Министерство образования Республики Беларусь

Учреждение образования «Полоцкий государственный университет»

Факультет информационных технологий

Кафедра технологий программирования

**Лабораторная работа №7**

**По дисциплине «Основы алгоритмизации и программирования»**

**Тема: «Методы организации списочных структур в динамической памяти»**

Выполнил: Макеёнок Д.И.

Группа 21-ИТ-1

Проверила: Пантелейко А.Ф.

Преподаватель-стажер кафедры ТП

Полоцк 2022

**Цель:** изучить методы организации списочных структур в динамической памяти. Реализовать алгоритмы добавления, удаления и сортировки списков.

**Вариант № 1**

**Задание 1.** Функции для создания нового списка и его удаления.

**Задание 2.** Функции добавления элемента в начало и после указанного элемента.

**Задание 3.** Функция удаления элемента.

**Задание 4.** Функция печати всех элементов списка.

**Дополнительное задание \*** Функция перестановки двух элементов списка (например, если на вход функции подается k-тый и g-тый элементы, то после срабатывания функции в списке на позиции k-того элемента окажется g-тый, а на позиции g-того – k-тый).

#include <iostream>

#include <stdlib.h>

#include <cstdlib>

#include <time.h>

#include <list>

#include <iostream>

#include <Windows.h>

using namespace std;

class List

{

int x, num;

List\* Head, \* Tail, \* Next, \* Prev;

public:

List() : num(0), Head(NULL), Tail(NULL) {}; //Инициализация с помощью конструктора по умолчанию

void Add(int);

void Add\_Start(int);

int Add\_After(int, int);

void Show();

int Replace(int, int);

void Del(int); //Функция принимает целочисленный параметр, обозначающий номер удаляемого элемента

~List();

};

int main()

{

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

system("CLS");

List List;

int n = -1, value, element, number, k, g;

while (n != 0)

{

cout << "Меню\n";

cout << "1. Добавить элемент\n";

cout << "2. Добавить элемент в начало\n";

cout << "3. Добавить элемент после элемента\n";

cout << "4. Удалить элемент\n";

cout << "5. Вывести список\n";

cout << "6. Поменять элементы местами\n";

cout << "7. Очистить список\n";

cout << "0. Выход\n";

cin >> n;

switch (n)

{

case 1: // добавить элемент в конец

system("CLS");

cout << "Элемент: ";

cin >> value;

system("CLS");

List.Add(value);

break;

case 2: // добавить элемент в начало

system("CLS");

cout << "Элемент: ";

cin >> value;

List.Add\_Start(value);

break;

case 3: // добавить элемент после элемента

system("CLS");

cout << "Элемент: ";

cin >> value;

cout << "Номер элемента после которого: ";

cin >> element;

List.Add\_After(value, element);

break;

case 4: // удалить элемент

system("CLS");

cout << "Номер элемента: ";

cin >> number;

system("CLS");

List.Del(number);

cout << "Элемент " << number << " удалён\n";

break;

system("CLS");

List.Show();

break;

case 5: //вывести список

system("CLS");

List.Show();

break;

case 6: // поменять местами

system("CLS");

cout << "Первая позиция: ";

cin >> k;

cout << "Вторая позиция: ";

cin >> g;

system("CLS");

List.Replace(k, g);

break;

system("CLS");

List.Show();

break;

case 7: // очистить список

system("CLS");

List.~List();

cout << "Список очищен\n";

break;

case 0:

system("CLS");

return 0;

break;

default:

system("CLS");

cout << "Проверьте правильность ввода\n";

break;

}

}

return 0;

}

void List::Add(int x)

{

List\* temp = new List;

temp->x = x;

temp->Next = NULL;

num++;

if (!Head)

{

temp->Prev = NULL;

Head = temp;

Tail = Head;

}

else

{

temp->Prev = Tail; //Указываем, что предыдущим элементом списка относительно добавленного, будет последний элемент существующего списка

Tail->Next = temp; //Следующий за последним существующим это непосредственно сейчас добавляемый элемент списка

Tail = temp; //После того как указали что есть настоящий и что предыдущий, объявляем, что последний существующий это только что добавленный элемент

}

}

void List::Add\_Start(int x)

{

List\* temp = new List;

temp->x = x;

temp->Next = NULL;

num++;

if (!Head)

{

temp->Prev = NULL;

