top КОЛЛЕДЖ

Часть 1. Исключения

Когда кода возврата недостаточно

Обработка ошибок

top КОЛЛЕДЖ

- Давайте представим, что некоторые функции выполняют какую-то важную работу, и нам для контроля исполнения кода обязательно нужно проверить, не выполнилась ли некоторая функция с ошибкой.
- Например, мы пытаемся прочитать набор символов и интерпретировать его исключительно как число — при наличии в наборе символов букв или знаков препинания число сформировать не получится.

```
// некоторая функция
int strtod();

int main()
{
	char buf[128];
	std::cin >> buf;
	double a = strtod(buf);
	// как проверить
	// корректность значения
	// в переменной a?
	...
}
```

Обработка ошибок

top КОЛЛЕДЖ

- Рассмотрим обычные возможности языков программирования, которые использовались, например, в языке С. Разберем каждую по порядку.

Коды возврата

top КОЛЛЕДЖ

- Один из наиболее распространенных способов — это использование ***кодов возврата*** — специальных целочисленных значений, имеющих определенную семантику.
- Такой способ применяется в функции main — при выходе из главной функции код возврата попадает в операционную систему, где интерпретируется определенным образом (0 — корректное завершение работы, не 0 — возникновение ошибки).

```
int main()
{
	...
	if (error)
	{
		// ошибка
		return 1;
	}
	...
	return 0; // успешно
}
```

Коды возврата

top КОЛЛЕДЖ

Плюсы

- Нет влияния на производительность
- Кодов возврата достаточно (2^{32}) для любой задачи
- Легко определить факт ошибки

Минусы

- Нетипичная сигнатура функций, требующая возвращаемое значение передавать аргументом в виде указателя
- Сами по себе не несут никакой информации кроме наличия или отсутствия факта ошибки
- Требуют множества логических ветвлений

Пример

top КОЛЛЕДЖ

- Создадим функцию, вычисляющую обратное значение числа. Если в качестве параметра подается ноль, то функция возвращает код ошибки.

```
int main()
{
    double value;
    std::cin >> value;

    int code;
    double result;
    code = reciprocal(value, &result);
    if (code)
    {
        // Если code ненулевой, значит функция завершилась с ошибкой
        std::cerr << "Error in function reciprocal!" << std::endl;
        return code;
    }

    std::cout << "Value 1/" << value << " = " << result << std::endl;

    return 0;
}
```

```
int reciprocal(double value, double* result)
{
    if (value == 0) return 1;
    *result = 1.0 / value;
    return 0;
}
```

```
10
Value 1/10 = 0.1
```

```
0
Error in function reciprocal!
```

Коды возврата

top КОЛЛЕДЖ

- Коды возврата хороши тем, что не требуют большой производительности от системы. Наиболее часто их используют в различных процедурах (функции, которые ничего не возвращают) для контроля исполнения программы.
- В то же время, коды возврата нарушают читаемость кода из-за непривычных объявлений функций.
- Тем не менее, коды возврата до сих пор часто используются в встраиваемых системах за счет легкости. Однако в больших и сложных проектах их использовать не рекомендуется.

Вопрос

top КОЛЛЕДЖ

Какое решение можно предложить для сохранения привычной сигнатуры функции без использования кода возврата?

Флаги

top КОЛЛЕДЖ

- Часто флаги используют для фиксации нескольких состояний работы функции, однако их можно использовать и для сигнализировании об ошибках.
- Флаги часто используют в межпроцессном взаимодействии и сложном API.
- Решение с ***флагами*** довольно банальное — следует просто поменять код возврата с реальным возвращаемым результатом так, чтобы реальное значение действительно возвращалось из функции, а код был ее аргументом.

```
double reciprocal(double value, int* flag = 0)
{
    if (flag && value == 0)
    {
        // Используем побитовые операции
        *flag |= 1;
        return NAN;
    }

    return 1.0 / value;
}
```

```
int main()
{
    double value;
    std::cin >> value;

    int flag = 0;
    double result = reciprocal(value, &flag);
    if (flag)
    {
        // Если есть хотя бы один ненулевой бит, произошла ошибка
        std::cerr << "Error in function reciprocal!" << std::endl;
        return flag;
    }

    std::cout << "Value 1/" << value << " = " << result << std::endl;

    return 0;
}
```

Флаги

top КОЛЛЕДЖ

Плюсы

- Сигнатура функции дополняется только одним необязательным параметром
- Функционал тот же, что и у кодов возврата.

Минусы

- При использовании флагов возвращаемое значение в случае ошибки должно быть так или иначе чем-то заполнено (даже если использовать указатель, то нужно его заранее обнулить). Без флагов об ошибке можно судить по данному значению, но иногда это же значение получается в результате нормальной работы функции.

