**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Информационных систем**

отчет

**по лабораторной работе №2**

**по дисциплине «Программирование»**

Тема: Линейные структуры данных

|  |  |  |
| --- | --- | --- |
| Студент гр. 0323 |  | Голубцов В.В. |
| Преподаватель |  | Глущенко А.Г. |

Санкт-Петербург

2021

**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.

**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.

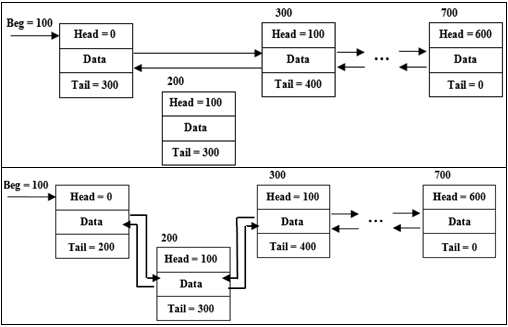
### Удаление элементов двусвязного списка

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.

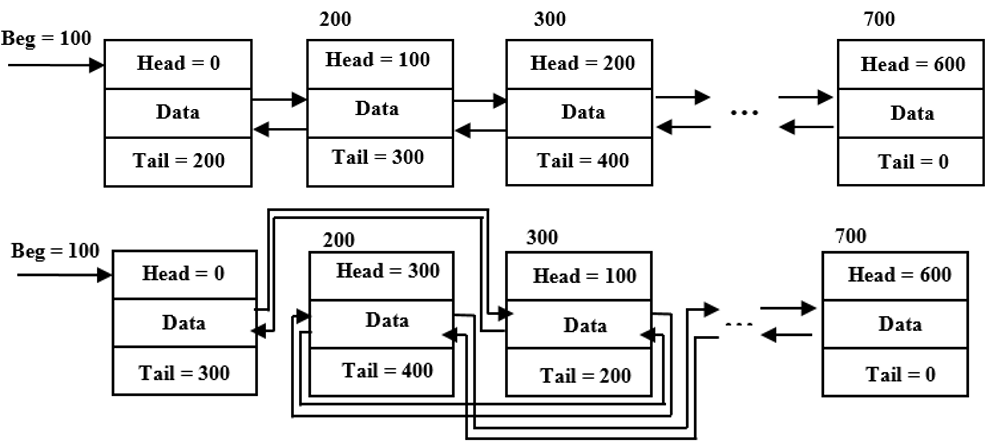


Вставка нового корневого узла отличается от удаления тем, что необходимо изменить **Head**-указатель прошлого корневого узла, указатель на корневой узел и правильно связать новый узел со списком.

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.

**Задача**

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.

