**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по практической работе №2**

**по дисциплине «Программирование»**

Тема: List

|  |  |  |
| --- | --- | --- |
| Студент гр. 3372 |  | Поконечный А.О |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2023

**Цель работы.**

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

**Основные теоретические положения.**

Структуры представляют собой группы связанных между собой, как правило, разнотипных переменных, объединенных в единый объект, в отличие от массива, все элементы которого однотипны. В языке C++ структура является видом класса и обладает всеми его свойствами. Чаще всего ограничиваются тем, как структуры представлены в языке С:

struct [имя\_типа] {

тип\_1 элемент\_1;

тип \_2 элемент\_2;

…

тип\_k элемент\_k;

} [ список\_описателей ];

Описание структуры начинается ключевым словом struct. Каждая входящая в структуру переменная называется членом (полем, элементом) структуры и описывается типом данных и именем. Поля структуры могут быть любого типа данных. Их количество не лимитировано.

Вся эта конструкция является инструкцией языка программирования, поэтому после нее всегда должен ставиться символ ‘;’.

При описании структуры память для размещения данных не выделяется. Работать с описанной структурой можно только после того, как будет определена переменная (переменные) этого типа данных, только при этом компилятор выделит необходимую память.

Для инициализации структуры значения ее элементов перечисляют в фигурных скобках в порядке их описания:

struct complex{

float real, im;

} data [2][2] = {

{{1,1}, {2,2}},

{{3,3}, {4,4}}

};

Все поля структурных переменных располагаются в непрерывной области памяти одно за другим. Общий объем памяти, занимаемый структурой, равен сумме размеров всех полей структуры. Для определения размера структуры следует использовать инструкцию sizeof().

Для того чтобы записать данные в структурную переменную, необходимо каждому полю структуры присвоить определенное значение. Для этого необходимо использовать оператор ‘’ («точка»):

struct Stack { // Cтек

float arr[100];

short topIndex;

};

…

Stack stack; // Объявляем переменную типа Stack

Stack.arr[0] = 1;

…

При доступе к определенному полю его следует рассматривать как обычную переменную, тип данных которой соответствует типу этого поля. Поля структур могут участвовать в качестве операндов любых выражений, допускающих использование операндов соответствующего типа данных.

Копирование данных из одной структурной переменной в другую осуществляется простой операцией присваивания, независимо от количества полей и размера структуры (это можно делать только в том случае, когда обе переменные одного и того же типа).

В программировании очень часто используются такие конструкции, как массивы структур. Например, сведения о студентах некоторой учебной группы можно хранить в массиве студентов:

t\_Student Gruppa\_N [30];

Был определен 30-элементный массив, каждый элемент которого предназначен для хранения данных одного студента. Получение доступа к данным некоторого студента из группы *N* осуществляется обычной индексацией переменной массива. Поскольку поля структуры могут быть любого типа данных, то они в свою очередь могут быть другой структурой или массивом других структур:

struct Stud

{

char FN[100];

short listNumber;

};

struct Group

{

int groupNumber;

short students;

Stud stud[30];

};

Но в структуре поля нельзя использовать элемент, тип которого совпадает с типом самой структуры, так как рекурсивное использование структур запрещено.

Любая структурная переменная занимает в памяти определенное положение, характеризующееся конкретным адресом. Для работы с адресами структурных переменных (как и для простых переменных) можно использовать указатели. Указатели на структурные переменные определяются точно так же, как и для обычных переменных. Разыменование указателя (обращение к данным по адресу, хранящемуся в указателе) осуществляется также обычным образом.

Через указатели можно работать с отдельными полями структур. Для доступа к полю структуры через указатель используется оператор ‘’ («стрелка»), а не «точка».

Структуры можно использовать в качестве параметров функций, как и обычные переменные. Для структур поддерживаются все три механизма передачи данных: по значению, через указатели и по ссылке.

Передачу структур в функции по значению необходимо использовать аккуратно:

void WriteStudent ( t\_Student S )

{

cout << "Фамилия: " << S.Fam << endl;

cout << "Имя: " << S.Name << endl;

cout << "Год рождения: " << S.Year << endl;

if ( S.Sex )

cout << "Пол: " << "М\n";

else

cout << "Пол: " << "Ж\n";

cout << "Средний балл: " << S.Grade << endl;

}

Вызов такой функции сопровождается дополнительным расходом памяти для создания локальной переменной *S*и дополнительными затратами времени на физическое копирование данных из аргумента в параметр *S*. Учитывая то, что объем структур может быть очень большим, эти дополнительные затраты вычислительных ресурсов могут быть чрезмерными.

