**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

Кафедра «Информационные системы»

Курсовая РАБОТА

**по дисциплине «Программирование»**

Тема: **"ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ С++. СТРУКТУРЫ ДАННЫХ"**

|  |  |  |
| --- | --- | --- |
| Студент гр. 0323 |  | Наумов А.С. |
| Преподаватель |  | Глущенко А.Г. |

Санкт-Петербург

2021

**курсовая работа**

|  |  |  |
| --- | --- | --- |
| Студент: Голубцов В.В | | |
| Группа: 0323 | | |
| Тема проекта: "Основы программирования на языке С++. Структуры данных" | | |
| Исходные данные: описание каждого подраздела курса и технические задания программ на информационно-обучающем сайте Stepik. | | |
| Содержание пояснительной записки:  «Аннотация», «Содержание», «Введение», «Проектные работы», «Заключение», «Список использованных источников», «Программный код». | | |
|  | | |
| Дата выдачи задания: 21.2.2021 | | |
| Дата сдачи реферата: 07.06.2021 | | |
| Дата защиты реферата: 07.6.2021 | | |
| Студент |  | Наумов А.С. |
| Преподаватель |  | Глущенко А.Г. |

**Аннотация**

Курсовой проект состоял из обучающих разделов "Основы алгоритмизации и программирования на языке С++". В каждом разделе были подразделы с описанием главной задачи и тестовыми промежутками в виде написания небольшой программы или теста. Так всего было 8 обучающих подразделов, курсовая работа и 3 контрольные точки. Проект содержал в себе 4 проектных работы, в которых требовалось написать 4 программы по техническому заданию каждой из работы. По итогу были написаны все 4 проектные работы, выполнено >80% курса степика и выполнена курсовая работа в виде объединения всех проектных работ в один.

**Содержание**

1. Введение \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 5
2. Первая практическая работа \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 6
3. Вторая практическая работа \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 9
4. Курсовая работа \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 31
5. Заключение \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 34
6. Список использованных источников \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 35
7. Программный код практических работ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 36

**введение**

Целью работы заключалось обучение программирования и понимая работы программ на языке C++.

Требовалось по ходу курса выполнять различные тестовые промежуточные точки и выполнять проектные работы по пройденному разделу.

**1. ПервАЯ ПРАКТИЧЕСКАЯ РАБОТА**

**(ТИПЫ ДАННЫХ И ИХ ВНУТРЕННЕЕ ПРЕДСТАВЛЕНИЕ В ПАМЯТИ)**

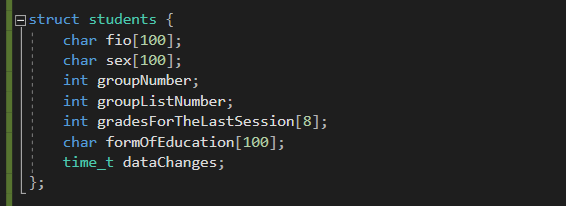
**Цель работы.**

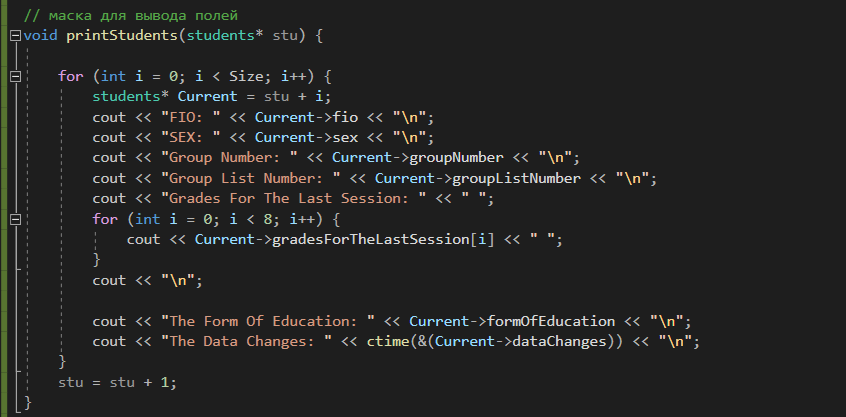
Изучение и организация структур; получение практических навыков работы со структурами; определение преимуществ и недостатков использования структур.

**Экспериментальные результаты.**

Программа разделена на две подпрограммы. В обоих из них алгоритм работы одинков, за исключением условий вывода данных на экран. Изначально данный берутся из текстового файла, преобразуются в структуру и дальше идет работа с ней.

Структуры выглядят следующим образом:





**Выводы.**

В ходе данной лабораторной работы я изучил работу с структурами и операциями над ними.

**2. ВТОРАЯ ПРАКТИЧЕСКАЯ РАБОТА**

**(ОДНОМЕРНЫЕ СТАТИЧЕСКИЕ МАССИВЫ)**

Цель работы.

изучение свойств и организация динамических массивов и двусвязных списков; получение практических навыков в работе с динамическими массивами и двусвязными списками; проведение сравнительной характеристики скорости вставки, получения и удаления элементов из них.

Постановка задачи.

Необходимо реализовать программу, которая выполняет следующие действия.

1.   Формирование целочисленного одномерного массива размерности N, где:

a) пользователь вводит количество элементов в массиве, который будет автоматически заполняться случайными числами (0 до 99);

б) пользователь вводит в консоль элементы массива, N определяется автоматически по количеству введенных элементов;

в) \* массив считывается с файла, N определяется как количество элементов массива в файле.

