**Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования**

**"Уфимский государственный авиационный технический университет"**

**Кафедра** Высокопроизводительных вычислительных технологий и систем

**Дисциплина:** Программирование

**Отчет по лабораторной работе № 9**

**Тема: «Использование стандартной библиотеки шаблонов STL и**

**работа с файлами»**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Группа ПМИ-148 | Фамилия И.О. | Подпись | Дата | Оценка |
| Студент | Фаизова А.Н. |  |  |  |
| Принял | Гайнетдинова А.А. |  |  |  |

**Уфа 2018**

**Цель:** изучить на практике использование стандартной библиотеки шаблонов(STL), также организацию ввода-вывода с использованием файлов.

**Теоретический материал**

С точки зрения языка C, любой файл представляет собой последовательный поток байтов. Каждый файл заканчивается специальным маркером конца файла. Когда файл открывается, то ему ставится в соответствие поток ввода или вывода, в зависимости от того, для каких целей (можно также сказать с какими опциями) он был открыт.

В начале работы программы открываются три файла и связанные с ними потоки: стандартный ввод (*stdin*), стандартный вывод (*stdout*) и стандартная ошибка (*stderr*). Потоки обеспечивают каналы передачи данных между файлами и программами. Открытый файл возвращает указатель на структуру *FILE*, определенную в *<stdio.h>*.

Стандартная библиотека поддерживает многочисленные функции чтения данных из файла и записи данных в файл.

*fopen* (имя файла, опция открытия) - открытие файла (w - на запись; r - на чтение, а - на добавление; w+ - создается файл для чтения и записи; r+ - открывается файл для чтения и записи, а+ - открывается файл для чтения и записи с помещением маркера текущей позиции в файле в конец файла).

*fclose* (указатель на файл) – закрытие файла.

*feof* (указатель на файл) – возвращает true,если достигнут конец файла.

*fgetc* (указатель на файл) – считывает из файла один символ. Например, *fgetc* (*stdin*) считает один символ из стандартного потока, что эквивалентно использованию функции *getchar*().

*fputc* (символ, указатель на файл) – запись одного символа в файл. Например, *fputc('a', stdout)* эквивалентно *putchar('a').*

*fgets* (указатель на файл) – считывает из файла строку.

*fputs* (строка, указатель на файл) – запись строки в файл.

*fscanf* (указатель на файл-поток, строка формата ввода) – считывание из файла.

*fprintf* (указатель на файл-поток,строка формата вывода, список переменных для вывода) – запись в файл.

*rewind* (указатель на файл) – переместить указатель текущей позиции в файле на начало.

*fwrite* (адрес первой ячейки памяти, размер одного блока, количество блоков, указатель на файл) – вывести в файл указанное количество блоков в файл.

*fread* (адрес первой ячейки памяти, размер одного блока, количество блоков, указатель на файл) – прочитать из файла указанное количество блоков в файл.

В С++ для работы с потоками ввода/вывода существует библиотека *iostream.h.* В этой библиотеке представлены следующие классы:

Класс *streambuf* предоставляет память для буфера, а также методы для его наполнения, доступа к содержимому, сброса буфера и управления памятью буфера.

Класс *ios\_base* предоставляет общие средства потока, такие как признак того, открыт ли поток для чтения, и является ли он бинарным или текстовым.

Класс *ios* базируется на *ios\_base* и включает член-указатель на объект касса *streamuf*.

Класс *ostream* наследуется от *ios* и предоставляет методы вывода.

Класс *istream* наследуется от *ios* и предоставляет методы ввода.

Класс *iostream* базируется на кассах *istream* и *ostream* и потому наследует и методы вывода, и методы ввода.

Для того, чтобы выводить в файловый поток, необходимо выполнить следующее:

Создать объект *ofstream* для управления выходным потоком.

Ассоциировать этот объект с конкретным файлом.

Использовать объект тем же способом, как используется *cout*. Единственным отличием будет то, что вывод направляется в файл вместо экрана.

**Ответы на контрольные вопросы**

1. Что такое поток?

Поток – это часть уже самого процесса, выполняющая определенный список действий. У каждого процесса есть как минимум один поток, и их увеличение обеспечивает распараллеливание процесса.

Сам поток  представляет собой стек команд со счетчиком, обладающий несколькими важными свойствами, такими как состояние и приоритет. Состояний потока всего три:  состояние активности, то есть поток выполняется на данный момент, состояние неактивности, когда поток ожидает выделения процессора для  перехода в состояние активности, и третье – состояние блокировки, когда потоку не выделяется время (соответственно он не занимает место в очереди, освобождая ресурсы) вне зависимости от его приоритета.

Из ресурса Википедия:

Поток данных (англ. stream) в программировании — абстракция, используемая для чтения или записи файлов, сокетов и т. п. в единой манере.

Потоки являются удобным унифицированным программным интерфейсом для чтения или записи файлов (в том числе специальных и, в частности, связанных с устройствами), сокетов и передачи данных между процессами.

