МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №6b

по дисциплине: Основы программирования
тема: «Исключения»

Выполнил: ст. группы ПВ-211
Чувилко Илья Романович

Проверили:
Притчин Иван Сергеевич
Черников Сергей Викторович

Белгород 2022 г.

Цель работы: получение навыков работы с исключениями, осознание необходимости данных языковых средств.

Содержание отчета:

- Тема лабораторной работы.
- Цель лабораторной работы.
- Тексты заданий с набранными фрагментами и ответами на необходимые вопросы. Даже если пункт сделан в ознакомительных целях, в процессе набора вы акцентируетесь на нужных моментах.

Задания к лабораторной работе:

- Рассмотрим всё же плохой сценарий обработки (вывод сообщения). Самостоятельно наберите пример ниже и опишите его поведение при b равном и неравном нулю

```cpp
#include <iostream>

int division(int a, int b) {
  if (b != 0)
    return a / b;
  else
    std::cerr << "b must not be equal to 0" << std::endl;
}

int main() {
  int a, b;
  std::cin >> a >> b;

  auto res = division(a, b);
  std::cout << res;

  return 0;
}
```

Какие недостатки вы можете выделить?

Ответ:

- При вводе корректных данных, программа безошибочно выдает результат деления:

```
"D:\BGTU\Programming Basics\6b\cmake-build-debug\6b.exe"
10 5
2
Process finished with exit code 0
```

- При вводе b равного нулю, функция начинает себя некорректно вести. Результат становится непредсказуем, и как следствие, вся программа работает с неправильным результатом деления. Это связанно с тем, что функция не прекращает выполнение всей программы и в случае, когда b = 0 функция не возвращает никаких логичных значений.

```
"D:\BGTU\Programming Basics\6b\cmake-build-debug\6b.exe"
10 0
1875946112b must not be equal to 0

Process finished with exit code 0
```

- Каким образом функция main из прошлого примера поймёт, что что-то в функции division пошло не так? Да, вы можете сказать, что это было известно ещё до вызова функции. Однако заранее знать ситуации, при которых могут возникнуть исключительные ситуации не всегда возможно. Функция division хотела бы как-нибудь информировать main о том, что произошла проблема. Она могла бы делать это при помощи возвращаемого значения

```cpp
#include <iostream>

bool division(int a, int b, int &res) {
  if (b != 0) {
    res = a / b;
    return true;
  } else {
    return false;
  }
}

int main() {
  int a, b;
  std::cin >> a >> b;

  int res;
  if (division(a, b, res)) {
    std::cout << res;
  } else {
    std::cerr << "b must not be equal to 0" << std::endl;
  }

  return 0;
}
```

Ответ: Если значение переменной res не будет использоваться в дальнейшем, то такое решение, хоть и не идеально, но на работоспособность программы не влияет. Если же значение переменной будет использоваться в программе после вывода ошибки, то результатом такой программы станет некорректный результат. Так как переменная res будет указывать на мусор.

- Для сигнализирования исключительной ситуации в языке программирования C++ используется ключевое слово throw

```cpp
int division(int a, int b) {
  if (b == 0)
    throw std::string(" Division by zero ");
  return a / b;
}
```

```
"D:\BGTU\Programming Basics\6b\cmake-build-debug\6b.exe"
10 0
terminate called after throwing an instance of 'std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<
char> >'

Process finished with exit code 3
```

```cpp
#include <iostream>

int division(int a, int b) {
  if (b == 0)
    // выкидываем строку, по которой можно будет понять
    // и при необходимости устранить проблему
    throw std::string(" Division by zero ");

  return a / b;
}

int main() {
  int a, b;
  std::cin >> a >> b;

  try {
    // блок кода, в котором потенциально могут возникнуть исключения
    auto res = division(a, b);
    std::cout << res;
  } catch (const std::string &s) {  // если ошибка произойдёт
                                    // перехваченное значение
                                    // запишется в s
                                    // переменной s можно как-то оперировать
    std::cerr << s;
  }

  return 0;
}
```

Сообщение выведенное в консоль:

```
"D:\BGTU\Programming Basics\6b\cmake-build-debug\6b.exe"
10 0
 Division by zero
Process finished with exit code 0
```

```cpp
int main() {
  try {
    throw 42;
                                // код ниже не выполнится
                                // если код выше возбудил исключение
    std::cout << "В дверь постучали 256 раз. \'char\' - подумал Штирлиц";
  } catch (int a) {
    std::cerr << a;
  }

  return 0;
}
```

```
"D:\BGTU\Programming Basics\6b\cmake-build-debug\6b.exe"
42
Process finished with exit code 0
```

Ответ: В дверь не постучали. Штирлиц ни о чем не подумал. Анекдота не случилось.

- На основании описания поведения абзацем выше объясните, каким образом работает данный фрагмент

```cpp
#include <iostream>

void f1() {
  throw std::string(" ERROR !!! ");
}

void f2() {
  try {
    f1();
  } catch (int a) {
    std::cerr << " CATCH IN F2";
    std::cerr << "(" << a << ")";
  }
}


int main() {
  try {
    f2();
  } catch (std::string &s) {
    std::cerr << " CATCH IN MAIN ";
    std::cerr << "(" << s << ")";
  }

  return 0;
}
```

Ответ: Когда `main` вызвал функцию `f2`, а та в свою очередь вызвала `f1`, то `f1` вызвало исключение, которое прекратило выполнение дальнейшего кода, поэтому программа вернулась к первому блоку `catch`