**Обработка результатов эксперимента.**

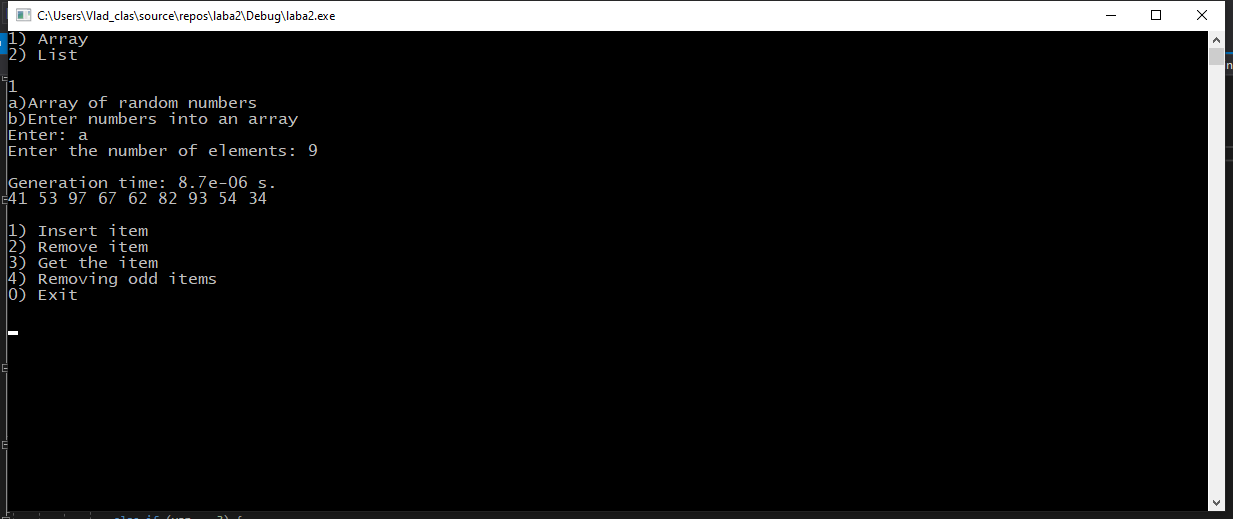


Рисунок Создание массива случайных чисел

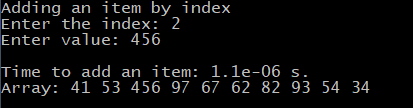
****

Рисунок Добавление элемента в массив

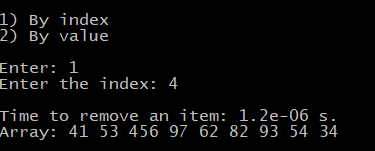


Рисунок Удаление элемента

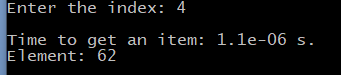


Рисунок Получение элемента

**Выводы.**

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

ПРИЛОЖЕНИЕ А

Листинг программы

#include <iostream>

#include <ctime> // содержит time()

#include <chrono>

using namespace std;

typedef chrono::high\_resolution\_clock Clock;

typedef chrono::duration<double> sec;

Clock::time\_point timeStart;

Clock::time\_point timeEnd;

int countElements = 0;

struct Node

{

int data;

Node \*Next, \*Prev;

};

class List

{

Node \*Head, \*Tail; //Указатели на адреса начала списка и его конца

public:

List() :Head(NULL), Tail(NULL) {}; //Инициализируем адреса как пустые

~List();

void Show(); //Отображение списка на экран

void Add(int x); //Добавлениу элементов в список

void AddTail(int n); //Добавление в конец списка

void Insert(); //Вставка элемента

void Del(); //Удаление элемента

void Del(int x); //Удаление элемента

Node\* GetElem(); // Получить элемент списка

};

List::~List() //Деструктор

{

while (Head)

{

Tail = Head->Next; //Резервная копия адреса следующего звена списка

delete Head;

Head = Tail; //Смена адреса начала на адрес следующего элемента

}

}

void List::Add(int x)

{

Node \*temp = new Node;

temp->Next = NULL;

temp->data = x;

if (Head != NULL)

{

temp->Prev = Tail; //Указываем адрес на предыдущий элемент в соотв. поле

Tail->Next = temp; //Указываем адрес следующего за хвостом элемента

Tail = temp; //Меняем адрес хвоста

}

else //Если список пустой

{

temp->Prev = NULL;

Head = Tail = temp; //Голова=Хвост=элемент

}

}

void List::Show()

{

Node \*temp = Head;

while (temp != NULL)

{

cout << temp->data << " "; //Выводим каждое считанное значение на экран

temp = temp->Next; //Смена адреса на адрес следующего элемента

}

cout << "\n";

}

List \*GenerateLst(List \*lst) {

char option;

cout << "a)List of random numbers\n" << "b)Enter numbers into an array\n" << "Enter: ";

cin >> option;

if (option == 'a') {

int count = 0;

cout << "Enter the number of elements: ";

cin >> count;

timeStart = Clock::now();

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

lst->Add(rand() % 99);

}

timeEnd = Clock::now();

countElements = count;

cout << endl;

cout << "Generation time: " << sec(timeEnd - timeStart).count() << " s." << endl;

}

else {

int count = 0;

int value = 0;

while (cin >> value)

{

char c = static\_cast<char>(cin.get());

count++;

if (isdigit(c) || c == '-' || c == '+')

{

//nope

}

else if (c == ' ')

{

lst->Add(value);

continue;

}

else

{

lst->Add(value);

break;

}

}

countElements = count;

}

return lst;

}

void List::AddTail(int n)

{

Node \* temp = new Node;

// Следующего нет

temp->Next = 0;

temp->data = n;

temp->Prev = Tail;

// Если элементы есть?

if (Tail != 0)

Tail->Next = temp;

if (countElements == 0)

Head = Tail = temp;

else

Tail = temp;

countElements++;

}

void List::Insert()

{

cout << "\nInsert element.\n";

int pos;

cout << "Input position: ";

cin >> pos;