1. Что представляет собой файловый указатель?

Особую роль при программировании операций с файлом играет файловый указатель, который определяет, в какое место файла выводится запись или откуда она вводится. При последовательной обработке файла файловый указатель является счетчиком уже прочитанных или записанных байт. Имеются специальные функции, которые позволяют работать с файловым указателем:

а) определение значения файлового указателя:

long ftell(FILE \*stream); // возвращает текущее значение файлового указателя;

б) установка файлового указателя на начало файла:

int rewind(FILE \*stream); // файловый указатель устанавливается на начало файла;

в) установка файлового указателя в указанное место:

int fseek(FILE \*stream,long offset,int whenсe);

Поток связывается с файлом посредством обращения к функции fopen(), и вся дальнейшая работа с потоком происходит благодаря использованию файлового указателя (который является указателем типа FILE \*). В определенном смысле файловый указатель играет роль клея, который обеспечивает целостность всей системы.

1. Перечислите режимы доступа к файлу.

Режим доступа к файлу.

Возможные значения данного параметра следующие:

w – файл открывается для записи (write); если файла с заданным именем нет, то он будет создан; если же такой файл уже существует, то перед открытием прежняя информация уничтожается;

r – файл открывается для чтения (read); если такого файла нет, то возникает ошибка;

a – файл открывается для добавления (append) новой информации в конец;

r+ (w+) – файл открывается для редактирования данных, т.е. возможны и запись, и чтение информации;

a+ – то же, что и для a, только запись можно выполнять в любое место файла (доступно и чтение файла);

t – файл открывается в текстовом режиме;

b – файл открывается в двоичном режиме;

Последние два режима используются совместно с рассмотренными выше. Возможны следующие комбинации режимов доступа: “w+b”, “wb+”, “rw+”, “w+t”, “rt+”, а также некоторые другие комбинации.

1. Как открыть и как закрыть файл?

В C++ файл открывается путем стыковки его с потоком. Имеется три типа потоков: ввода, вывода и ввода/вывода. Для открытия потока ввода необходимо объявить поток как объект класса ifstream. Для открытия потока вывода необходимо объявить его как объект класса ofstream. Потоки, которые выполняют как ввод, так и вывод, должны быть объявлены как объекты класса fstream. Например, следующий фрагмент программы создает один поток ввода, один поток вывода и один поток, способный выполнять как ввод, так и вывод:

ifstream in; // ввод

ofstream out; // вывод

fstream both; // ввод и вывод

После создания потока единственным способом ассоциировать его с файлом является использование функции ореn(). Эта функция является членом каждого из трех классов потока. Она имеет следующий прототип:

void open(const char "filename, int mode, int access=filebuf::openprot);

Здесь filename является именем файла и может включать указание пути. Величина mode определяет, каким способом открывается файл. Mode может принимать одно или более из следующих значений, (определенных в заголовочном файле fstream.h):

ios::app

ios::ate

ios::binary

ios::in

ios::nocreate

ios::noreplace

ios::out

ios::trunc

Можно комбинировать два и более из этих значений, используя побитовое ИЛИ.

ios::app указывает на то, что выводимые данные будут добавляться в конец файла. Это значение может быть использовано только для файлов, для которых возможен вывод. Использование ios::ate вызывает поиск конца файла в момент открытия файла.

ios::in задает возможность ввода из файла. ios::out указывает, что файл предназначен для вывода. Однако создание потока с использованием ifstream задает режим ввода, а создание потока с использованием ofstream задает режим вывода, и в этих случаях нет необходимости задавать указанные выше величины.

ios::nocreate задает такой режим, при котором функция open() может открыть только существующий файл. ios::noreplace не позволяет открыть файл функции ореn(), если файл уже существует, но не указаны атрибуты ios::ate или ios::app.

ios::trunc обусловливает уничтожение содержимого файла с заданным именем и усечение длины файла до нуля.

Режим ios::binary обусловливает открытие файла для двоичных операций. Это означает, что не будет никаких преобразований символов.

В следующем фрагменте открывается нормальный файл для вывода:

ofstream out;

out.open ( "test", ios::out, 0);

Тем не менее, практически никогда не бывает подобного вызова функции ореn(), поскольку оба параметра имеют значения по умолчанию: для ifstream значением по умолчанию mode служит ios::in; а для ofstream значение по умолчанию равно ios::out. Параметр доступа access по умолчанию соответствует нормальному файлу, как это уже отмечалось выше. Поэтому предыдущая инст­рукция будет выглядеть следующим образом:

out.open("test"); // значения по умолчанию для вывода и нормального файла

Для того, чтобы открыть поток одновременно для ввода и вывода, необходимо задать обе величины ios::in и ios::out для режима mode, как показано в следующем примере:

fstream mystream;

mystream.open("test", ios::in | ios::out);

Если функции open() не удается открыть поток, то поток остается равным NULL.

