**Федеральное государственное автономное образовательное учреждение**

**высшего образования**

**«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ**

**УНИВЕРСИТЕТ имени академика С.П. КОРОЛЕВА»**

**Факультет информатики**

**Кафедра технической кибернетики**

**Отчет по лабораторной работе №3**

**по курсу «Языки программирования»**

Выполнил: Борисов Д. С.

Проверил: Головастиков Н.В.

Группа: 6110

САМАРА 2017

**Задание**

**Основное задание на ЛР:**

Реализовать консольное приложение, предназначенное для сортировки целочисленного вектора стандартными средствами и собственной функцией, реализующей алгоритм сортировки согласно варианту. **Входные данные:**количество элементов, тип входного вектора, вектор. **Выходные данные:**. отсортированный массив, время работы библиотечной и реализованной функций сортировок, ускорение библиотечной сортировки.

**Особенности:**

**-**должна присутствовать возможность выбора исходного (сортируемого) вектора: вводимый с клавиатуры или генерируемый согласно варианту (два возможных для выбора типа генерируемых данных);

- в случае, если число элементов вектора менее некоторого разумного значения, исходный и отсортированные двумя методами векторы должны выводиться на экран для сравнения и проверки правильности результатов сортировки;

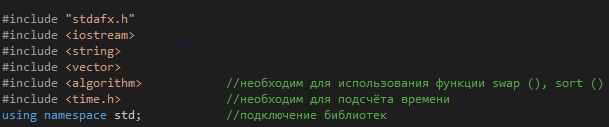
- в случае, если число элементов вектора более некоторого разумного значения, должны выводиться на экран для сравнения и проверки правильности результатов сортировки некоторое количество элементов исходного и отсортированных двумя методами векторов.

**Вариант 11.** Сортировка пузырьком. **Входные данные:** Строго случайные данные и «отсортированные наоборот» случайные данные

**Основная часть**

Для решения поставленной задачи воспользуемся программой Microsoft Visual Studio, используя язык программирования С++.

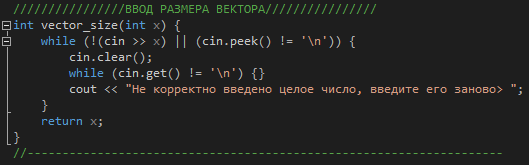
Для успешного выполнения задания необходимо подключить необходимые библиотеки, в которых хранятся команды для работы.



*Рис. 1. Подключение библиотек*

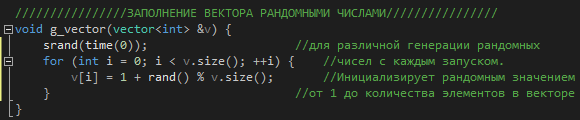
Теперь создадим все необходимые функции, выполняющие определённую работу:

1. Первая функция проверяет введённое значение ‘x’, обозначающее размер вектора. Если введено то, что не подходит под тип переменной, или после того, как переменная изъяла из потока целое число, внутри потока остался не символ перевода строки ‘\n’, то мы с помощью cin.clear() удаляем флаг ошибки для дальнейшей работы и уже с помощью цикла (пока следующий элемент не равен символу перевода строки) мы очищаем поток до ‘\n’. Потом выводим текст с ошибкой и просим ввести значение заново. Если введено корректное целое число, то возвращаем его через return.



*Рис. 2. Ввод размера вектора*

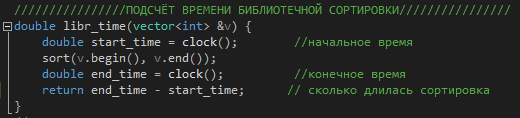
2.Вторая функция void g\_vector необходима для заполнения вектора рандомными числами (void мы поставили, так как мы ничего не возвращаем). Для этого мы по ссылке передаём нужный вектор (который инициализируем в функции main) для того, чтобы не создавать новый вектор, а изменить уже существующий. В самой функции пишем srand(time(0)), а потом цикл for с количеством итераций от 0 до размера вектора, в котором заменяем каждый элемент вектора рандомным числом, не превышающим размер вектора.



