**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Отчет по лабораторной работе №2

«**Разработка класса Планета и создание класса по варианту**»

Вариант 24: «**Заявки на выполнение работ**»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Выполнил:** | |  | **Принял:** | |
| ФИО: | \_Цыпышев Т. А.\_\_\_\_\_ |  | ФИО: | \_Правдина А. Д.\_\_\_\_ |
| Группа: | \_ИУ5-21Б\_\_\_\_\_\_\_\_\_\_ |  | Должность: | \_Преподаватель\_\_\_\_ |
| Дата: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  | Дата: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Подпись: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  | Подпись: | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Москва, 2022 г.

**Постановка задачи**

**Цель работы:**

* создание программных объектов пользовательских типов с использованием классов;
* создание много-файловых проектов (заголовочный файл для класса, файл с определением методов класса и файл с main-функцией);
* ввод-вывод объектов пользовательских типов в файл;
* перегрузка операций потокового ввода-вывода (<<, >>), операций отношения (<, ==) для пользовательских типов;
* перегрузка конструктора копирования и операции присваивания;
* сортировка массивов объектов пользовательских типов, хранящихся в файле, с использованием перегруженных операций.

**Разработка алгоритма**

int main()

Эта функция - это реализация консольного интерфейса для работы с словарем "Statement".

В начале определяется константа **max\_size**, которая указывает на максимальное количество элементов, которые можно хранить в словаре.

Далее создается указатель на массив типа **Statement** с помощью оператора **new**, чтобы выделить память под хранение элементов словаря. Затем определяется переменная **len**, которая представляет текущее количество элементов в словаре. В начале программы **len** равно 0.

Далее программа начинает бесконечный цикл **while (true)**, в котором пользователю предлагается выбрать действие из меню. Если пользователь вводит число от 1 до 7, то вызывается соответствующая функция:

* **reading()** - чтение элементов из файла и добавление их в словарь;
* **logging()** - вывод элементов словаря в файл;
* **sorting()** - сортировка элементов словаря;
* **adding()** - добавление нового элемента в словарь;
* **removing()** - удаление элемента из словаря;
* **viewing()** - просмотр элементов словаря;
* **help()** - вывод справки.

Если пользователь выбирает ввод некорректных значений, программа выводит сообщение об ошибке и предлагает выбрать элемент меню еще раз.

После выполнения действия программа выводит сообщение с просьбой выбрать следующее действие.

Если пользователь выбирает "8", программа завершает работу, освобождает память, выделенную под массив элементов, с помощью оператора **delete[]**, и возвращает 0.

void reading(Statement \*universe, int \*len)

Функция **reading** читает данные из файла "database.txt" и заполняет ими массив **dictionary** типа **Statement**.

Параметры функции:

* **dictionary** - указатель на массив типа **Statement**, который нужно заполнить данными из файла;
* **len** - указатель на переменную типа **int**, хранящую текущее количество элементов в массиве.

Функция открывает файл "database.txt" для чтения и проверяет, открыт ли файл успешно. Если файл не удалось открыть, функция выводит сообщение об ошибке.

Если файл открылся успешно, то функция начинает чтение данных из файла с помощью оператора **>>**. Чтение продолжается до тех пор, пока не будет достигнут конец файла. Каждый считанный элемент добавляется в массив **dictionary** и переменная **len** увеличивается на 1 после каждого добавления.

После завершения чтения функция закрывает файл и выводит сообщение о завершении чтения данных.

void logging(Statement \*universe, const int len)

Функция **logging** записывает содержимое массива **dictionary** в файл "database.txt".

Параметры функции:

* **dictionary** - указатель на массив типа **Statement**, содержимое которого нужно записать в файл;
* **len** - константа типа **int**, определяющая количество элементов в массиве.

Функция открывает файл "database.txt" для записи и проверяет, открыт ли файл успешно. Если файл не удалось открыть, функция выводит сообщение об ошибке.

Если файл открылся успешно, то функция начинает записывать элементы массива в файл с помощью оператора **<<**

После записи данных функция закрывает файл и выводит сообщение о завершении записи данных.

void sorting(Statement \*universe, const int len)

Функция **sorting** сортирует элементы массива **dictionary** в порядке убывания.