Хотя открытие файла с использованием функции ореn() является вполне допустимым, в большинстве случаев так не делается, потому что классы ifstream, ofstream и fstream содержат конструкторы, автоматически открывающие файл. Конструкторы имеют те же параметры и значения по умолчанию, что и функция ореn(). Поэтому наиболее типичный способ открытия файла выглядит следующим образом:

ifstream mystream("myfile"); // открытие файла для чтения

Если по каким-либо причинам файл не может быть открыт, значение ассоциированного потока будет равно NULL. Можно использовать следующий код для того, чтобы убедиться, что файл действительно открыт:

ifstream mystream("myfile"); // открытие файла для чтения

if (!mystream) {

cout << "Cannot open file.\n"; // обработка ошибки

}

Для того, чтобы закрыть файл, следует использовать функцию-член close(). Например, чтобы закрыть файл, пристыкованный к потоку mystream, используется следующая инструкция:

mystream.close ();

Функция close() не имеет ни параметров, ни возвращаемого значения.

1. Что такое шаблоны и с какой целью они используются?

Шаблоны (англ. template) — средство языка C++, предназначенное для кодирования обобщённых алгоритмов, без привязки к некоторым параметрам (например, типам данных, размерам буферов, значениям по умолчанию).

В C++ возможно создание шаблонов функций и классов.

Шаблоны позволяют создавать параметризованные классы и функции. Параметром может быть любой тип или значение одного из допустимых типов (целое число, enum, указатель на любой объект с глобально доступным именем, ссылка).

Мы можем абстрагироваться от конкретных типов и использовать шаблоны с параметрами. Синтаксис: в начале, перед объявлением класса напишем слово template и укажем параметры в угловых скобках.

Получается, что механизм шаблонов предлагает совершенное решение, позволяющее отделить общий алгоритм от его реализации применительно к конкретным типам данных. Вы можете составить текст подпрограммы сейчас, а используемые типы уточнять позднее. Это возможно, так как используемый тип данных является в этом случае параметром. Шаблоны сочетают в себе преимущества однократной подготовки фрагментов программы (аналогично макрокомандам) и контроль типов, присущий переопределяемым функциям

1. Какого типа шаблоны используются в программах?

В языке С++ имеются два типа шаблонов - шаблоны функций и шаблоны классов.

1. Как оформляются шаблоны функций?

Объявление шаблона функции начинается с заголовка, состоящего из ключевого слова **template**, за которым следует список параметров шаблона.

// Описание шаблона функции

template <class T>

T min (T a, T b)

{ return a<b ? a : b; }

Ключевое слово **class** в описании шаблона означает тип, идентификатор в списке параметров шаблона T означает имя некоторого типа.

В описании заголовка функции этот же идентификатор означает тип возвращаемого функцией значения и типы параметров функции.

...

// Использование шаблона функции

int m, k, l;

cout<<”Input k, l”;

cin>>k>>l;

m = min (k, l);

...

Компилятор генерирует (порождает) экземпляр функции для указанного типа параметров:

int min (int a, int b)

{ return a<b ? a : b;

}

В списке параметров шаблона слово **class** может относиться и к любому из стандартных типов данных. Таким образом, список параметров шаблона <class T> просто означает, что Т представляет собой фиктивный тип, который будет определен при вызове. Так как Т является параметром, обозначающим тип, шаблоны иногда называют параметризованными типами.

Так как компилятор генерирует экземпляры шаблонов функций согласно типам, заданным при их вызовах, то очень важным моментом является передача корректных типов, особенно если шаблон функции имеет два или более параметров.

1. Какие требования предъявляются к фактическим параметрам шаблонов?

Шаблон функции может быть использован почти для любого типа данных. Предостережение "почти" проистекает из характера операций, выполняемых над параметрами и переменными в теле функции. Какой бы тип мы ни использовали в функции, эти операции для нее должны быть определены. В противном случае компилятор не будет знать, что ему делать.

Таким образом, используя классы в шаблонах функций, убедитесь в том, что вы знаете, какие действия с ними выполняются в шаблоне функции, и определены ли для класса эти действия. Если вы не снабдили класс необходимыми функциями, возникнут различные невразумительные сообщения об ошибках.

1. Какие преимущества программы обеспечиваются при использовании шаблонов классов?

Параметризовать некоторый класс так, чтобы он работал для любого типа данных - это только половина того, что шаблоны обеспечивают для классов. Другой аспект состоит в том, чтобы дать возможность задания числовых параметров. Это позволяет Вам, например, создавать объекты типов "Вектор из 20 целых", "Вектор из 1000 целых" или "Вектор из 10 переменных типа double". Основная идея проста, хотя используемый синтаксис может показаться сложным.