Глобальная переменная

top КОЛЛЕДЖ

- Полное решение проблем с сигнатурой может быть достигнуто путем использования ***глобальной переменной***. По сути, это будет код возврата, являющийся общим для любых функций.
- В WinAPI и стандартной библиотеки C в качестве такой переменной используется определение препроцессора `errno`.
- Данное определение можно использовать как переменную — то есть в нее при желании можно записать собственный код ошибки, представляющий собой произвольное целое число типа `int`.

Глобальная переменная

top КОЛЛЕДЖ

Плюсы

- Сохраняется функционал кодов возврата и флагов.
- Сохраняется привычная сигнатура функций.
- Обработка ошибок выполняется с общей для любой функции переменной.

Минусы

- В случае разделяемости переменной множеством разных функций может быть потеряна уникальность кода возврата, что ведет к неправильной интерпретации ошибки.
- Для каждой функции используется одна и та же глобальная переменная, что затрудняет ее использование в случае многопоточного приложения.

Пример

top КОЛЛЕДЖ

```cpp
int main()
{
    double value;
    std::cin >> value;

    double result = reciprocal(value);
    if (errno)
    {
        // Если глобальная переменная ненулевая, то произошла ошибка
        std::cerr << "Error in function reciprocal! (code " << errno << ")" << std::endl;
        return errno;
    }

    std::cout << "Value 1/" << value << " = " << result << std::endl;

    return 0;
}
```

```cpp
double reciprocal(double value)
{
    if (value == 0)
    {
        // Используем глобальную переменную
        errno = 1;
        return NAN;
    }

    return 1.0 / value;
}
```

```
10
Value 1/10 = 0.1
```

```
0
Error in function reciprocal! (code 1)
```

Результаты и решение

top КОЛЛЕДЖ

- У каждого способа есть свои недостатки. Однако в языке C++ (как и во многих других языках высокого уровня) появился способ обработки ошибок, устраняющий недостатки любых способов практически полностью — ***исключения***.

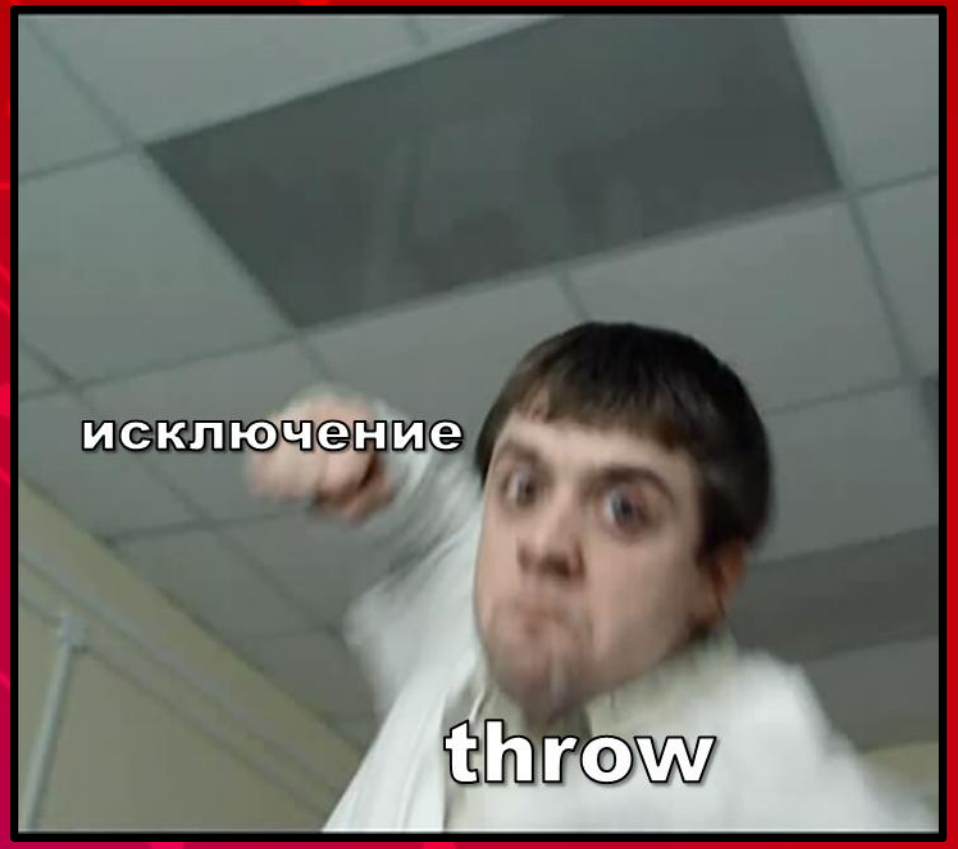
Примечание: до этого всегда использовалось слово «ошибка», однако это неправильно.