// Позиция от 1 до Count?

if (pos < 1 || pos > countElements + 1)

{

cout << "Incorrect position !!!\n";

return;

}

// Если вставка в конец списка

if (pos == countElements + 1)

{

int data;

cout << "Input new number: ";

cin >> data;

timeStart = Clock::now();

Add(data); // Добавление в конец списка

timeEnd = Clock::now();

cout << "\nTime to add an item: " << sec(timeEnd - timeStart).count() << " s." << endl;

return;

}

else if (pos == 1)

{

int data;

cout << "Input new number: ";

cin >> data;

timeStart = Clock::now();

AddTail(data); // Добавление в начало списка

timeEnd = Clock::now();

cout << "\nTime to add an item: " << sec(timeEnd - timeStart).count() << " s." << endl;

}

int i = 1; // Счетчик

Node \* temp = new Node;

cout << "Input new number: "; cin >> temp->data;

timeStart = Clock::now();

Node \*Ins = Head;

while (i < pos)

{

// Доходим до элемента,

// перед которым вставляемся

Ins = Ins->Next;

i++;

}

// Доходим до элемента,

// который предшествует

Node \* PrevIns = Ins->Prev;

// настройка связей

if (PrevIns != 0 && countElements != 1)

PrevIns->Next = temp;

temp->Next = Ins;

temp->Prev = PrevIns;

Ins->Prev = temp;

countElements++;

timeEnd = Clock::now();

cout << "\nTime to add an item: " << sec(timeEnd - timeStart).count() << " s." << endl;

}

void List::Del()

{

int var = -1;

cout << "\nRemoving an item.\n";

cout << "\n1) By position\n"

<< "2) By value\n";

cout << endl;

cout << "Enter: "; cin >> var;

if (var == 1) {

int pos;

cout << "Input position: "; cin >> pos;

timeStart = Clock::now();

if (pos < 1 || pos > countElements) // Неверная позиция

{

cout << "Incorrect position !\n";

timeEnd = Clock::now();

return;

}

int i = 1; // Счетчик

Node \*Del = Head;

while (i < pos) // Доходим до элемента, который удаляется

{

Del = Del->Next;

i++;

}

Node \* PrevDel = Del->Prev; // Доходим до элемента, который предшествует удаляемому

Node \* AfterDel = Del->Next; // Доходим до элемента, который следует за удаляемым

if (PrevDel != 0 && countElements != 1) // Если удаляем не голову

PrevDel->Next = AfterDel;

if (AfterDel != 0 && countElements != 1) // Если удаляем не хвост

AfterDel->Prev = PrevDel;

if (pos == 1) // Удаляются крайние?

Head = AfterDel;

if (pos == countElements)

Tail = PrevDel;

delete Del; // Удаление элемента

timeEnd = Clock::now();

cout << "\nTime to remove an item: " << sec(timeEnd - timeStart).count() << " s." << endl;

}

else {

int i = 1; // Счетчик

int count = 1; // Счетчик

int value;

cout << "Input value: "; cin >> value;

timeStart = Clock::now();

Node \*Del = Head;

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

if (Del->data == value) {

count = i;

}

Del = Del->Next;

}

if (count < 0) // Неверное значение

{

cout << "Incorrect value!\n";

timeEnd = Clock::now();

return;

}

Del = Head;

while (i < count) // Доходим до элемента, который удаляется

{

Del = Del->Next;

i++;

}

Node \*PrevDel = Del->Prev; // Доходим до элемента, который предшествует удаляемому

Node \*AfterDel = Del->Next; // Доходим до элемента, который следует за удаляемым

if (PrevDel != 0 && countElements != 1) // Если удаляем не голову

PrevDel->Next = AfterDel;

if (AfterDel != 0 && countElements != 1) // Если удаляем не хвост

AfterDel->Prev = PrevDel;

if (count == 1) // Удаляются крайние?