**Индивидуальное задание №1**

Создайте приложение с использованием класса двусторонняя очередь deque. В программе должны обрабатываться три различных вида очередей: символьная, строковая, логическая (тип bool). Для обработки создать шаблонную функцию поиска суммы результатов логического сравнения на равенство элементов, расположенных зеркально относительно центра очереди. Например, в очереди 12 элементов, сравниваем 1 и 12, 2 и 11 и т.д.

Исходный код программы

#include <iostream>

#include <deque>// подключаем заголовочный файл деков

#include <string>

using std::cout;

using std::endl;

template <typename T>

int Sum\_equal( std::deque<T> &d )

{

int S=0;

do

{

if (d.back() == d.front()){S++;}

d.pop\_back();

d.pop\_front();

}while((d.size()>1));

return S;

}

int main()

{

std::deque<std::string> str; // создаем пустой дек типа string

str.push\_front("One");//1+

str.push\_back("Two");//2

str.push\_back("Th");//3+

str.push\_back("One");//4+

str.push\_back("On");//5

str.push\_back("One");//6

str.push\_back("Th");//7

str.push\_back("One");//8

str.push\_back("One");//9

int x=0;

x=Sum\_equal(str);

std::deque<char> ch; // создаем пустой дек типа char

ch.push\_front('O'); //1+

ch.push\_back('O'); //2+

ch.push\_back('T');//3+

ch.push\_back('O');//4

ch.push\_back('On');//5+

ch.push\_back('On');//6

ch.push\_back('T');//7

ch.push\_back('T');//8

ch.push\_back('O');//9

ch.push\_back('O');//10

int y=0;

y=Sum\_equal(ch);

std::deque<bool> log;// создаем пустой дек типа bool

log.push\_front(true); //1

log.push\_back(false); //2+

log.push\_back(false);//3

log.push\_back(false);//4+

log.push\_back(true);//5

log.push\_back(true);//6

log.push\_back(false);//7

log.push\_back(true);//8

log.push\_back(false);//9

log.push\_back(false);//10

int z=0;

z=Sum\_equal(log);

cout<<" Sum of symmetric elements in an string double-ended-queue = "<<x<<";"<<endl<<endl;

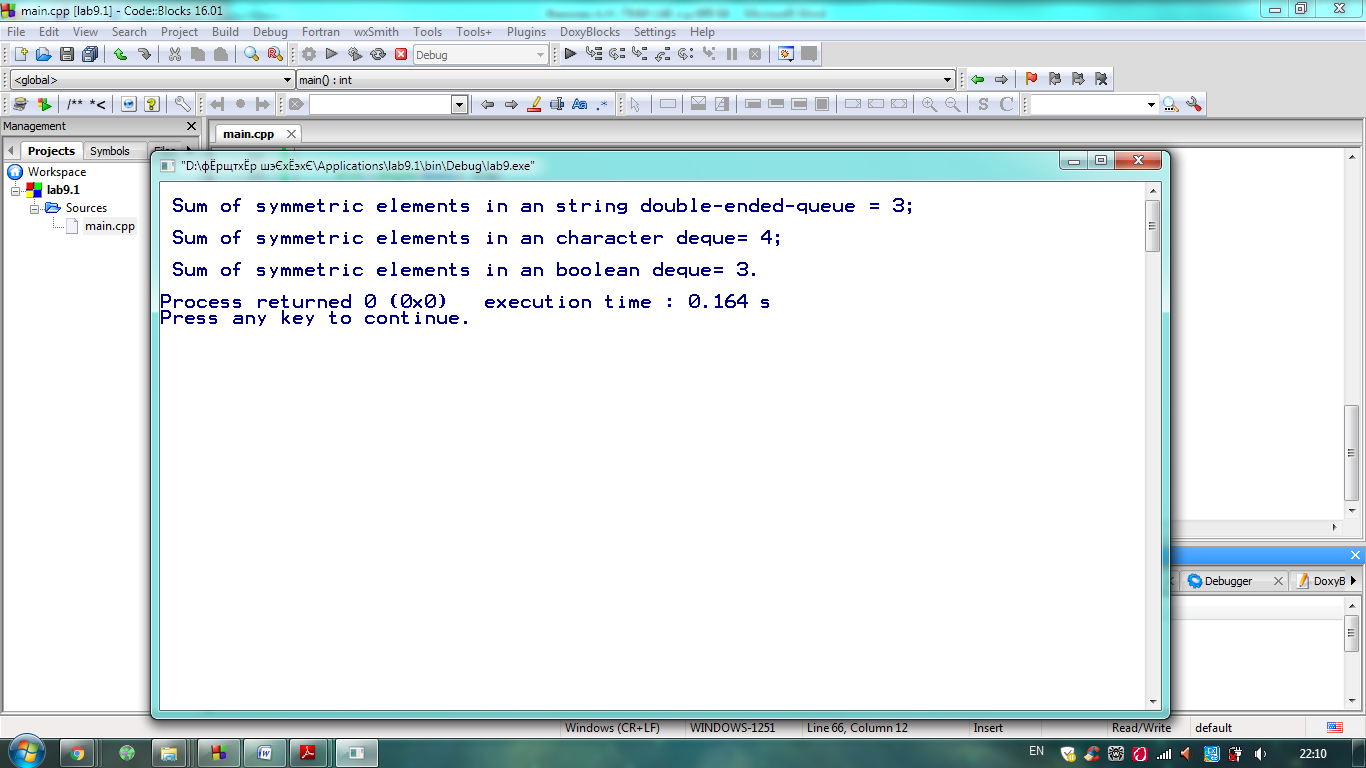
cout<<" Sum of symmetric elements in an character deque= "<<y<<";"<<endl<<endl;

cout<<" Sum of symmetric elements in an boolean deque= "<<z<<"."<<endl;

return 0;