Параметры функции:

* **dictionary** - указатель на массив типа **Statement**, который нужно отсортировать;
* **len** - константа типа **int**, определяющая количество элементов в массиве.

Для сортировки функция использует алгоритм пузырьковой сортировки. Внешний цикл проходит по всем элементам массива от первого до предпоследнего. Внутренний цикл проходит от первого элемента до последнего элемента, находящегося на расстоянии i от начала массива. На каждом шаге внутреннего цикла происходит сравнение двух соседних элементов. Если значение левого элемента меньше значения правого элемента, то они меняются местами.

После сортировки функция выводит сообщение о завершении сортировки.

void adding(Statement \*universe, const int len)

Функция **adding** добавляет новый элемент в массив **dictionary**.

Параметры функции:

* **dictionary** - указатель на массив типа **Statement**, в который нужно добавить новый элемент;
* **len** - константа типа **int**, определяющая количество элементов в массиве.

Функция выводит сообщение с запросом на ввод данных для нового элемента и считывает их с помощью оператора **>>**. Затем новый элемент добавляется в конец массива. После этого функция выводит сообщение об успешном добавлении нового элемента в массив.

void removing(Statement \*universe, const int len)

Функция **removing** удаляет элемент из массива **dictionary**.

Параметры функции:

* **dictionary** - указатель на массив типа **Statement**, из которого нужно удалить элемент;
* **len** - константа типа **int**, определяющая количество элементов в массиве.

Функция выводит сообщение с запросом на ввод номера удаляемого элемента и считывает его с помощью оператора **>>**. Затем функция проверяет, что введенное число является допустимым индексом элемента в массиве и, если это так, удаляет элемент из массива, перемещая последний элемент вместо удаленного. После этого функция выводит сообщение об успешном удалении элемента из массива. Если введенное число не является допустимым индексом элемента в массиве, функция выводит сообщение об ошибке.

void viewing(Statement \*universe, int len)

Функция **viewing** выводит на консоль содержимое базы данных в виде таблицы. Она принимает в качестве параметров указатель на массив объектов типа **Statement** и его длину.

Переменные, используемые в функции:

* **dictionary** - указатель на массив объектов типа **Statement**, содержащий записи базы данных
* **len** - длина массива **dictionary** (количество записей в базе данных)

void help()

Функция **help()** выводит на экран список доступных операций с кратким описанием каждой из них. Она не принимает никаких параметров и не возвращает результатов.

Переменные не используются в этой функции.

class Statement

Класс Statement представляет запись о состоянии дела.

Переменные класса:

* name: указатель на символьный массив, содержащий имя состояния дела (максимальная длина имени - 255 символов);
* type: целое число, указывающее тип состояния дела;
* status: булево значение, указывающее на текущий статус состояния дела;
* estimation: целое число, указывающее оценку времени, необходимого для завершения состояния дела.

Класс содержит конструкторы, деструктор, перегрузки операторов ввода/вывода, операторы сравнения и геттеры/сеттеры для доступа к переменным класса.

**Текст программы**

**main.cpp**

*/// Copyright 2022 ttsypyshev <ttsypyshev01@gmail.com>*#include <iostream>  
#include "../include/class.h"  
#include "../include/functions.h"  
  
int main() {  
 const size\_t max\_size = 100;  
 Statement \*dictionary = new Statement[max\_size];  
 int len = 0;  
  
 int item = 7;  
 while (true) {  
 if (item == 1) reading(dictionary, &len);  
 else if (item == 2) logging(dictionary, len);  
 else if (item == 3) sorting(dictionary, len);  
 else if (item == 4) adding(dictionary, len++);  
 else if (item == 5) removing(dictionary, len--);  
 else if (item == 6) viewing(dictionary, len);  
 else if (item == 7) help();  
 else if (item == 8) return 0;  
 else { std::cout << "Incorrect values have been entered. Enter them again\n"; }  
 std::cout << "\nSelect a menu item (1-8):\n";  
 std::cin >> item;  
 }  
  