- ***Ошибка*** — это ситуация, при которой выполнение программы приводит к непредвиденному и необрабатываемому результату.
- ***Исключение*** — это ситуация нормальной работы программы, требующая обработки.

Механизм исключений

top КОЛЛЕДЖ

- Для организации работы исключений используется две взаимосвязанные конструкции:
  - Инструкция throw обеспечивает «***выбрасывание***» исключения,
  - Блок try-catch обеспечивает «***отлавливание***» исключений.
- В качестве исключения может быть любое значение произвольного типа данных.



Использование throw

top КОЛЛЕДЖ

- Инструкция throw организует выбрасывание исключения. После указания ключевого слова должно следовать значение (в том числе выражение):

```
throw 1;
throw std::string(“Hello”);
throw sqrt(2.0) / 2;
```

- Это значение интерпретируется как некоторый объект, который необходимо вернуть вверх из функции по стеку вызовов вместо возвращаемого значения. Однако, так как объект необязательно представляет тот же самый тип, что и возвращаемое значение, то исключение необходимо где-то перехватить с нужным типом.

Использование try-catch

top КОЛЛЕДЖ

- Для перехватывания исключений используется блок `try-catch`.
- Он состоит как минимум из двух блоков, одним из которых обязательно должен быть единственный `try` — данный блок включает код, который потенциально может содержать исключения.

```
try
{
   // здесь могут быть отловлены исключения
   ...
}
```

Использование try-catch

top КОЛЛЕДЖ

- После блока `try` должен быть как минимум один блок `catch` — данный блок содержит объявление переменной исключения с типом и код, выполняемый при перехвате исключения данного типа.

```
catch ({тип} {имя_переменной})
{
    // здесь происходит перехват
    // исключений типа {тип}.
    ...
}
...
```

Использование try-catch

top КОЛЛЕДЖ

- Так как блок `catch` перехватывает исключение определенного типа, то поля и методы этого типа можно использовать внутри блока для получения дополнительной информации.
- В примере справа в качестве исключения используется тип строки `std::string`.

```
double reciprocal(double value)
{
    if (value == 0)
    {
        // выбрасываем исключение типа std::string
        throw std::string("Value cannot be zero!");
    }

    return 1.0 / value;
}

int main()
{
    double value;
    std::cin >> value;

    try
    {
        double result = reciprocal(value);
        std::cout << "Value 1/" << value << " = " << result << std::endl;
    }
    catch (std::string str)
    {
        // Используем переменную str из исключения
        std::cerr << "Error in function reciprocal: " << str << std::endl;
        return 1;
    }

    return 0;
}
```

```
0
Error in function reciprocal: Value cannot be zero!
```

Использование try-catch

top КОЛЛЕДЖ

- Обратите внимание, при использовании исключений ключевое слово `return` в функции не используется, вместо него достаточно указать `throw`, так как это гарантированно приведет к выходу из функции.
- Переменная в блоке `try`, выделенная под результат работы функции, также не используется.

Несколько блоков catch

top КОЛЛЕДЖ

- Блоков catch может быть несколько, идущих друг за другом.
- У них должен быть обязательно разный тип, так как обработка исключений сильно зависит от типов.
- Располагать их в таком случае следует в порядке от наибольшей информативности к наименьшей (например, от произвольного типа вплоть до примитивного).

```
try
{
        ...
}
catch (std::string e)
{
        ...
}
catch (int e)
{
        ...
}
```

catch с произвольным исключением

top КОЛЛЕДЖ

- У блока catch есть также специальная сигнатура, позволяющая отловить любое исключение, происходящая в блоке `try`. Для этого вместо переменной используется многоточие:

```
catch (...) // любое исключение
{
    ...
}
```

- Заметьте, что в таком случае можно получить только информацию о факте ошибки, но не о ее происхождении, так как ни переменная, ни ее тип не будут неизвестны.