}

Пример выполнения программы

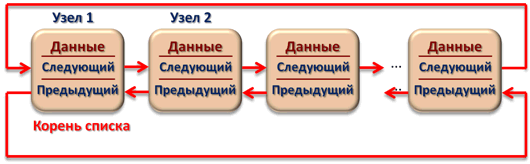


**Индивидуальное задание №2**

Создайте приложение для обработки файла с данными. Данные в файле расположены в несколько колонок. Имя файла и номер базовой колонки вводятся пользователем. Для хранения данных создается структура для каждой строки, а для всего файла двусвязный список структур. При чтении данных из файла происходит помещение элемента базовой колонки в двусвязный список в позицию, соответствующую сортировке списка по возрастанию. Отсортированные данные выводятся в другой файл в виде таблицы.

Описание программы**:**

Двусвязный список



Исходный код программы

#include <iostream>

#include <fstream>

#include <sstream>

#include <vector>

#include <string>

using namespace std;

class DList {

struct Node {

vector<int> numbers; // вектор чисел

Node \*prev; // указатель на предыдущий элемент

Node \*next; // указатель на следующий элемент

};

Node \*head; // начало списка

Node \*tail; // конец списка

public:

DList();

void Insert(const vector<int> &numbers, int index);

void Print(const string &path);

bool isEmpty();

~DList();

};

DList::DList() {

head = NULL; // изначально список

tail = NULL; // пуст и указатели пусты

}

// вставка массива в список с сортировкой по индексу index

void DList::Insert(const vector<int> &numbers, int index) {

Node \*node = new Node; // создаём новый элемент списка

node->numbers = numbers; // присваиваем вектор

node->prev = NULL; // обнуляем указатели на следующий

node->next = NULL; // и предыдущий элементы списка

if (!head) { // если список пуст

head = node; // начало и конец списка -

tail = node; // это этот элемент

}

else {

Node \*prev = NULL; // указатель на предыдущий элемент

Node \*curr = head; // указатель на текущий элемент

// ищем место для вставки (пока не кончился список и вставляемое значение больше текущего)

while (curr && curr->numbers[index] <= numbers[index]) {

prev = curr; // запоминаем предыдущий элемент

curr = curr->next; // переходим к следующему элементу

}

// если остались на месте, то вставляем в начало

if (curr == head) {

node->next = head;

head->prev = node;

head = node;

}

else if (curr) { // иначе вставляем в середину между prev и curr

node->next = curr;

node->prev = prev;

curr->prev = node;

prev->next = node;

}

else { // иначе вставляем в конец

node->prev = tail;

tail->next = node;

tail = node;

}

}

}

// вывод списка

void DList::Print(const string &path) {

ofstream output;

output.open(path.c\_str());

if (!head) {

output << "(empty)" << endl; // если список пуст, выводим сообщение

return;

}

// иначе проходимся по всем элементам списка

for(Node \*node = head; node; node = node->next) {

for (size\_t i = 0; i < node->numbers.size(); i++) // и выводим каждый массив

output << node->numbers[i] << " ";

output << endl;

}

output.close();

}

bool DList::isEmpty() {

return head == NULL;

}

DList::~DList() {

while (head) {

Node \*tmp = head->next;

delete head; // удаляем все элементы списка

head = tmp;

}

tail = NULL;

}

DList readFromFile(const string& path, int index) {

DList dlist;

ifstream input;

input.open(path.c\_str());//Функция c\_str() возвращает указатель на символьный массив, который содержит строку объекта string в том виде, в котором она размещалась бы, во встроенном строковом типе

if (!input) {

cout << "Incorrect file!" << endl;

return dlist;

}

string line;

while (getline(input, line)) {

stringstream ss(line);

vector<int> numbers;

int number;

while (ss >> number)

numbers.push\_back(number);

dlist.Insert(numbers, index);

}

input.close();

return dlist;

}

int main() {

string path;

int index;

cout << "Enter path to input file: ";

getline(cin, path);// ф-я getline предназначена для ввода данных из потока

cout << "Index: ";

cin >> index;

cin.ignore();

DList dlist = readFromFile(path, index);

if (!dlist.isEmpty()) {

cout << "Enter path to output file: ";

getline(cin, path);

dlist.Print(path);