 delete[] dictionary;  
}

**class.h**

*/// Copyright 2022 ttsypyshev <ttsypyshev01@gmail.com>*#ifndef **INF\_LAB\_2\_CLASS\_H**#define **INF\_LAB\_2\_CLASS\_H**#include <cstring>  
#include <iostream>  
#include <fstream>  
#include <iomanip>  
  
class Statement {  
private:  
 char \*name;  
 unsigned type;  
 bool status;  
 unsigned estimation;  
public:  
 Statement() {  
 name = new char[1];  
 strcpy(name, "");  
 type = 0;  
 status = false;  
 estimation = 0;  
 }  
  
 Statement(char \*name\_, unsigned type\_, bool status\_, unsigned estimation\_) {  
 name = new char[strlen(name\_) + 1];  
 strcpy(name, name\_);  
 type = type\_;  
 status = status\_;  
 estimation = estimation\_;  
 }  
  
 Statement(Statement &other) {  
 name = new char[strlen(other.name) + 1];  
 strcpy(name, other.name);  
 type = other.type;  
 status = other.status;  
 estimation = other.estimation;  
 }  
  
 ~Statement() { delete[] name; }  
  
 Statement &operator=(Statement &other) {  
 if (this != &other) {  
 delete[] name;  
 name = new char[strlen(other.name) + 1];  
 strcpy(name, other.name);  
 type = other.type;  
 status = other.status;  
 estimation = other.estimation;  
 }  
 return \*this;  
 }  
  
 friend std::istream &operator>>(std::istream &input, Statement &statement) {  
 char \*name\_ = new char[256];  
 input >> name\_ >> statement.type >> statement.status >> statement.estimation;  
 statement.set\_name(name\_);  
 delete[] name\_;  
 return input;  
 }  
  
 friend std::ostream &operator<<(std::ostream &output, const Statement &statement) {  
 output << "|" << std::setw(16) << statement.name  
 << "|" << std::setw(10) << statement.type  
 << "|" << std::setw(8) << statement.status  
 << "|" << std::setw(12) << statement.estimation  
 << "|\n";  
 return output;  
 }  
  
 friend std::ifstream &operator>>(std::ifstream &input, Statement &statement) {  
 char \*name\_ = new char[256];  
 input >> name\_ >> statement.type >> statement.status >> statement.estimation;  
 statement.set\_name(name\_);  
 delete[] name\_;  
 return input;  
 }  
  
 friend std::ofstream &operator<<(std::ofstream &output, const Statement &statement) {  
 output << statement.name << " " << statement.type << " " << statement.status << " " << statement.estimation;  
 return output;  
 }  
  
 friend bool operator<(const Statement &statement1, const Statement &statement2) {  
 return statement1.type < statement2.type;  
 }  
  
 friend bool operator==(const Statement &statement1, const Statement &statement2) {  
 return statement1.type == statement2.type;  
 }  
  
 char \*get\_name() const { return name; }  
 unsigned get\_type() const { return type; }  
 bool get\_status() const { return status; }  
 unsigned get\_estimation() const { return estimation; }  
  
 void set\_name(const char \*name\_) {  
 delete[] name;  
 size\_t len = strlen(name\_) + 1;  
 name = new char[len];  
 strcpy\_s(name, len, name\_);  
 }  
 void set\_type(unsigned type\_) { type = type\_; }  
 void set\_status(bool status\_) { status = status\_; }  
 void set\_estimation(unsigned estimation\_) { estimation = estimation\_; }  
};  
  
