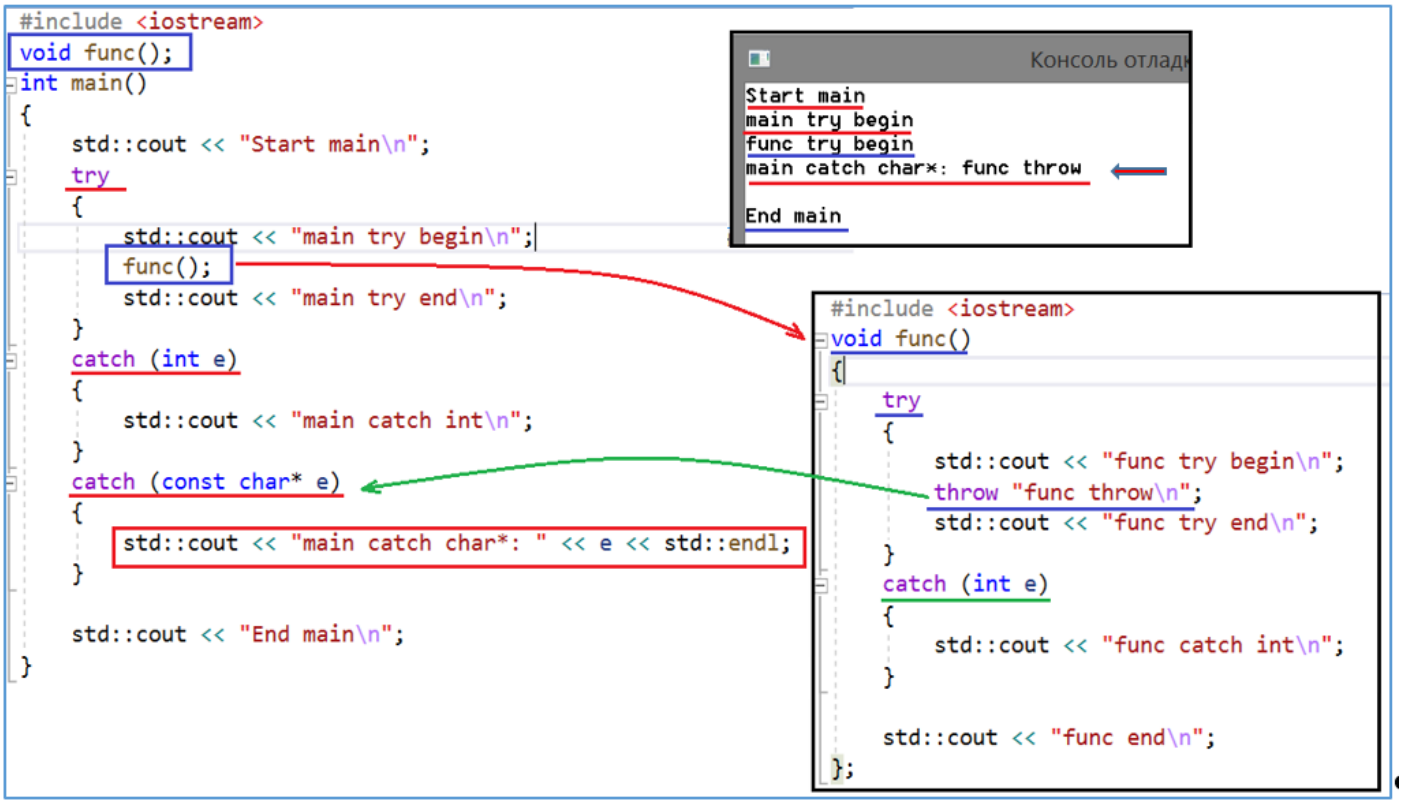
Таким образом, оператор `cout`, следующий за оператором `throw`, никогда не выполняется.

```
int _tmain(int argc, _TCHAR* argv[])
{
    try
    {
        std::cout << "try begin"<< std::endl;
        EEE eee = {25, 'M'};
        throw eee;
        std::cout << "try end"  << std::endl;
    }
    catch (int e)
    {
        std::cout << "catch int begin"<< std::endl;
        std::cout << "catch e = "<<e<<std::endl;
        std::cout << "catch int end"  << std::endl;
    }
    catch(char e)
    {
        std::cout << "catch char begin"<< std::endl;
        std::cout << "catch e = "<<e<<std::endl;
        std::cout << "catch char end"  << std::endl;
    }
    catch(EEE e)
    {
       std::cout << "catch EEE begin"<< std::endl;
       std::cout << "catch e = "<<e.c << e.k<<std::endl;
       std::cout << "catch EEE end"  << std::endl;
    }
```