Предпочтительно использование передачи структуры по указателю или ссылке:

void WriteStudent ( t\_Student \*S )

{

cout << "Фамилия: " << S -> Fam << endl;

cout << "Имя: " << S -> Name << endl;

cout << "Год рождения: " << S -> Year << endl;

if ( S -> Sex )

cout << "Пол: " << "М\n";

else

cout << "Пол: " << "Ж\n";

cout << "Средний балл: " << S -> Grade << endl;

}

Фактической передачи данных в функцию не осуществляется. Дополнительные затраты памяти для создания локальной переменной небольшие – это адрес памяти (4 байта, независимо от размера самой структуры). Вызов такой функции будет происходить быстрее, а расход памяти будет существенно меньше, чем при передаче данных по значению.

Передача по ссылке по эффективности эквивалентна передаче данных через указатель. Однако, поскольку при передаче данных по ссылке все адресные преобразования берет на себя компилятор, существенно упрощается программирование действий со структурами. При использовании ссылочных параметров структурных типов доступ к членам структуры осуществляется обычным способом – с помощью оператора «точка».

Недостатком этих способов является то, что случайные изменения значений полей структуры внутри функции отразятся на значении аргумента после окончания работы функции. Если необходимо предотвратить изменения переданных по адресу аргументов, можно при определении соответствующего параметра объявить его константой (использовать спецификатор const).

**Постановка задачи.**

Необходимо создать массив структур, содержащий информацию о студентах: ФИО, пол, номер группы, номер в списке группы, оценки за прошедшую сессию (всего 3 экзамена и 5 дифференцированных зачетов), форма обучения, отметка времени о внесении или изменении данных. Ввод и изменение данных обо всех студентах должен осуществляться в файл students.

Написать функции, реализующие операции со структурами (ввод данных с клавиатуры):

1.   Создание новой записи о студенте.

2.   Внесение изменений в уже имеющуюся запись.

3.   Вывод всех данных о студентах.

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

5.   Вывод топа самых успешных студентов с наивысшим по рейтингу средним баллом за прошедшую сессию.

6.   Вывод количества студентов мужского и женского пола.