2.   Определение скорости создания динамического массива п. 1.

3.   Вставка, удаление и получение элемента массива. Удаление и получение элемента необходимо реализовать по индексу и по значению.

4.   Определение скорости вставки, удаления и получения элемента массива п. 3.

5.   Формирование двусвязного списка размерности N, где:

a) пользователь вводит количество элементов в списке, который будет автоматически заполняться случайными числами (0 до 99);

б) пользователь вводит в консоль элементы списка, N определяется автоматически по количеству введенных элементов;

в) \* список считывается с файла, N определяется как количество элементов списка в файле.

6.   Определение скорости создания двусвязного списка п. 5.

7.   Вставка, удаление и получение элемента двусвязного списка. Удаление и получение элемента необходимо реализовать по индексу и по значению.

8.   Определение скорости вставки, удаление и получения элемента двусвязного списка п. 7.

Должна быть возможность запуска каждого пункта многократно, если есть возможность (если в списке/массиве нет элементов, то нельзя ничего удалить и об этом нужно сообщить пользователю). Необходимо сравнить результаты. Для этого пункты 1–4 и 5–8 должны принимать одинаковые значения.

**Основные теоретические положения.**

## Одномерные динамические массивы

Для того чтобы создать в динамической области некоторый объект, необходима одна обычная переменная-указатель (не динамическая переменная). Сколько таких объектов понадобится для одновременной обработки, столько необходимо иметь обычных переменных-указателей. Таким образом, проблема задач неопределенной размерности созданием одиночных динамических объектов решена быть не может.

Решить эту проблему поможет возможность создавать в динамической области памяти массивы объектов с таким количеством элементов, которое необходимо в данный момент работы программы, т. е. создание динамических массивов. Действительно, для представления массива требуется всего одна переменная-указатель, а в самом массиве, на который ссылается этот указатель, может быть столько элементов, сколько требуется в данный момент времени.

Очень часто в процессе работы программы требуется изменять размеры уже созданных и заполненных данными массивов. Общий алгоритм решения этой задачи таков:

1.      создать исходный массив размерности N1 и заполнить его данными;  
2.      создать промежуточный массив размерности N2 (пусть N2 > N1);  
3.      скопировать данные из исходного массива в промежуточный массив;  
4.      освободить память от исходного массива;  
5.      переменной-указателю исходного массива присвоить значение переменной-указателя промежуточного массива;  
6.      заполнить новые элементы массива данными.

Для того чтобы получить двумерный массив, необходимо:

1)  создать одномерный динамический массив из **RowCount** указателей на базовый тип элементов массива (в нашем случае – указателей на тип **int**);  
2)  в цикле создать **RowCount** одномерных динамических массивов, каждый из которых содержит **ColCount** элементов базового типа (в нашем  случае –  указателей на тип **int**), и адреса их первых элементов записать в соответствующие элементы «вертикального» массива.

Обычный одномерный массив определяется как указатель на базовый тип данных элементов этого массива. Базовым типом элементов этого массива являются указатели **int**\*. Для того чтобы определить указатель на указатель достаточно использовать следующую конструкцию: **(int\*)\*** или проще **int\*\***.

Для создания динамических двумерных массивов с другими базовыми типами элементов достаточно в предыдущих примерах заменить тип данных **int**, на необходимый тип данных. Ну, и конечно, изменить работу с элементами массива в соответствии с их типом данных. Обязательные места исправлений выделены красным цветом.

По аналогии с двумерными динамическими массивами можно создавать и массивы большей мерности.

Одним из недостатков односвязных списков является то, что узел (элемент списка) имеет указатель только на следующий элемент. Вернуться из текущего элемента к предыдущему явным способом невозможно.

Каждый узел двусвязного (двунаправленного) линейного списка содержит два поля указателей – на следующий и на предыдущий узлы. Указатель на предыдущий узел корня списка содержит нулевое значение. Указатель последнего узла также содержит нулевое значение.

Основные действия, производимые над узлами двусвязного линейного списка (ДЛС):

1)  инициализация списка;  
2)  добавление узла в список;  
3)  удаление узла из списка;  
4)  удаление корня списка;  
5)  вывод элементов списка;  
6)  вывод элементов списка в обратном порядке;  
7)  взаимообмен двух узлов списка.

Порядок действия очень похож на односвязный линейный список, но необходимо учитывать, что в двусвязном списке имеется два указателя: на следующий и предыдущий элементы.

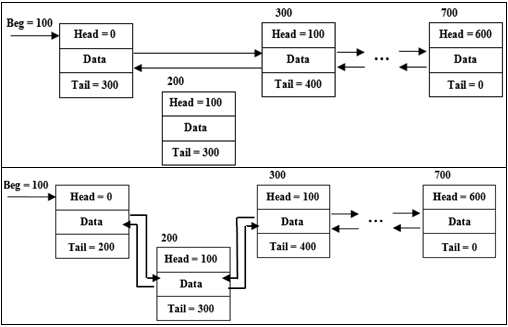
### Удаление элементов двусвязного списка

Удаление первого элемента и последнего элемента практически аналогично удалению элемента из односвязного списка. Нужно изменить значение указателя на первый элемент, обнулить значение указателя головы/хвоста, освободить динамическую память.