Head = AfterDel;

if (count == countElements)

Tail = PrevDel;

delete Del; // Удаление элемента

timeEnd = Clock::now();

cout << "\nTime to remove an item: " << sec(timeEnd - timeStart).count() << " s." << endl;

}

countElements--;

}

void List::Del(int pos)

{

int i = 1;

Node \*Del = Head;

while (i < pos)

{

Del = Del->Next;

i++;

}

Node \*PrevDel = Del->Prev;

Node \*AfterDel = Del->Next;

if (PrevDel != 0 && countElements != 1)

PrevDel->Next = AfterDel;

if (AfterDel != 0 && countElements != 1)

AfterDel->Prev = PrevDel;

if (pos == 1)

Head = AfterDel;

if (pos == countElements)

Tail = PrevDel;

delete Del;

countElements--;

}

Node \*List::GetElem()

{

int var = -1;

cout << "\n1) By position\n"

<< "2) By value\n";

cout << endl;

cout << "Enter: "; cin >> var;

if (var == 1) {

int pos;

cout << "Input position: "; cin >> pos;

timeStart = Clock::now();

Node \*temp = Head;

if (pos < 1 || pos > countElements) // Неверная позиция

{

cout << "Incorrect position!\n";

timeEnd = Clock::now();

return 0;

}

int i = 1;

while (i < pos && temp != 0) // Ищем нужный нам элемент

{

temp = temp->Next;

i++;

}

timeEnd = Clock::now();

cout << "\nTime to remove an item: " << sec(timeEnd - timeStart).count() << " s." << endl;

if (temp == 0)

return 0;

else

return temp;

}

else {

int count = -1; // Счетчик

int value;

int i = 1;

cout << "Input value: "; cin >> value;

timeStart = Clock::now();

Node \*temp = Head;

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

if (temp->data == value) {

count = i;

}

temp = temp->Next;

}

if (count < 0) // Неверное значение

{

cout << "Incorrect value!\n";

timeEnd = Clock::now();

return 0;

}

temp = Head;

while (i < count) // Доходим до элемента, который удаляется

{

temp = temp->Next;

i++;

}

timeEnd = Clock::now();

cout << "\nTime to remove an item: " << sec(timeEnd - timeStart).count() << " s." << endl;

if (temp == 0)

return 0;

else

return temp;

}

}

int \*GenerateArr(int \*ar) {

//srand(time(NULL));

char option;

cout << "a)Array of random numbers\n" << "b)Enter numbers into an array\n" << "Enter: ";

cin >> option;

if (option == 'a') {

int count = 0;

cout << "Enter the number of elements: ";

cin >> count;

timeStart = Clock::now();

ar = new int[count];

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

ar[i] = rand() % 99;

}

timeEnd = Clock::now();

countElements = count;

cout << endl;

cout << "Generation time: " << sec(timeEnd - timeStart).count() << " s." << endl;

}

else {

int count = 0;

while (cin >> ar[count])

{

char c = static\_cast<char>(cin.get());

count++;

if (isdigit(c) || c == '-' || c == '+')

{

}

else if (c == ' ')

{

continue;

}

else

{

break;

}

}

countElements = count;

}

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

cout << ar[i] << " ";

}

return ar;

}

int \*AddElementByIndex(int \*ar) {

int index = 0;

int value = 0;

int \*arrNew = new int[countElements + 1];

cout << "Enter the index: "; cin >> index;

if (index <= countElements - 1) {

cout << "Enter value: "; cin >> value;

timeStart = Clock::now();

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

arrNew[i] = ar[i];

}

arrNew[index] = value;

for (int i = index + 1; i <= countElements+1; i++) {

arrNew[i] = ar[i-1];

}

countElements++;

timeEnd = Clock::now();

cout << "\nTime to add an item: " << sec(timeEnd - timeStart).count() << " s." << endl;

return arrNew;

}

else {

cout << "Element missing!\n";

return ar;

}

}

int \*DeliteElement(int \*ar) {

int var = -1;

int index = -1;

int value = 0;

int \*arrNew = new int[countElements - 1];

cout << endl;

cout << "\n1) By index\n"

<< "2) By value\n";

cout << endl;

cout << "Enter: "; cin >> var;

if (var == 1) {

cout << "Enter the index: "; cin >> index;

if (index <= countElements - 1 && index >= 0) {

timeStart = Clock::now();

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

arrNew[i] = ar[i];

}

for (int i = index + 1; i <= countElements - 1; i++) {

arrNew[i - 1] = ar[i];

}

countElements--;

timeEnd = Clock::now();

cout << "\nTime to remove an item: " << sec(timeEnd - timeStart).count() << " s." << endl;

return arrNew;

}

else {

cout << "Element missing!\n";

return ar;

}

}

else {

cout << "Enter the value: "; cin >> value;

timeStart = Clock::now();

