**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: Линейные структуры данных. Динамические массивы и двусвязные списки

|  |  |  |
| --- | --- | --- |
| Студентка 0324 | Косенко А.Р |  |
| Преподаватель | Глущенко А.Г. |  |

Санкт-Петербург

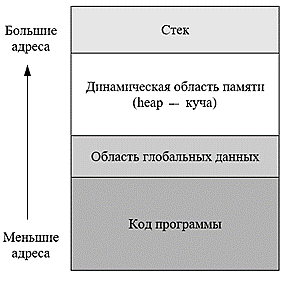
2021

**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.**

Схема распределения памяти под программу показана на рис. 1.



*Рис.1.* Схема распределения памяти под программу

Область кода программы предназначена для хранения инструкций функций программы, обеспечивающих обработку данных.

Данные в программе представляются переменными и константами.

Для хранения глобальных данных предназначена область глобальных данных.

Стек программы используется при вызове функций для передачи параметров и хранения локальных данных.

Распределение памяти для хранения всех обычных переменных осуществляется компилятором, адреса и объемы соответствующих участков памяти (в области глобальных данных) жестко закреплены за этими переменными на все время работы программы и изменены быть не могут.

Однако во многих задачах невозможно заранее предсказать, сколько места (количество переменных, объемы массивов и т.д.) потребуется для решения задачи – это так называемые задачи с неопределенной размерностью.

Решить эту проблему можно лишь в том случае, если иметь механизм, позволяющий создавать новые объекты по мере возникновения необходимости в этих объектах или изменять объемы памяти, выделенные под эти объекты (например, объемы массивов).

Между областью глобальных данных и стеком располагается так называемая динамическая область памяти, которую и можно использовать в процессе работы программы для реализации механизма динамического управления памятью.

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.

Для создания одномерного динамического массива, элементами которого являются, например, действительные числа, используется следующий синтаксис инструкции new (стиль С++):

double \*Arr = newdouble [100];

Освободить динамическую область от этого массива можно с помощью инструкции delete:

delete [] Arr;

После этого занятый участок памяти будет возвращен в список свободной памяти и может быть повторно использован для размещения других динамических объектов.

Язык C++ поддерживает и «старый», заимствованный от языка C, стиль работы с динамической областью. Довольно часто бывает полезно использовать именно этот механизм управления динамической памятью.

В языке C отсутствуют инструкции new и delete. Вместо них для управления динамической памятью используются библиотечные функции:

// Блок прототипов функций

{

void \*malloc (size);

void \*calloc(num, size);

void free( void \*memblock);

void \*realloc( void \*memblock, size);

}

Функция malloc выделяет в динамической области size байт памяти и возвращает адрес этого участка в виде указателя (void \*).

Поскольку возвращаемый указатель не привязан ни к какому типу данных, при работе с ним потребуется явное приведение типов данных (см. пример далее).

Функция calloc выделяет в динамической области size \* num байт памяти и возвращает адрес этого участка в виде указателя (void \*).

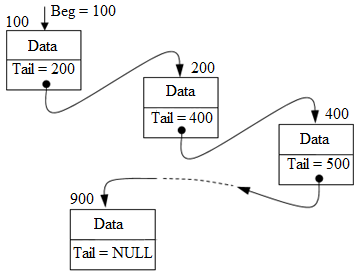
Функция free освобождает участок динамической памяти по адресу memblock и возвращает его в список свободной памяти для повторного использования.

Функция realloc позволяет изменить размер (уменьшить или увеличить) ранее выделенной по адресу memblock памяти, установив новый размер выделенного участка равным size байт. При увеличении размера выделенного участка данные, которые хранились в старом участке, копируются в новый участок памяти. При уменьшении объема выделенного участка данные, которые хранились в нем, усекаются до нового размера. Функция возвращает указатель на область памяти нового размера.

Работа с одномерным динамическим массивом осуществляется так же, как и с обычным. При этом стиль использования динамических массивов С имеет весомое преимущество над С++, которое заключается в изменении размерности массива. Дело в том, что в C++ нет функций увеличения размерности. Увеличить размер массива можно, создав новый динамический массив нужной размерности, скопировав данные из старого массива в новый и освободив память от старого массива.

Одномерный однонаправленный список представляет собой совокупность отдельных элементов, каждый из которых содержит две части – информационную () и адресную ().

Информационная часть предназначена для хранения полезных данных и может иметь практически любой тип. Адресная часть каждого элемента содержит адрес следующего элемента списка. Схематическое изображение такого списка представлено на рис. 2.



*Рис. 2.* Схематическое изображение односвязного списка

Для работы со списком достаточно знать только адрес его первого элемента (). Зная адрес первого элемента списка можно последовательно получить доступ к любому другому его элементу.

Поскольку каждый элемент списка должен иметь две части, логичнее всего представить его в виде следующей структуры:

struct list

{

int data;

list \*tail;

};

Типовыми операциями при работе со списками являются:

1. создание списка;
2. освобождение памяти от списка (удаление списка);
3. доступ к заданному элементу списка для манипуляций с его информационной частью;
4. добавление нового элемента к списку;
5. удаление элемента из списка;
6. перестановка элемента списка на новую позицию внутри списка.

Достоинством подобных структур является простота добавления, удаления и перестановки элементов списка, которые осуществляются путем манипуляций с адресными частями без перезаписи всего списка.

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.

Поскольку каждый элемент списка должен иметь три части, логичнее всего представить его в виде следующей структуры:

struct list

{

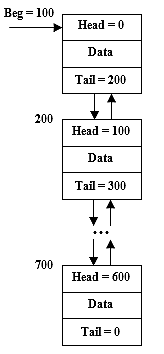
int data;

list \*head;

list \*tail;

};

На рис. 3 показано схематическое представление двусвязного списка. Поле содержит адрес предыдущего элемента, поле содержит адрес следующего элемента списка. Такая организация списка позволяет перемещаться по его элементам в двух направлениях.



*Рис. 3.* Схематическое изображение двусвязного списка

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.

**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.

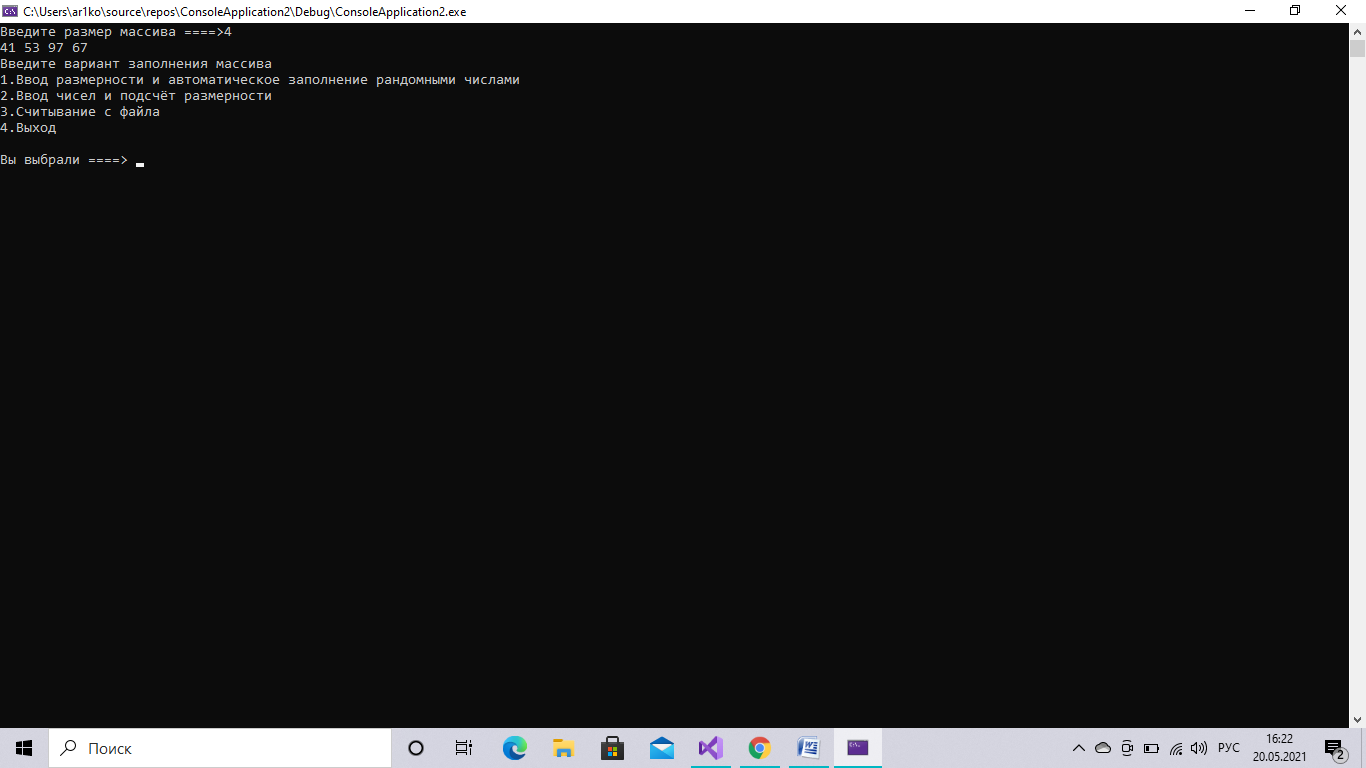
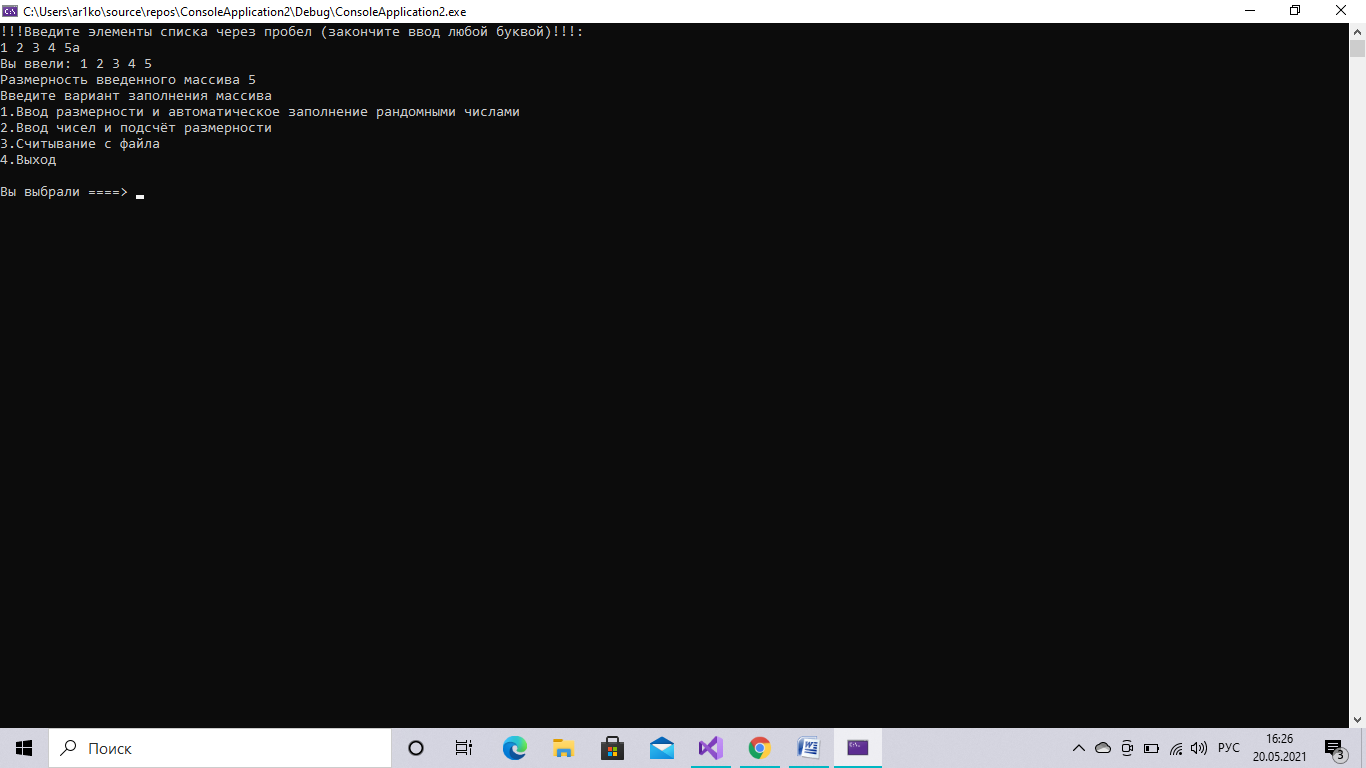
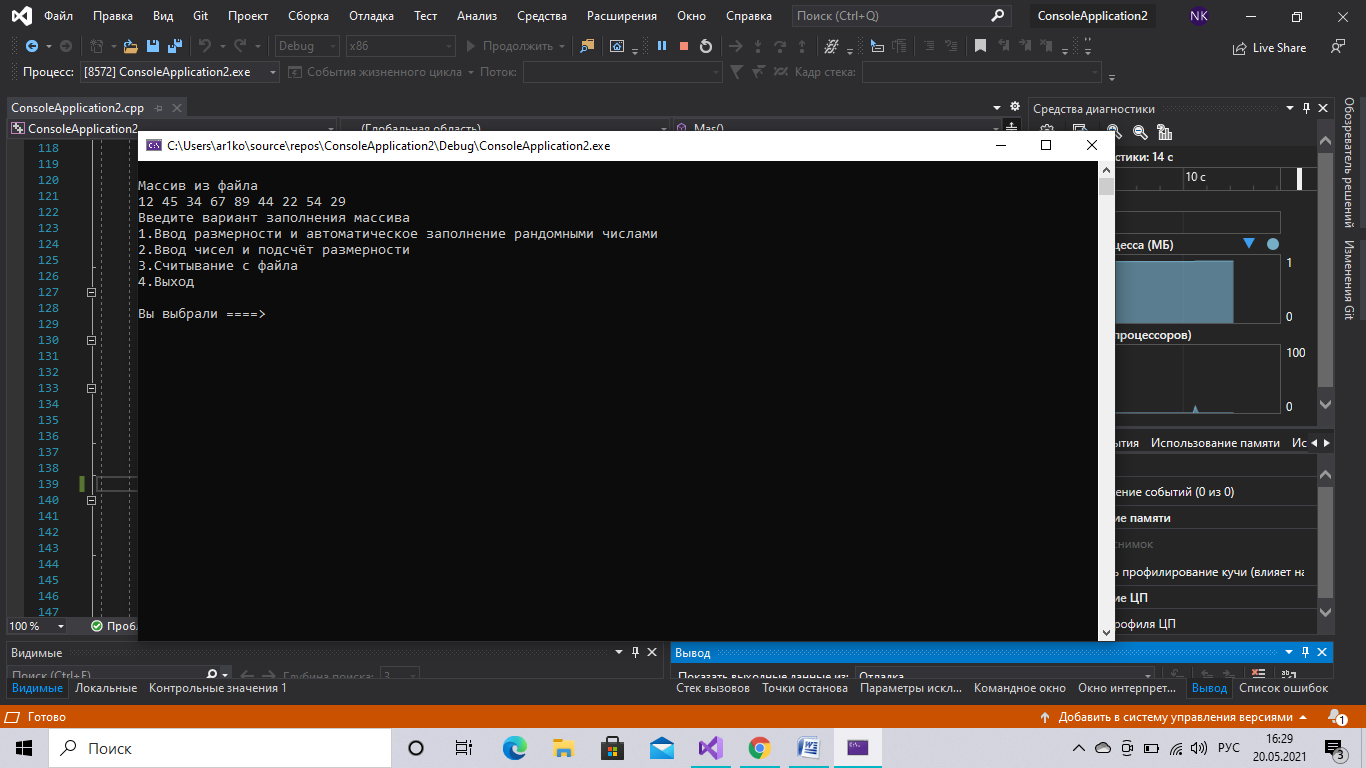
8. Определение скорости вставки, удаление и получения элемента двусвязного списка п. 7.

