Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Вятский государственный университет»

Колледж ВятГУ

**ОТЧЕТ**

**ПО ДОМАШНЕЙ КОНТРОЛЬНОЙ РАБОТЕ №6**

**«Реализация элементарных структур данных на основе статической памяти»**

**ПО МДК 05.02 РАЗРАБОТКА КОДА ИНФОРМАЦИОННЫХ СИСТЕМ**

Выполнил: студент учебной группы

ИСПк-204-52-00

Зяблицева Артема Павловича

Преподаватель:

Сергеева Елизавета Григорьевна

Киров

2023

В отчете должны отображаться:

1. Цель работы
2. Скриншот задания (с вариантом)
3. Описание алгоритма
4. Код программы
5. Результат выполнения программы
6. Вывод

**Цель работы:** изучение принципов работы с базовыми структурами данных, получение навыков организации case-menu.

Задание:

1. Написать программу для работы со структурой данных «Двусвязный список».
2. Структура данных должна быть реализована на основе статической памяти.
3. Работа со структурой должна быть осуществляется с помощью case-menu. Предусмотреть наглядную содержимого структуры.

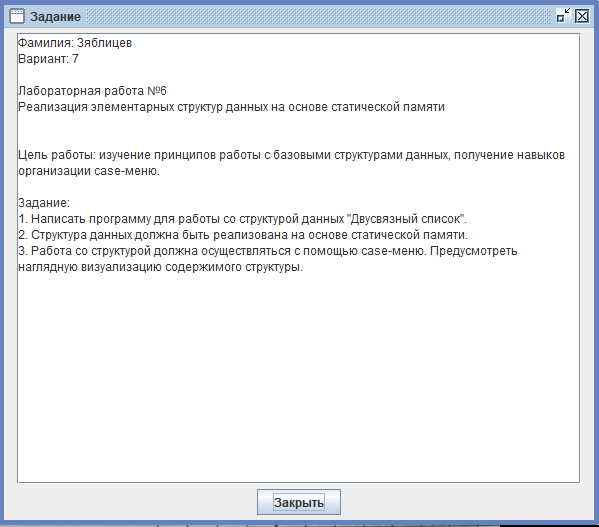


Рисунок 1 – Индивидуальное задание ДКР №6

**Описание алгоритма:**

Формируется двусвязный список посредством использования массива, список имеет ограниченное количество элементов. В список добавляется 5-10 случайных элементов, работа со списком представляет собой возможность добавить элемент в начало и конец списка, добавить элемент перед или после существующего элемента, удаление и нахождение элемента.

Процедура addToBeginning добавляет в свободный элемент массива значение введенного элемента, ссылка на предыдущий элемент равна 0, ссылка на следующий элемент равна head (первый элемент списка).

Процедура delete принимает значение элемента и проходит по списку пока узел со значением элемента не будет найден. Если указатель на предыдущий элемент не равен 0, то указатель на следующий элемент

предыдущего элемента равен указателю на следующий элемент удаляемого элемента, иначе head равен указателю на следующий элемент удаляемого элемента. Если указатель на следующий элемент не равен 0, то указатель на предыдущий элемент следующего элемента равен указателю на предыдущий элемент удаляемого элемента, иначе tail (последний элемент списка) равен указателю на предыдущий элемент удаляемого элемента.

**Код программы:**

**uses** CRT;

**const**

MAX\_SIZE = 100;

**type**

ElementType = Integer;

Node = **record**

data: ElementType;

prev: Integer;

next: Integer;

**end**;

**var**

list: **array** [1..MAX\_SIZE] **of** Node;

head: Integer = 0;

tail: Integer = 0;

free: Integer = 1;

**procedure** initializeList;

**var**

i: Integer;

**begin**

**for** i := 1 **to** MAX\_SIZE **do**

**begin**

list[i].prev := i - 1;

list[i].next := i + 1;

**end**;

list[MAX\_SIZE].next := 0;

**end**;

**function** getFreeNode: Integer;

**begin**

**if** free = 0 **then**

getFreeNode := 0

**else**

**begin**

getFreeNode := free;

free := list[free].next;

**end**;

**end**;

**procedure** releaseNode(index: Integer);

**begin**

list[index].prev := 0;

list[index].next := free;

free := index;

**end**;

**procedure** addToBeginning(value: ElementType);

**var**

index: Integer;

**begin**

index := getFreeNode;

**if** index <> 0 **then**

**begin**

list[index].data := value;

list[index].prev := 0;

list[index].next := head;

**if** head <> 0 **then**

list[head].prev := index;

head := index;

**if** tail = 0 **then**

tail := head;

**end**;

**end**;

**procedure** addToEnd(value: ElementType);

**var**

index: Integer;

**begin**

index := getFreeNode;

**if** index <> 0 **then**

**begin**

list[index].data := value;

list[index].prev := tail;

list[index].next := 0;

**if** tail <> 0 **then**

list[tail].next := index;

tail := index;

