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 №20

по дисциплине: Основы программирования
тема: «Потоки. Ссылки»

Выполнил: ст. группы ПВ-223
Пахомов Владислав Андреевич

Проверили:
Притчин Иван Сергеевич
Черников Сергей Викторович

Код-ревьер: ст. группы ПВ-223
Голуцкий Георгий Юрьевич

Белгород 2023 г.

Лабораторная работа № 20

Содержание отчёта:

- Тема лабораторной работы.
- Цель лабораторной работы.
- Тексты заданий с набранными фрагментами кода к каждому из пунктов и ответами на вопросы по ходу выполнения работы.
- Работа над ошибками (код ревью)
- Вывод по работе.

Тема лабораторной работы: Потоки. Ссылки

Цель лабораторной работы: получение навыков работы с потоками, ссылками, управляющими конструкциями; осознание необходимости появления данных языковых средств.

19.20.1 Буферизация в выходных потоках

1. Создайте функцию `infinitivePause`, которая в своём теле будет воспроизводить бесконечный цикл.

```
void infinitivePause() {
    while (true);
}

int main() {
    return 0;
}
```

2. Отправьте в объект cout, связанный со стандартным потоком вывода, произвольное сообщение без использования endl. После чего сделайте вызов `infinitivePause`. Запустите программу. Опишите наблюдаемое поведение.

```
#include <iostream>

void infinitivePause() {
    while (true);
}

int main() {
    std::cout << "Fish attack!";
    infinitivePause();

    return 0;
}
```

В консоли не появляется отправленное в cout сообщение, программа не останавливает своё выполнение.

3. Выполните аналогичные пункту 2 действия, только с использованием endl. Опишите наблюдаемое поведение.

```
#include <iostream>

void infinitivePause() {
    while (true);
}
```

```
int main() {
    std::cout << "Fish attack!" << std::endl;
    infinitivePause();

    return 0;
}
```

В консоли появилось отправленное в cout сообщение, программа не останавливает своё выполнение.

4. Для случая 2, после вывода сообщения вставьте инструкцию ввода какой-нибудь переменной. Запустите приложение. Опишите наблюдаемое поведение.

```
#include <iostream>

void infinitivePause() {
    while (true);
}

int main() {
    std::cout << "Fish attack!";

    int a;
    std::cin >> a;

    infinitivePause();

    return 0;
}
```

В консоли появилось отправленное в cout сообщение, после ввода значения в консоль программа не останавливает своё выполнение.

5. Выполните создание строки, состоящей из 10000 символов ’a’. Осуществите её вывод перед вызовом `infinitivePause`. Опишите наблюдаемое поведение.

```
#include <iostream>

void infinitivePause() {
    while (true);
}

int main() {
    std::string s(10000, 'a');
    std::cout << s;

    infinitivePause();

    return 0;
}
```

В консоли появилось отправленная в cout строка, состоящая из 10000 символов `a`, программа не останавливает своё выполнение.

6. На основании прошлых трёх пунктов выделите 3 случая, когда очищается буфер вывода. Дополнительно добавьте четвертый: буфер очищается по окончанию работы программы.

Буфер вывода очищается в 4 случаях:

- При отправке в cout std::endl

- При ввода значения в консоли (при использовании std::cin)
- При переполнении буфера
- По окончанию работы программы

7. Говорят, что частый сброс буфера способен повлиять на производительность приложения. Проведите эксперимент, доказывающий это. Например, он мог выглядеть так: выполним создание файла, в который будем записывать данные. В одном случае будем сбрасывать буфер, в другом не будем:

```
#include <iostream>
#include <fstream> // для работы с файлами #include <ctime>
// вывод осуществляется в поток, связанный с логами
#define TIME_TEST(testCode, message) { \
    clock_t start_time = clock () ; \
    testCode \
    clock_t end_time = clock () ;\
    clock_t sort_time = end_time - start_time ; \
    std::clog << message << ": "  \
              << (double) sort_time/CLOCKS_PER_SEC << std::endl; \
}

int main() {
    const char *filename = "tmp.txt";
    std::ofstream file(filename); // выполняем создание файла
    TIME_TEST({
                  for (int i = 0; i < 1000000; i++)
                      file << 'a' << std::endl;
              }, "Often buffer reset");
    TIME_TEST({

                  for (int i = 0; i < 1000000; i++)
                      file << 'a' << '\n';
              }, "Buffer opt");
    file.close();           // закрываем файл
    std::remove(filename); // удаляем временный файл
    return 0;
}
```

```
Often buffer reset: 1.724
Buffer opt: 0.057941
```