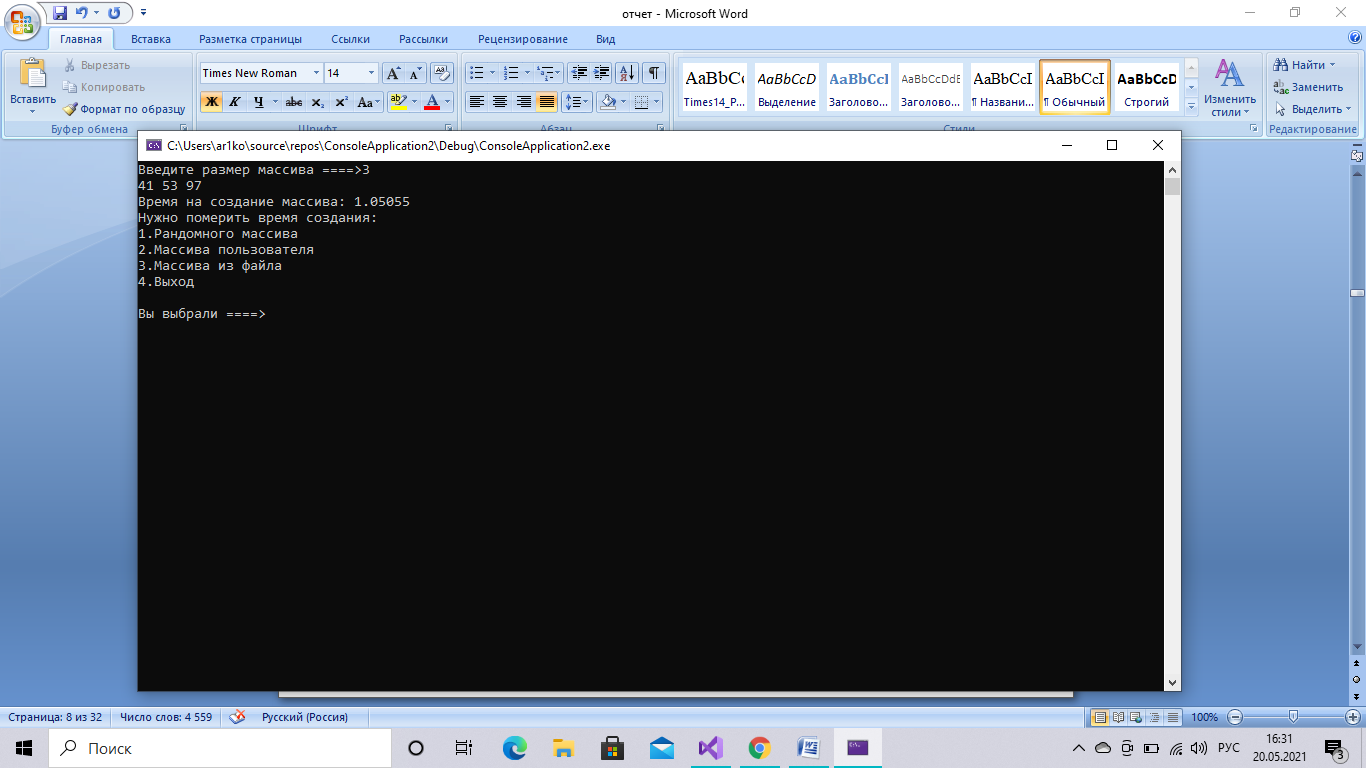
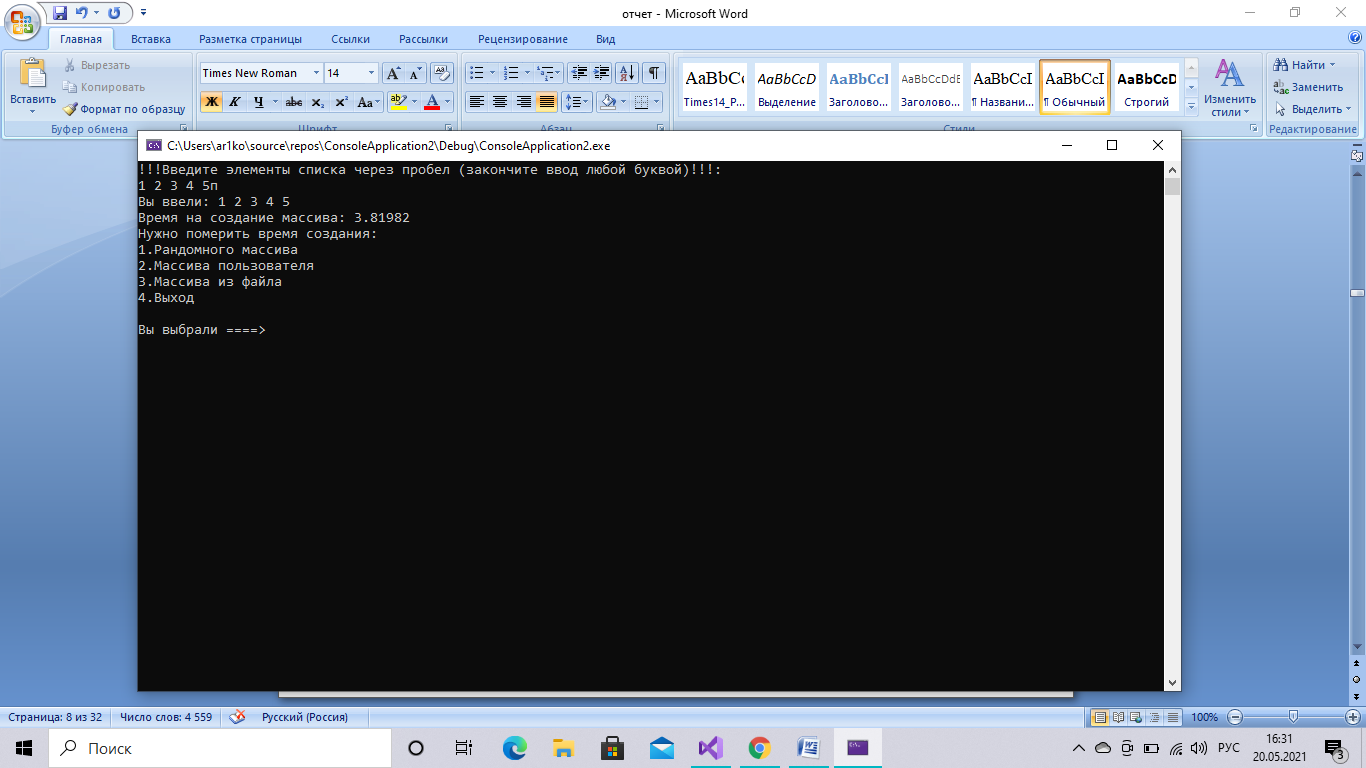
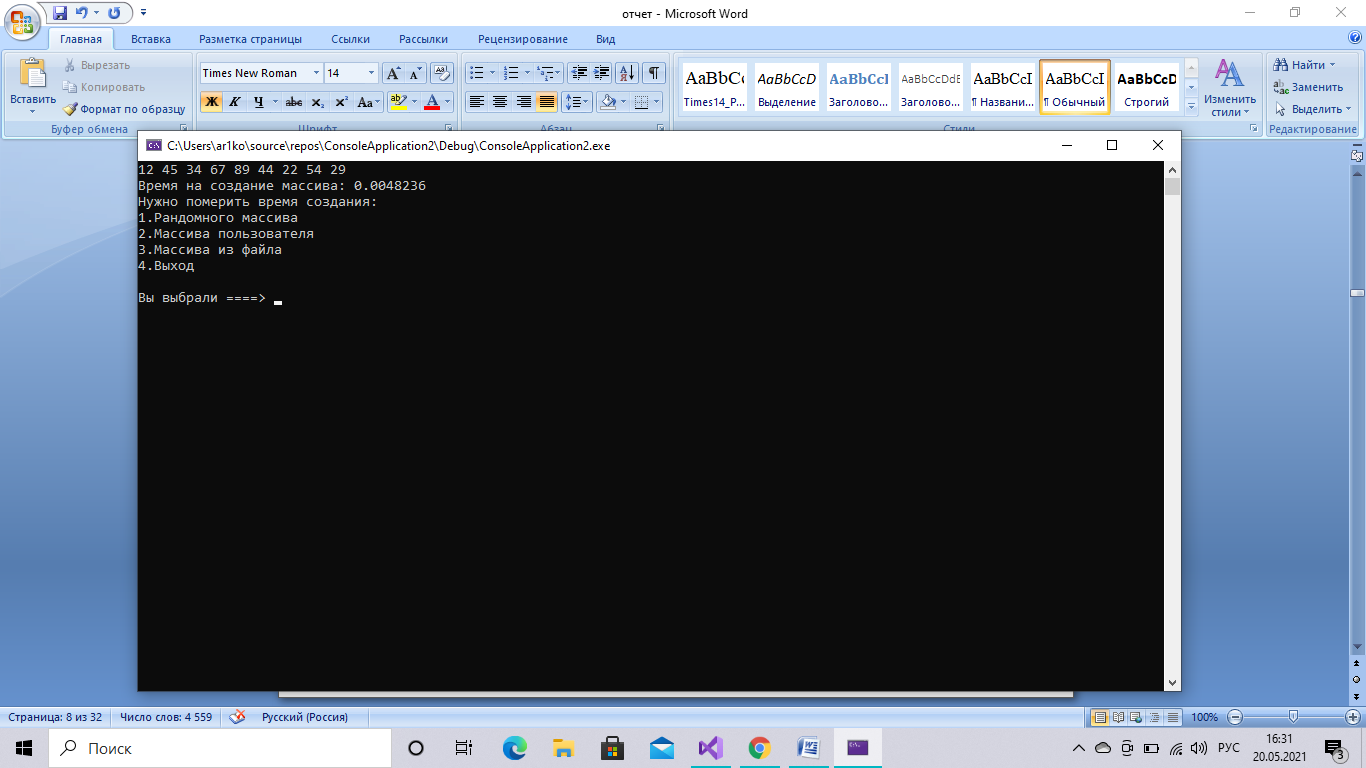
Должна быть возможность запуска каждого пункта многократно, если есть возможность (если в списке/массиве нет элементов, то нельзя ничего удалить и об этом нужно сообщить пользователю). Необходимо сравнить результаты. Для этого пункты 1–4 и 5–8 должны принимать одинаковые значения.