С удалением элемента из середины списка дело обстоит сложнее: необходимо проделать аналогичные операции по отношению к двум узлам, а не к одному.

### Вставка узла

Вставка нового узла в двусвязный линейный список проводится аналогично. Также присутствует три ситуации: вставка нового корня, вставка последним элементом и вставка узла в середину списка.

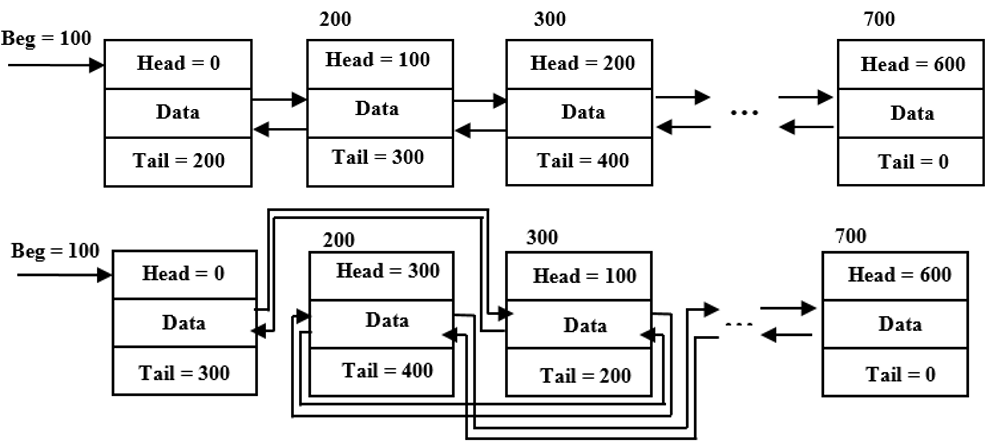


Вставка нового корневого узла отличается от удаления тем, что необходимо изменить **Head**-указатель прошлого корневого узла, указатель на корневой узел и правильно связать новый узел со списком.

Вставка узла в конец списка не должна вызывать затруднений: необходимо изменить **Tail**-указатель прежнего последнего элемента на адрес нового узла, затем правильно связать новый узел с двусвязным списком (**Head** – адрес на предыдущий элемент, **Tail** – NULL-указатель).

### Обмен элементов местами

Больше всего трудностей может возникнуть именно с взаимообменом элементов двусвязного списка. Главная проблема – возможная путаница в указателях. Нужно правильно изменить указатели всех затрагиваемых узлов.



Видно, сколько необходимо изменить адресов указателей, чтобы двумерный список работал корректно. Если хоть один указатель будет ссылаться неправильно, то список будет работать некорректно и может вовсе зациклиться. Причем программист этого может не заметить, если поле **Data** однородно или вовсе отсутствует. Поэтому (как и с любыми указателями) нужно быть внимательным.