**if** head = 0 **then**

head := tail;

**end**;

**end**;

**procedure** insertBefore(existingValue: ElementType; newValue: ElementType);

**var**

index, currentNode: Integer;

**begin**

currentNode := head;

**while** (currentNode <> 0) **and** (list[currentNode].data <> existingValue) **do**

currentNode := list[currentNode].next;

**if** currentNode <> 0 **then**

**begin**

index := getFreeNode;

**if** index <> 0 **then**

**begin**

list[index].data := newValue;

list[index].prev := list[currentNode].prev;

list[index].next := currentNode;

**if** list[currentNode].prev <> 0 **then**

list[list[currentNode].prev].next := index

**else**

head := index;

list[currentNode].prev := index;

**end**;

**end**;

**end**;

**procedure** insertAfter(existingValue: ElementType; newValue: ElementType);

**var**

index, currentNode: Integer;

**begin**

currentNode := head;

**while** (currentNode <> 0) **and** (list[currentNode].data <> existingValue) **do**

currentNode := list[currentNode].next;

**if** currentNode <> 0 **then**

**begin**

index := getFreeNode;

**if** index <> 0 **then**

**begin**

list[index].data := newValue;

list[index].prev := currentNode;

list[index].next := list[currentNode].next;

**if** list[currentNode].next <> 0 **then** list[list[currentNode].next].prev := index

**else** tail := index;

list[currentNode].next := index;

**end**;

**end**;

**end**;

**procedure** delete(value: ElementType);

**var**

currentNode: Integer;

**begin**

currentNode := head;

**while** (currentNode <> 0) **and** (list[currentNode].data <> value) **do**

currentNode := list[currentNode].next;

**if** currentNode <> 0 **then**

**begin**

**if** list[currentNode].prev <> 0 **then**

list[list[currentNode].prev].next := list[currentNode].next

**else**

head := list[currentNode].next;

**if** list[currentNode].next <> 0 **then**

list[list[currentNode].next].prev := list[currentNode].prev

**else**

tail := list[currentNode].prev;

releaseNode(currentNode);

**end**;

**end**;

**function** find(value: ElementType): Boolean;

**var**

currentNode: Integer;

**begin**

currentNode := head;

**while** (currentNode <> 0) **and** (list[currentNode].data <> value) **do**

currentNode := list[currentNode].next;

find := currentNode <> 0;

**end**;

**procedure** displayList;

**var**

currentNode: Integer;

**begin**

println('Элементы списка:');

currentNode := head;

**while** currentNode <> 0 **do**

**begin**

print(list[currentNode].data);

currentNode := list[currentNode].next;

**end**;

println();

**end**;

**begin**

initializeList;

**var** r:integer;

**var** g:integer;g:=random(5,10);

**for var** i:=1 **to** g **do begin**

r:=random(-15,45);

addtobeginning(r);

**end**;

displayList;

**var** c:byte;

**repeat**

println('1 - Добавить элемент в начало');

println('2 - Добавить элемент в конец');

println('3 - Вставить после элемента');

println('4 - Вставить перед элементом');

println('5 - Найти элемент в списке');

println('6 - Удалить элемент Кирилла');

println('0 - Выход');

read(c);

**case** c **of**

1:**begin**

**var** el:integer;

println('zxczxczxczxczxczxczxczxczxczxczxczxczxc');

el:=readinteger('Введите элемент: ');

addtobeginning(el);

displayList;

**end**;

2:**begin**

**var** el:integer;

println('zxczxczxczxczxczxczxczxczxczxczxczxczxc');

el:=readinteger('Введите элемент: ');

addtoend(el);

displayList;

**end**;

3:**begin**

**var** el,aft:integer;

println('zxczxczxczxczxczxczxczxczxczxczxczxczxc');

el:=readinteger('Введите элемент: ');

aft:=readinteger('Введте элемент, после которого вставить: ');

insertAfter(aft, el);

displayList;

**end**;

4:**begin**

**var** el,bef:integer;

println('zxczxczxczxczxczxczxczxczxczxczxczxczxc');

el:=readinteger('Введите элемент: ');

bef:=readinteger('Введте элемент, перед которым вставить: ');

insertBefore(bef, el);

displayList;

**end**;

5:**begin**

**var** el:integer;

**var** fl:boolean;

println('zxczxczxczxczxczxczxczxczxczxczxczxczxc');

el:=readinteger('Введите элемент: ');

fl:=find(el);

**if** fl **then** print('Элемент в списке присутствует')

**else** print('Элемент в списке отсутствует');

println();

**end**;

6:**begin**

**var** el:integer;

println('zxczxczxczxczxczxczxczxczxczxczxczxczxc');

el:=readinteger('Введите элемент: ');

delete(el);

displayList;