**Выполнение работы.**

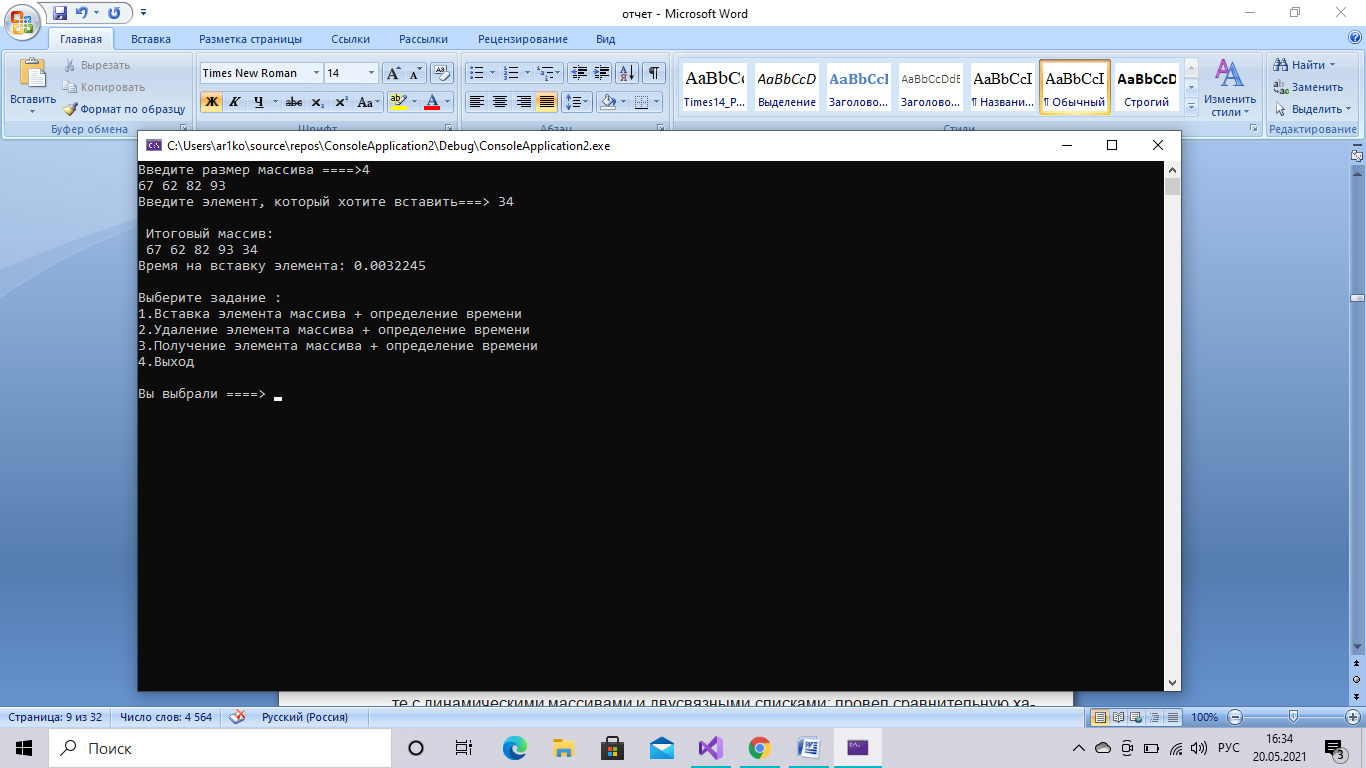
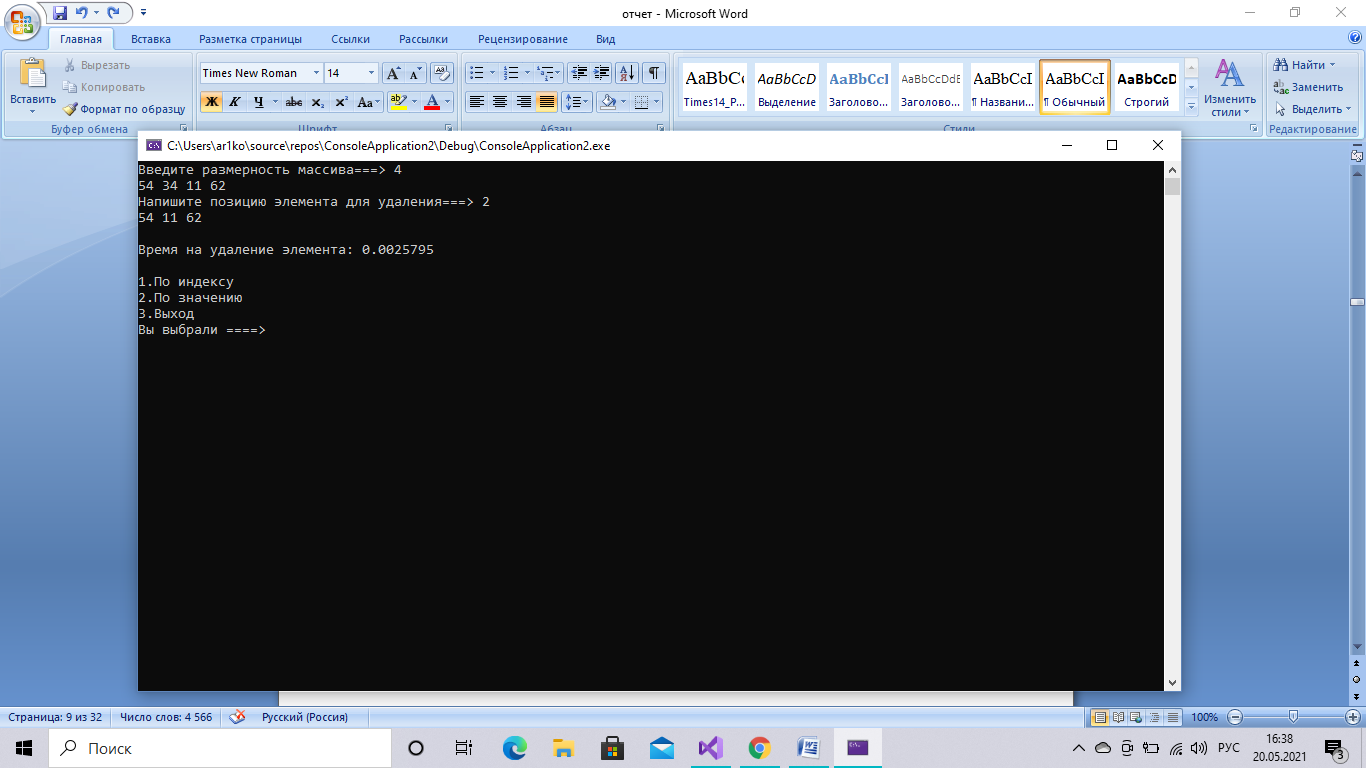
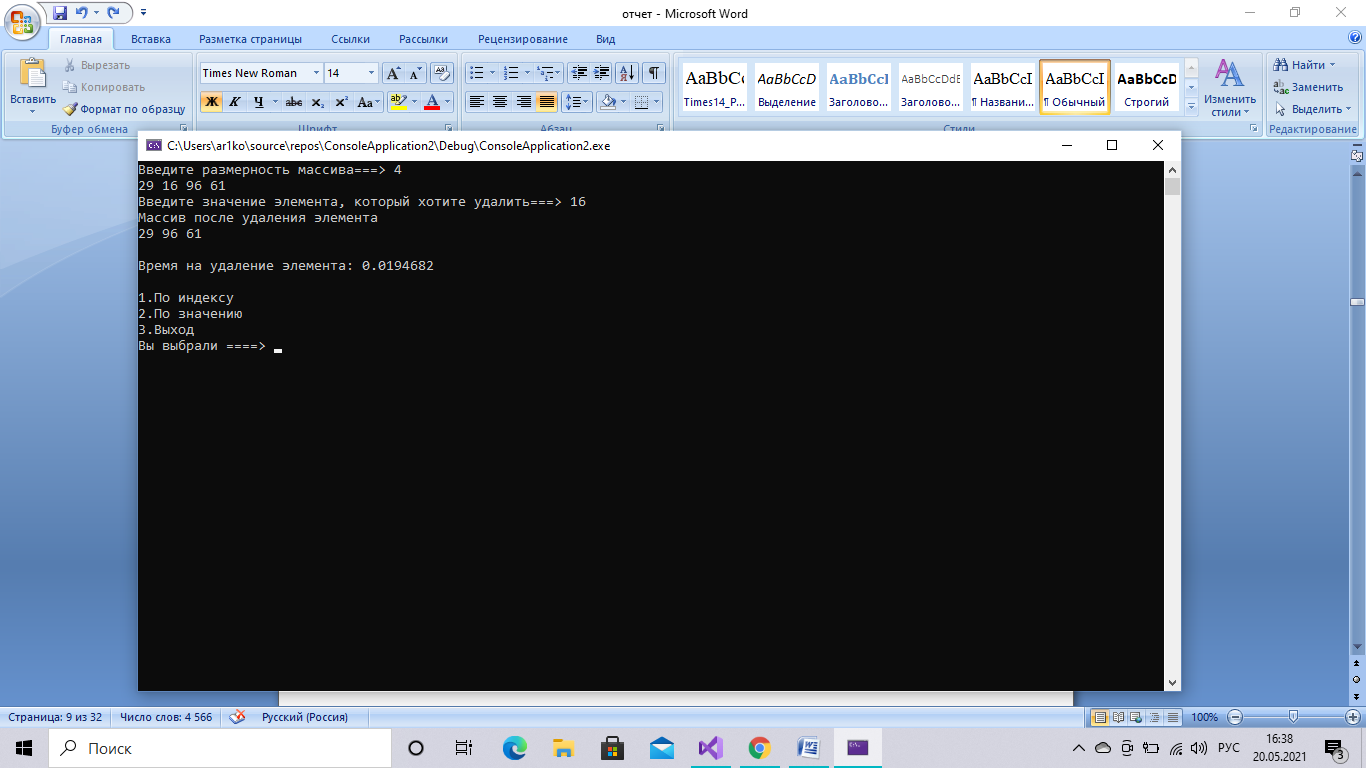
**1 задание.**

