Оглавление

[Для чего нужен механизм исключений? 1](#_Toc127433853)

[Типы исключений. 1](#_Toc127433854)

[Знакомство с механизмом исключений. Программа №1. 2](#_Toc127433855)

[Дополнительные материалы по данной теме: 7](#_Toc127433856)

# Для чего нужен механизм исключений?

В C++ различают ошибки **времени компиляции** и ошибки **времени выполнения**.

* Ошибки первого типа обнаруживает компилятор **до** запуска программы.   
  К ним относятся, например, синтаксические ошибки в коде.
* Ошибки второго типа проявляются **при** запуске программы.   
  Примеры ошибок времени выполнения: ввод некорректных данных, некорректная работа с памятью, недостаток места на диске и т. д. Часто такие ошибки могут привести к неопределённому поведению программы.

Некоторые ошибки времени выполнения можно обнаружить **заранее** с помощью проверок в коде. Например, такими могут быть ошибки, нарушающие инвариант класса в конструкторе. ***Обычно, если ошибка обнаружена, то дальнейшее выполнение функции не имеет смысла, и нужно сообщить об ошибке в то место кода, откуда эта функция была вызвана.*** Для этого предназначен **механизм исключений**.

Исключения предпочтительнее использовать в современном C++ по следующим причинам:

* Исключение заставляет вызывающий код распознавать условие ошибки и обрабатывать его. Необработанное исключение останавливает выполнение программы.
* Исключение переходит к точке стека вызовов, которая может обрабатывать ошибку. Промежуточные функции могут позволить распространению исключения. Они не должны координироваться с другими слоями.
* Механизм очистки стека исключений уничтожает все объекты в области после создания исключения в соответствии с четко определенными правилами.
* Исключение позволяет четко разделить код, который обнаруживает ошибку и код, обрабатывающий ошибку.

# Типы исключений.

Кроме типа exception в C++ есть еще несколько производных типов исключений, которые могут использоваться при различных ситуациях. Основные из них:

**runtime\_error**: общий тип исключений, которые возникают во время выполнения

**range\_error**: исключение, которое возникает, когда полученный результат превосходит допустимый диапазон

**overflow\_error**: исключение, которое возникает, если полученный результат превышает допустимый диапазон

**underflow\_error**: исключение, которое возникает, если полученный в вычислениях результат имеет недопустимые отрицательное значение (выход за нижнюю допустимую границу значений)

**logic\_error**: исключение, которое возникает при наличии логических ошибок к коде программы

**domain\_error**: исключение, которое возникает, если для некоторого значения, передаваемого в функцию, не определено результата

**invalid\_argument**: исключение, которое возникает при передаче в функцию некорректного аргумента

**length\_error**: исключение, которое возникает при попытке создать объект большего размера, чем допустим для данного типа

**out\_of\_range**: исключение, которое возникает при попытке доступа к элементам вне допустимого диапазона

# Знакомство с механизмом исключений. Программа №1.

#include <iostream>

double divide(int, int);

int main()

{

int x = 500;

int y = 0;

double z = divide(x, y);

std::cout << z << std::endl;

std::cout << "The End..." << std::endl;

return 0;

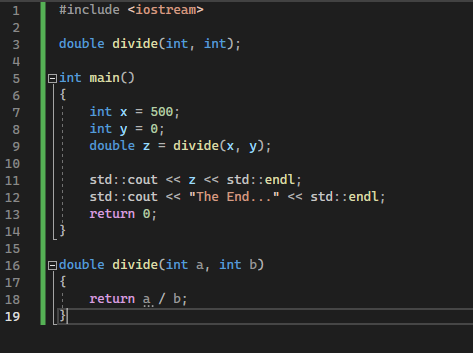
}

double divide(int a, int b)

{

return a / b;

}



Эта программа успешно скомпилируется, но при ее выполнении возникнет ошибка, поскольку в коде производится деление на ноль, после чего программа **аварийно завершится**.

С одной стороны, мы можем в функции divide определить проверку и выполнять деление, если параметр b не равен 0. Однако нам в любом случае надо возвращать из функции divide некоторый результат - некоторое число. То есть мы не можем просто написать:

#include <iostream>

double divide(int a, int b)

{

if (b != 0)

return a / b;

else

std::cout << "Error! b must not be equal to 0" << std::endl;

}

int main()

{

int x = 500;

int y = 0;

double z = divide(x, y);

std::cout << z << std::endl;

std::cout << "The End..." << std::endl;

return 0;

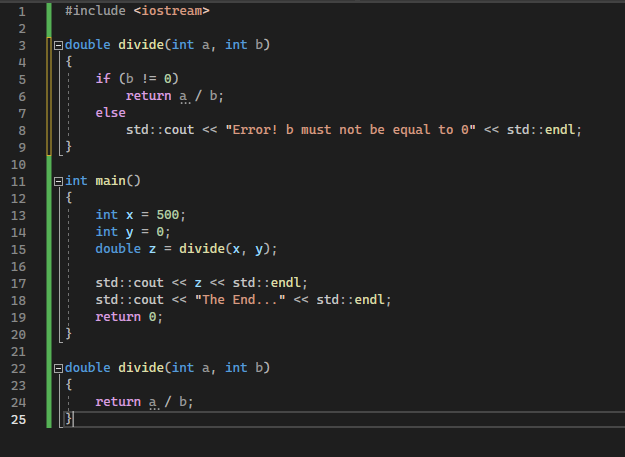
}

double divide(int a, int b)

{

return a / b;

}



И в этом случае нам надо известить систему о возникшей ошибке. Для этого используется оператор **throw**.