8. Рассмотрим случай, когда частый сброс буфера вывода из-за чередования ввода / вывода оказывает влияние на производительность. Выполните решение задачи Минимальная OR сумма (1635A) на codeforces и приложите вердикт тестирующей системы.

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

int main() {
    int t;
    std::cin >> t;

    for (int i = 0; i < t; i++) {
        int n;
        std::cin >> n;
```

```
        std::vector<int> a(n);
        for (auto &el : a)
            std::cin >> el;

        for (int j = 0; j < a.size() - 1; j++) {
            for (int k = j + 1; k < a.size(); k++) {
                int sDiff = a[j] & a[k];
                a[j] -= sDiff;
            }
        }

        int sum = 0;
        for (auto &el : a)
            sum += el;

        std::cout << sum << "\n";
    }
}
```

Основное

№	Отправитель	Задача	Язык	Вердикт	Время	Память	Отослано	Протест.		
202750838	Дорешивание: VladOS4052	1635A - 30	GNU C++17	Полное решение	140 мс	164 КБ	2023-04-19 23:17:48	2023-04-19 23:17:48		Сравнить

9. В условиях задачи 4 добавьте

```
std::cin.tie(nullptr);
```

Опишите наблюдаемое поведение.

```
#include <iostream>

void infinitivePause() {
    while (true);
}

int main() {
    std::cin.tie(nullptr);

    std::cout << "Fish attack!";

    int a;
    std::cin >> a;

    infinitivePause();

    return 0;
}
```

Поведение программы не изменилось. В консоли появилось отправленное в cout сообщение, после ввода значения в консоль программа не останавливает своё выполнение.

10.В решение вашей задачи для пункта 8 добавьте строки:

```
std::cin.tie(nullptr);
std::ios_base::sync_with_stdio(false);
```

И снова приложите вердикт тестирующей системы.

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

int main() {
```

```
    std::cin.tie(nullptr);
    std::ios_base::sync_with_stdio(false);

    int t;
    std::cin >> t;

    for (int i = 0; i < t; i++) {
        int n;
        std::cin >> n;

        std::vector<int> a(n);
        for (auto &el : a)
            std::cin >> el;

        for (int j = 0; j < a.size() - 1; j++) {
            for (int k = j + 1; k < a.size(); k++) {
                int sDiff = a[j] & a[k];
                a[j] -= sDiff;
            }
        }

        int sum = 0;
        for (auto &el : a)
            sum += el;

        std::cout << sum << "\n";
    }
}
```

Основное

№	Отправитель	Задача	Язык	Вердикт	Время	Память	Отослано	Протест.		
202751425	Дорешивание: VladOS4052	1635A - 30	GNU C++17	Полное решение	30 мс	148 КБ	2023-04-19 23:25:54	2023-04-19 23:25:54		Сравнить

11. Существуют объекты, связанные с потоками вывода ошибок (cerr) и логов (clog). Приведите код, который может проверить, является ли вывод в них буферизированным. Как вы считаете, почему было принято именно такое решение по буферизации данных потоков? Ответ обоснуйте.

```
#include <iostream>

void infinitivePause() {
    while (true);
}

int main() {
    std::cerr << "Cerr is not buffered!\n";
    std::clog << "Clog is not buffered!\n";
    infinitivePause();

    return 0;
}
```

В консоли появилось отправленное в cerr и clog сообщения, программа не останавливает своё выполнение. cerr и clog не имеют буфера. cerr используется для вывода ошибок, следовательно логично выполнять их вывод в консоль сразу как только о них сообщается в cerr, поэтому cerr буффер не нужен. В clog может выводиться информация о текущем состоянии ПО, следовательно небуферризированный вывод в консоль также необходим.