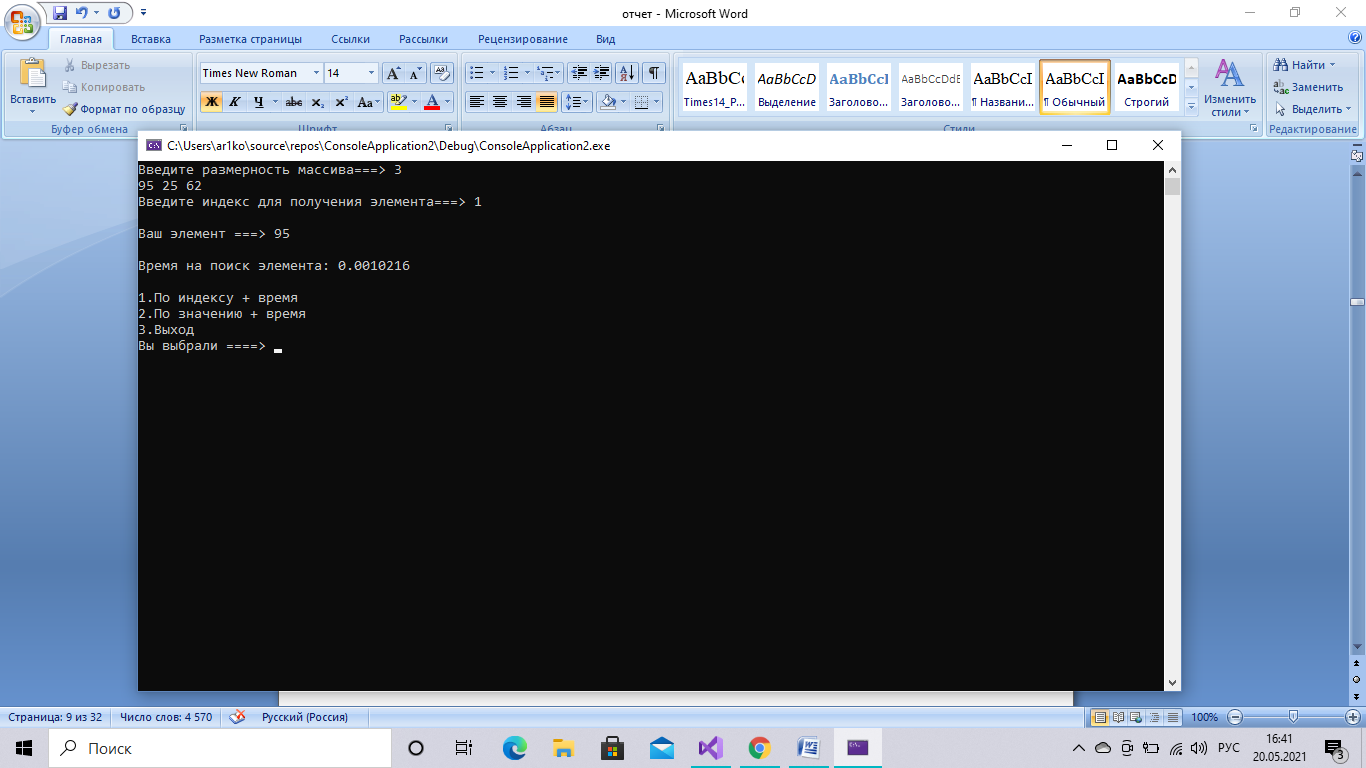
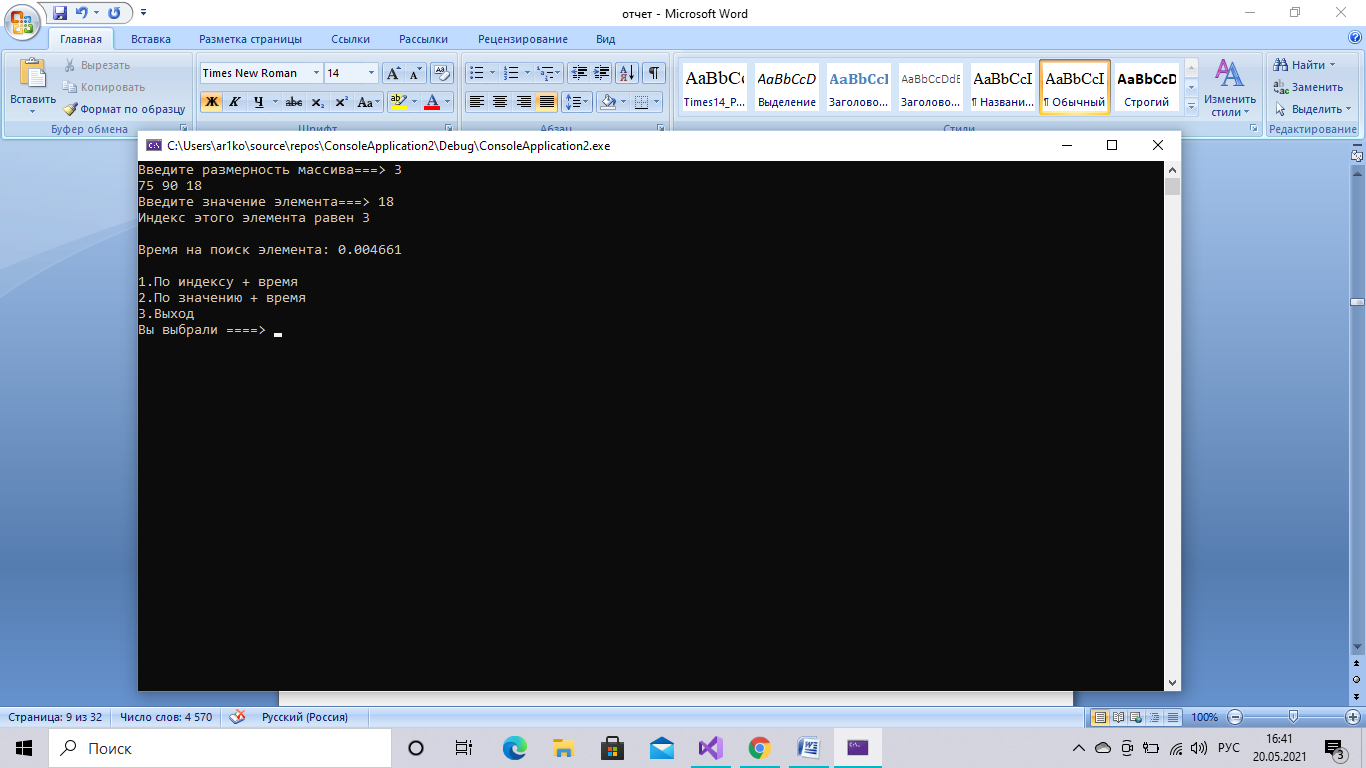
**А)**

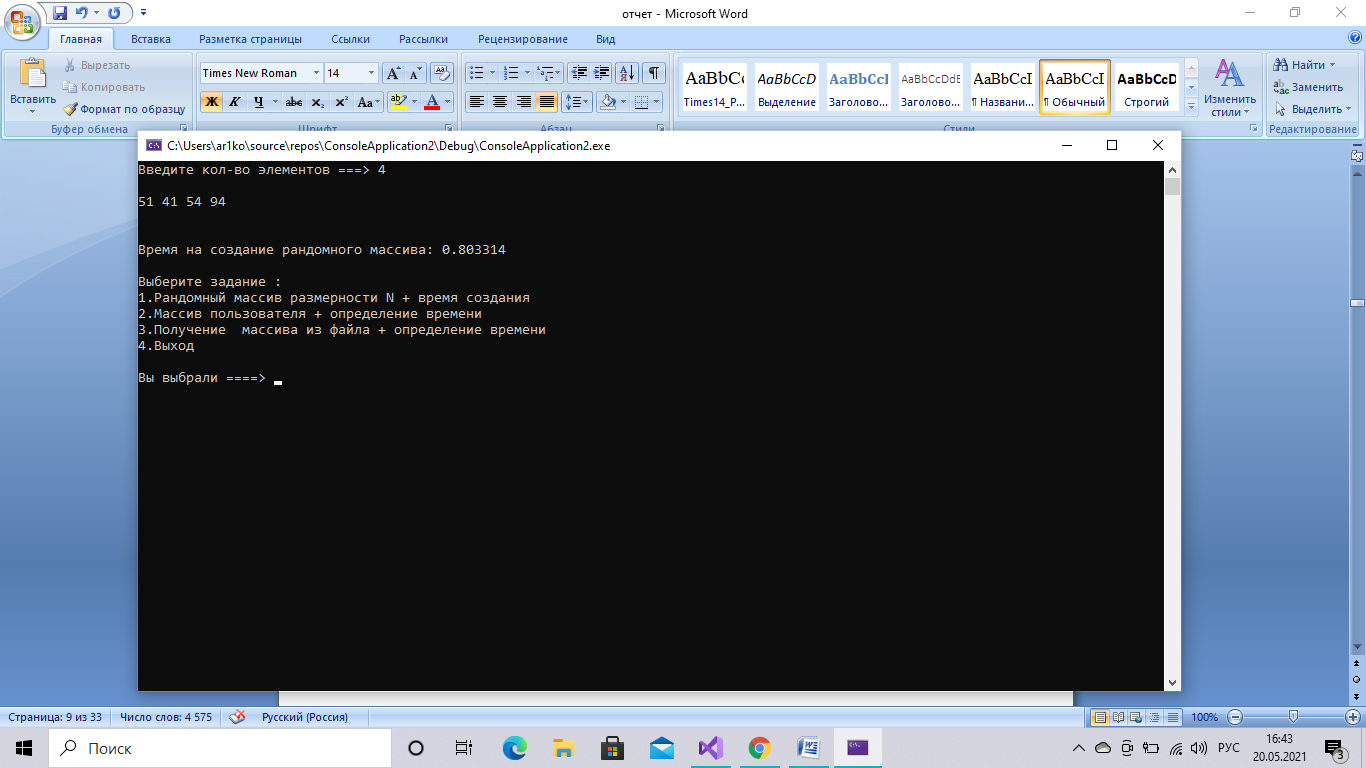
**  
б)  
  