**Обработка результатов эксперимента.**

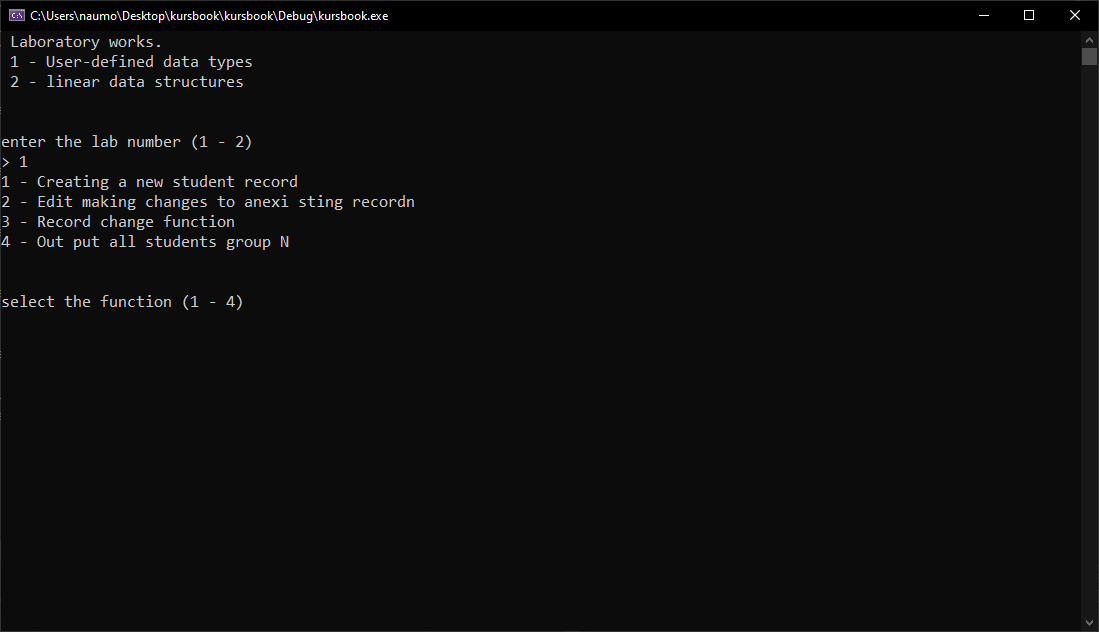


Рисунок Создание массива случайных чисел

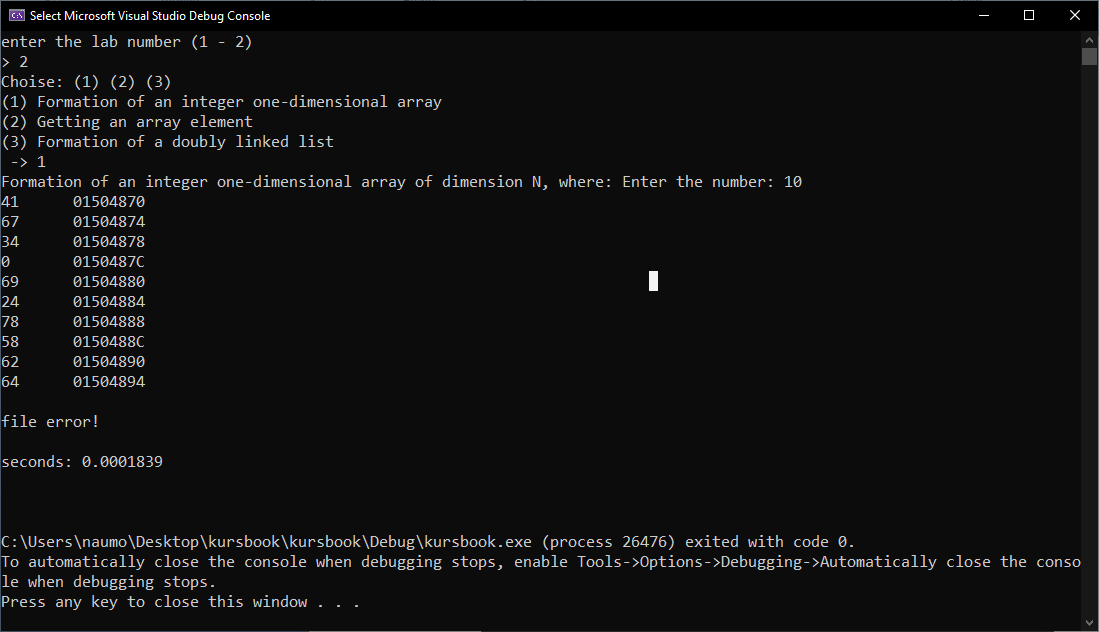


Рисунок Добавление элемента в массив

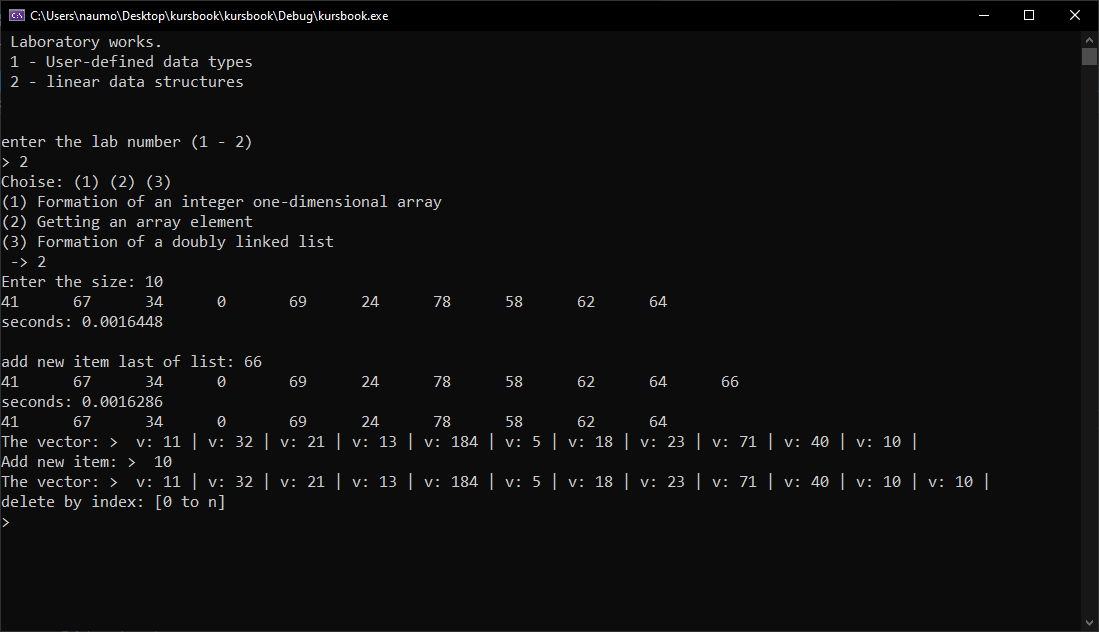


Рисунок Удаление элемента

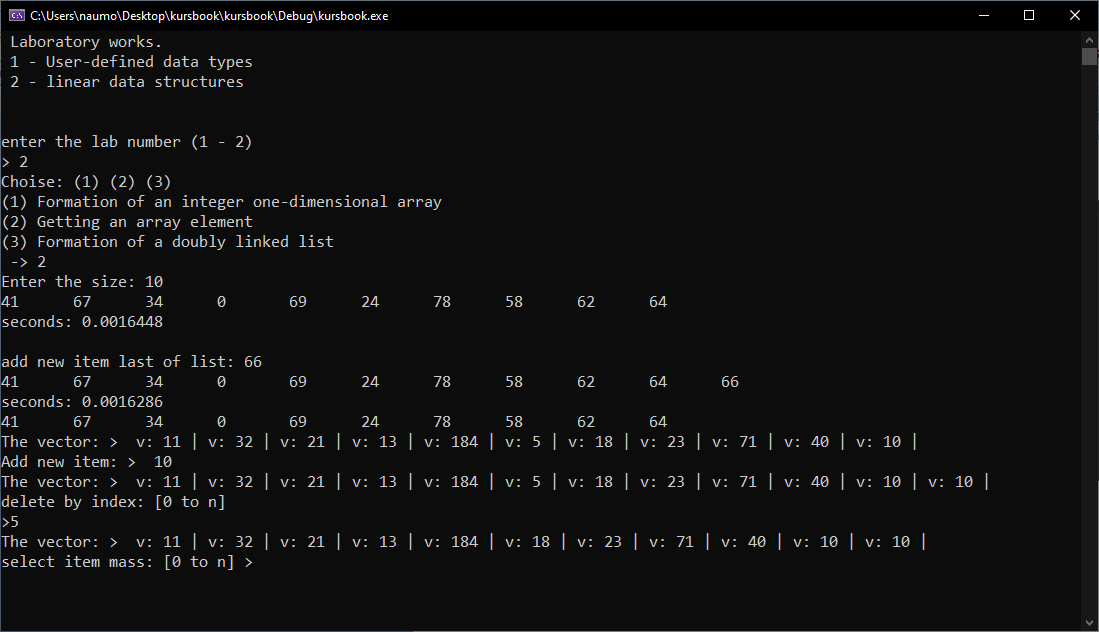


Рисунок Получение элемента

Выводы.

В ходе данной лабораторной работы я изучил работу с динамическими массивами и двусвязными линейными списками, а так же выполнил различные операции над ними.

**КУРСОВАЯ работа**

Цель работы.

Совместить все программы практических работ в одну и реализовать интерфейс переключения между ними.

Постановка задачи.

Необходимо объединить все 2 лабораторные работы в единый проект. Нужно добавить инфраструктуру переключения между заданиями (интерактивное меню).

**Выводы.**

Такое решение удобнее, чем закрывать одну программу и открывать другую, чтобы воспроизвести ее.

**заключение**

Курс был довольно насыщен практической частью закрепления пройденного материала, что хорошо сказалось на обучении и усвоении алгоритмов программирования на языке С++. Было изучено 3 основых разделов в которых выполнены все тестовые точки. Написано 2 программы для практических точек курса и в заключении собран курсовой проект состоящий из одной программы, выполняющая роль агрегатора всех практических программ, в которой можно переключаться между ними.

**список использованных источников**

•https://stepik.org/ **приложение А**

**Программный код**

**Объединение двух работ:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <iomanip>

#include <ctime>

#include <cstring>

#include <cstring>

#include <chrono>

#include <string>

#include <random>

#include <vector>

#include <limits>

#include <fstream>

using namespace std;

using namespace chrono;

struct students {

char fio[100];

char sex[100];