19.20.2 Некоторые особенности вывода

1. При помощи cout можно выводить в поток вывода значения переменных и адреса:

```
int i = 10;
std::cout << i << ' ' << &i;
```

Создайте переменную s типа const char*, выполните её инициализацию и попробуйте вывести её в поток вывода. Получившееся поведение опишите.

```
#include <iostream>

int main() {
    const char* s = "Batman!";
    std::cout << s;

    return 0;
}
```

Сохранённое в s сообщение вывелось в консоль, программа успешно завершилась.

2. Несмотря на то, что переменная типа const char* и так является адресом, вы не должны были его увидеть. Чтобы увидеть адрес, необходимо такую переменную привести к типу. Однако одним приведением типа вы не отделаетесь, так как мешает . Цепочка преобразований будет такова:

$$const\ char* \rightarrow char* \rightarrow void*$$

```
std::cout << static_cast<void*>(const_cast<char*>(s));
```

```
#include <iostream>

int main() {
    const char* s = "Batman!";
    std::cout << static_cast<void*>(const_cast<char*>(s));

    return 0;
}
```

В консоль вывелся адрес переменной s, программа успешно завершилась.

3. Выведите литералы `true` и `false` в поток вывода. Какие значения были отображены на экране?

```
#include <iostream>

int main() {
    std::cout << true << ' ' << false;

    return 0;
}
```

Вывод в консоли:

```
1 0
```

4. Добавьте манипулятор std::boolalpha. Опишите отображаемый вывод. Проверьте, распространяется ли действие манипулятора на последующие выводы.

```
#include <iostream>

int main() {
    std::cout << std::boolalpha << true << ' ' << false << '\n';
    std::cout << true << ' ' << false << '\n';

    return 0;
}
```

Вывод в консоли:

```
true false
true false
```

Значение true и false вывелось в консоли в строковом представлении “true” и “false” соответственно. Манипулятор std::boolalpha распространился на дальнейший вывод.

5. Чтобы переключиться на вывод логических значений 1 и 0 используйте манипулятор std::noboolalpha.

```
#include <iostream>

int main() {
    std::cout << std::boolalpha << true << ' ' << false << '\n';
    std::cout << true << ' ' << false << '\n';
    std::cout << std::noboolalpha << true << ' ' << false << '\n';
    std::cout << true << ' ' << false << '\n';

    return 0;
}
```

Манипулятор `std::noboolalpha` отменяет форматирование `std::boolalpha` и распространяется на дальнейший вывод.

19.20.3 Ввод

1. Создайте переменную целочисленного типа, но при осуществлении ввода введите строку, состоящую из символов-букв. Опишите наблюдаемое поведение.

```
#include <iostream>

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

    return 0;
}
```

Ничего не происходит, программа успешно завершилась.

2. Переменная, связанная с потоком ввода будет возвращать значение "истина", если поток ввода не находится в состоянии ошибки. Попробуйте запустить пример из прошлого пункта, добавив обработку.

```
#include <iostream>

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

    if (std::cin)
        std::cout << "Success" << std::endl;
    else
        std::cout << "Error" << std::endl;

    return 0;
}
```

В консоль выводится сообщение "Success", если была введена последовательность из цифр, им можем предшествовать whitespace. Дальнейший ввод может содержать любые символы, например при вводе “ 42HelloWorld!” будет выведено сообщение "Success". Иначе – выводится сообщения “Error”. Программа в каждом из случаев успешно завершается.

3. Создайте переменную типа char и попробуйте выполнить ввод пробельного символа. Опишите наблюдаемое поведение.

```
#include <iostream>

int main() {
    char a;
    std::cin >> a;

    return 0;
}
```

Ничего не происходит, программа не завершается.

4. Проделайте то же самое но при считывании осуществляйте ввод через cin.get():

```
#include <iostream>

int main() {
    char c;

    std::cin.get(c);

    std::cout << c;
}
```

Программа успешно считывает пробел и выводит его, программа успешно завершается. cin.get позволяет считать один символ из консоли.