в)  
  
2 задание**

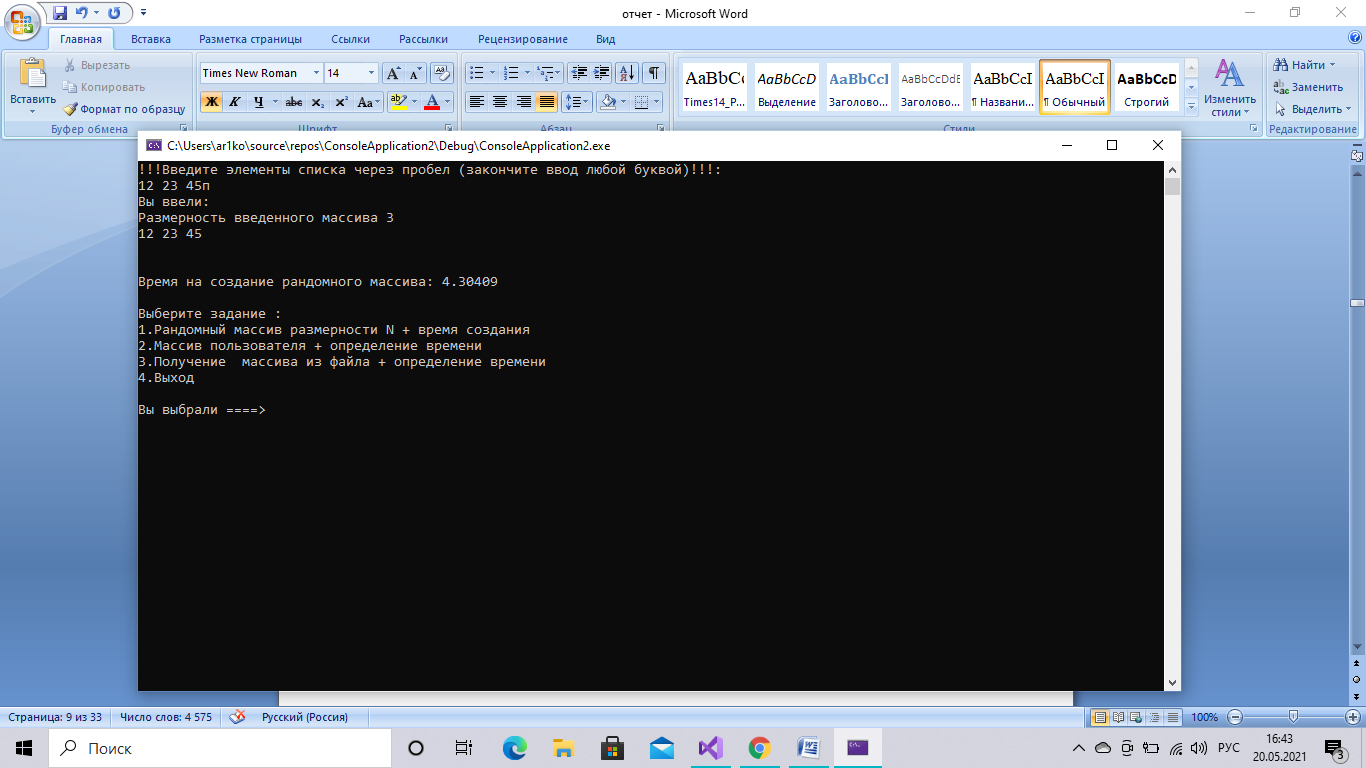
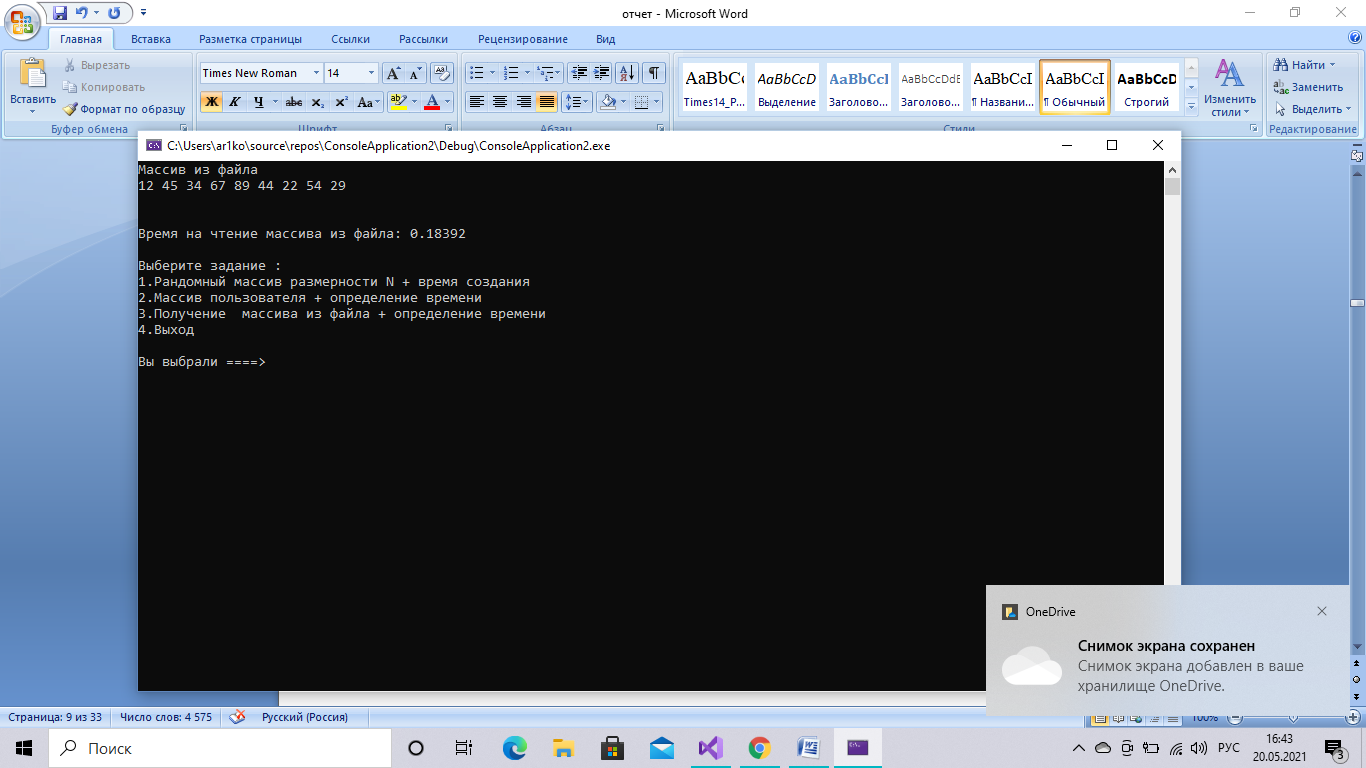
**А)  
  
б)  
  