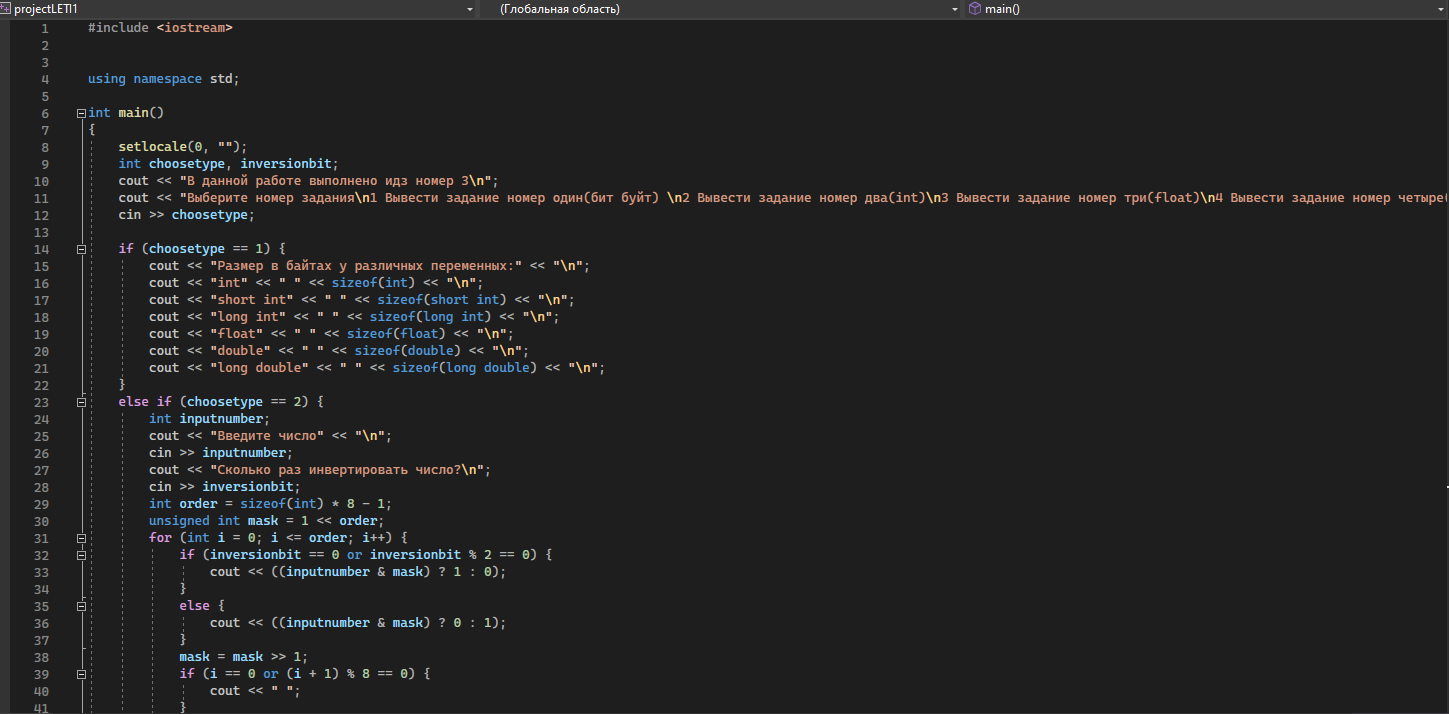
7.   Определение количества студентов, которые будут получать стипендию (стипендия начисляется, если у студента нет троек и очная форма обучения).

8.   Вывод данных о студентах, которые не получают стипендию; учатся только на «хорошо» и «отлично»; учатся только на «отлично»;

9.   Вывод данных о студентах, имеющих номер в списке – k.

10.   Вывод всех записей, сделанных в день, который введет пользователь. Вывод всех записей, сделанных после полудня. Вывод всех записей, сделанных до полудня.

**Выполнение работы.**

**Код программы представлен в приложении А**.

Часть кода

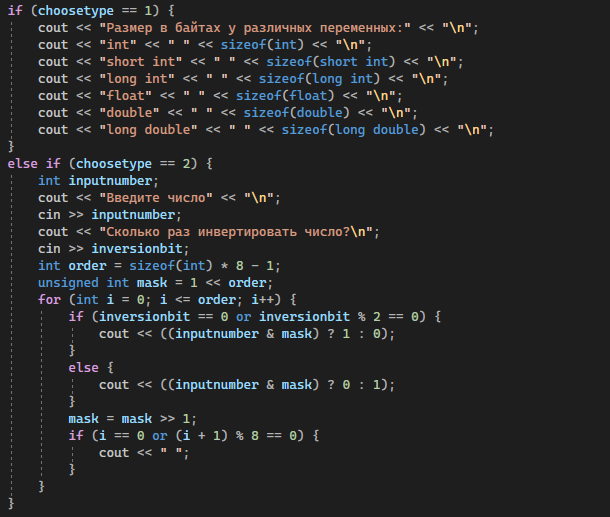
**Блок описания кода и использованных алгоритмов**

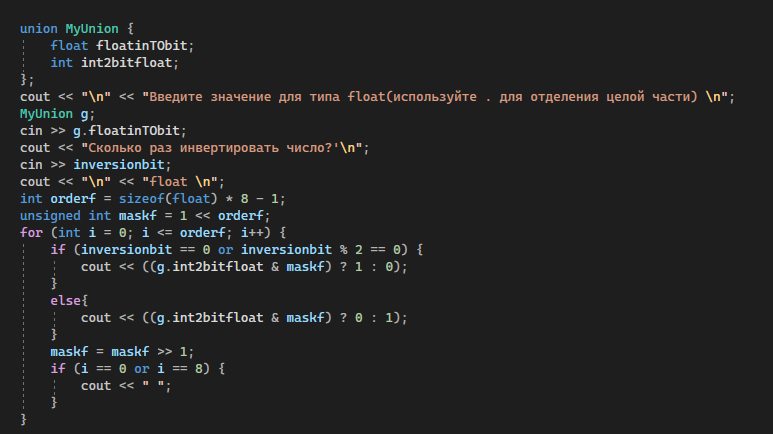
Использованы cin, cout, if, else, union

Пользователю даётся возможность выбрать то, что он хочет сделать.