5. Манипуллятор std::noskipws заставит выполнить ввод и пробельного символа.

```
#include <iostream>

int main() {
    char c;
    std::cin >> std::noskipws >> c;
```

```
    std::cout << c;
    std::cin >> std::noskipws >> c;

    std::cout << c;

    return 0;
}
```

Поведение аналогично поведению программы из пункта 4. Манипулятор продолжает своё действие при дальнейшем вводе.

6. Иногда возникает потребность пропустить некоторые символы в потоке ввода. Например, вводится дата в формате "DD.MM.YYYY"(например 19.02.2003). И стоит цель вычленить отдельно день, месяц и год в переменные типа int. Выполните чтение числа до точки и после точки. Например "14.341"должно быть разбито на 14 и 341.

```
#include <iostream>

int main() {
    int firstNumber;
    std::cin >> firstNumber;
    std::cin.ignore(1);

    int secondNumber;
    std::cin >> secondNumber;

    std::cout << firstNumber << '.' << secondNumber;

    return 0;
}
```

Программа корректно распознаёт оба числа, выводит их и успешно завершается.

7. Обсудим строковый ввод. Несмотря на возможность осуществлять ввод как строк в стиле C, так и за счёт класса `std::string` в C++, вам следует остановиться на последнем, так как происходит автоматическое управление памятью.

```
#include <iostream>

int main() {
    std::string s;
    std::cin >> s;

    std::cout << s;

    return 0;
}
```

Программа корректно считывает строку и выводит её в консоль. Программа успешно завершается.

19.20.4 Файловый ввод / вывод

1. Очень часто приходится оперировать работой с файлами – именованной областью данных на носителе информации. В рамках данной лабораторной разберёмся, как осуществляется работа с текстовыми файлами. Создайте вручную (не при помощи кода) файл `input.txt` содержащий несколько строк, по одному числу в каждой.

input.txt

```
4
42
42
42
8
41
12312
314
12421
12
11
```

main.cpp

```
int main() {
    return 0;
}
```

2. Убедитесь, что можете использовать оператор » для считывания строк из файла:

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

int main() {
    // чтение из файла
    std::ifstream inputFile("input.txt");

    std::string s;
    while (inputFile >> s)
        std::cout << s << '\n';

    return 0;
}
```

Программа считывает строки из файла и выводит их в консоль. Программа успешно завершается.

3. Код описанный выше несколько небезопасен. Файл с таким именем мог и не существовать. Попробуйте запустить прошлый фрагмент для файла, который не был создан до этого и опишите наблюдаемое поведение.

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

int main() {
    // чтение из файла
    std::ifstream inputFile("input.txt");

    std::string s;
    while (inputFile >> s)
        std::cout << s << '\n';

    return 0;
}
```

Ничего не происходит, программа успешно завершается.

4. В прошлом примере для проверки на то, открыт ли файл, мог использоваться функция-член .is_open():

```
std::ifstream inputFile("notExists.txt");
if (inputFile.is_open())
    std::cout << "Exists";
else
    std::cout << "Not exists";
```

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

int main() {
    // чтение из файла
    std::ifstream inputFile("input.txt");

    if (inputFile.is_open()) {
        std::cout << "Exists\n";

        std::string s;
        while (inputFile >> s)
            std::cout << s << '\n';
    } else
        std::cout << "Not exists";


    return 0;
}
```

Программа выводит “Non exists”, если файл не существует, иначе – “Exists” и затем выводит содержимое файла. В обоих случаях программа завершается успешно.

однако и сама переменная, ассоциированная с файлом может быть использована для проверки на то, прошла ли операция открытия успешно:

```
std::ifstream inputFile("notExists.txt");
if (inputFile)
    std::cout << "Exists";
else
    std::cout << "Not exists";
```

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

int main() {
    // чтение из файла
    std::ifstream inputFile("input.txt");

    if (inputFile) {
        std::cout << "Exists\n";

        std::string s;
        while (inputFile >> s)
            std::cout << s << '\n';
    } else
        std::cout << "Not exists";
```

```
    return 0;
}
```

Поведение программы аналогично предыдущему примеру

5. В примере выше мы осуществляли ввод в строку, но с таким же успехом вы можете считывать данные в переменные, например, целочисленного типа.

```
int x;
while (inputFile >> x) {
}
```

Имея данные сведения, реализуйте функцию для вычисления суммы чисел, записанных в файл с именем `long long getSum(const std::string &filename)`. Если файла с таким именем нет, функция должна возвращать -1.