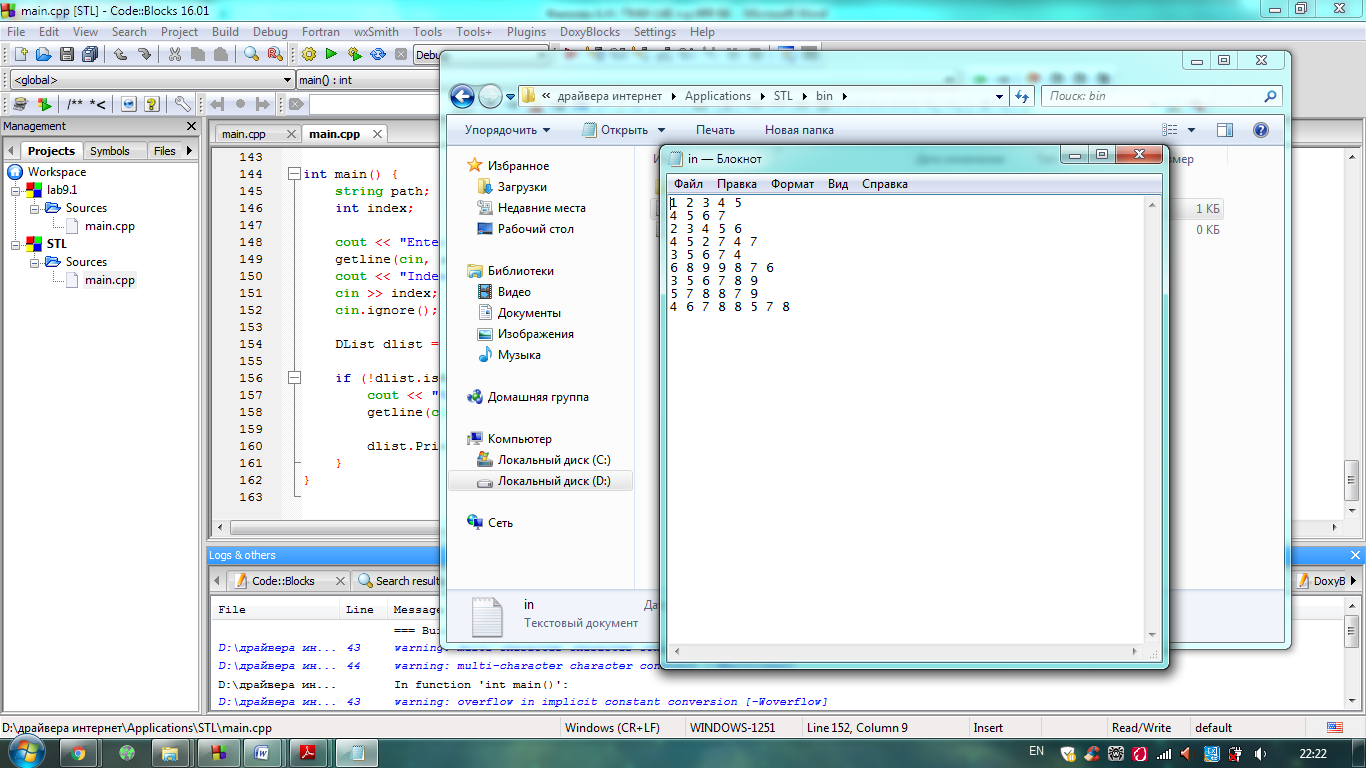
}

}

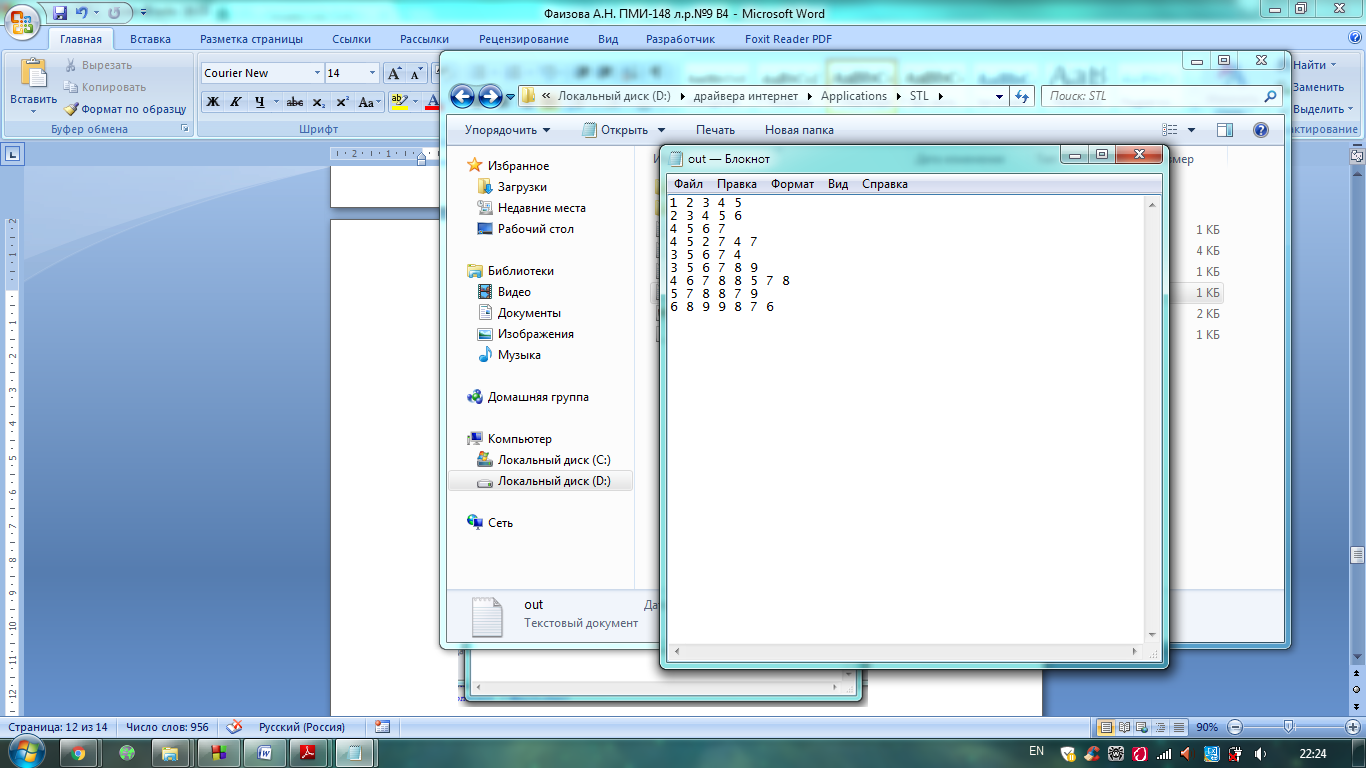
Пример выполнения программы

Сортировка по второй колонке

Исходный файл

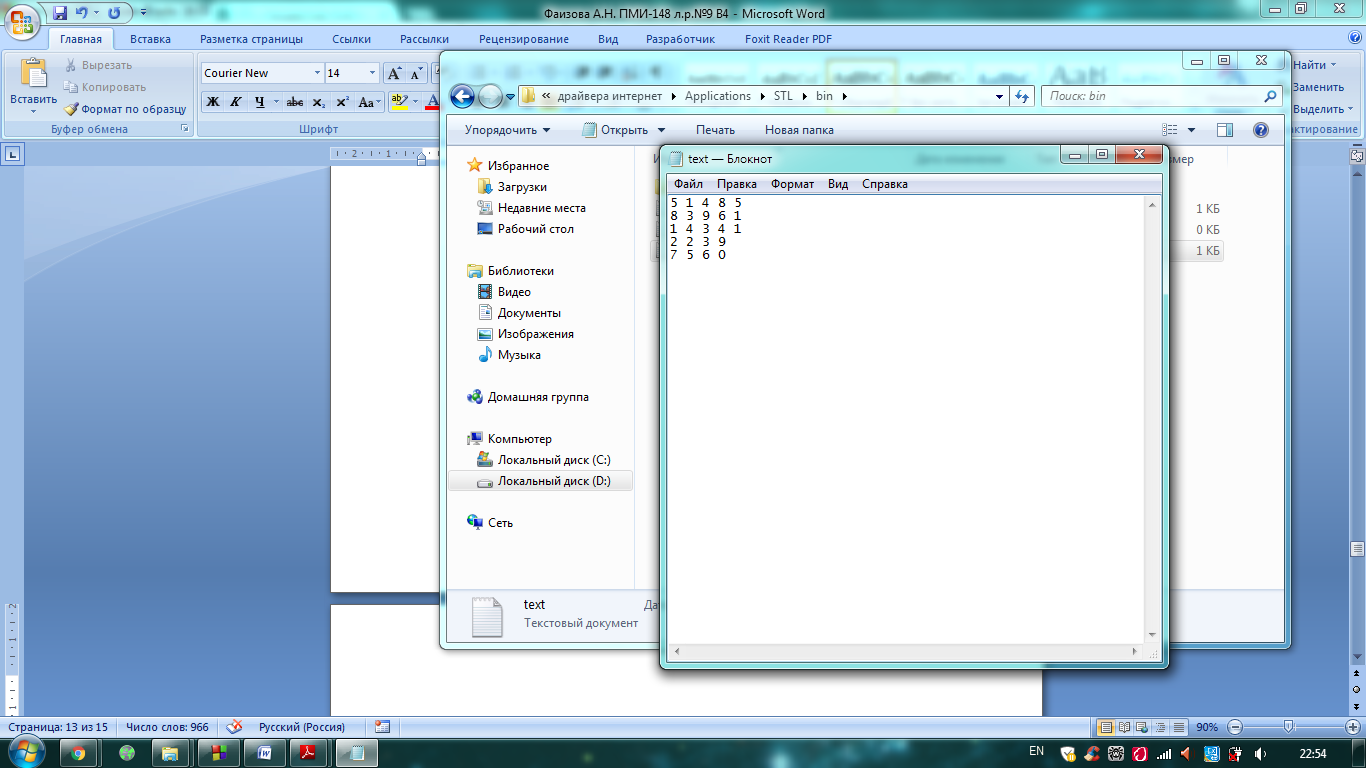


Принимающий файл

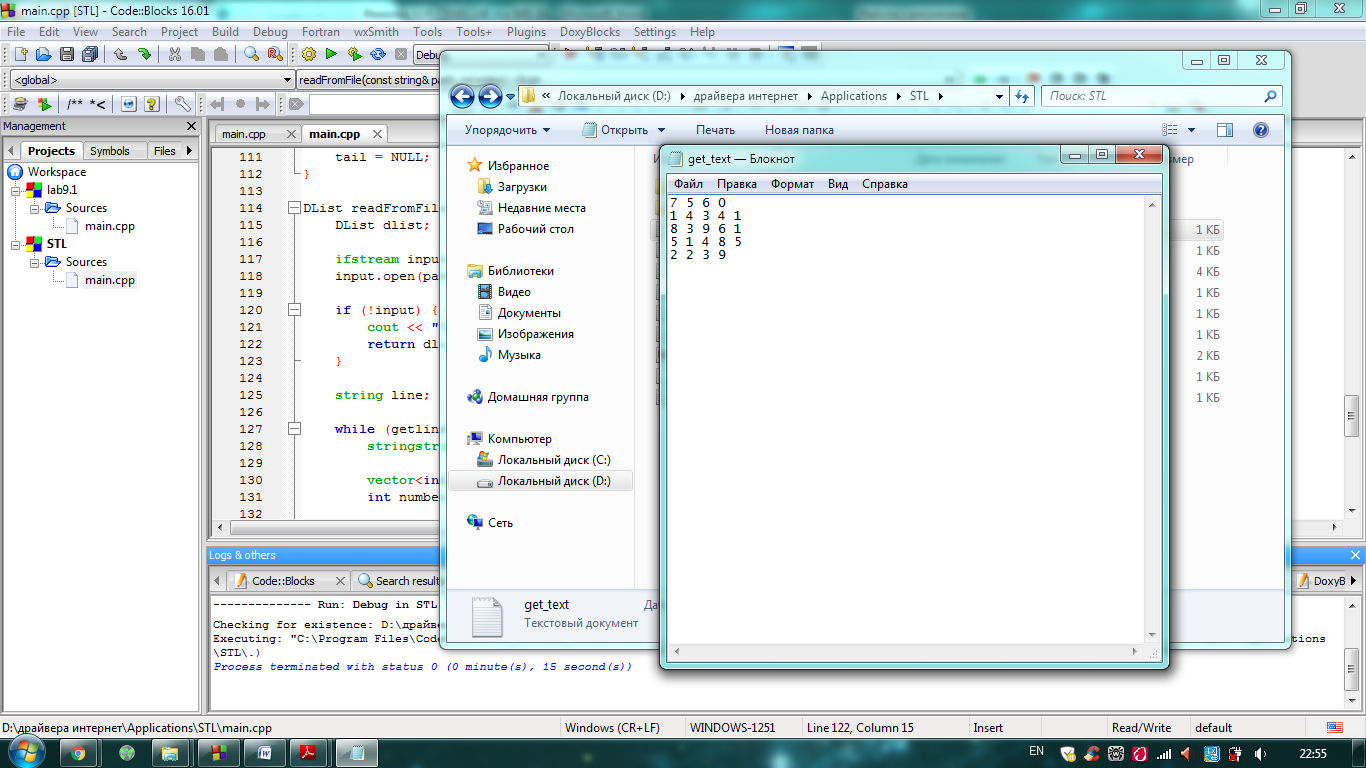


Сортировка по четвёртой колонке

Исходный файл



Принимающий файл



**Вывод**

На данной лабораторной работе мы изучили на практике использование стандартной библиотеки шаблонов(STL), также организацию ввода-вывода с использованием файлов.

**Список использованной литературы**

1. Бьерн Страуструп. Язык программирования C++. Специальное издание. Пер. с англ. — М.: Издательство Бином, 2011 г. — 1136 с.
2. Стенли Б. Липпман. C++ для начинающих: Пер. с англ. 2тт. - Москва: Унитех; Рязань: Гэлион, 1992, 304-345сс.
3. Шилдт, Герберт. C++: руководство для начинающих, 2-е издание. : Пер. с англ. — М. : Издатель- ский дом “Вильямс”, 2005. — 672 с.