Ему не требуется пользоваться всеми функциями программы для получения чего-то нужного.

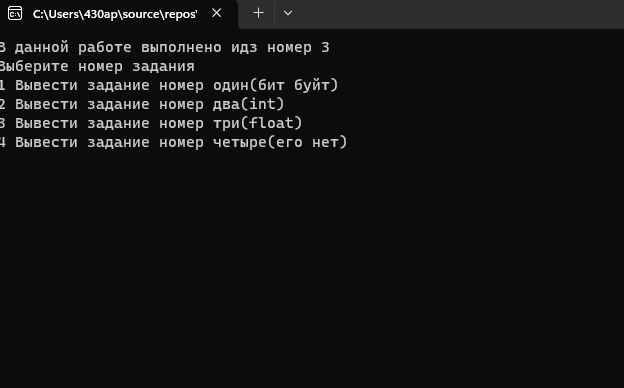
Значения пользователь вводит сам во время работы программы.

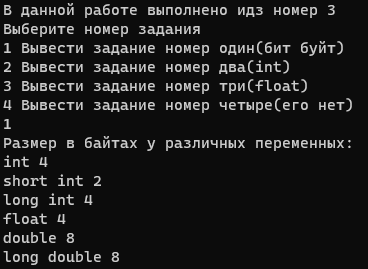




**Блок скриншотов работы программы**

Выбор задания

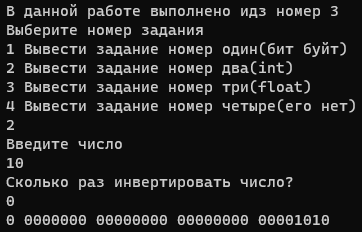




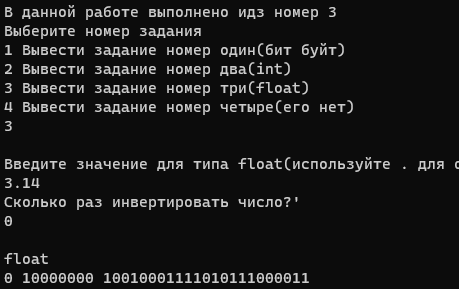
**Блок таблицы с тестовыми данными**

Данные пользователь вводит сам.

В данной ситуации введено число 10



В данной ситуации мы запрашиваем у программы двоичное представление типа данных float. Мы просим вывести у программы значение числа Пи, а именно 3.14.



**Выводы.**

Выполнив данную работу мы поняли, как представляется число в памяти компьютера при использовании С++, как объединить в одну ячейку памяти несколько переменных. Так же мы усовершенствовали свои навыки использования конструкций if, else, list.

Приложение А

рабочий код

#include <iostream>

#include <vector>

#include <string>

#include <sstream>

#include <iomanip>

#include <chrono>

using namespace std;

struct list {

int data;

list\* next;

list\* prev;

};

void createListRND(list\*& curr, int& length) {

list\* next = nullptr;

cout << "Enter length:";

cin >> length;

auto start = std::chrono::high\_resolution\_clock::now();

for (int i = 1; i <= length; ++i)

{

curr = new list;

curr->data = rand() % 100;

curr->next = next;

if (next) {

next->prev = curr;

}

next = curr;

}

curr->prev = 0;

auto stop = std::chrono::high\_resolution\_clock::now();

auto duration = std::chrono::duration\_cast<std::chrono::microseconds>(stop - start);

cout << "Time taken by function: " << duration.count() << " microseconds" << endl;

}

void addToList(list\*&start, int& length) {

int newElIndex = -100;

list\* next = nullptr;

cout << "Enter index of element(new will be added after this)" << endl;

cout << "Or 1, new element will take place at first position " << endl;

while (newElIndex < 1 || newElIndex > length) {

cout << "0<index<=" << length << endl;

cin >> newElIndex;