int groupNumber;

int groupListNumber;

int gradesForTheLastSession[8];

char formOfEducation[100];

time\_t dataChanges;

};

struct List {

int data;

List\* next;

List\* prev;

int list[10];

};

int Size;

// Создание новой записи о студенте

void createStudents() {

students stu;

string path = "test\_2.txt";

fstream fs;

fs.open(path, fstream::in | fstream::out | fstream::app);

if (!fs.is\_open()) {

cout << "Eror!";

}

else {

cout << "\nEnter the FIO: ";

cin >> stu.fio;

cout << "\nEnter the sex: ";

cin >> stu.sex;

cout << "\nEnter the Number group: ";

cin >> stu.groupNumber;

cout << "\nEnter the Group List Number: ";

cin >> stu.groupListNumber;

cout << "\nEnter the Grades For The Last Session: ";

for (int i = 0; i < 8; i++) {

cin >> stu.gradesForTheLastSession[i];

}

cout << "\nEnter the Form Of Education: ";

cin >> stu.formOfEducation;

stu.dataChanges = time(NULL);

cout << "\n";

cout << "FIO: " << stu.fio << "\n";

cout << "SEX: " << stu.sex << "\n";

cout << "Group Number: " << stu.groupNumber << "\n";

cout << "Group List Number: " << stu.groupListNumber << "\n";

cout << "Grades For The Last Session: " << stu.gradesForTheLastSession << "\n";

cout << "The Form Of Education: " << stu.formOfEducation << "\n";

cout << "The Data Changes: " << stu.dataChanges << "\n";

fs << stu.fio << '\n';

fs << stu.sex << '\n';

fs << stu.groupNumber << '\n';

fs << stu.groupListNumber << '\n';

for (int i = 0; i < 8; i++) {

fs << stu.gradesForTheLastSession[i] << " ";

}

fs << "\n";

fs << stu.formOfEducation << '\n';

fs << stu.dataChanges << '\n';

}

fs.close();

}

// edit makingChangesToAnexiStingRecord

void makingChangesToAnexiStingRecord() {

students stu;

string path = "test\_2.txt";

fstream fs;

fs.open(path, fstream::in | fstream::out | fstream::app);

if (!fs.is\_open()) {

cout << "Eror!";