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

long long getSum(const std::string &filename) {
    long long sum = 0;
    std::ifstream inputFile(filename);

    if (inputFile) {
        long long s;
        while (inputFile >> s)
            sum += s;

        return sum;
    } else
        return -1;
}

int main() {
    std::cout << getSum("input.txt");

    return 0;
}
```

6. История с возвратом значения -1, если файл не был найден, смотрится несколько нелепо (ответьте для себя, а почему?). В C++ появились другие средства для сигнализирования об ошибках, называемых исключениями. Мы не будем вдаваться в подробности сейчас. Если вы хотите сигнализировать, что возникла некоторая ошибка времени исполнения, можно использовать такой подход:

```
#include <stdexcept>

long long getSum(const std::string &filename) {
    std::ifstream inputFile(filename);

    if (!inputFile)
        throw std::runtime_error("File doesn't exist");

    // работа с файлом; ветку else стоит опустить
}
```

Модифицируйте программу из пункта 5, запустите её для несуществующего файла. Вставьте в отчёт полученное сообщение.

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

long long getSum(const std::string &filename) {
    long long sum = 0;
    std::ifstream inputFile(filename);

    if (!inputFile)
        throw std::runtime_error("File doesn't exist");

    long long s;
    while (inputFile >> s)
        sum += s;

    return sum;
}

int main() {
    std::cout << getSum("input.txt");

    return 0;
}
```

```
/Users/vlad/Desktop/C/programming-and-algorithmization-basics/main
libc++abi: terminating with uncaught exception of type std::runtime_error: File doesn't exist

Process finished with exit code 134 (interrupted by signal 6: SIGABRT)
```

7. Пусть в файле inputFile записаны размеры матрицы, а далее записаны непосредственно элементы матрицы, например:

```
2 4
1 2 3 4
5 3 4 2
```

Функция long long getSumOfMaxesInRows(const std::string &filename) должна выполнять поиск суммы максимальных элементов строк. Для примера выше сумма равняется 9. Реализуйте её.

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

long long getSumOfMaxesInRows(const std::string &filename) {
    long long sum = 0;
    std::ifstream inputFile(filename);

    if (!inputFile)
        throw std::runtime_error("File doesn't exist");

    long long height, width;

    inputFile >> height >> width;

    for (int i = 0; i < height; i++) {
        long long max;
        inputFile >> max;

        for (int j = 1; j < width; j++) {
            long long tmp;
```

```
            inputFile >> tmp;

            if (tmp > max)
                max = tmp;
        }

        sum += max;
    }

    return sum;
}

int main() {
    std::cout << getSumOfMaxesInRows("input.txt");

    return 0;
}
```

19.20.5 sstream

19.20.6 Определение операций ввода / вывода для произвольных типов

1. Опишите произвольную (но отличную от примера) структуру.

```
struct Lab {
    std::string name;
    int id;
};
```

2. Реализуйте функции ввода и вывода структуры:

```
void inputLab(Lab &p) {
    std::cin >> p.id >> p.name;
}

void outputLab(const Lab &p) {
    std::cout << "Id: " << p.id << "\nName: \"" << p.name << "\"\n";
}
```

3. В main создайте переменную данного типа и выполните ввод и вывод структуры на экран:

```
int main() {
    Lab lab;

    inputLab(lab);
    outputLab(lab);

    return 0;
}
```

Такой подход немного неудобен. Ведь когда мы вводим переменные, нам бы хотелось использовать cin, а для вывода – cout. Для этого придётся для нашей структуры определить операцию ввода и вывода:

```
void operator>>(std::istream &in, Lab &p) {
    in >> p.id >> p.name;
}

void operator<<(std::ostream &out, Lab &p) {
    out << "Id: " << p.id << "\nName: \"" << p.name << "\"\n";
}
```