}

newElIndex -= 1;

auto startTime = chrono::high\_resolution\_clock::now();

list\* curr = start;

if (newElIndex) {

for (int i = 1; i <= newElIndex; ++i) {

curr = curr->next;

}

next = curr->next;

}

else {

next = start;

}

list\* currnew = new list;

newElIndex ? currnew->prev = curr : currnew->prev = nullptr;

newElIndex ? curr->next = currnew : start = currnew;

cout << "Enter value for new element" << endl;

cin >> currnew->data;

if (next) {

currnew->next = next;

next->prev = currnew;

}

else

currnew->next = nullptr;

auto stop = chrono::high\_resolution\_clock::now();

auto duration = chrono::duration\_cast<chrono::microseconds>(stop - startTime);

cout << "Time taken adding new element: " << duration.count() << " microseconds" << endl;

length += 1;

}

void showToList(list\* start, int length) {

list\* curr = start;

int i = 1;

cout << "/////////////" << endl;

while (curr) {

cout<<"index "<< i<<" " << right << setw(7) << curr->data << endl;

curr = curr->next;

i += 1;

}

cout << "/////////////" << endl;

}

vector<int> readNumbersFromConsole() {

cout << "Enter numbers (separated by spaces): ";

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

string input;

getline(cin, input);

vector<int> numbers;

stringstream ss(input);

int number;

while (ss >> number) {

numbers.push\_back(number);

}

return numbers;

}

void createListInputMainFunc(list\*& curr, int & length) {

list\* next = nullptr;

vector<int> numbers = readNumbersFromConsole();

auto start = std::chrono::high\_resolution\_clock::now();

for (int number : numbers) {

curr = new list;

length += 1;

curr->data = number;

curr->next = next;

if (next) {

next->prev = curr;

}

next = curr;

}

curr->prev = 0;

auto stop = std::chrono::high\_resolution\_clock::now();

auto duration = std::chrono::duration\_cast<std::chrono::microseconds>(stop - start);

cout << "Time taken by function: " << duration.count() << " microseconds" << endl;

}

void swaplist(list\*& start,int length) {

cout << "Enter index of first element to swap" << endl;

int index1 = 0;

cin >> index1;

cout << "Enter index of second element to swap" << endl;

int index2 = 0;

cin >> index2;

if (index1 == index2 || index1 < 0 || index1 > length || index2 < 0 || index2 > length) return;

list\* node1 = nullptr, \* node2 = nullptr, \* prev1 = nullptr, \* prev2 = nullptr;

list\* curr = start;

int count = 0;

index1 -= 1;

index2 -= 1;

auto start\_time = std::chrono::high\_resolution\_clock::now();

while (curr != nullptr) {

if (count == index1) {

node1 = curr;

if (curr->prev != nullptr) prev1 = curr->prev;

}

else if (count == index2) {

node2 = curr;

if (curr->prev != nullptr) prev2 = curr->prev;

}

curr = curr->next;

count++;

}

if (node1 != nullptr && node2 != nullptr) {

if (prev1 != nullptr) prev1->next = node2;

if (prev2 != nullptr) prev2->next = node1;

curr = node1->next;

node1->next = node2->next;

node2->next = curr;

if (node1->next != nullptr) node1->next->prev = node1;

if (node2->next != nullptr) node2->next->prev = node2;

curr = node1->prev;

node1->prev = node2->prev;

node2->prev = curr;

if (index1 == 0) start = node2;

if (index2 == 0) start = node1;

}

auto stop\_time = std::chrono::high\_resolution\_clock::now();

auto duration = std::chrono::duration\_cast<std::chrono::microseconds>(stop\_time - start\_time);

cout << "Time taken by function: " << duration.count() << " microseconds" << endl;

}

void removebyindex(list\*&start, int& length) {

int index=-100;

cout << "Enter index of element" << endl;

while (index < 0 || index > length) {

cout << "0<Index< " << length << endl;

cin >> index;

}

auto startTime = chrono::high\_resolution\_clock::now();

list\* curr = start;

index -= 1;

if (index == 0) {

start = curr->next;

if (curr->next != nullptr)

curr->next->prev = nullptr;

delete curr;

length--;

return;