**end**;

**end**;

**until** c=0;

**end**.

**Результат выполнения работы:**

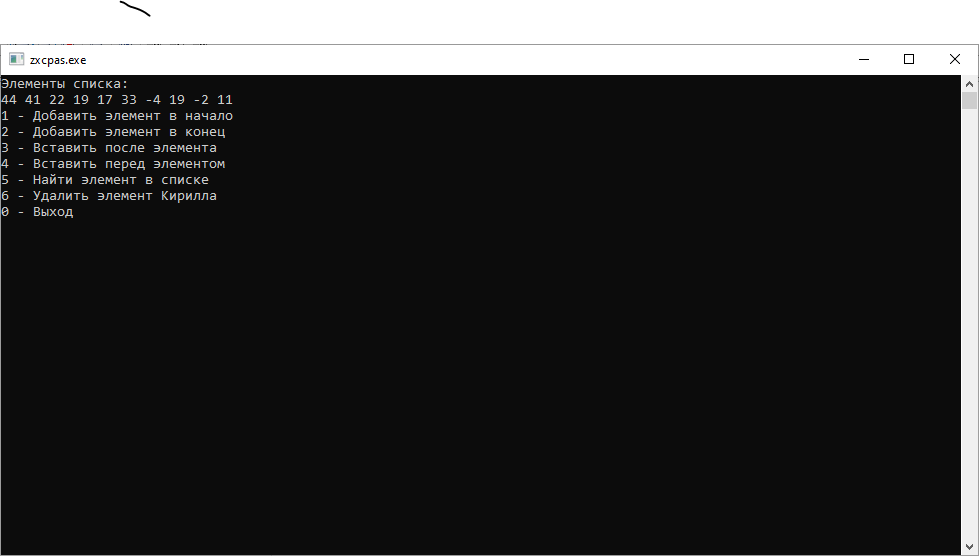


Рисунок 2 – программа

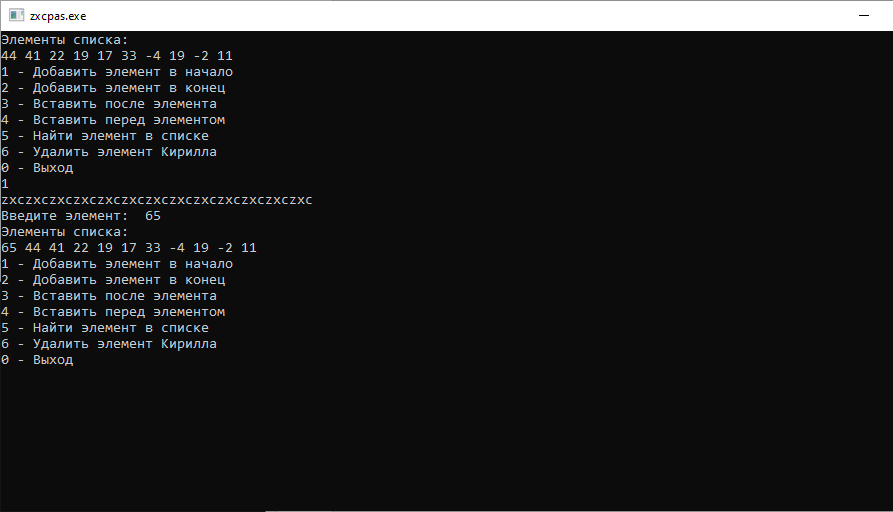


Рисунок 3 – программа

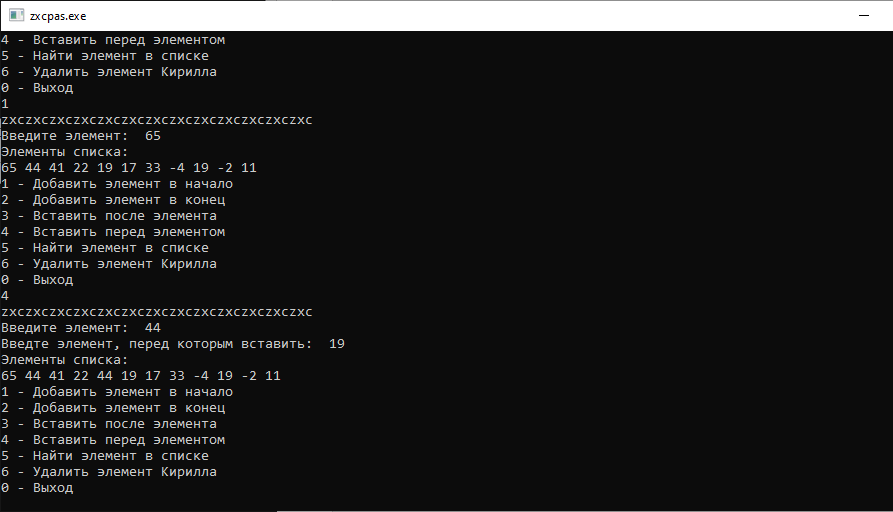


Рисунок 4 – программа

**Вывод:**

Для выполнения задания была использована реализация структуры данных двусвязный список. Список был реализован посредством статической памяти. Для организации case-меню была использована библиотека CRT.