}

else {

cout << "\nEdit FIO: ";

cin >> stu.fio;

cout << "\nEdit the sex: ";

cin >> stu.sex;

cout << "\nEdit the Number group: ";

cin >> stu.groupNumber;

cout << "\nEdit the Group List Number: ";

cin >> stu.groupListNumber;

cout << "\nEdit the Grades For The Last Session: ";

for (int i = 0; i < 8; i++) {

cin >> stu.gradesForTheLastSession[i];

}

cout << "\nEdit the Form Of Education: ";

cin >> stu.formOfEducation;

cout << "FIO: " << stu.fio << "\n";

cout << "SEX: " << stu.sex << "\n";

cout << "Group Number: " << stu.groupNumber << "\n";

cout << "Group List Number: " << stu.groupListNumber << "\n";

cout << "Grades For The Last Session: " << stu.gradesForTheLastSession << "\n";

cout << "The Form Of Education: " << stu.formOfEducation << "\n";

cout << "The Data Changes: " << stu.dataChanges << "\n";

fs << stu.fio << '\n';

fs << stu.sex << '\n';

fs << stu.groupNumber << '\n';

fs << stu.groupListNumber << '\n';

fs << stu.gradesForTheLastSession << '\n';

fs << stu.formOfEducation << '\n';

fs << stu.dataChanges << '\n';

}

fs.close();

}

// функция возврат заначения

students\* getAllDataFromFile() {

students\* List = NULL; //

ifstream file;

file.open("test\_2.txt");

while (!file.is\_open()) {

cout << "file error! \n";

return NULL;

}

Size = 0;

while (!file.eof()) {

List = (students\*)realloc(List, (Size + 1) \* sizeof(students));

students\* newStudent = List + Size;

file.getline(newStudent->fio, 100);

file.getline(newStudent->sex, 100);

file >> newStudent->groupNumber;

file.ignore(numeric\_limits<streamsize>::max(), '\n');

file >> newStudent->groupListNumber;

file.ignore(numeric\_limits<streamsize>::max(), '\n');

for (int i = 0; i < 8; i++) {

file >> newStudent->gradesForTheLastSession[i];

}

file.ignore(numeric\_limits<streamsize>::max(), '\n');

file.getline(newStudent->formOfEducation, 100);

file >> newStudent->dataChanges;

file.ignore(numeric\_limits<streamsize>::max(), '\n');

Size++;

}

file.close();

return List; // возращает указатель впамати

}

// маска для вывода полей

void printStudents(students\* stu) {

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

students\* Current = stu + i;

cout << "FIO: " << Current->fio << "\n";

cout << "SEX: " << Current->sex << "\n";

cout << "Group Number: " << Current->groupNumber << "\n";

cout << "Group List Number: " << Current->groupListNumber << "\n";

cout << "Grades For The Last Session: " << " ";

for (int i = 0; i < 8; i++) {

cout << Current->gradesForTheLastSession[i] << " ";

}

cout << "\n";

cout << "The Form Of Education: " << Current->formOfEducation << "\n";

cout << "The Data Changes: " << ctime(&(Current->dataChanges)) << "\n";

}

stu = stu + 1;

}

// функция изменения записи

void editPrintStudents(students\* stu) {

ofstream file;

string path = "test\_2.txt";

file.open(path, std::ios\_base::trunc);

if (!file.is\_open()) {

cout << "file error! \n";

}

else {

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

students\* Current = stu + i;

file << Current->fio << '\n';

file << Current->sex << '\n';

file << Current->groupNumber << '\n';

file << Current->groupListNumber << '\n';

for (int i = 0; i < 8; i++) {

file << Current->gradesForTheLastSession[i] << " ";

}

file << "\n";

file << Current->formOfEducation << '\n';

file << Current->dataChanges;

//file << ctime(&(Current->dataChanges));

if (!(i == Size - 1)) {

file << '\n';

}

}

file.close();

}

}

// функция записывает в память

void memoryEdit(students\* stu) {

students\* Current;

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

Current = stu + i;

cout << i + 1 << ". " << Current->fio << '\n';

}

cout << "Enter the students: ";

int number;

cin >> number;

Current = stu + number - 1;

cout << "FIO: ";

cin >> Current->fio;

cout << "SEX: ";

cin >> Current->sex;

cout << "Group Number: ";

cin >> Current->groupNumber;

cout << "Group List Number: ";

cin >> Current->groupListNumber;

cout << "Grades For The Last Session: ";

for (int i = 0; i < 8; i++) {

cin >> Current->gradesForTheLastSession[i];

}

cout << "The Form Of Education: ";

cin >> Current->formOfEducation;

cout << "The Data Changes: ";

cin.getline(Current->sex, 100);

editPrintStudents(stu);

}

//Вывод информации обо всех студентах группы N. N – инициализируется пользователем

void outputAllStudentsGroupN(students\* List) {

int n;

cout << "Enter the group number:" << endl;

cin >> n;

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

if (List[i].groupNumber == n) {

cout << List[i].fio << endl;

cout << List[i].sex << endl;

cout << List[i].groupNumber << endl;

cout << List[i].groupListNumber << endl;

cout << "Term grades:";

for (int j = 0; j < 8; j++) {

cout << List[i].gradesForTheLastSession[j] << " ";

}

cout << endl;

cout << List[i].formOfEducation << endl;

cout << ctime(&(List[i].dataChanges)) << endl;

}

}

}

void listTop(students\* List)