#endif *//INF\_LAB\_2\_CLASS\_H*

**functions.ccp**

*/// Copyright 2022 ttsypyshev <ttsypyshev01@gmail.com>*#include <iostream>  
#include <fstream>  
#include "../include/class.h"  
  
void reading(Statement \*dictionary, int \*len) {  
 std::ifstream file("../other/database.txt", std::ios\_base::in);  
 if (!file) std::cout << "The file does not exits\n";  
 else {  
 while (!file.eof()) {  
 file >> dictionary[(\*len)++];  
 }  
 }  
 std::cout << "The database is reading\n";  
 file.close();  
}  
  
void logging(Statement \*dictionary, const int len) {  
 std::ofstream file("../other/database.txt", std::ios\_base::out);  
 if (!file) std::cout << "The file does not exits\n";  
 else {  
 for (int i = 0; i < len; ++i) {  
 file << dictionary[i];  
 if (i != len - 1) file << "\n";  
 }  
 }  
 std::cout << "The database is saved\n";  
 file.close();  
}  
  
void sorting(Statement \*dictionary, const int len) {  
 for (int i = 0; i < len - 1; ++i) {  
 for (int j = 0; j < len - i - 1; ++j) {  
 if (dictionary[j] < dictionary[j + 1]) {  
 Statement tmp(dictionary[j]);  
 dictionary[j] = dictionary[j + 1];  
 dictionary[j + 1] = tmp;  
 }  
 }  
 }  
 std::cout << "The database is sorted\n";  
}  
  
void adding(Statement \*dictionary, const int len) {  
 std::cout << "Enter the data of the new statement (separated by a space/enter):\n";  
 std::cin >> dictionary[len];  
 std::cout << "New statement added\n";  
}  
  
void removing(Statement \*dictionary, const int len) {  
 int i = 0;  
 std::cout << "Enter the statement number from the database:\n";  
 std::cin >> i;  
 if (--i > 0 && i <= len + 1) {  
 dictionary[i] = dictionary[len - 1];  
 std::cout << "New statement deleted\n";  
 } else { std::cout << "This statement is not in the database\n"; }  
}  
  
void viewing(Statement \*dictionary, int len) {  
 std::cout << "|" << std::setw(16) << "NAME"  
 << "|" << std::setw(10) << "TYPE"  
 << "|" << std::setw(8) << "STATUS"  
 << "|" << std::setw(12) << "ESTIMATION"  
 << "|\n";  
  
 for (int i = 0; i < len; ++i)  
 std::cout << dictionary[i];  
  
}  
  
void help() {  
 std::cout << "1 - Reading a database from a file \n"  
 << "2 - Writing a database to a file \n"  
 << "3 - Sorting the database \n"  
 << "4 - Adding a new statement \n"  
 << "5 - Removing a statement \n"  
 << "6 - Viewing the database \n"  
 << "7 - Help \n"  
 << "8 - Completion of the program \n";  
}

**functions.h**

*/// Copyright 2022 ttsypyshev <ttsypyshev01@gmail.com>*#ifndef **INF\_LAB\_2\_FUNCTIONS\_H**#define **INF\_LAB\_2\_FUNCTIONS\_H**void reading(Statement \*dictionary, int \*len);  
  
void logging(Statement \*dictionary, const int len);  
  
void sorting(Statement \*dictionary, const int len);  
  
void adding(Statement \*dictionary, const int len);  
  
void removing(Statement \*dictionary, const int len);  
  
void viewing(Statement \*dictionary, int len);  
  
void help();  
  