}

for (int i = 0; curr != nullptr && i < index - 1; i++)

curr = curr->next;

if (curr == nullptr || curr->next == nullptr)

return;

list\* next = curr->next->next;

delete curr->next;

curr->next = next;

if (next != nullptr)

next->prev = curr;

auto stop = chrono::high\_resolution\_clock::now();

auto duration = chrono::duration\_cast<chrono::microseconds>(stop - startTime);

cout << "Time taken removing element: " << duration.count() << " microseconds" << endl;

length--;

}

void removeByValue(list\*& start, int& length) {

int value;

cout << "Enter value of element to remove:" << endl;

cin >> value;

list\* curr = start;

auto startTime = chrono::high\_resolution\_clock::now();

if (curr != nullptr && curr->data == value) {

start = curr->next;

if (curr->next != nullptr)

curr->next->prev = nullptr;

delete curr;

length--;

return;

}

while (curr != nullptr && curr->data != value) {

curr = curr->next;

}

if (curr == nullptr) return;

list\* next = curr->next;

if (curr->prev != nullptr)

curr->prev->next = next;

if (next != nullptr)

next->prev = curr->prev;

delete curr;

auto stop = chrono::high\_resolution\_clock::now();

auto duration = chrono::duration\_cast<chrono::microseconds>(stop - startTime);

cout << "Time taken removing element: " << duration.count() << " microseconds" << endl;

length--;

}

void deleteList(list\*& start,int& length) {

list\* curr = start;

while (curr) {

list\* next = curr->next;

delete curr;

curr = next;

}

length = 0;

start = nullptr;

}

void getvaluebyindex(list\* start, int length) {

int index = -100;

cout << "Enter index of element" << endl;

while (index < 0 || index >= length) {

cout << "0 <= Index < " << length << endl;

cin >> index;

}

index -= 1;

auto startTime = chrono::high\_resolution\_clock::now();

list\* curr = start;

for (int i = 0; curr != nullptr && i < index; i++)

curr = curr->next;

if (curr == nullptr)

return;

cout << "Element = " << curr->data << endl;

auto stop = chrono::high\_resolution\_clock::now();

auto duration = chrono::duration\_cast<chrono::microseconds>(stop - startTime);

cout << "Time taken: " << duration.count() << " microseconds" << endl;

}

void getByValue(list\* start) {

int value, countelem(0), countall(1);

vector<int> indexElements;

cout << "Enter value of element:" << endl;

cin >> value;

auto startTime = chrono::high\_resolution\_clock::now();

list\* curr = start;

while (curr != nullptr) {

if (curr->data == value) {

countelem += 1;

indexElements.push\_back(countall);

}

curr = curr->next;

countall += 1;

}

if (countelem == 0) {

cout << "Element with this value not found." << endl;

return;

}

for (int number : indexElements) {

cout << "Element founded: " << number << endl;

}

auto stop = chrono::high\_resolution\_clock::now();

auto duration = chrono::duration\_cast<chrono::microseconds>(stop - startTime);

cout << "Time taken: " << duration.count() << " microseconds" << endl;

}

void shakerSortArra(int\* arr, int count) {

bool swapped;

do {

swapped = false;

for (int i = 0; i < count - 1; i++) {

if (arr[i] > arr[i + 1]) {

swap(arr[i], arr[i + 1]);

swapped = true;

}

}

if (!swapped)

break;

swapped = false;

for (int i = count - 2; i >= 0; i--) {

if (arr[i] > arr[i + 1]) {

swap(arr[i], arr[i + 1]);

swapped = true;

}

}

} while (swapped);

}

void shakerSort(list\*& start) {

if (start == nullptr || start->next == nullptr)

return;

int count = 0;

list\* curr = start;

while (curr != nullptr) {

count++;

curr = curr->next;

}

int\* arr = new int[count];

curr = start;

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

arr[i] = curr->data;

curr = curr->next;

}

auto startTime = chrono::high\_resolution\_clock::now();

shakerSortArra(arr, count);

auto stop = chrono::high\_resolution\_clock::now();

auto duration = chrono::duration\_cast<chrono::microseconds>(stop - startTime);

cout << "Time taken sorting arr: " << duration.count() << " microseconds" << endl;

startTime = std::chrono::high\_resolution\_clock::now();

bool swapped;

do {

swapped = false;

list\* curr = start;

while (curr->next != nullptr) {

if (curr->data > curr->next->data) {

swap(curr->data, curr->next->data);

swapped = true;

