МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «Полоцкий государственный университет»

Факультет информационных технологий

Кафедра технологий программирования

**Отчёт по лабораторной работе № 1 по курсу «Программирование сетевых приложений»**

«Реализация линейной структуры данных «Список» и основные

алгоритмы обработки»

Вариант №11

ВЫПОЛНИЛ студент группы 21-ИТ-1

Шиковец Е.А.

ПРОВЕРИЛ преподаватель

Виноградова А.Д.

Полоцк, 2022 г.

**ЦЕЛЬ РАБОТЫ:** ознакомиться с основами линейной структуры данных «Список», изучить основные алгоритмы обработки ЛСД «Список», научиться применять полученные знания на практике.

**ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ:**

1. Определение понятия список.

Это структура данных, которая построена на двусвязных списках. Это значит, что любой элемент знает только о предыдущем и о следующем элементах.

2. Классификация списков по количеству полей указателей.

По количеству полей указателей различают однонаправленный

(односвязный) и двунаправленный (двусвязный) списки.

3. Классификация списков по способу связи элементов.

По способу связи элементов различают линейные и циклические списки.

4. Виды списков. Отличие списков.

Различают 4 основных вида списков.

• Односвязный линейный список (ОЛС). Каждый узел ОЛС содержит 1

поле указателя на следующий узел. Поле указателя последнего узла содержит нулевое значение (указывает на NULL).

• Односвязный циклический список (ОЦС). Каждый узел ОЦС

содержит 1 поле указателя на следующий узел. Поле указателя последнего узла содержит адрес первого узла (корня списка).

• Двусвязный линейный список (ДЛС). Каждый узел ДЛС содержит

два поля указателей: на следующий и на предыдущий узел. Поле указателя на следующий узел последнего узла содержит нулевое значение (указывает на NULL). Поле указателя на предыдущий узел первого узла (корня списка) также содержит нулевое значение (указывает на NULL).

• Двусвязный циклический список (ДЦС). Каждый узел ДЦС содержит

два поля указателей: на следующий и на предыдущий узел. Поле указателя на следующий узел последнего узла содержит адрес первого узла (корня списка). Поле указателя на предыдущий узел первого узла (корня списка) содержит адрес последнего узла.

5. Основные действия, производимые над элементами односвязного

линейного списка.

Основные действия, производимые над элементами ОЛС:

• Инициализация списка

• Добавление узла в список

• Удаление узла из списка

• Удаление корня списка

• Вывод элементов списка

• Взаимообмен двух узлов списка

6. Какие аргументы принимает функция добавления узла в список

(ОЛС).

Функция добавления узла в список принимает два аргумента:

• Указатель на узел, после которого происходит добавление

• Данные для добавляемого узла.

7. Сколько полей указателей содержит каждый узел двусвязного

линейного списка?

Каждый узел двунаправленного (двусвязного) линейного списка (ДЛС) содержит два поля указателей — на следующий и на предыдущий узлы.

8. Сколько полей указателей содержит каждый узел односвязного

линейного списка?

Каждый узел однонаправленного (односвязного) линейного списка (ОЛС) содержит одно поле указателя на следующий узел. Поле указателя последнего узла содержит нулевое значение (указывает на NULL).

9. Основные действия, производимые над элементами двусвязного

линейного списка.

Основные действия, производимые над узлами ДЛС:

• Инициализация списка

• Добавление узла в список

• Удаление узла из списка

• Удаление корня списка

• Вывод элементов списка

• Вывод элементов списка в обратном порядке

• Взаимообмен двух узлов списка

**ОПИСАНИЕ ПРОДЕЛАННОЙ РАБОТЫ:**

При выполнении работы я пользовался IDE Visual Studio Code. В результате выполнения работы был написан следующий код:

#include <iostream>

#include <stdlib.h>

#include <cstdlib>

#include <time.h>

#include <list>

#include <iostream>

using namespace std;

class MyList

{ //Список MyList

int x, count\_;

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

public:

MyList() : count\_(0), Head(NULL), Tail(NULL){}; //Инициализация с помощью конструктора по умолчанию

void Add(int);

void Show();

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

~MyList();

};

/\*ДОБАВЛЕНИЕ ЭЛЕМЕНТА В СПИСОК\*/

void MyList::Add(int x)

{

MyList \*temp = new MyList;

temp->x = x;

temp->Next = NULL;

count\_++;

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

}

}

/\*ПОКАЗЫВАЕТ СПИСОК НА ЭКРАНЕ\*/

void MyList::Show()

{

MyList \*t = Head;

while (t)

{

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

t = t->Next;

}

cout << "\n\n";

}

/\*ФУНКЦИЯ УДАЛЕНИЯ КОНКРЕТНОГО ЭЛЕМЕНТА ДВУСВЯЗНОГО СПИСКА\*/

void MyList::Del(int x)

{

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

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

MyList \*temp = Head; //Указываем, что нам нужно начало списка

Head = Head->Next; //Сдвигаем начало на следующий за началом элемент

Head->Prev = NULL; //Делаем так, чтоб предыдущий началу элемент был пустым

delete temp; //Удаляем удаляемое начало

count\_--; //Обязательно уменьшаем счетчик

return; //И выходим из функции

}

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

count\_ = 0;

return;

}

// удаляемый элемент является последним элементом списка

if (x == count\_)

{

MyList \*temp = Tail;

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

Tail->Next = NULL;

delete temp; //Очищаем память от бывшего хвоста

count\_--;

return;

}

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

MyList \*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;

count\_--;

}

MyList::~MyList()

{

while (Head)

{

Tail = Head->Next;

delete Head;

Head = Tail;

}

}

int getRandomNumber(int min, int max)

{

int num = min + rand() % (max - min + 1);

return num;

}

int main()

{

MyList List1, List2;

int rand[20];

// Заполнение списка

for (int i = 0; i < 20; i++)

{

rand[i] = getRandomNumber(-50, 50);

}

rand[2] = 30;

for (int i = 0; i < 20; i++)

{

List1.Add(rand[i]);

}

cout << "List at start: " << endl;

List1.Show();

cout << endl;

// Форматирование списка

int it2 = 20;

for (int i = 0; i < 19; i++)

{

for (int j = i + 1; j < 20; j++)

{

if (rand[i] == rand[j])

{

rand[j] = 51;

it2--;

}

}

}

for (int i = 0; i < it2; i++)

{

if (rand[i] != 51)

{

List2.Add(rand[i]);

}

}

cout << "List at end: " << endl;

List2.Show();

// Удаление списка

for (int i = 0; i < 20; i++)

{

List1.Del(i);

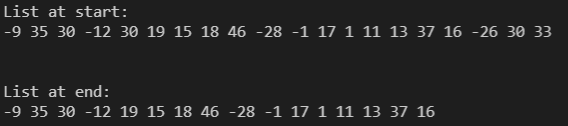
List2.Del(i);

}

return 0;

}

**РЕЗУЛЬТАТЫ ВЫПОЛЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ**:



**Вывод:** в результате выполнения лабораторной работы был реализован двухсвязный список с числами в диапазоне от -50 до 50, было выполнено индивидуальное задание. Также я научился работать со списками на ЯП C++.