Необработанное исключение распространяется по стеку вызова

Необработанное в функции исключение ***распространяется*** по стеку вызова функций:



Очистки стека

В блоке `try` главной функции `main` операторы выполняются последовательно. Выполнение потока переходит в функцию `func()`. В стеке создаются локальные объекты, объявленные в инструкциях и определениях этой функции. Последовательно выполняются операторы блока `try`. В вызванной функции `func()` генерируется исключение типа `char`. В функции `func()` соответствующего обработчика нет и происходит возврат к операторам `catch` главной функции `main`. Т.е. поиск подходящего обработчика продолжается в вызывающей функции. При этом прекращают свое существование локальные объекты, объявленные в функции `func`. Этот процесс называется ***очисткой стека***.

Обратите внимание, что завершается обработка исключения в функции `main`, содержащей походящий обработчик `catch` (для обработки исключение типа `char`).

> Говорят, что **«необработанное *в функции исключение распространяется* по стеку вызова функций».**

Процесс, в результате которого программа последовательно покидает составные инструкции и определения функций в поисках оператора `catch`, способного обработать возникшее исключение, называется ***раскруткой стека***. По мере раскрутки прекращают существование локальные объекты, объявленные в

составных инструкциях и определениях функций, из которых произошел выход. C++ гарантирует, что во время описанного процесса вызываются деструкторы локальных объектов классов.

2. Пример обработка ошибок:

```
Джон фон Нейман прожил 19402 дней
Ошибка: Date:  год должен быть 1 или больше
```

```
#include "stdafx.h"
#include <locale>
#include <iostream>
namespace Date
{
    // количество дней между датами
    unsigned long  distance(short yyyy1, short mm1, short dd1,
                            short yyyy2, short mm2, short dd2);
};

int _tmain(int argc, _TCHAR* argv[])
{
    setlocale(LC_ALL, "rus");
    try
    {
       // Джон фон Нейман (28.12.1903 - 9.02.1957)
       long  d1 =  Date::distance (1957, 2, 9, 1903, 12, 28);
       std::cout<<" Джон фон Нейман прожил "<<d1<< " дней"<<std::endl;
       // от рождества Христова.
       long  d2 =  Date::distance (0, 1, 7, 2015, 3, 24);
       std::cout<<"От рождества Христова прошло "<<d2<< " дней"<<std::endl;
       // Ада Лавлейс  (10.12.1815 - 27.12.1852)
       long  d3 =  Date::distance (1852, 12, 27, 2015, 3, 24);
        std::cout<<"Со дня смерти Ады Лавлейс прошло "<<d3<< " дней"<<std::endl;
    }
    catch (char* e) { std::cout<<" Ошибка: "<<e <<std::endl;};

    system("pause");
    return 0;
}
```

```
#include "stdafx.h"
#include <locale>
#include <iostream>
namespace Date
{

    unsigned long  datetoday(short yyyy, short mm, short dd)
    {
        bool G = (yyyy < 1582) || (yyyy == 1582 && mm < 10) || (yyyy == 1582 && mm == 10 && dd <15);
        //int A = (G?0:2-(yyyy/100) + (yyyy/400));   // это правильно
        int A = 2-(yyyy/100) + (yyyy/400);           // так у Microsoft
        mm = (mm <=2? (yyyy--, mm+12): mm);
        unsigned long rc =  (1461L * long(yyyy))/4L;
        unsigned long k =  (306001L * long(mm+1))/10000L;
        rc += k + dd  +1720995L + A;
        return rc;
    };

    unsigned long distance(short yyyy1, short mm1, short dd1, short yyyy2, short mm2, short dd2)
    {
     if (yyyy1 < 1 || yyyy2 < 1)                          throw "Date:  год должен быть 1 или больше";
     if (mm1 < 1   || mm1 > 12 || mm2 < 1   || mm2 > 12)  throw "Date:  месяц должен быть в интервале от 1 до 12";
     if (dd1 < 1   || mm1 > 31 || dd2 < 1   || dd2 > 31)  throw "Date: день должен быть в интервале от 1 до 31";
     if (dd1 > 28 && yyyy1%4 > 0)                         throw "Date: день должен быть в интервале от 1 до 28";
     if (dd1 > 29 && yyyy1%4 == 0)                        throw "Date: день должен быть в интервале от 1 до 29";
     return  datetoday(yyyy1, mm1, dd1) - datetoday(yyyy2, mm2, dd2);
    };
```

3. Обработка ошибок:

в стандартной библиотеке определено несколько стандартных типов исключений (в файле заголовка <stdexcept>).