- Напишите фрагмент кода, который выбирает необходимый catch в зависимости от типа исключения (конструкцию с несколькими блоками catch)

```cpp
#include <iostream>

bool isGoodX(int x) {
  bool a = false;
  if (x <= 0)
    throw std::string("x > 0 ");
  else if (x >= 10)
    throw a;
  return true;
}


int main() {
  std::cout << "Input x. (0 < x < 10)\n";
  int x;
  std::cin >> x;

  try {
    isGoodX(x);
  } catch (const std::string &s) {
    std::cout << s;
  } catch (bool a) {
    std::cout << "x < 10";
  }

  return 0;
}
```

- Напишите фрагмент кода в котором выкидывается исключение, но никак не обрабатывается. Опишите наблюдаемое поведение.

```cpp
#include <iostream>

int main() {
  try {
    throw std::out_of_range("1");
  } catch (std::out_of_range &exc) {
    std::cerr << "CATCH IN MAIN ";
    std::cerr << "(" << exc.what() << ")";
  }

  return 0;
}
```

- Можно перехватывать исключения любого типа используя такой синтаксис

```
#include <iostream>

bool isGoodX(int x) {
  bool a = false;
  if (x <= 0)
    throw std::string("x > 0 ");
  else if (x >= 10)
    throw a;
  return true;
}


int main() {
  std::cout << "Input x. (0 < x < 10)\n";
  int x;
  std::cin >> x;

  try {
    isGoodX(x);
  } catch (...) {
    std::cout << "... Error";
  }

  return 0;
}
```

- Оператор `throw` может и не содержать параметров. Он используется тогда, когда возникает необходимость прокинуть исключение дальше. В примере ниже функция `f()` с какой-то вероятностью не может выделить память. Предположим, если память не смогла быть выделена, наша программа должна освободить память под все ранее выделенные объекты, и при этом `main` должен узнать, что произошла ошибка:

```
#include <iostream>
#include <vector>
#include <ctime>

using namespace std;

int *getmem(float successProbability) {
  float r = static_cast <float > ( rand()) / static_cast <float > (
    RAND_MAX );
  // имитируем работу системы:
  // с веростянотью 1 - successProbability выкидывается ошибка
  if (r > successProbability)
    // прокидываем исключение bad_alloc()
    throw bad_alloc();

  return new int[10];
}

// возвращает вектор из nParts фрагментов динамической памяти
// в случае возникновения ошибки освобождает память всем выделенным фрагментам;
// прокидывает исключение
vector<int *> getParts(int nParts, float successProbability) {
  vector<int *> parts;
  for (int i = 0; i < nParts; i++) {
    try {
      parts.push_back(getmem(successProbability));
    } catch (const bad_alloc &e) {
      clog << " free dynamic memory " << '\n';
```

```
    for (auto &part: parts)
        delete[] part;
      throw; // прокидываем исключение дальше
    }
  }
  return parts;
}


int main() {
  srand(time(nullptr));

  try {
    auto parts = getParts(10, 0.95);
    cout << " Success !";
  } catch (const exception &e) {
    // перехватываем исключение типа exception
    // - стандартный класс исключений C++
    // у стандартных исключений C++ имеется метод .what(),
    // позволяющий получить информацию от исключения
    cerr << e.what();
  }

  return 0;
}
```

- Перехватите исключение out_of_range при помощи logic_error. Чтобы исключение сгенерировалось более осмысленно, создайте вектор размера n, и обратитесь к элементу вектора при помощи метода .at(), допуская выход за пределы массива. Напишите код таким образом, чтобы при каких-то значениях исходных данных исключение выкидывалось, а при каких-то – нет

```
#include <iostream>
#include <vector>

int main() {
  size_t n;
  std::cin >> n;
  std::vector<int> arr(n);
  try {
    arr.at(n - 1);
  } catch (std::logic_error &error) {
    std::cerr << error.what();
  }

  return 0;
}
```

Для векторов размера n > 0 исключение не выкидывалось.

Для n = 0:

```
"D:\BGTU\Programming Basics\6b\cmake-build-debug\6b.exe"
0
vector::_M_range_check: __n (which is 18446744073709551615) >= this->size() (which is 0)
Process finished with exit code 0
```

Для n < 0:

```
"D:\BGTU\Programming Basics\6b\cmake-build-debug\6b.exe"
-1
terminate called after throwing an instance of 'std::bad_alloc'
  what():  std::bad_alloc

Process finished with exit code 3
```

- Пусть есть некоторая функция, которая возвращает фрагмент свободной динамической памяти:

```
int *getmem(size_t partSize) {
  return new int[partSize];
}
```

Напишите функцию:

```
vector<int *> getParts(int nParts, size_t partSize)
```

которая по окончанию своей работы должна вернуть вектор указателей на выделенные фрагменты памяти. Если память выделить при помощи `new` не удалось, будет выброшен `bad_alloc`, который должен быть перехвачен в main.

```
#include <iostream>
#include <vector>

int *getmem(size_t partSize) {
  return new int[partSize];
}

std::vector<int *> getParts(int nParts, size_t partSize) {
  std::vector<int *> result(nParts);
  try {
    for (auto &part: result)
      part = getmem(partSize);
  } catch (std::bad_alloc &error) {
    for (auto &part: result)
      free(part);
    throw;
  }
  return result;
}
```

```cpp
int main() {
  int nParts;
  size_t partSize;
  std::cin >> nParts >> partSize;
  try {
    auto result = getParts(nParts, partSize);

    for (auto &part: result)
      free(part);
  } catch (std::bad_alloc &error) {
    std::cerr << error.what();
  } catch (std::length_error &error) {
    if (nParts < 0)
      std::cerr << "nParts < 0 \n";
    std::cerr << error.what();
  }

  return 0;
}
```

Вывод: Получил навыки работы с ошибками и исключениями в C++