в)  
**

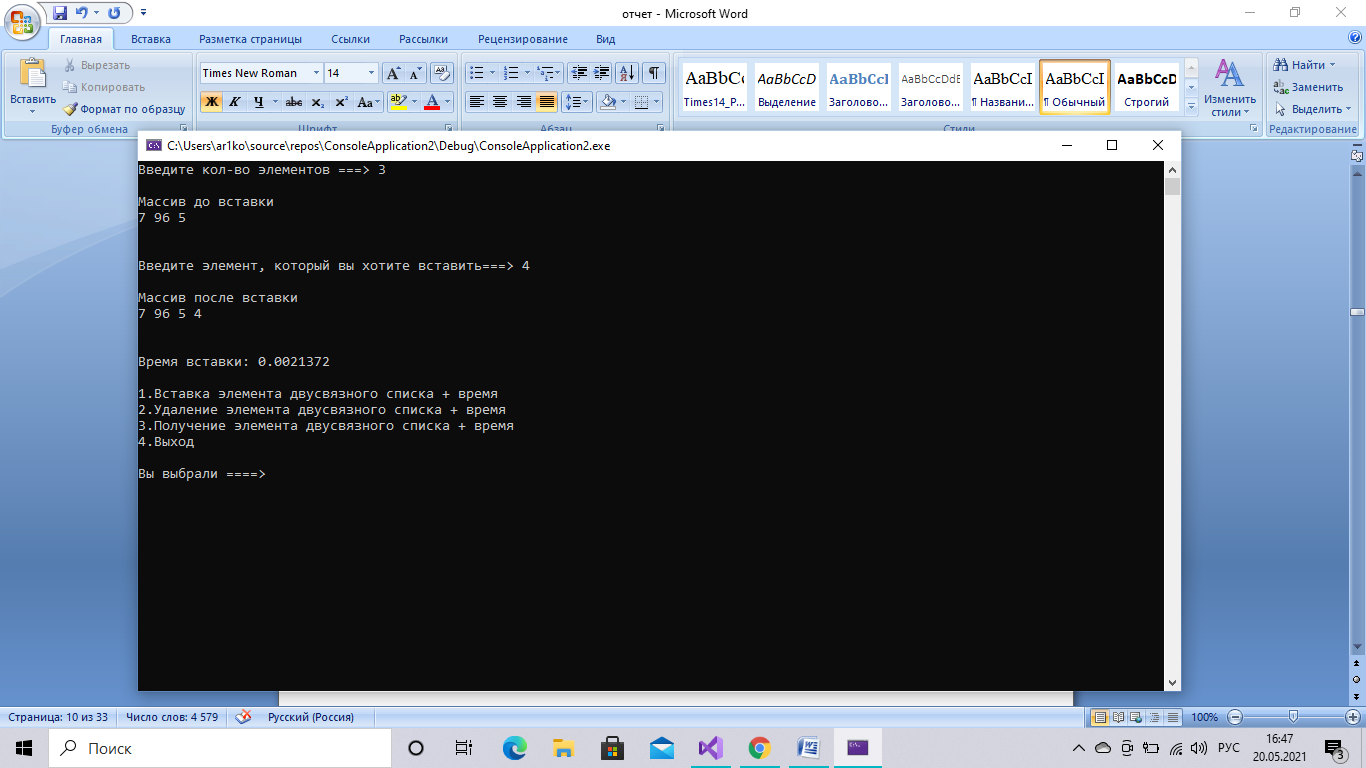
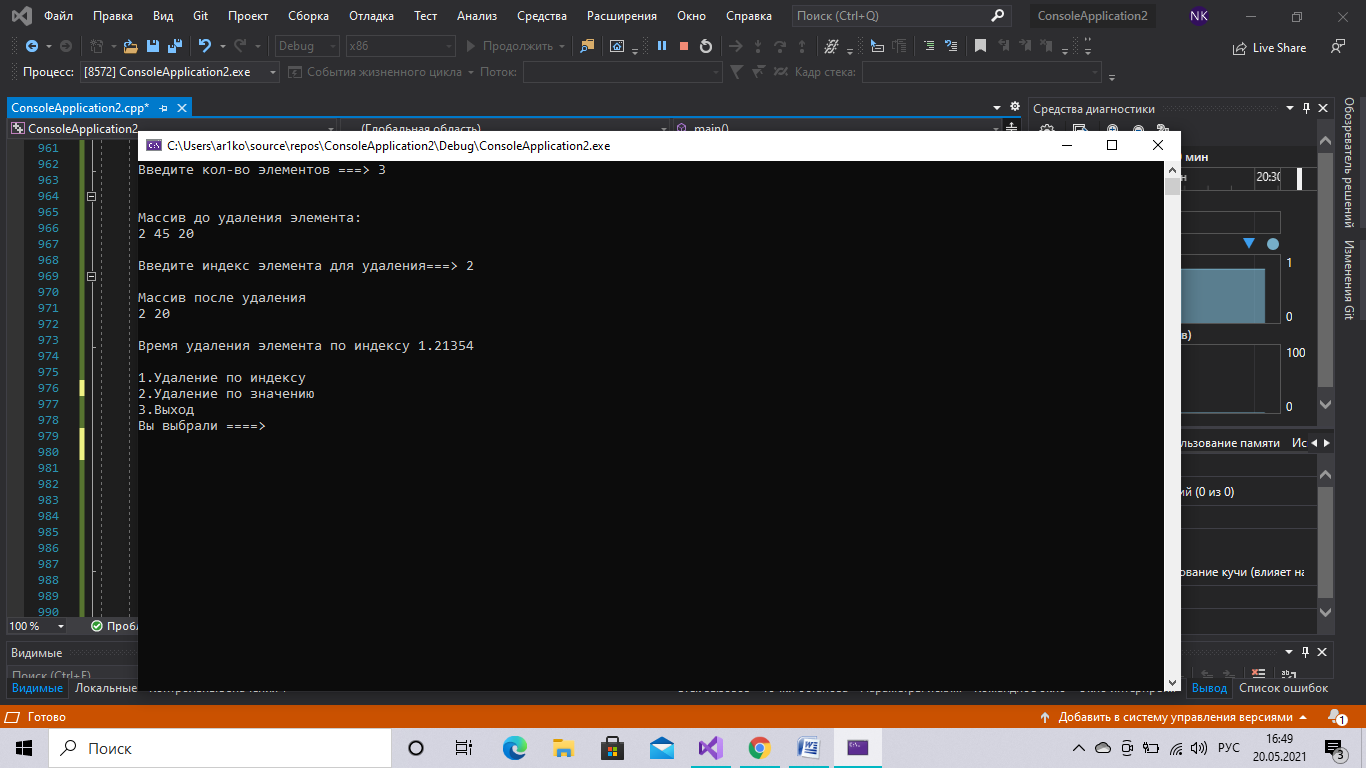
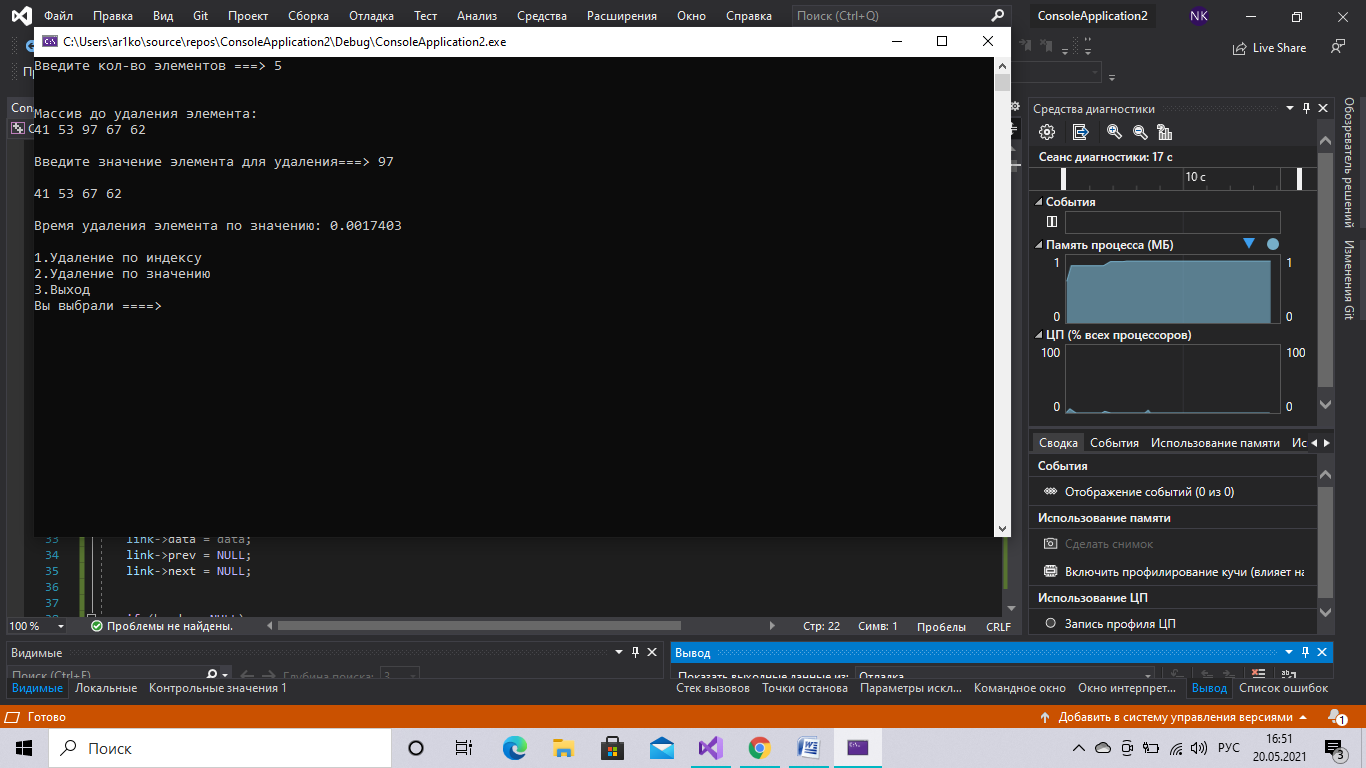
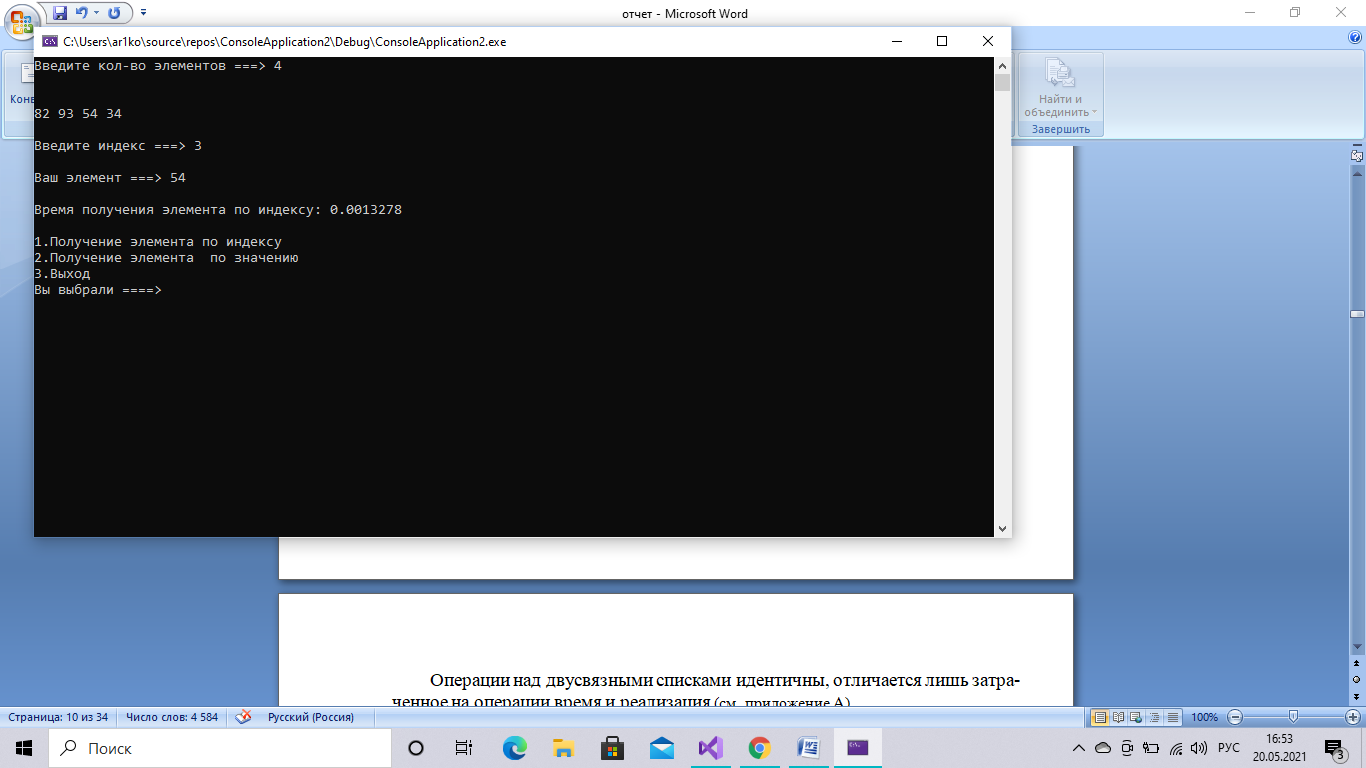
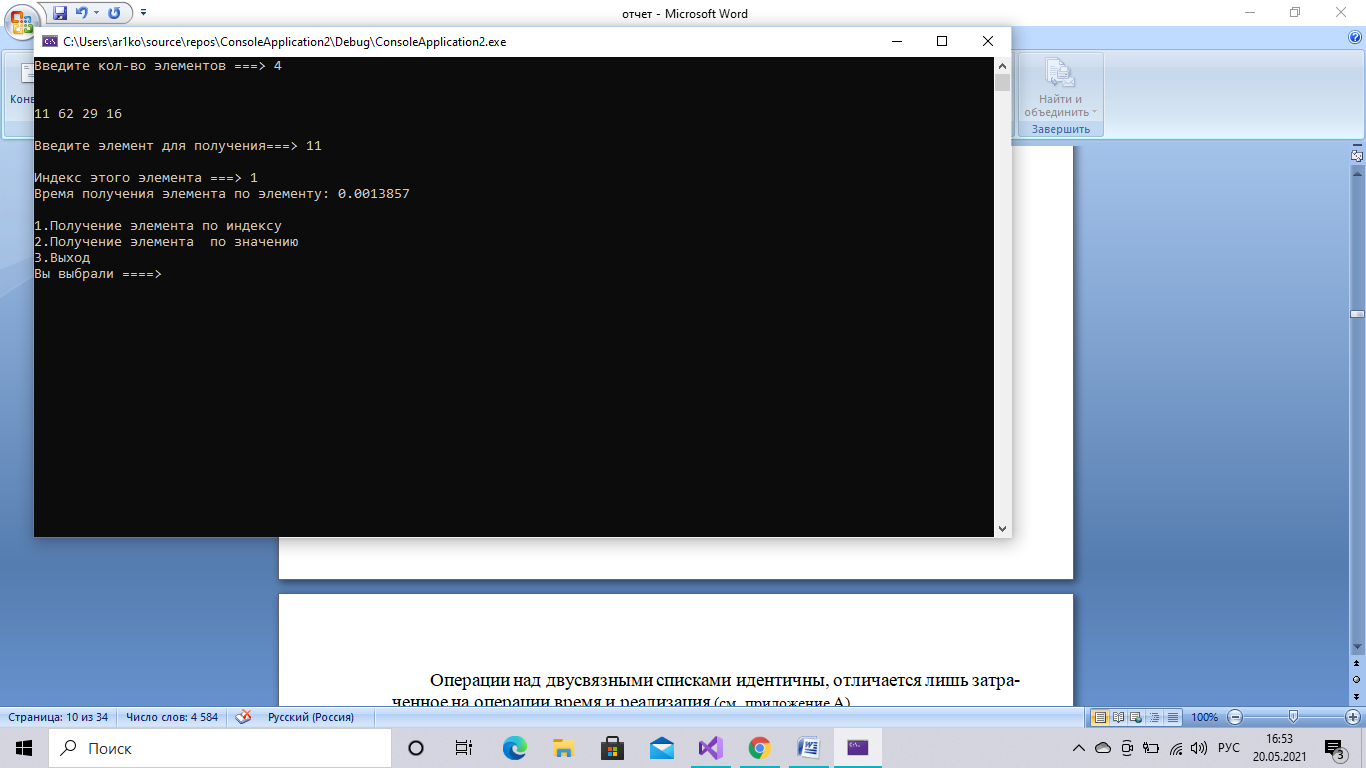
**3 задание +4 задание**