*Рис. 3. Заполнение вектора случайными числами*

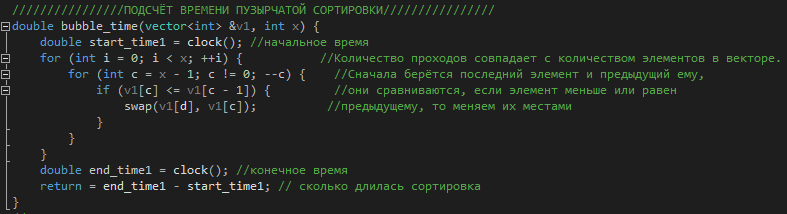
3.Третья функция void vector\_filling(int x, vector<int>& v) даёт выбор пользователю метод заполнения вектора целочисленными значениями (см. примечание). Она ничего не возвращает, принимает значение ’x’ (размер вектора) и собственно ссылку на сам вектор. Создаётся бесконечный цикл, выйти из которого можно лишь введя букву, соответствующего метода заполнения. Если вводится ‘k’, то выполняется заполнение вектора с клавиатуры заменой каждого его элемента на новый, только что введённый. Если вводится ‘r’, то вектор заполняется «отсортированными наоборот» случайными данными, с помощью функциями g\_vector (см. вторую функцию) и sort, преобразованный под обратную сортировку. Если вводится ‘g’, то вектор заполняется случайными числами без какой-либо сортировки. Если введено что-то другое, то пользователя просят ввести символ заново.

4.Четвёртая функция сортирует вектор с помощью библиотечной функции sort и возвращает разницу между временами конца и начала сортировки с помощью clock, которая в свою очередь возвращает количество тиков программы на данный момент.



*Рис. 4. Подсчёт времени работы библиотечной сортировки*

5.Пятая функция уже сортирует вектор «пузырьком» и возвращает также разницу между временами конца и начала сортировки с помощью clock. Засекается время начала, потом начинаются два цикла: первый – обозначает количество проходов (совпадает с размером вектора), второй (вложенный в первый) – начинает брать последний элемент, постепенно уменьшая его на единицу до тех пор, пока он не дойдёт до нулевого элемента. Каждый раз этот элемент сравнивается с числом, стоящим до него, и если предыдущий больше следующего, то они меняются местами. Получается, что второй цикл заново выполняется ровно столько раз, сколько имеет итераций первый цикл. В результате получается отсортированный вектор. Засекается время конца работы и возвращается разница конца и начала работы.



*Рис. 5. Подсчёт времени работы пузырчатой сортировки*

6.Шестая функция (см. примечание) мало чем отличается от предыдущей, за исключением того, что она эффективней. Здесь также количество проходов равно количеству чисел в векторе, но теперь в первом цикле появляется буловая переменная count, инициализированная false. Если в проходе не была использована функция swap, и count всё ещё имеет значение false, то это означает, что вектор уже отсортирован и сортировка заканчивается. Это позволяет меньше затрачивать времени на сортировку «пузырьком». После чего засекается время конца работы и возвращается разница конца и начала сортировки «пузырьком».

7.Седьмая функция (см. примечание) необходима лишь для того, чтобы выводить элементы вектора через cout, вот только она для удобства чтения выводит лишь первые 100 элементов вектора, так как большее их количество было бы трудно читать. Здесь я использовал проход по каждому числу в векторе с их выводом (если их меньше ста), и вывод первых ста элементов (если их больше ста).

Теперь таки начинается главная функция main(). Для начала подключим возможность выводить русские буквы на экран с помощью setlocale(LC\_CTYPE, "rus"). Потом создаётся бесконечный цикл с выводом текста «Продолжить? (Да (Y) | Нет (N))\n)», после которого нужно ответить либо ‘Y’, после которого идёт сам код, либо ‘N’, после которого программа заканчивает свою работу, иначе потребуется вводить ответ заново. После введённого ‘Y’, пользователя просят ввести размер вектора. Инициализируется переменная x нулём (для того, чтобы использовать её в функции 1), после чего идёт функция 1. Создаётся вектор v размером x (изначально заполненный нулями), потом выводится сообщение о выборе метода его заполнения, следом идёт функция 3 (vector\_filling). Создаются ещё два вектора (v1 и v2), равные самому первому вектору, после чего с помощью функции 7 (out\_val) выводится старый вектор, потом результаты функций 4, 5, 6 (libr\_time, bubble\_time, bubble\_time1) заносятся в соответствующие переменные l, b, b1. Затем выводится время работы библиотечной сортировки и отсортированный с помощью неё старый вектор, следом выводится время работы сортировки «пузырьком» и отсортированный им v2, ну и наконец выводится время работы ускоренной сортировки «пузырьком» и отсортированный им вектор v2. Под конец выводится отношение времени работы sort к времени работы «пузырька» и если один из них равен нулю, то выводится сообщение о невозможности вычисления отношения. Программа закончена.