Оператор **throw** генерирует исключение.   
***Через оператор throw можно передать информацию об ошибке.***   
Например, функция divide могла бы выглядеть следующим образом:

#include <iostream>

double divide(int a, int b)

{

if (b == 0)

throw "Division by zero!";

return a / b;

}

int main()

{

int x = 500;

int y = 0;

double z = divide(x, y);

std::cout << z << std::endl;

std::cout << "The End..." << std::endl;

return 0;

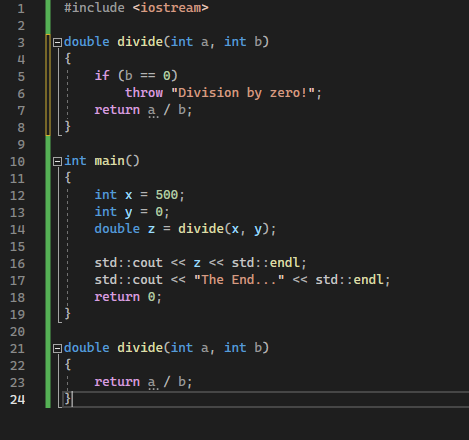
}

double divide(int a, int b)

{

return a / b;

}



То есть если параметр b равен 0, то генерируем исключение.

Но это **исключение** еще надо обработать в коде, где будет вызываться функция divide. ***Для обработки исключений применяется конструкция try...catch***. В блок после ключевого слова **try** помещается код, который потенциально может сгенерировать исключение.

После ключевого слова **catch** в скобках идет параметр, который передает информацию об исключении. Затем в блоке производится собственно обработка исключения.

#include <iostream>

double divide(int, int);

int main()

{

int x = 500;

int y = 0;

try

{

double z = divide(x, y);

std::cout << z << std::endl;

}

catch (...)

{

std::cout << "Error!" << std::endl;

}

std::cout << "The End..." << std::endl;

return 0;

}

double divide(int a, int b)

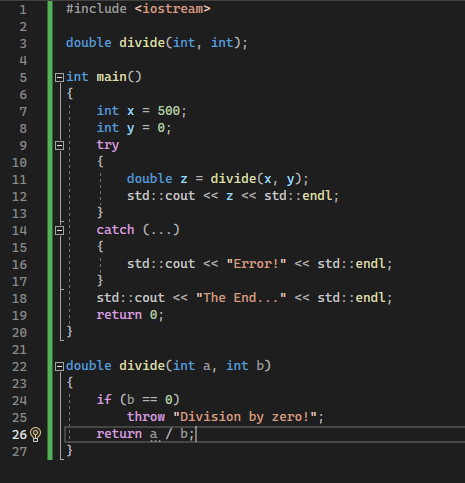
{

if (b == 0)

throw "Division by zero!";

return a / b;

}



Код, который потенциально может сгенерировать исключение - вызов функции divide помещается в блок try.

В блоке catch идет обработка исключения. Причем многоточие в скобках после оператора catch (catch(...)) позволяет обработать любое исключение.

В итоге когда выполнение программы дойдет до строки double z = divide(x, y);, будет сгенерировано исключение, поэтому последующие инструкции из блока try выполняться не будут, а управление перейдет в блок catch, в котором на консоль просто выводится сообщение об ошибке. После выполнения блока catch программа аварийно не завершится, а продолжит свою работу, выполняя операторы после блока catch

!!Однако в данном случае мы только знаем, что произошла какая-то ошибка, а какая именно, неизвестно. Поэтому в выражении catch мы можем получить то сообщение, которое передается оператору throw:

#include <iostream>

double divide(int, int);

int main()

{

int x = 500;

int y = 0;

try

{

double z = divide(x, y);

std::cout << z << std::endl;

}

catch (const char\* msg)

{

std::cout << msg << std::endl;

}

std::cout << "The End..." << std::endl;

return 0;

}

double divide(int a, int b)

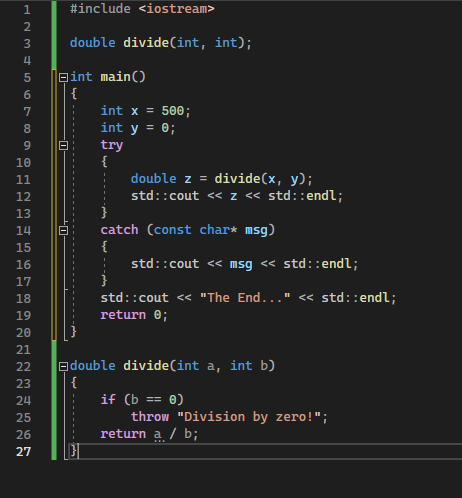
{

if (b == 0)

throw "Division by zero!";

return a / b;

}



С помощью параметра const char\* msg получаем сообщение, которое предано оператору throw, и выводит это сообщение на консоль.

Таким образом, мы можем узнать суть возникшего исключения.

# Дополнительные материалы по данной теме:

<http://www.c-cpp.ru/books/obrabotka-isklyucheniy>

<https://ravesli.com/urok-182-obrabotka-isklyuchenij/>

<https://unetway.com/tutorial/c-obrabotka-isklucenij>