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

if (value == ar[i]) {

index = i;

}

}

if (index >= 0) {

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

arrNew[i] = ar[i];

}

for (int i = index + 1; i <= countElements - 1; i++) {

arrNew[i - 1] = ar[i];

}

countElements--;

timeEnd = Clock::now();

cout << "\nTime to remove an item: " << sec(timeEnd - timeStart).count() << " s." << endl;

return arrNew;

}

else {

cout << "Item not found!\n";

return ar;

}

}

}

int \*DeliteElements(int \*ar, int index) {

int \*arrNew = new int[countElements - 1];

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

arrNew[i] = ar[i];

}

for (int i = index + 1; i <= countElements - 1; i++) {

arrNew[i - 1] = ar[i];

}

countElements--;

return arrNew;

}

int \*DeliteOddArray(int \*arrr) {

timeStart = Clock::now();

int count = countElements;

int i = 0;

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

if (j % 2 == 0) {

arrr = DeliteElements(arrr, j - i);

i++;

}

}

timeEnd = Clock::now();

cout << "\nTime to remove an odd item: " << sec(timeEnd - timeStart).count() << " s." << endl;

return arrr;

}

int GetElement(int \*ar) {

int var = -1;

int \*element = new int;

int index = -1;

int value = 0;

cout << "\n1) By index\n"

<< "2) By value\n";

cout << endl;

cout << "Enter: "; cin >> var;

if (var == 1) {

cout << "Enter the index: "; cin >> index;

timeStart = Clock::now();

if (index <= countElements - 1 && index >= 0) {

\*element = ar[index];

}

}

else {

cout << "Enter the value: "; cin >> value;

timeStart = Clock::now();

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

if (value == ar[i]) {

index = i;

}

}

if (index >= 0) {

\*element = ar[index];

}

else {

cout << "Item not found!\n";

}

}

timeEnd = Clock::now();

cout << "\nTime to get an item: " << sec(timeEnd - timeStart).count() << " s." << endl;

return \*element;

}

void PrintArray(int \*ar) {

cout << "Array: ";

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

cout << ar[i] << " ";

}

}

int main()

{

int varOne = -1;

do {

cout << "1) Array\n"

<< "2) List\n";

cout << endl;

cin >> varOne;

if (varOne == 1) {

int element = -1;

int var = -1;

int \*arr = new int[1000]; // выделяем память под массив

arr = GenerateArr(arr);

do {

cout << endl;

cout << "\n1) Insert item\n"

<< "2) Remove item\n"

<< "3) Get the item\n"

<< "4) Removing odd items\n"

<< "0) Exit\n";

cout << endl;

cin >> var;

if (var == 1) {

cout << "Adding an item by index\n";

arr = AddElementByIndex(arr);

PrintArray(arr);

}

else if (var == 2) {

cout << "Remove item\n";

arr = DeliteElement(arr);

PrintArray(arr);

}

else if (var == 3) {

cout << "Get item\n";

element = GetElement(arr);

cout << "Element: " << element;

}

else if (var == 4) {

cout << "Removing odd items\n";

arr = DeliteOddArray(arr);

PrintArray(arr);

}

else {

break;

}

} while (var != 0);

}

else if (varOne == 2) {

int var = -1;

List \*lst = new List;

lst = GenerateLst(lst);

lst->Show();

do {

cout << endl;

cout << "\n1) Insert item\n"

<< "2) Remove item\n"

<< "3) Get the item\n"

<< "4) Removing odd items\n"

<< "0) Exit\n";

cout << endl;

cin >> var;

if (var == 1) {

cout << "Adding an item by index\n";

lst->Insert();

lst->Show();

}

else if (var == 2) {

cout << "Remove item\n";

lst->Del();

lst->Show();

}

else if (var == 3) {

cout << "Get item\n";

cout << "Element: " << lst->GetElem();

}

else if (var == 4) {

cout << "Removing odd items\n";

timeStart = Clock::now();

int count = countElements;

int i = 0;

for (int j = 1; j < count+1; j++) {

if (j % 2 == 1) {

lst->Del(j-i);

i++;

}

}

timeEnd = Clock::now();

cout << "\nTime to remove an odd item: " << sec(timeEnd - timeStart).count() << " s." << endl;

lst->Show();

}

else {

break;

}

} while (var != 0);

}

} while (varOne > 0);

}