Вывод: даже ускоренная пузырчатая сортировка работает быстрее библиотечной, так как sort имеет сложность *O(n log n)*, а «пузырь» *O(n2)*, следовательно затрачивает меньше времени.

Заключение

Цель лабораторной работы - написать программу, которая сортирует массив целочисленных значений с помощью библиотечной сортировки и сортировки «пузырьком», с последующим выводом времени их работы их их отношения, достигнута. В ходе выполнения лабораторной работы я изучил принцип работы сортировки «пузырьком», один из способов вычисления времени работы определённого фрагмента кода, приобрел навыки по написанию сортировок.

Литература

1. Безус Е.А. Козлова Е.С. Лабораторные работы по курсу «Языки программирования» (язык программирования C++): Методические указания / Сост. Е.А. Безус, Е.С. Козлова; Самара, 2016. 36 с.
2. Страуструп Б. Язык программирования С++ / Б. Страуструп; М:Бином, 2011. 1136 с.

Приложение

#include "stdafx.h"

#include <iostream>

#include <string>

#include <vector>

#include <algorithm> //необходим для использования функции swap (), sort ()

#include <time.h> //необходим для подсчёта времени

using namespace std; //подключение библиотек

////////////////ВВОД РАЗМЕРА ВЕКТОРА////////////////

int vector\_size(int x) {

while (!(cin >> x) || (cin.peek() != '\n')) {

cin.clear();

while (cin.get() != '\n') {}

cout << "Не корректно введено целое число, введите его заново> ";

}

return x;

}

//--------------------------------------------------------------------

////////////////ЗАПОЛНЕНИЕ ВЕКТОРА РАНДОМНЫМИ ЧИСЛАМИ////////////////

void g\_vector(vector<int> &v) {

srand(time(0)); //для различной генерации рандомных

for (int i = 0; i < v.size(); ++i) { //чисел с каждым запуском.

v[i] = 1 + rand() % v.size(); //Инициализирует рандомным значением

} //от 1 до количества элементов в векторе

}

//--------------------------------------------------------------------

////////////////ВЫБИРАЕТСЯ МЕТОД ЗАПОЛНЕНИЯ ВЕКТОРА////////////////

void vector\_filling(int x, vector<int>& v) {

char k\_g\_r;

while (cin >> k\_g\_r) {

if (k\_g\_r == 'k') {

int value;

cout << "Введите целое число в вектор> ";

for (int i = 0; i < x; ++i) { //инициализация каждого элемента вектора

cin >> value; //соответствующим введённым значением

v[i] = value;

}

break;

}

else if (k\_g\_r == 'r') {

g\_vector(v);

sort(v.rbegin(), v.rend()); //Обратная сортировка с помощью reverse

break;

}

else if (k\_g\_r == 'g') {

g\_vector(v);

break;

}

else { //вывод ошибки и просьба ввести символ заново

cout << "Неправильный символ! Введите его заново> ";

continue;

}

}

}

////////////////ПОДСЧЁТ ВРЕМЕНИ БИБЛИОТЕЧНОЙ СОРТИРОВКИ////////////////

double libr\_time(vector<int> &v) {

double start\_time = clock(); //начальное время

sort(v.begin(), v.end());

double end\_time = clock(); //конечное время

return end\_time - start\_time; // сколько длилась сортировка

}

//--------------------------------------------------------------------

////////////////ПОДСЧЁТ ВРЕМЕНИ ПУЗЫРЧАТОЙ СОРТИРОВКИ////////////////

double bubble\_time(vector<int> &v1, int x) {

double start\_time1 = clock(); //начальное время

for (int i = 0; i < x; ++i) { //Количество проходов совпадает с количеством элементов в векторе.

