C++

Урок 30

Обработка исключений 2

Обработка исключений

Исключение — это условие ошибки, возможно вне элемента управления программы, которое не позволяет продолжать выполнение программы по обычному пути выполнения.

Операции, включая создание объектов, входные и выходные данные файлов и вызовы функций из других модулей, являются потенциальными источниками исключений, даже если программа работает правильно. В надежном коде можно **предвидеть** и **обработать** исключения.

Обработка исключений

Механизм: чтобы поймать исключение, его нужно где-то (внутри ф-ии) бросать (**throw**), затем проверять эту ф-ию внутри **try** и ,если возникнет исключение, то отреагировать на него в **catch**.

Try позволяет определить блок кода, который будет **проверяться** на **наличие ошибок** во время его выполнения;

Throw нужен для создания и **отображения исключений** и используется для перечисления ошибок, которые генерирует функция, но не может самостоятельно обрабатывать исключения;

Catch - блок кода, который выполняется при **возникновении** определенного **исключения** в блоке try

Обработка исключений

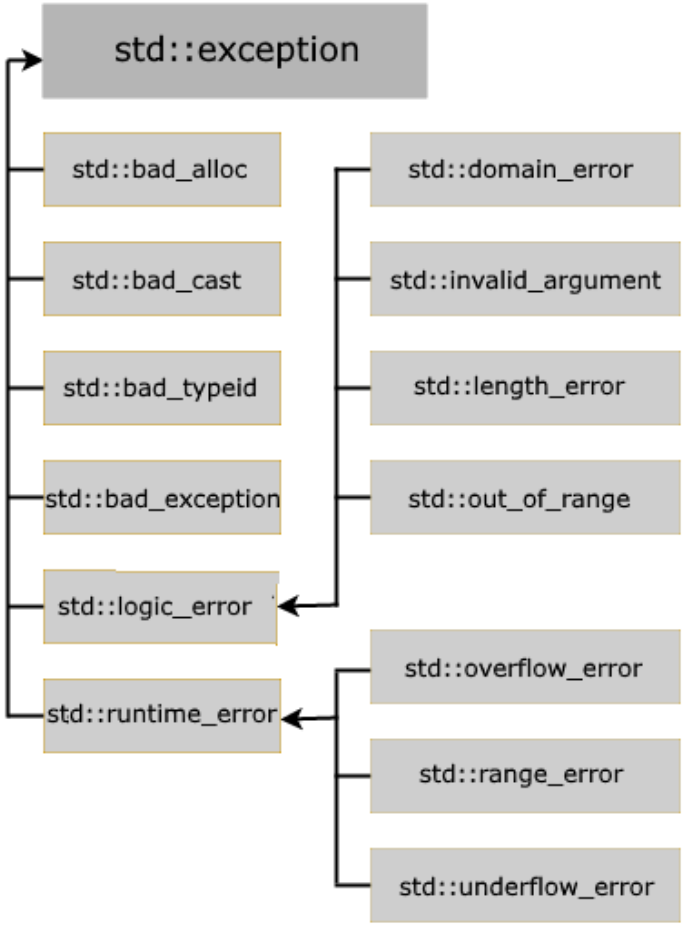
```
try
{
    // проверяем (генерируем) исключительную ситуацию
}
catch (const std::exception &e) // передача информации из try
{
    // ловим и проводим обработку
    std::cout << e.what() << std::endl;
}
```

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

double sqrt_val(const int &);

int main()
{
    int val = -1;
    try
    {
        double rez = sqrt_val(val);
        std::cout << rez << std::endl;
    }
    catch (const char *msg)
    {
        std::cout << "Ошибка: " << msg << std::endl;
    }
    std::cout << "Завершение работы" << std::endl;
    return 0;
}
double sqrt_val(const int &test)
{
    if (test < 0)
        throw "Корень из отриц числа!";
    return sqrt(test);
}
```

Обработали исключение типа ”строка” при отрицательном числе

Обработка исключений

Типы исключений



Обработка исключений

Типы исключений

- **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;

```cpp
#include <iostream>
#include <cmath>
#include <exception>
#include <stdexcept>

double sqrt_val(const int &);

int main()
{
    int val = -1;
    try
    {
        double rez = sqrt_val(val);
        std::cout << rez << std::endl;
    }

    catch (std::overflow_error err)
    {
        std::cout << "Overflow_error: " << err.what() << std::endl;
    }
    catch (std::runtime_error err)
    {
        std::cout << "Runtime_error: " << err.what() << std::endl;
    }
    catch (std::exception err)
    {
        std::cout << "Exception!!!" << std::endl;
    }

    std::cout << "Завершение работы" << std::endl;
    return 0;
}
double sqrt_val(const int &test)
{
    if (test < 0)
        throw std::runtime_error("value <0! ");
    return sqrt(test);
}
```

Обработали под тип исключения

RAII

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

using namespace std;

// C style

int main()
{
    FILE *f = fopen("30_01.txt", "w");

    if (f != nullptr)
    {
        cout << "LOCATETD";
        fputs("Some info for file", f);

        //....

        fclose(f);
    }
    else
        cout << "File not found" << endl;
}
```

Открытие и чтение в Си стиле

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

using namespace std;

// C style

class File
{
    FILE *f;

public:
    File(const char *str)
    {
        f = fopen(str, "r+w");
        if (f == nullptr)
        {
            throw("file not generated");
        }
    }

    void write_info(const char *str)
    {
        fputs(str, f);
    }
    ~File()
    {
        fclose(f);
        cout << "worked destr" << endl;
    }
};

int main()
{
    try
    {
        File file("30_02.txt");
        file.write_info("hello world!");
    }
    catch (const char *err)
    {
        cout << err << endl;
    }
    catch (...) // исключение всех типов
    {
        //
    }

    cout << "\nEND of program" << endl;
}
```

Обернем код в класс + расширим его

```cpp
int main()
{
    srand(time(0));
    try
    {
        File file("30_02.txt");
        File f2 = file;

        file.write_info("hello world!");
    }
    catch (const char *err)
    {
        cout << err << endl;
    }
    catch (...) // исключение всех типов
    {
        //
    }

    cout << "\nEND of program" << endl;
}
```

Что делать в таком случае?

```
File(const File &test)
{
    this->f = fopen("new_file.txt", "w");
    if (f == nullptr)
    {
        throw("new file not generated");
    }
}
```

Можно так, но ....

```
File(const File &test) = delete;           // запрет на использование конструктора копирования
File operator=(const File &test) = delete; // запрет на присваивание
```

Лучше полностью запретить копирование и присваивание

RAII

Идиома RAII (resource equitization is initialization).

Идея: нужно запрашивать ресурс в конструкторе некоторого объекта, а освобождать — в деструкторе. На этой идее построены стандартные контейнеры ,lock_guard и ”умные указатели”.

RAII

В конструкторе соответствующего класса организуется **получение доступа к ресурсу**, а в деструкторе — **освобождение этого ресурса**.

Поскольку деструктор локальной (автоматической) переменной вызывается при выходе её из области видимости, то ресурс гарантированно **освобождается** при **уничтожении переменной**. Это справедливо даже в ситуациях, в которых возникают исключения.

RAII - ключевая концепция для написания безопасного кода в C++ (и других языках программирования, где конструкторы и деструкторы автоматических объектов вызываются автоматически).

Следует **контролировать** возможность создания объекта через операцию клонирования и корректно переопределить (или запретить) операцию присваивания для подобных объектов.

Задача

Задача: напишите собственный класс Vector или String с учетом идиомы Raii.