```
int main() {
    Lab lab;

    std::cin >> lab;
    std::cout << lab;

    return 0;
}
```

Переменная in выше имеет тип std::istream&. Функция принимает объект типа std::istream&. Описание функции в таком ключе позволит осуществлять ввод структуры для произвольного объекта, ассоциированных с вводом хоть с клавиатуры, хоть с файла.

4. Создайте файл intput.txt, и выполните чтение структуры и с клавиатуры, и из файла

input.txt

```
1
Алгоритмы разветвляющейся структуры
3
Алгоритмы разветвляющейся структуры
10
Бинарный поиск
```

main.cpp

```
#include <string>
#include <iostream>
#include <fstream>

struct Lab {
    std::string name;
    int id;
};

void operator>>(std::istream &in, Lab &p) {
    in >> p.id >> p.name;
}

void operator<<(std::ostream &out, Lab &p) {
    out << "Id: " << p.id << "\nName: \"" << p.name << "\"\n";
}

int main() {
    Lab p;
// ввод с клавиатуры
    std::cin >> p;
// вывод в консоль
    std::cout << p;

// ввод с файла
    std::ifstream inputFile("input.txt");
// вывод в файл
    std::ofstream outputFile("output.txt");
// ввод с файла

    inputFile >> p;
    outputFile << p;
```

```
    return 0;
}
```

output.txt

```
Id: 1
Name: "Алгоритмы"
```

5. Попробуйте написать цепочку вида:

```
int main() {
    Lab p, q, r, s;
    std::cin >> p >> q >> r >> s;
    std::cout << p << q << r << s;

    return 0;
}
```

Опишите наблюдаемое поведение.

Код не компилируется

6.

```
std::istream& operator>>(std::istream &in, Lab &p) {
    in >> p.id >> p.name;
    return in;
}

std::ostream& operator<<(std::ostream &out, Lab &p) {
    out << "Id: " << p.id << "\nName: \"" << p.name << "\"\n";
    return out;
}
```

```
int main() {
    Lab p, q, r, s;
    std::cin >> p >> q >> r >> s;

    std::cout << p << q << r << s;

    return 0;
}
```

Программа завершается успешно.

19.20.7 Ссылки

1. Реализуйте функцию sort2 с использованием указателей. Введите два значения в main и выполните вызов функции sort2.

```
#include <iostream>

template<typename T>
void swap(T *a, T *b) {
    T tmp = *a;
    *a = *b;
    *b = tmp;
}

template<typename T>
void sort2(T *a, T *b) {
    if (*a > *b)
        swap(a, b);
}

int main() {
```

```
    int a1, a2;
    std::cin >> a1 >> a2;

    sort2(&a1, &a2);

    std::cout << a1 << " " << a2;

    return 0;
}
```

2. Реализуйте функцию sort2 с использованием ссылок. Введите два значения в main и выполните вызов функции sort2

```
#include <iostream>

template<typename T>
void swap(T &a, T &b) {
    T tmp = a;
    a = b;
    b = tmp;
}

template<typename T>
void sort2(T &a, T &b) {
    if (a > b)
        swap(a, b);
}

int main() {
    int a1, a2;
    std::cin >> a1 >> a2;

    sort2(a1, a2);

    std::cout << a1 << " " << a2;

    return 0;
}
```

3. Создайте в функции main вектор из 100000000 целых значений. Напишите функции которые в своём теле ничего не делают, но принимают вектор следующими способами:

- По значению (vector<int> v)
- По значению указателя (vector<int> *v)
- По ссылке (vector<int> &v)

Замерьте время вызова каждой из функций и приложите к отчёту вместе с кодом эксперимента. Сделайте выводы.

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

#define TIME_TEST(testCode, message) { \
    clock_t start_time = clock () ; \
    testCode \
    clock_t end_time = clock () ;\
    clock_t sort_time = end_time - start_time ; \
    std::clog << message << ": "  \
              << (double) sort_time/CLOCKS_PER_SEC << std::endl; \
```

```
}

void emptyFunctionByVal(std::vector<int> v) {

}

void emptyFunctionByRef(std::vector<int> &v) {

}

void emptyFunctionByPointer(std::vector<int> *v) {

}

int main() {
    std::vector<int> v(100000000);

    TIME_TEST({
                  emptyFunctionByVal(v);
    }, "Passing argument by value")

    TIME_TEST({
                  emptyFunctionByPointer(&v);
    }, "Passing argument by pointer")

    TIME_TEST({
                  emptyFunctionByRef(v);
    }, "Passing argument by reference")

    return 0;
}
```

```
/Users/vlad/Desktop/C/programming-and-algorithmization-basics/main
Passing argument by value: 0.65705
Passing argument by pointer: 1e-06
Passing argument by reference: 1e-06


Process finished with exit code 0
```