{

cout << "Top students:" << std::endl << std::endl;

// структура для хранения средних балов

struct sortList

{

unsigned int number;

double score;

};

sortList\* SortList = (sortList\*)malloc(sizeof(sortList) \* Size);

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

int sum = 0;

(SortList + i)->number = i;

for (int j = 0; j < 8; j++) {

sum = (sum + List[i].gradesForTheLastSession[j]);

(SortList + i)->score = (double)sum / 8;

}

}

int sorted = 0;

int currentMaxUnsorted = Size - 1;

while (!sorted)

{

sorted = 1;

for (int i = 0; i < currentMaxUnsorted; i++)

{

if ((SortList + i)->score - (SortList + i + 1)->score > 0)

{

sortList Buffer;

Buffer.number = (SortList + i)->number;

Buffer.score = (SortList + i)->score;

(SortList + i)->number = (SortList + i + 1)->number;

(SortList + i)->score = (SortList + i + 1)->score;

(SortList + i + 1)->number = Buffer.number;

(SortList + i + 1)->score = Buffer.score;

sorted = 0;

}

}

currentMaxUnsorted--; // последний уже отсортирован и проверсять не надо

}

for (unsigned int i = Size - 1, j = 0; i >= 0 && j < 3; i--, j++)

{

cout << List[(SortList + i)->number].fio << endl;

cout << List[(SortList + i)->number].sex << endl;

cout << List[(SortList + i)->number].groupNumber << endl;

cout << List[(SortList + i)->number].groupListNumber << endl;

cout << "Term grades:";

for (int j = 0; j < 8; j++) {

cout << List[(SortList + i)->number].gradesForTheLastSession[j] << " ";

}

cout << endl;

cout << List[(SortList + i)->number].formOfEducation << endl;

cout << ctime(&(List[(SortList + i)->number].dataChanges)) << endl;

}

std::cout << std::endl << std::endl;

free(SortList); // осовобождения памяти функции малок

}

/\*----------------------------------------------\*/

void getItemArray() {

int size;

cout << "Enter the number: ";

cin >> size;

if (size <= 100) {

int\* arr = new int[size];

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

arr[i] = rand() % 100;

}

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

cout << arr[i] << "\t" << arr + i << "\n";

}

delete[] arr;

}

else {

cout << "Error! ";

}

}

List\* getAllDataFromFile2() {

List\* listFile = NULL; //

ifstream file;

file.open("list.txt");

while (!file.is\_open()) {

cout << "file error! \n";

return NULL;

}

Size = 0;

while (!file.eof()) {

listFile = (List\*)realloc(listFile, (Size + 1) \* sizeof(List));

List\* newList = listFile + Size;

for (int i = 0; i < 10; i++) {

file >> newList->list[i];

}

file.ignore(numeric\_limits<int>::max(), '\n');

Size++;

}

file.close();

return listFile; // возращает указатель впамати

}

void getListFile(List\* N) {

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

List\* Current = N + i;

cout << "the array is read from the file, N is defined as the number of array elements in the file" << endl;

cout << "Array: " << " ";

for (int i = 0; i < 10; i++) {

cout << Current->list[i] << " ";

}

cout << "\n";

cout << "count item array: ";

cout << sizeof(Current->list) / sizeof(int);

}

N = N + 1;

}

void FillArray(int\* const arr, const int size) {

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

arr[i] = rand() % 100;

}

}

void ShowArray(const int\* const arr, const int size) {

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

cout << arr[i] << "\t";

}

cout << endl;

}

void push\_back(int\*& arr, int& size, const int value) {

int\* newArray = new int[size + 1];

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

newArray[i] = arr[i];

}

newArray[size] = value;

size++;

delete[] arr;

arr = newArray;

}

void pop\_back(int\*& arr, int& size) {

size--;

int\* newArray = new int[size];

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

newArray[i] = arr[i];

}

delete[] arr;

arr = newArray;

}

void editArray() {

vector <double> array = { 11, 32, 21, 13, 184, 5, 18, 23, 71, 40, 10 };

/\*for (int i = 0; i < 11; ++i) {

array[i] = rand() % 100;

}\*/

cout << "The vector: > " << " ";

for (int i = 0; i < array.size(); i++) {

cout << "v: " << array[i] << " | ";

}

cout << endl;

cout << "Add new item: > " << " ";

int addItem;

cin >> addItem;

array.push\_back(addItem);

cout << "The vector: > " << " ";

for (int i = 0; i < array.size(); i++) {

cout << "v: " << array[i] << " | ";

}

cout << endl;

cout << "delete by index: [0 to n]" << endl;

cout << ">";

int indexToRemove;

cin >> indexToRemove;

array.erase(array.begin() + indexToRemove);

cout << "The vector: > " << " ";

for (int i = 0; i < array.size(); i++) {

cout << "v: " << array[i] << " | ";

}

cout << endl;

cout << "select item mass: [0 to n] > " << " ";

int select;

cin >> select;

if (select < 12) {

cout << "v: " << array[select] << "\t";

}

else {

cout << "error!";

}

}

//inserts List at the front of the list

void insert\_front(struct List\*\* head, int new\_data)

{

//выделяю память для нового списка

struct List\* newList = new List;

//назначить данные в новый список

newList->data = new\_data;

//новый список является заголовком, а предыдущий - нулевым, так как мы добавляем впереди

newList->next = (\*head);

newList->prev = NULL;