for (int c = x - 1; c != 0; --c) { //Сначала берётся последний элемент и предыдущий ему,

if (v1[с] <= v1[c - 1]) { //они сравниваются, если элемент меньше или равен

swap(v1[d], v1[c]); //предыдущему, то меняем их местами

}

}

}

double end\_time1 = clock(); //конечное время

return end\_time1 - start\_time1; // сколько длилась сортировка

}

//-------------------------------------------------------------------

////////////////ПОДСЧЁТ ВРЕМЕНИ УСКОРЕННОЙ ПУЗЫРЧАТОЙ СОРТИРОВКИ////////////////

double bubble\_time1(vector<int> &v2, int x) {

double start\_time1 = clock(); //начальное время

for (int i = 0; i < x; ++i) { //Количество проходов совпадает с количеством элементов в векторе.

bool count = false; //Определяет, был ли использован swap в проходе

for (int c = x - 1; c != 0; --c) {

if (v2[c] <= v2[c-1]) {

swap(v2[c-1], v2[c]);

count = true;

}

}

if (!count) break; //Если функция swap не использовалась, то вектор отсортирован

}

double end\_time1 = clock(); //конечное время

return end\_time1 - start\_time1; // сколько длилась сортировка

}

//-------------------------------------------------------------

////////////////ВЫВОД НА ЭКРАН ЗНАЧЕНИЙ ВЕКТОРА////////////////

void out\_val(vector<int>& v, int x) {

if (x < 100) {

for (int i : v) {

cout << i << ' ';

}

}

else {

for (int i = 0; i < 100; ++i) {

cout << v[i] << ' ';

}

}

}

//-------------------------------------------------------------

int main() {

setlocale(LC\_CTYPE, "rus"); //функция, позволяющая выводить на экран буквы русского алфавита

while (cout << "Продолжить? (Да (Y) | Нет (N))\n") {

string y\_n;

cin >> y\_n;

if (y\_n == "Y" || y\_n == "y") {

cout << "Введите размер вектора> ";

int x = 0;

x = vector\_size(x); //Проверка на правильность введённого х

vector<int>v(x);

cout << "Выберите метод заполнения вектора" << endl

<< "Клавиатура (k), генерация (g) или <<обратная>> генерация (r)? (k|g|r)> ";

vector\_filling(x, v); //функция, заполняющая вектор

vector<int> v1 = v; //создание и копирование нового вектора

vector<int> v2 = v;

cout << "Старый вектор: ";

out\_val(v, x); //вывод значений вектора v

double l = libr\_time(v); //время, потраченное на сортировку sort

double b = bubble\_time(v1, x); //время, потраченное на сортировку bubble

double b1 = bubble\_time1(v2, x);

cout << endl << "Время библиотечной сортировки = "

<< l << "\n" << "Отсортированный вектор (sort): ";

out\_val(v, x); //вывод значений отсортированного sort-ом вектор v

cout << endl << "Время пузырчатой сортировки = "

<< b << "\n" << "Отсортированный вектор (пузырьком): ";

out\_val(v1, x); //вывод значений отсортированного пузырьком вектор v

cout << endl << "Время ускоренной пузырчатой сортировки = "

<< b1 << "\n" << "Отсортированный вектор (уск. пузырьком): ";

out\_val(v2, x); //вывод значений отсортированного пузырьком вектор v

//-----------------------------------------------------------------------------

if (l != 0 && b != 0) {

cout << endl << "отношения сортировки пузырьком к сортировки sort - ом: "

<< b / l << endl; //вывод отношения сортировки пузырьком к сортировки sort - ом

}

//-----------------------------------------------------------------------------

else {

cout << "\n" << "Время одной из сортировок равно нулю! "

<< "Найти отношение двух сортировок невозможно!\n";

}

}

//-----------------------------------------------------------------------------

else if (y\_n == "N" || y\_n == "n") {

return 0;

}

//-----------------------------------------------------------------------------

else {

cout << "Неправильный символ!\n";

continue;

}

}

}