**1)  
  
2)  
а)  
  
б)  
  
3)  
а)**

**  
б)  
  
 5+6 задание**

**1)  
  
2)**

**  
3)  
  
7+8 задание**

**1)  
  
2)  
а)  
  
б)  
  
3)  
а)  
  
б)  
**

Операции над двусвязными списками идентичны, отличается лишь затраченное на операции время и реализация (см. приложение A)

**Выводы**.

В ходе выполнения практической работы я изучила свойства и организацию динамических массивов и двусвязных списков; получил практические навыки в работе с динамическими массивами и двусвязными списками; провел сравнительную характеристику скорости вставки, получения и удаления элементов из них.

ПриложениеА

**КОДПРОГРАММЫ**

#include <iostream>

#include <fstream>

#include <string>

#include <chrono>

#include <cstdlib>

using namespace std;

struct node

{

int data;

struct node\* prev;

struct node\* next;

};

struct node\* head = NULL;

struct node\* last = NULL;

struct node\* current = NULL;

void display()

{

struct node\* ptr = head;

while (ptr != NULL)

{

cout << ptr->data << " ";

ptr = ptr->next;

}

cout << endl;

}

void insert(int data)

{

struct node\* link = (struct node\*)malloc(sizeof(struct node));

link->data = data;

link->prev = NULL;

link->next = NULL;

if (head == NULL)

{

head = link;

return;

}

current = head;

while (current->next != NULL)

current = current->next;

current->next = link;

last = link;

link->prev = current;

}

void Mas()

{

int variant;

do

{

cout << "Введите вариант заполнения массива\n";

cout << "1.Ввод размерности и автоматическое заполнение рандомными числами\n";

cout << "2.Ввод чисел и подсчёт размерности\n";

cout << "3.Считывание с файла\n";

cout << "4.Выход\n";

cout << "\n";

cout << "Вы выбрали ====> ";

cin >> variant;

system("cls");

switch (variant)

{

case 1:

{

int kol;

cout << "Введите размер массива ====>";

cin >> kol;

if (kol == 0)

cout << "В массиве нет элементов";

int\* mas = new int[kol];

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

{

mas[i] = 0 + rand() % 99;

cout << mas[i] << " ";

}

cout << "\n";

delete[] mas;

break;

}

case 2:

{

int\* arr = new int[0];

int\* temp;

int n = 0, num;

std::cout << "!!!Введите элементы списка через пробел (закончите ввод любой буквой)!!!: " << '\n';

std::cin >> num;

while (!std::cin.fail()) {

temp = new int[n + 1];

for (int i = 0; i != n; i++) {

temp[i] = arr[i];

}

temp[n] = num;

delete[] arr;

arr = temp;

n++;

std::cin >> num;

}

cout << "Вы ввели: ";

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

{

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

}

if (n==0)

cout << "В массиве нет элементов";

else

cout << "\nРазмерность введенного массива "<< n;

cout << "\n";

std::cin.clear();

std::cin.ignore(32767, '\n');

break;

}

case 3:

{

string arr[9];

ifstream file("massiv.txt");

if (file.is\_open())

{

for (int i = 0; i < 9; ++i)

{

file >> arr[i];

}

}

for (int i = 0; i < 9; ++i)

{

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

}

cout << endl;

break;

}

case 4:

break;

}

} while (variant != 4);

}

void deleteAllNodes() {

node\* temp = new node();

while (head != NULL) {

temp = head;

head = head->next;

delete[] temp;

}

}

void Delete(int l)//Удаление по значению

{

node\* temp = head;

while ((temp != NULL) && (temp->data != l))

temp = temp->next;

if (temp == NULL)

return;

node\* prev = temp->prev;

node\* next = temp->next;

if (prev != NULL)

prev->next = next;

if (next != NULL)

next->prev = prev;

delete temp;

if (prev == NULL)

{

cout << "Ошибка: Такого элемента нет в массиве\n";

display();

deleteAllNodes();

}

else

{ // found match

cout << "\n";

display();

deleteAllNodes();

}

}

int displayEle(struct node\* head, int x)

{

if (head == NULL)

return -1;

if (x == 0)

return head->data;

return displayEle(head->next, x - 1);

}

int search(node\* head, int x)

{

node\* temp = head;

int pos = 0;

while (temp->data != x

&& temp->next != NULL) {

pos++;

temp = temp->next;

}

if (temp->data != x)

return -1;

return (pos + 1);

}

void addElement(node\*& head, int newdata)

{

//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

node\* tmp = new node;

tmp->data = newdata;

tmp->next = NULL;

tmp->prev = NULL;

if (!head)

head = tmp;

else

{

node\* last = head;

while (last->next != NULL)

last = last->next;

last->next = tmp;

tmp->prev = last;

}

}

//для удаления первого элемента

void delFirst()

{

struct node\* NodeToDel;

NodeToDel = head;

head = head->next;

head->prev = NULL;

free(NodeToDel);

}

//для удаления последнего элемента

void delLast()

{

struct node\* NodeToDel;

NodeToDel = last;

last = last->prev;

last->next = NULL;

free(NodeToDel);

}

//удаление элементов по индексу

void deleteNode(int pos)

{

struct node\* curNode;

int i;

curNode = head;

for (i = 1; i < pos && curNode != NULL; i++)

{

curNode = curNode->next;

}

if (pos == 1)

{

delFirst();

}

else if (curNode == last)

{

delLast();

}

else if (curNode != NULL)

{

curNode->prev->next = curNode->next;

curNode->next->prev = curNode->prev;

free(curNode);

}

}

int main()

{

std::chrono::system\_clock::time\_point start;

std::chrono::system\_clock::time\_point end;

std::chrono::duration <double> time;

int task, nom, nam;

int kol=0;

setlocale(LC\_ALL,"Russian");

do

{

cout << "Введите номер задания:\n";

cout << "1.Формирование целочисленного одномерного массива размерности N\n";

cout << "2.Время создания массива\n";

cout << "3.Вставка, удаление и получение элемента массива + определение времени\n";

cout << "4.Формирование двусвязного списка по вариантам + определение времени\n";

cout << "5.Вставка, удаление и получение элемента двусвязного списка + определение времени\n";

cout << "6.Выход\n";

cout << "\n";

cout << "Вы выбрали ====> ";

cin >> task;

system("cls");

switch (task)

{

case 1:

Mas();

break;

case 2:

{

do

{

cout << "Нужно померить время создания:\n";

cout << "1.Рандомного массива\n";

cout << "2.Массива пользователя\n";

cout << "3.Массива из файла\n";

cout << "4.Выход\n";

cout << "\n";

cout << "Вы выбрали ====> ";

cin >> nom;

system("cls");

switch (nom)

{

case 1:

{

start = std::chrono::system\_clock::now();

int kol;

cout << "Введите размер массива ====>";

cin >> kol;

if (kol == 0)

cout << "В массиве нет элементов";

int\* mas = new int[kol];

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

{

mas[i] = 0 + rand() % 99;

cout << mas[i] << " ";

}

cout << "\n";

delete[] mas;

end = std::chrono::system\_clock::now();

time = end - start;

std::cout << "Время на создание массива: " << time.count() << '\n';

break;

}

case 2:

{

start = std::chrono::system\_clock::now();

int\* arr = new int[0];

int\* temp;

int n = 0, num;

std::cout << "!!!Введите элементы списка через пробел (закончите ввод любой буквой)!!!: " << '\n';

std::cin >> num;

while (!std::cin.fail()) {

temp = new int[n + 1];

for (int i = 0; i != n; i++) {

temp[i] = arr[i];

}

temp[n] = num;

delete[] arr;

arr = temp;

n++;

std::cin >> num;

}

cout << "Вы ввели: ";

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

{

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

}

cout << "\n";

std::cin.clear();

std::cin.ignore(32767, '\n');

end = std::chrono::system\_clock::now();

time = end - start;

std::cout << "Время на создание массива: " << time.count() << '\n';

break;

}

case 3:

{

start = std::chrono::system\_clock::now();

string arr[9];

ifstream file("massiv.txt");

if (file.is\_open())

{

for (int i = 0; i < 9; ++i)

{

file >> arr[i];

}

}

for (int i = 0; i < 9; i++)

{

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

}

cout << "\n";

end = std::chrono::system\_clock::now();

time = end - start;

std::cout << "Время на создание массива: " << time.count() << '\n';

break;

}

case 4:

break;

}

} while (nom != 4);

}

case 3:

{

int k, l;

do

{

cout << "\nВыберите задание :\n";

cout << "1.Вставка элемента массива + определение времени\n";

cout << "2.Удаление элемента массива + определение времени\n";

cout << "3.Получение элемента массива + определение времени\n";

cout << "4.Выход\n";

cout << "\n";

cout << "Вы выбрали ====> ";

cin >> nam;

system("cls");

switch (nam)

{

case 1:

{

int kol;

cout << "Введите размер массива ====>";

cin >> kol;

if (kol == 0)

cout << "В массиве нет элементов";

int\* mas = new int[kol];

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

{

mas[i] = 0 + rand() % 99;

cout << mas[i] << " ";

}

cout << "\n";

int vstavka;

cout << "Введите элемент, который хотите вставить===> ";

cin >> vstavka;

start = std::chrono::system\_clock::now();

cout << "\n";

int element= kol + 1;

int\* arr = new int[element];

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

{

arr[i] = mas[i];

}

delete[] mas;

mas = arr;

arr[kol]=vstavka;

cout << " Итоговый массив:\n ";

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

{

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

}

cout << endl;

end = std::chrono::system\_clock::now();

time = end - start;

std::cout << "Время на вставку элемента: " << time.count() << '\n';

delete[] arr;

break;

}

case 2:

{

do

{

cout << "\n1.По индексу \n";

cout << "2.По значению\n";

cout << "3.Выход\n";

cout << "Вы выбрали ====> ";

cin >> k;

system("cls");

switch (k)

{

case 1:

{

cout << "Введите размерность массива===> ";

int n;

cin >> n;

if (n == 0)

{

cout << "В массиве ноль элементов";

}

else

{

int\* arr = new int[n];

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

{

arr[i] = 0 + rand() % 99;

}

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

{

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

}

cout << endl;

int\* massiv = new int[n - 1];

int index = 0;

cout << "Напишите позицию элемента для удаления===> ";

cin >> index;

start = std::chrono::system\_clock::now();

if ((index > n) || (index<0))

{

cout << "!!!Неверный индекс!!!\n ";

delete[] arr;

delete[] massiv;

}

else

if ((index <= n) && (index > 0))

{

index--;

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

{

massiv[i] = arr[i];

}

for (int i = index + 1; i < n; i++)

{

massiv[i - 1] = arr[i];

}

delete[] arr;

arr = massiv;

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

{

cout << massiv[i] << " ";

}

cout << endl;

n--;

if (n == 0)

{

cout << endl;

cout << "Нет элементов для удаления" << endl;

cout << endl;

delete[] massiv;

break;

}

cout << endl;

end = std::chrono::system\_clock::now();

time = end - start;

std::cout << "Время на удаление элемента: " << time.count() << '\n';

}

}

break;

}

case 2:

{

cout << "Введите размерность массива===> ";

int n;

cin >> n;

if (n == 0)

{

cout << "В массиве ноль элементов";

}

int\* arr = new int[n];

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

arr[i] = 0 + rand() % 99;

}

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

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

}

cout << endl;

int\* massiv = new int[n - 1];

int znachenie = 0;

cout << "Введите значение элемента, который хотите удалить===> ";

cin >> znachenie;

start = std::chrono::system\_clock::now();

int rast = distance(arr, find(arr, arr + n, znachenie));

//std::distance - это функция объединения интерфейсов, цель которой - позволить

//вычислить расстояния между различными типами итераторов: произвольный, двунаправленный,

//прямой и т. Д. Эта функция предназначена для сокрытия неэффективности прямого вычисления

//расстояния для итераторы с неслучайным доступом в ситуациях, когда вы действительно знаете,

//что делаете, и действительно хотите принять эту неэффективность.

if (rast != n)

{

int\* mass = new int[n - 1];

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

{

mass[i] = arr[i];

}

for (int i = rast + 1; i < n; i++)

{

mass[i - 1] = arr[i];

}

delete[] arr;

arr = mass;

cout << "Массив после удаления элемента\n";

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

{

cout << mass[i] << " ";

}

cout << endl;

n--;

end = std::chrono::system\_clock::now();

time = end - start;

std::cout << "\nВремя на удаление элемента: " << time.count() << '\n';

}

break;

}

case 3:

break;

}

} while (k != 3);

break;

}

case 3:

{

do

{

cout << "\n1.По индексу + время\n";

cout << "2.По значению + время\n";

cout << "3.Выход\n";

cout << "Вы выбрали ====> ";

cin >> l;

system("cls");

switch (l)

{

case 1:

{

cout << "Введите размерность массива===> ";

int n;

cin >> n;

if (n == 0)

{

cout << "В массиве ноль элементов";

}

int\* arr = new int[n];

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

arr[i] = 0 + rand() % 99;

}

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

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

}

cout << "\nВведите индекс для получения элемента===> ";

int ind;

cin >> ind;

start = std::chrono::system\_clock::now();

if ((ind > 0) && (ind <= n))

{

cout << "\nВаш элемент ===> " << arr[ind - 1] << endl;

delete[] arr;

}

else

{

cout << "\nПРОВЕРЬТЕ ВВОД\n";

delete[] arr;

break;

}

end = std::chrono::system\_clock::now();

time = end - start;

std::cout << "\nВремя на поиск элемента: " << time.count() << '\n';

break;

}

case 2:

{

cout << "Введите размерность массива===> ";

int n;

bool l=0;

cin >> n;

if (n == 0)

{

cout << "В массиве ноль элементов";

}

int\* arr = new int[n];

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

arr[i] = 0 + rand() % 99;

}

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

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

}

cout << "\nВведите значение элемента===> ";

int znach;

cin >> znach;

start = std::chrono::system\_clock::now();

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

{

if (arr[i] == znach)

{

cout << "Индекс этого элемента равен " << i + 1 << "\n";

l = 1;

}

}

if (l == 0)

{

cout << "\n!!!Такого элемента нет!!!\n ";

break;

}

end = std::chrono::system\_clock::now();

time = end - start;

std::cout << "\nВремя на поиск элемента: " << time.count() << '\n';

delete[] arr;

break;

}

case 3:

break;

}

}

while (l != 3);

break;

}

case 4:

break;

}

} while (nam != 4);

}

case 4:

{

int k;

do

{

cout << "\nВыберите задание :\n";

cout << "1.Рандомный массив размерности N + время создания \n";

cout << "2.Массив пользователя + определение времени\n";

cout << "3.Получение массива из файла + определение времени\n";

cout << "4.Выход\n";

cout << "\n";

cout << "Вы выбрали ====> ";

cin >> k;

system("cls");

switch (k)

{

case 1:

{

int n;

cout << "Введите кол-во элементов ===> ";

start = std::chrono::system\_clock::now();

cin >> n;

cout << endl;

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

{

insert(0 + rand() % 99);

}

display();

cout << endl;

end = std::chrono::system\_clock::now();

time = end - start;

std::cout << "\nВремя на создание рандомного массива: " << time.count() << '\n';

deleteAllNodes();

break;

}

case 2:

{

int\* arr = new int[0];

int\* temp;

int n = 0, num;

std::cout << "!!!Введите элементы списка через пробел (закончите ввод любой буквой)!!!: " << '\n';

start = std::chrono::system\_clock::now();

std::cin >> num;

while (!std::cin.fail()) {

temp = new int[n + 1];

for (int i = 0; i != n; i++) {

temp[i] = arr[i];

}

temp[n] = num;

delete[] arr;

arr = temp;

n++;

std::cin >> num;

}

cout << "Вы ввели: ";

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

{

insert(arr[i]);

}

if (n == 0)

cout << "В массиве нет элементов";

else

cout << "\nРазмерность введенного массива " << n << "\n";

display();

cout << "\n";

end = std::chrono::system\_clock::now();

time = end - start;

std::cout << "\nВремя на создание рандомного массива: " << time.count() << '\n';

deleteAllNodes();

std::cin.clear();

std::cin.ignore(32767, '\n');

break;

}

case 3:

{

start = std::chrono::system\_clock::now();

int znachenie;

ifstream file("massiv.txt");

if (file.is\_open())

{

for (int i = 0; i < 9; ++i)

{

file >> znachenie;

insert(znachenie);

}

}

cout << "Массив из файла\n";

display();

cout << endl;

end = std::chrono::system\_clock::now();

time = end - start;

std::cout << "\nВремя на чтение массива из файла: " << time.count() << '\n';

deleteAllNodes();

break;

}

case 4:

break;

}

}

while (k != 4);

break;

}

case 5:

{

int l;

do

{

cout << "\n1.Вставка элемента двусвязного списка + время\n";

cout << "2.Удаление элемента двусвязного списка + время\n";

cout << "3.Получение элемента двусвязного списка + время\n";

cout << "4.Выход\n";

cout << "\n";

cout << "Вы выбрали ====> ";

cin >> l;

system("cls");

switch (l)

{

case 1:

{

int n, vstavka;

cout << "Введите кол-во элементов ===> ";

cin >> n;

cout << endl;

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

{

insert(0 + rand() % 99);

}

cout << "Массив до вставки\n";

display();

cout << endl <<"\nВведите элемент, который вы хотите вставить===> ";

cin >> vstavka;

start = std::chrono::system\_clock::now();

addElement(head, vstavka);

cout << "\nМассив после вставки\n";

display();

cout << endl;

end = std::chrono::system\_clock::now();

time = end - start;

std::cout << "\nВремя вставки: " << time.count() << '\n';

deleteAllNodes();

break;

}

case 2:

{

int j;

do

{

cout << "\n1.Удаление по индексу\n";

cout << "2.Удаление по значению\n";

cout << "3.Выход\n";

cout << "Вы выбрали ====> ";

cin >> j;

system("cls");

switch (j)

{

case 1:

{

int n;

cout << "Введите кол-во элементов ===> ";

cin >> n;

cout << endl;

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

{

insert(0 + rand() % 99);

}

cout << "\nМассив до удаления элемента: \n";

display();

int l;

start = std::chrono::system\_clock::now();

cout << "\nВведите индекс элемента для удаления===> ";

cin >> l;

if ((l <= n) && (l > 0))

{

deleteNode(l);

cout << "\nМассив после удаления\n";

display();

deleteAllNodes();

}

else

{

cout << "Ошибка : Индекс не подходит условию";

deleteAllNodes();

break;

}

end = std::chrono::system\_clock::now();

time = end - start;

std::cout << "\nВремя удаления элемента по индексу " << time.count() << '\n';

break;

}

case 2:

{

int n;

cout << "Введите кол-во элементов ===> ";

cin >> n;

cout << endl;

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

{

insert(0 + rand() % 99);

}

cout << "\nМассив до удаления элемента: \n";

display();

start = std::chrono::system\_clock::now();

int l;

cout << "\nВведите значение элемента для удаления===> ";

cin >> l;

Delete(l);

end = std::chrono::system\_clock::now();

time = end - start;

std::cout << "\nВремя удаления элемента по значению: " << time.count() << '\n';

deleteAllNodes();

break;

}

case 3:

break;

}

}

while (j != 3);

break;

}

case 3:

{

int d;

do

{

cout << "\n1.Получение элемента по индексу\n";

cout << "2.Получение элемента по значению\n";

cout << "3.Выход\n";

cout << "Вы выбрали ====> ";

cin >> d;

system("cls");

switch (d)

{

case 1:

{

int n,l;

cout << "Введите кол-во элементов ===> ";

cin >> n;

cout << endl;

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

{

insert(0 + rand() % 99);

}

cout << "\n";

display();

start = std::chrono::system\_clock::now();

cout << "\nВведите индекс ===> ";

cin >> l;

if ((l <= n) && (l > 0))

{

cout <<"\nВаш элемент ===> " <<displayEle(head, l - 1) << endl;

deleteAllNodes();

}

else

{

cout << "Ошибка : Индекс не подходит условию";

deleteAllNodes();

break;

}

end = std::chrono::system\_clock::now();

time = end - start;

std::cout << "\nВремя получения элемента по индексу: " << time.count() << '\n';

break;

}

case 2:

{

int n , x;

cout << "Введите кол-во элементов ===> ";

cin >> n;

cout << endl;

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

{

insert(0 + rand() % 99);

}

cout << "\n";

display();

cout << "\nВведите элемент для получения===> ";

start = std::chrono::system\_clock::now();

cin >> x;

int c = search(head, x);

if (c == -1)

{

cout << "\nЭтого элемента нет в массиве " << endl;

deleteAllNodes();

break;

}

else

{

cout << "\nИндекс этого элемента ===> "<< c;

deleteAllNodes();

}

end = std::chrono::system\_clock::now();

time = end - start;

std::cout << "\nВремя получения элемента по элементу: " << time.count() << '\n';

break;

}

case 3:

break;

}

}

while (d != 3);

}

case 4:

break;

}

}

while (l != 4);

}

case 6:

break;

}

} while (task != 6);

}