//previous of head is new List

if ((\*head) != NULL)

(\*head)->prev = newList;

//заголовок указывает на новый список

(\*head) = newList;

}

/\* вставка нового списка после нового \*/

void insert\_After(struct List\* prev\_List, int new\_data)

{

if (prev\_List == NULL) {

cout << "Previous List is required , it cannot be NULL";

return;

}

struct List\* newList = new List;

newList->data = new\_data;

//установить следующий из newList на следующий из предыдущего списка

newList->next = prev\_List->next;

//установить следующий из предыдущего списка в новый список

prev\_List->next = newList;

//теперь установите prev для newList на prev List

newList->prev = prev\_List;

//установить предыдущий новый список рядом с новым списком

if (newList->next != NULL) {

newList->next->prev = newList;

}

}

//вставить новый список в конец списка

void insert\_end(struct List\*\* head, int new\_data)

{

//выделяю памать для списка

struct List\* newList = new List;

struct List\* last = \*head; //установить последнее значение списка в заголовок

//установить данные для нового списка

newList->data = new\_data;

//установка для следующего нового списка значение null

newList->next = NULL;

//если список пуст - создаем новый

if (\*head == NULL) {

newList->prev = NULL;

\*head = newList;

return;

}

//в противном случае перейдите по списку, чтобы перейти к последнему списку

while (last->next != NULL)

last = last->next;

//установить предпоследний в новый список

last->next = newList;

//установить последний на предыдущий в новом списке

newList->prev = last;

return;

}

// вывод

void displayList(struct List\* List) {

struct List\* last;

while (List != NULL) {

cout << List->data << "<==>";

last = List;

List = List->next;

}

if (List == NULL) {

cout << "NULL";

}

}

void dellList(struct List\* list) {

struct List\* prev, \* next;

prev = list->prev;

next = list->next;

if (prev != NULL) {

prev->next = list->next;

}

if (next != NULL) {

next->prev = list->prev;

}

free(list);

}

int main()

{

int choice;

cout << " Laboratory works.\n";

cout << " 1 - User-defined data types\n";

cout << " 2 - linear data structures\n";

cout << endl;

cout << endl;

cout << "enter the lab number (1 - 2)\n";

cout << "> ";

cin >> choice;

if (choice == 1) {

int select;

cout << "1 - Creating a new student record\n";

cout << "2 - Edit making changes to anexi sting recordn\n";

cout << "3 - Record change function \n";

cout << "4 - Out put all students group N \n";

cout << endl;

cout << endl;

cout << "select the function (1 - 4)\n";

cin >> select;

if (select == 1) {

createStudents();

}

if (select == 2) {

makingChangesToAnexiStingRecord();

}

if (select == 3) {

cout << "oppppsss :(";

}

if (select == 4) {

students\* stu\_1 = getAllDataFromFile();

printStudents(stu\_1);

}

}

if (choice == 2) {

cout << "Choise: (1) (2) (3)" << endl;

cout << "(1) Formation of an integer one-dimensional array" << endl;

cout << "(2) Getting an array element" << endl;

cout << "(3) Formation of a doubly linked list" << endl;

cout << " -> ";

int input;

cin >> input;

if (input == 1) {

cout << "Formation of an integer one-dimensional array of dimension N, where: ";

getItemArray();

cout << endl;

system\_clock::time\_point start = system\_clock::now();

List\* newl = getAllDataFromFile2();

getListFile(newl);

cout << "\n";

system\_clock::time\_point end = system\_clock::now();

duration<double> sec = end - start;

cout << "seconds: " << sec.count() << endl << endl;

}

// определение скорости создания динамического массива

if (input == 2) {

int size;

cout << "Enter the size: ";

cin >> size;

int\* arr = new int[size];

FillArray(arr, size);

system\_clock::time\_point start\_1 = system\_clock::now();

ShowArray(arr, size);

system\_clock::time\_point end\_1 = system\_clock::now();

duration<double> sec\_ = end\_1 - start\_1;

cout << "seconds: " << sec\_.count() << endl << endl;

int add;

cout << "add new item last of list: ";

cin >> add;

system\_clock::time\_point start\_2 = system\_clock::now();

push\_back(arr, size, add);

ShowArray(arr, size);

system\_clock::time\_point end\_2 = system\_clock::now();

duration<double> sec\_2 = end\_2 - start\_2;

cout << "seconds: " << sec\_2.count() << endl;

pop\_back(arr, size);

ShowArray(arr, size);

editArray();

}

cout << endl;

if (input == 3) {

struct List\* head = NULL;

int x, x2, x3, x4, x5;

cout << "Enter the x" << " ";

cin >> x;

cout << "Enter the x2" << " ";

cin >> x2;

cout << "Enter the x3" << " ";

cin >> x3;

cout << "Enter the x4" << " ";

cin >> x4;

cout << "Enter the x5" << " ";

cin >> x5;

// Insert 40 as last node

insert\_end(&head, x4);

// Insert 10 at the beginning.

insert\_front(&head, x2);

// Insert 30, after 20.

insert\_After(head->next, x3);

// insert 20 at the head

insert\_front(&head, x);

// Insert 50 at the end.

insert\_end(&head, x5);

cout << "Doubly linked list is as follows: " << endl;

displayList(head);

}

}

}