#endif *//INF\_LAB\_2\_FUNCTIONS\_H*

**database.txt**

N980c3 140 1 16

N981a77 12 0 17

N720c3 511 0 55

N1033c3 49 1 2

N1034 127 1 1

N1035l 2 0 9

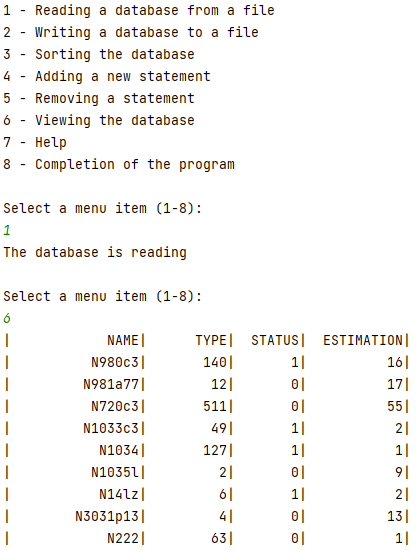
N14lz 6 1 2

N3031p13 4 0 13

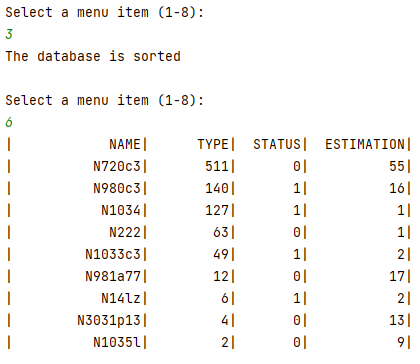
N222 63 0 1

**Тестирование функций**

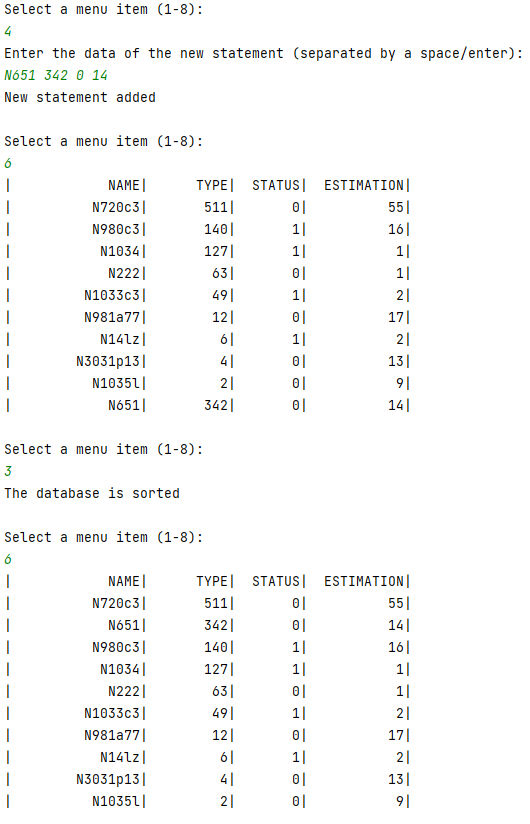
**Чтение из файла**

****

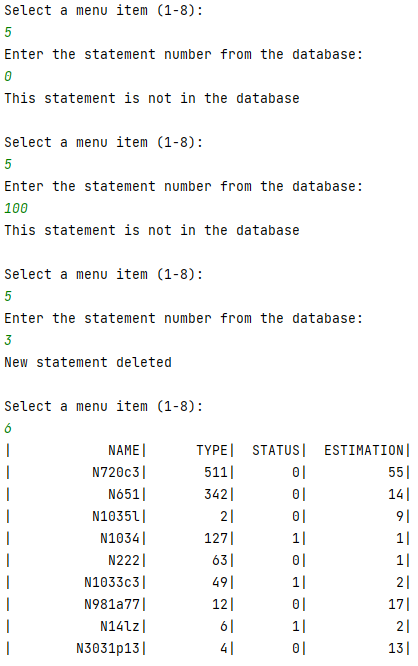
**Сортировка**

****

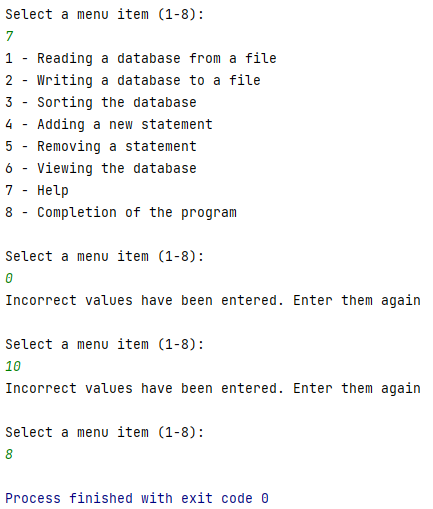
**Добавление**

****

**Удаление**

****

**Работа с меню**

****

**Вывод**

При выполнении данной работы я научился создавать программные объекты пользовательских типов с использованием классов, работать с много-файловыми проектами, вводить и выводить объекты пользовательских типов в файл, а также перегружать операции потокового ввода-вывода (<<, >>), операции отношения (<, ==), конструктор копирования и операцию присваивания.