Head = temp;

Tail = Head;

}

else

{

temp->Next = Head; //Указываем, что следующим элементом списка относительно добавленного, будет первый элемент существующего списка

Head->Prev = temp; //Предыдущий за последним существующим это непосредственно сейчас добавляемый элемент списка

Head = temp; //После того как указали что есть настоящий и что следующий, объявляем, что первый существующий это только что добавленный элемент

}

}

int List::Add\_After(int x, int elem)

{

List\* t = Head;

while (t)

{

if (t->x == elem)

{

List\* temp = new List;

temp->x = x;

temp->Next = NULL;

num++;

temp->Next = t->Next;

temp->Prev = t;

t->Next = temp;

t = t->Next;

t->Prev = temp;

cout << x << " Добавлен\n";

return 1;

}

t = t->Next;

}

cout << "Что-то не так\n";

return 0;

}

void List::Show()

{

List\* t = Head;

if (t)

{

while (t)

{

cout << t->x << " ";

t = t->Next;

}

cout << "\n";

}

else

cout << "Список пуст\n";

}

int List::Replace(int k, int g)

{

if (k > num - 1 || g > num - 1) // проверка на дурака

{

cout << "Неправильные номера\n";

return 0;

}

List\* pk = Head;

List\* pg = Head;

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

{

pk = pk->Next;

}

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

{

pg = pg->Next;

}

int temp;

temp = pk->x;

pk->x = pg->x;

pg->x = temp;

cout << "Элементы поменялись местами\n";

return 1;

}

void List::Del(int x)

{

if ((x == 1) && (Head->Next))

{ //Если удаляем первый, но есть и другие, то

List\* temp = Head;

Head = Head->Next;

Head->Prev = NULL;

delete temp;

num--;

return;

}

else if ((x == 1) && (Head == Tail))

{ //Если удаляем первый, но в списке только 1 элемент

Head->Next = NULL;

Head = NULL;

delete Head; //Удаляем указатель на начало

num = 0;

return;

}

// удаляемый элемент является последним элементом списка

if (x == num)

{

List\* temp = Tail;

Tail = Tail->Prev; //Отодвигаем хвост назад

Tail->Next = NULL;

delete temp; //Очищаем память от бывшего хвоста

num--;

return;

}

//Если же удаляемый элемент лежит где-то в середине списка, то тогда его можно удалить

List\* temp = Head, \* temp2; // temp-Удаляемый элемент, temp2 нужен, чтобы не потерять данные

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

temp = temp->Next;

temp2 = temp;

temp2->Prev->Next = temp->Next;

temp2->Next->Prev = temp->Prev;

delete temp;

num--;

}

List::~List()

{

while (Head)

{

Tail = Head->Next;

delete Head;

Head = Tail;

}

}

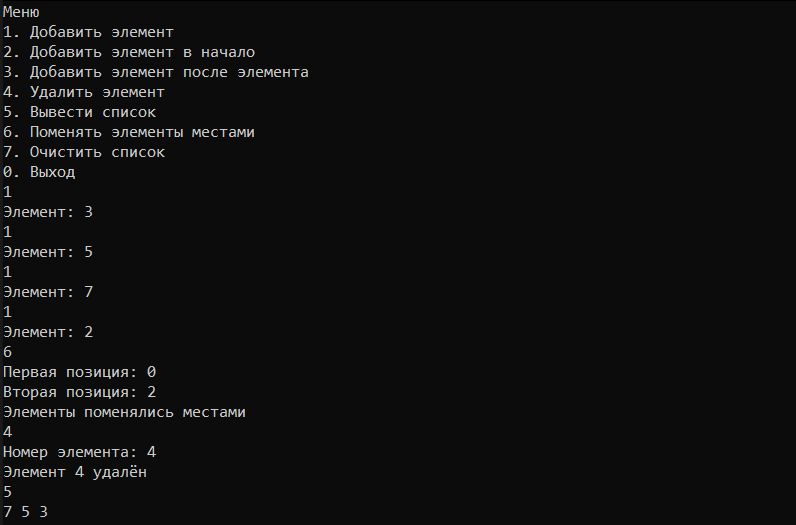


Рисунок 1 – Результат работы заданий

**Вывод:** в данной лабораторной работе были изучены методы организации списочных структур в динамической памяти. Реализованы алгоритмы добавления, удаления и сортировки списков.