Передача по значению является наиболее медленной по сравнению с передачей по ссылке или указателю.

4. Напишите следующие функции:

- Ввод вектора целых чисел

```
template<typename T>
void inputVector(std::vector<T> &array) {
    for (auto &item: array)
        std::cin >> item;
}
```

- Вывод вектора целых чисел

```
template<typename T>
void outputVector(std::vector<T> &array) {
    for (auto &item: array)
        std::cout << item << " ";
}
```

- Поиск минимального значения среди элементов вектора

```
template<typename T>
T min(std::vector<T> &array) {
    T minVal = array[0];

    for (size_t i = 1; i < array.size(); i++)
        if (array[i] < minVal)
            minVal = array[i];

    return minVal;
}
```

- Вводится последовательность с клавиатуры, признак конца ввода – 0. Изменить порядок следования элементов в векторе на обратный. Индексы в функции изменения порядка должны иметь тип size_t. Убедитесь, что функция корректно работает для пустого вектора (проще обработать данный случай до тела цикла).
  Убедитесь, что используете оптимальный способ передачи вектора в функцию (по ссылке или по ссылке на константу).

```
template<typename T>
void inputInverted(std::vector<T> &array) {
    T input;
    std::cin >> input;

    if (input) {
        do {
            array.push_back(input);
            std::cin >> input;
        } while (input);

        for (auto begin = array.begin(), end = array.end() - 1; end > begin;
begin++, end--) {
            std::swap(*begin, *end);
        }
    }
}
```

5. Решите следующую задачу, используя указатели: даны коэффици- енты a, b, c квадратного уравнения. Гарантируется, что количество корней равняется двум. Вывести полученные корни. Корни должны выводиться в функции main, а их вычисление – происходить в void getRoots(int a, int b, int c, double *x1, double *x2).

```
#include <iostream>

void getRoots(int a, int b, int c, double *x1, double *x2) {
    double sqrtD = sqrt(b * b  - 4 * a * c);
    *x1 = (-b + sqrtD) / (2 * a);
    *x2 = (-b - sqrtD) / (2 * a);
```

```
}

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

    double x1, x2;
    getRoots(a, b, c, &x1, &x2);

    std::cout << x1 << " " << x2 << std::endl;

    return 0;
}
```

6. Для условия задачи 5, выполните переход на ссылки: void getRoots(int a, int b, int c, double &x1, double &x2)

```
#include <iostream>

void getRoots(int a, int b, int c, double &x1, double &x2) {
    double sqrtD = sqrt(b * b  - 4 * a * c);
    x1 = (-b + sqrtD) / (2 * a);
    x2 = (-b - sqrtD) / (2 * a);
}

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

    double x1, x2;
    getRoots(a, b, c, x1, x2);

    std::cout << x1 << " " << x2 << std::endl;

    return 0;
}
```

7. При необходимости проведите эксперименты и заполните следующую таблицу:

Аспект	Указатель	Ссылка
Обязан быть инициализирован	Нет	Да
Особенности обращения к элементам	Выполняется через оператор ->	Выполняется через оператор .
Возможность перенацеливания	Да	Нет
Возможность получения адреса переменной (адреса указателя или адреса ссылки)	Да	Нет
Возможность непосредственной работы с динамической памятью	Да	Нет
Возможность создавать массивы	Да	Нет

Вывод: в ходе лабораторной работы получили навыки работы с потоками, ссылками, управляющими конструкциями; осознание необходимости появления данных языковых средств.