}

curr = curr->next;

}

if (!swapped)

break;

swapped = false;

while (curr->prev != nullptr) {

if (curr->data < curr->prev->data) {

swap(curr->data, curr->prev->data);

swapped = true;

}

curr = curr->prev;

}

} while (swapped);

stop = std::chrono::high\_resolution\_clock::now();

duration = std::chrono::duration\_cast<std::chrono::microseconds>(stop - startTime);

cout << "Time taken sorting list: " << duration.count() << " microseconds" << endl;

}

int main() {

list\* start = nullptr;

int choice = 0;

int length = 0;

bool listcreated = false;

bool listcreatedandreadyforwork = false;

do {

if (!listcreated) {

cout << "You should create a new list\n";

}

if (!length& listcreated) {

deleteList(start,length);

listcreated = false;

listcreatedandreadyforwork = false;

cout << "List was deleted automaticly because it had 0 elements \n";

cout << "You should create a new list\n";

}

cout << "\nChoose\n";

cout << "1. Random list generation\n";

cout << "2. Manual list generation\n";

cout << "3. Show list\n";

cout << "4. Add to list\n";

cout << "5. Remove from list\n";

cout << "6. Clear list\n";

cout << "7. Swap el of list\n";

cout << "8. Get value by index\n";

cout << "9. Get value by value\n";

cout << "10. Shaker sort(idz2)\n";

cout << "11. Exit\n";

cin >> choice;

system("cls");

switch (choice) {

case 1: {

if (listcreated == true) {

cout << "Error. List already created. Delete it with func for delete\n";

break;

}

createListRND(start, length);

listcreated = true;

listcreatedandreadyforwork = true;

break;

}

case 2: {

if (listcreated == true) {

cout << "Error. List already created. Delete it with func for delete\n";

break;

}

createListInputMainFunc(start, length);

listcreated = true;

listcreatedandreadyforwork = true;

break;

}

case 3: {

if (listcreatedandreadyforwork == false) {

cout << "Error. The list no exists\n";

break;

}

if (!length) {

cout << "List is empty" << endl;

break;

}

showToList(start, length);

break;

}

case 4: {

if (listcreatedandreadyforwork == false) {

cout << "Error. The list no exists\n";

break;

}

if (!length) {

cout << "List is empty" << endl;

break;

}

addToList(start, length);

break;

}

case 5: {

if (listcreatedandreadyforwork == false) {

cout << "Error. The list no exists\n";

break;

}

if (!start) {

cout << "List is empty" << endl;

break;

}

int choosef;

cout << "Choose what you want\nEnter 1 - By index\nEnter 2 - By value" << endl;

cin >> choosef;

system("cls");

switch(choosef){

case 1: {

removebyindex(start, length);

break;

}

case 2: {

removeByValue(start, length);

break;

}

}

break;

}

case 6: {

if (listcreatedandreadyforwork == false) {

cout << "Error. The list no exists\n";

break;

}

if (!length) {

cout << "List is empty" << endl;

break;

}

deleteList(start, length);

start = nullptr;

bool listcreated = false;

bool listcreatedandreadyforwork = false;

break;

}

case 7: {

if (listcreatedandreadyforwork == false) {

cout << "Error. The list no exists\n";

break;

}

if (!length) {

cout << "List is empty" << endl;

break;

}

swaplist(start, length);

break;

}

case 8: {

if (listcreatedandreadyforwork == false) {

cout << "Error. The list no exists\n";

break;

}

if (!length) {

cout << "List is empty" << endl;

break;

}

getvaluebyindex(start, length);

break;

}

case 9: {

if (listcreatedandreadyforwork == false) {

cout << "Error. The list no exists\n";

break;

}

if (!length) {

cout << "List is empty" << endl;

break;

}

getByValue(start);

break;

}

case 10: {

if (listcreatedandreadyforwork == false) {

cout << "Error. The list no exists\n";

break;

}

if (!length) {

cout << "List is empty" << endl;

break;

}

shakerSort(start);

break;

}

case 11: {

cout << "Exit";

break;

}

default: {

cout << "Invalid choice\n";

break;

}

}

} while (choice != 11);

return 0;

}