Лабораторная работа № 4

Тема:«связный список»

Цель работы: Изучение работы линейных списков в языке C#.

Список — это линейная динамическая структура данных, у каждого элемента может быть только один предок и только один потомок. По сути своей это очень похоже на обыкновенный массив, с той лишь разницей, что размер его не имеет ограничений. Списки также подразделяются на несколько типов.

Односвязный список — элемент имеет указатель только на своего потомка.

Двусвязный список — 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.

Замкнутый (кольцевой, циклический) список — головной и хвостовой элементы которого указывают друг на друга.

На базе простого однонаправленного списка могут быть построены такие структуры данных, как очередь (queue) и стек (stack).

Очередь есть ничто иное, как список, операции чтения и добавления элементов в котором подвержены определенным правилам. При этом, при чтении элемента, он удаляется из очереди. Все операции проводятся по принципу «Первый пришел, первый вышел» (FIFO — first in, first out). Таким образом, для чтения в очереди доступна только голова, в то время как добавление проводится только в хвост.

Программный код

```
using System;
using System.Collections.Generic;
using System.Collections;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using static System.Console;
namespace ConsoleApp1

{

    class Program

    {

        public class Node<T> // Класс Node  может хранить данные любого типа

        {

            public Node(T data)

            {
                Data = data;
            }

            public T Data { get; set; } //  для хранения данных.

            public Node<T> Next { get; set; } //  ссылка на следующий узел.

        }

        public class LinkedList<T> : IEnumerable<T> // Мой P(x)

        {

            Node<T> perv; // Первый элемент
            Node<T> posled; // Последний элемент
            int count; // Количество элементов в списке.
            public void Add(T data) // добавление элемента

            {

                Node<T> node = new Node<T>(data);

                if (perv == null)

                    perv = node;

                else
                posled.Next = node;
                posled = node;
                count++;

            }

            IEnumerator IEnumerable.GetEnumerator() // перебор во внешней прграмме с помощью
цикла foreach

            {
                return ((IEnumerable)this).GetEnumerator();
            }
```

```
        IEnumerator<T> IEnumerable<T>.GetEnumerator()

        {

            Node<T> current = perv;
            while (current != null)

            {
                yield return current.Data;
                current = current.Next;
            }

        }

    }

    static void Main(string[] args)

    {

        int[] m = new int[3]; // Постоянный коэффициент

        Random rand = new Random();

        for (int i = 0; i < m.Length; i++)
        {
            m[i] = rand.Next(0, 9);
        }
        WriteLine("Значение X: ");

        int x = int.Parse(ReadLine());

        LinkedList<int> linkedList = new LinkedList<int>();

        linkedList.Add(x);

        if (m[0] != 0) linkedList.Add(m[0]);

        else
            WriteLine("m = 0, звено в список не включено ");

        if (m[1] != 0)

            linkedList.Add(m[1]);

        else
            WriteLine("m = 0, звено в список не включено ");

        foreach (var item in linkedList) // ВЫВОД ЗНАЧЕНИЙ УЗЛОВ

        {
            WriteLine("Значения, в узле: " + item);
        }

        WriteLine("Многочлен  P(x) = {0} * {1} + {2} * {3} ", m[0], x, m[1], x);

        // Рассчёт многочлена

        int b1 = m[0] * x;
        int b2 = m[1] * x;
        int result = b1 + b2;

        WriteLine("\nМногочлен P = {0}", result); // Вывод значения многочлена в точке X
        ReadKey();
```

```
        }
    }
}
```

Результат выполнения программы показан на рисунке 1.

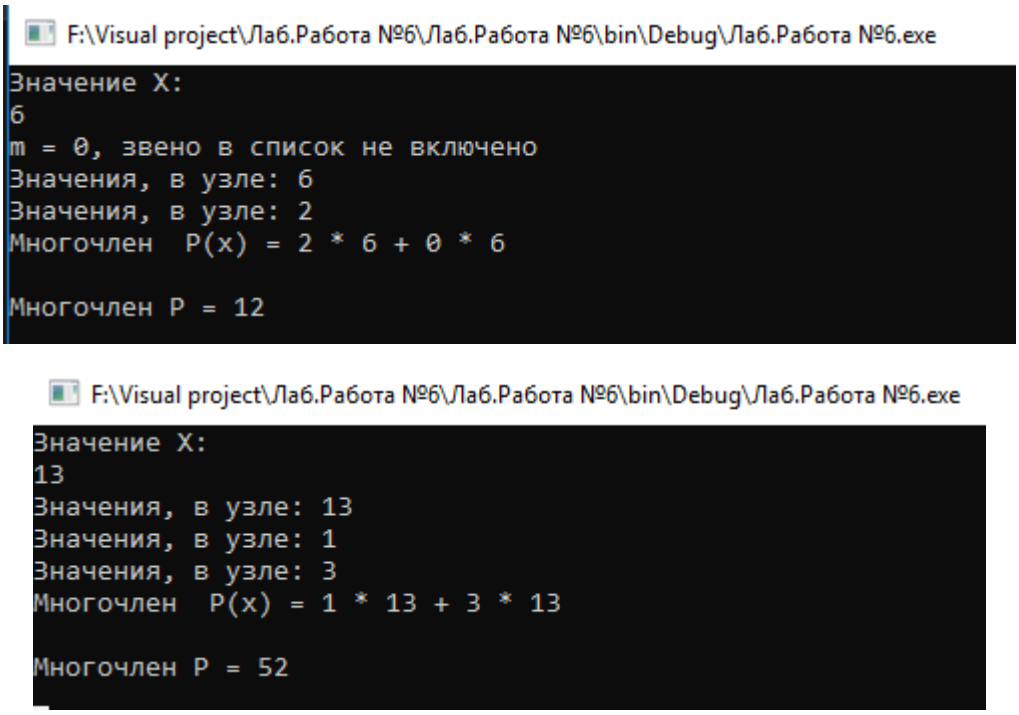

Рисунок 1 – Выполнение программного кода

Вывод: В данной лабораторной работе, были изучены структуры данных типа «связный список», применены полученные знания на практике.