Важное замечание

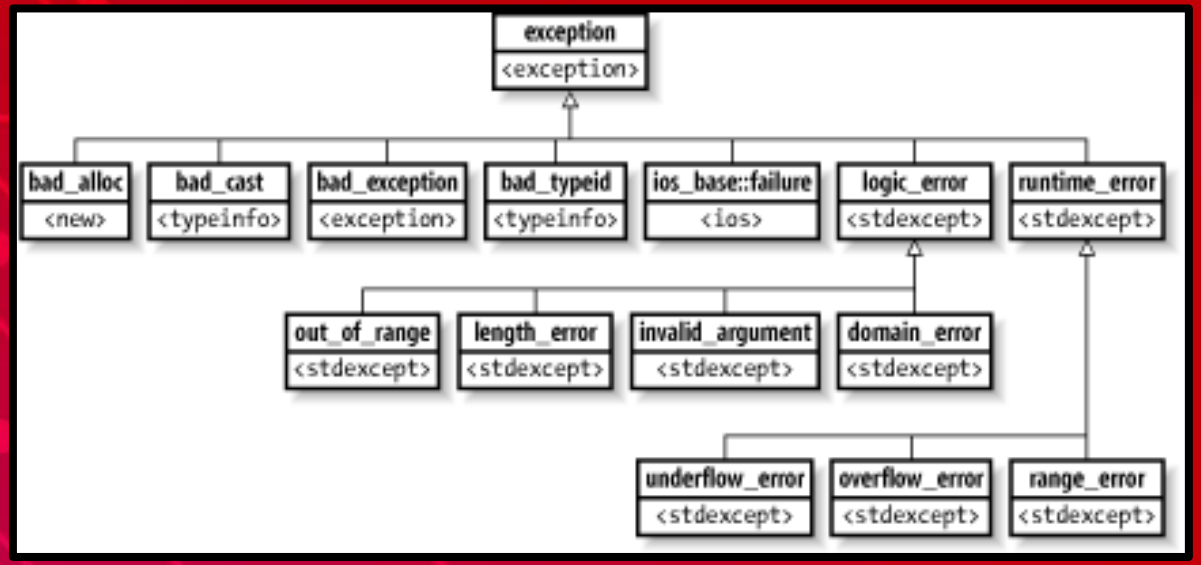
top КОЛЛЕДЖ

- Важно, чтобы исключение обязательно было поймано где-либо в коде. Иначе это приведет к попаданию исключения в код операционной системы, где она попытается его обработать самостоятельно.
- В большинстве случаев попытки обработать исключение операционной системой приводят к завершению работы программы, однако не во всех случаях — часто в графических приложениях исключения могут быть пропущены, а операционная система только лишь создаст запись в системном журнале.

Типы исключений

top КОЛЛЕДЖ

- Так как исключения позволяют передавать любые типы вверх по стеку вызовов, то в качестве унификации принято использовать специальные типы исключений. В C++ такие типы определены в заголовочном файле <stdexcept>:
  - std::invalid_argument
  - std::out_of_range
  - std::range_error
  - и некоторые другие



Снова несколько блоков catch

top КОЛЛЕДЖ

- Необходимо учитывать «приоритетность» тех или иных исключений, чтобы гарантировать верную работу программы.
- Например, `std::invalid_argument` должен идти раньше `std::logic_error`, потому что неверный аргумент логически является частным случаем общей логической ошибки.
- *(А еще потому что наследование, но об этом на другой дисциплине ☺)*

```
try
{
        ...
}
catch (std::invalid_argument e)
{
        ...
}
catch (std::logic_error e)
{
        ...
}
```

Важное замечание 2

top КОЛЛЕДЖ

- Так как при возврате из функций часто передается копия, то в случае исключений (как «специального» возвращаемого значения) множественное копирование при подъеме вверх по стеку вызовов занимает продолжительное время и требует относительно много ресурсов, особенно, когда используются структуры.
- Поэтому лучше принимать исключения в блоке `catch` по ссылке — это исключает возможность копирования данных.

```
catch (some_type& ex) { ... }
```

Функции и методы noexcept

top КОЛЛЕДЖ

- Механизм исключений является достаточно требовательным к ресурсам, при его использовании выполняется достаточно много операций по развертке стека и передач управления кодом. Поэтому, компилятор оказывается сильно ограничен в оптимизации фрагментов кода, в которых могут возникать исключения.
- Чтобы помочь компилятору использовать оптимизации, можно явно пометить функции или методы как те, которые не могут выбрасывать исключения. Это выполняется с помощью ключевого слова noexcept.

```
void foo(int a, double b) noexcept { … }
```

Пример noexcept-функции

top КОЛЛЕДЖ

```cpp
unsigned str_to_uint_noexcept(std::string& str) noexcept
{
    // Здесь гарантированно никогда
    // не будет выброшено исключение
    unsigned result = 0;
    for (size_t i = 0; i < str.size(); ++i)
    {
        if (str[i] < '0' || str[i] > '9')
        {
            return result;
        }
        else
        {
            result = result * 10 + (str[i] - '0');
        }
    }

    return result;
}
```

```cpp
int main()
{
    std::string str;
    unsigned u;

    str = "1578";
    u = str_to_uint_noexcept(&: str);

    std::cout << u / 2 << std::endl;

    str = "21 hello!";
    u = str_to_uint_noexcept(&: str);

    std::cout << u << std::endl;

    return 0;
}
```

```
789
21
```

Мини-задание

top КОЛЛЕДЖ

- Напишите свой собственный тип исключения, используя структуры. Реализуемый тип должен иметь поле, хранящее строку описания исключения.
- Покажите, как ваше исключение можно использовать в коде.
* Можете данное задание выполнить на основе имеющегося у вас кода с различных уроков по дисциплине, где было важно обрабатывать ошибки (математические функции или структуры данных).