**Лабораторная работа 4**

**«ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ. Списки»**

**Задание.** Составить программу на языке C#, содержащую библиотеку классов: узел списка и список. Программа выполняет следующие действия над списком:

1. добавление элемента по ключу;
2. удаление элемента по ключу;
3. выполнение запроса;
4. просмотр списка;

 На примере продемонстрировать состояние списка после выполнения операций

Вариант имеет вид xnadsp, где

x – вариант задания;

n - вид списка (1-однонаправленный, 2 – двунаправленный).

a – способ добавления (1 – добавление в начало, 2 - добавление в конец, 3-добавление после заданного ключа);

d – критерий удаления (1 – удаление первого, 2 – удаление последнего, 3-удаление по заданному ключу);

s – вид запроса;

p – вид просмотра (1-с начала, 2 – с конца ) – только для двунаправленного списка

Во все варианты к полям структуры добавляется ключ – числовое поле.

**варианты:**

х=1. Структура **“Справочное агентство”** c информацией о наличии мест на рейсы (название города, N рейса, тип самолета, цена билета, время вылета, время прибытия, наличие мест):

n=1

a=1

d=2

s= определение числа рейсов из определенного города;

х=2. Структура **"Абитуриент"** (шифр специальности, наименование специальности, план набора, количество поданных заявлений, из них медалистов):

n=1

a=2

d=1

s= определение плана набора по определенной специальности;

**Динамические структуры данных**

Простым способом хранения совокупности элементов является массив. Но массив имеет два недостатка – он способен хранить данные только одного типа, и его размер постоянен. Во многих случаях количество элементов данных может меняться в процессе работы. В таких случаях удобно рассматривать динамические структуры данных, число экземпляров которых может быть любым, в зависимости от потребности.

**Списки, их виды, алгоритмы основных операций**

Однонаправленный список представляет собой простейшую форму совокупности узлов, упорядоченных в линейную последовательность. Каждый узел является составным объектом, содержащим некий элемент, а также ссылку на следующий узел (рис. ).



Рис. Структура однонаправленного списка

Ссылка next внутри узла является указателем на следующий узел. Первый и последний узлы связного списка называются головой и хвостом списка соответственно. Хвост списка определим как узел с нулевым значением ссылки next, означающим, что список завершен. Описанный подобным образом связный список называется однонаправленным связным списком.

Как и массив, однонаправленный связный список объединяет элементы в определенную последовательность узлов и ссылок next между узлами. В отличие от массива, у этого списка нет заранее установленной длины, используемое им пространство пропорционально числу его элементов.

Для реализации списка в языке C# опишем класс узел списка:

*public class Node*

 *{ private Node next; private int data;*

 *public Node Next*

 *{get { return next; }*

 *set { next = value; } }*

 *public int Data*

 *{ get { return data; }*

 *set { data = value; } }*

 *public Node(int t)*

 *{ Next = null; Data = t; } }*

с полями data (данные) и next (указатель на следующий узел). В классе имеется конструктор, создающий экземпляр.

Второй необходимый класс описывает сам список:

*class list\_int*

*{private Node head;*

 *private int cnt;*

 *public Node Head*

 *{get { return head; }*

 *set { head = value; } }*

 *public int Cnt*

 *{ get { return cnt; }*

 *set { cnt= value; } }*

 *public list\_int() {Cnt = 0;*

 *Head = null; }*

*//Добавление элемента в начало списка*

 *public void AddHead(int t)*

 *{Node n = new Node(t);*

 *n.Next=Head;*

 *Head = n; Сnt++; }*

*//Получение всех элементов списка в виде массива*

*public int Getlist(out int[] cc)*

*{int i = 0;*

*cc = new int[20];*

*Node current = Head;*

*while (current != null)*

*{cc[i] = current.Data; i++;*

 *current = current.Next;} return Cnt;}*

*// Вставка элемента в любое место списка*

*public Node InsertSll(int t, Node prev)*

*// указатель на элемент, после которого делается вставка,*

*// если prev=null - вставка перед 1-ым эл-том*

 *{ //если head=null - список пустой*

 *Node cur; cur = new Node(t);*

 *if (prev != null)*

 *{//если есть предыдущий элемент - вставка в середину списка*

 *cur.Next=prev.Next; prev.Next=cur;}*

 *else*

 *{// вставка в начало списка*

 *cur.Next=Head;*

*// новый элемент указывает на бывший первый элемент списка;*

 *//если head=null, то новый элемент будет и последним*

 *Head = cur;} // новый элемент становится первым в списке,*

 *// указатель на начало теперь указывает на него*

 *Cnt++;*

 *return cur; }*

*//Поиск элемента списка по значению элемента*

 *public Node Find(int t)*

 *{Node current = Head;*

 *while (current != null)*

 *{if (t == current.Data) return current;*

 *current = current.Next;};*

 *return null; }*

*// Удаление элемента из любого места списка*

*public void DeleteSll(Node del)*

*//del указатель на элемент, который удаляется*

 *{ Node prev; // адрес предыдущего элемента }*

 *if (del == Head)*

*// если удаляемый элемент - первый в списке,*

*// то следующий за ним становится первым*

 *Head = del.Next;*

 *else*

*// удаление из середины списка*

*// приходится искать элемент, предшествующий удаляемому;*

*//поиск производится перебором списка с самого его начала,*

*//пока не будет найдет элемент, поле next которого совпадает*

*//с адресом удаляемого элемента.*

 *{ prev = Head;*

 *while (prev.Next != del && prev.Next!= null)*

 *prev = prev.Next;*

 *prev.Next=del.Next;*

*// предыдущий элемент теперь указывает на следующий за удаляемым*

 *}; Cnt--; } }*

Поля этого класса – head (голова списка) и cnt (количество элементов списка). Кроме конструктора имеются методы AddHead (добавление в начало), InsertSll(вставка элемента в любое место списка), Find (поиск указателя элемента списка по значению) и DeleteSll(удаление элемента списка по его указателю). Алгоритмы методов добавления и удаления понятны из рисунков и комментариев.



Рис.Вставка элемента в начало списка



Рис. Вставка элемента в середину списка



Рис. Удаление элемента из списка

[**Двусвязный список**](http://evilcoderr.blogspot.ru/2013/01/doubly-linked-list-c.html)

Двусвязный список - динамическая структура данных, каждый из элементов которой содержит ссылку на предыдущий элемент и на следующий элемент.

Пример двусвязного списка:



* Также есть головной элемент First, который указывает на начало списка. First.Prev=null
* Переменная Last указывает на конец списка. Last.Next=null
* Переменная Current указывает на текущий элемент (выступает обычно в роли итератора списка, то есть элемента с помощью которого можно перемещаться по списку)
* Переменная целого типа cnt, которая содержит информацию о количестве элементов в списке.

Показаны только самые базовые операции вставки в конец и начало,

Push\_Front: Вставка нового элемента в начало списка, сделав его первым, надо, чтобы, First ссылался на этот объект.

В начальном состоянии Prev и Next ссылаются на null, First.Prev = null, First.Next ссылается на след объект и т.д.

1. Мы сделаем так, чтобы новый объект ссылался на First
2. Потом мы сделаем так, чтобы и First и newNode ссылались на новый объект.
3. Теперь просто новый объект ссылается на предыдущий первый.

Push\_Back: Добавление нового элемента в конец, Last должен ссылаться на новый объект.

В начальном состоянии Last.Next=null.

1. Last.Next теперь ссылается на новый объект.
2. newNode.Prev ссылается на Last.
3. Last ссылается на новый объект.

Каждый объект описан классом Node (Узел)

* Data - наши данные, которые содержатся в узле.
* Next - ссылка на следующий элемент.
* Prev - ссылка на предыдущий элемент.

class two\_Node

 { private int data;

 private two\_Node next;

 private two\_Node prev;

 public int Data

 { get { return data; }

 set { data = value; } }

 public two\_Node(int Data)

 { this.Data = Data; }

 public two\_Node Next

 { get { return next; }

 set { next = value; } }

 public two\_Node Prev

 { get { return prev; }

 set { prev = value; } }

class two\_list\_int

 { private two\_Node first;

 private two\_Node current;

 private two\_Node last;

 private int cnt; private two\_Node head;

 public two\_Node Head

 { get { return head; }

 set { head = value; } }

 public int Cnt

 { get { return cnt; }

 set { cnt = value; } }

 public two\_Node Current

 { get { return current; }

 set { current = value; } }

 public two\_Node Last

 { get { return last; }

 set { last = value; } }

 public two\_Node First

 { get { return first; }

 set { first = value; } }

 public two\_list\_int()

 { Cnt = 0;

 First = Current = Last = null; }

 public bool isEmpty()//проверка на пустоту

 { return Cnt == 0; }

 public void Insert\_Index(int newElement, int index)

//вставить по индексу

 { if (index < 1 || index > Cnt)

//ошибка, если неправильный индекс

 { throw new InvalidOperationException(); }

 else if (index == 1) //если начало

 { Push\_Front(newElement); }

 else if (index == Cnt)

 //если конец

 { Push\_Back(newElement); }

 else

 //иначе ищем элемент с таким индексом

 { int count = 1;

 Current = First;

 while (Current != null && count != index)

 { Current = Current.Next;

 count++; }

 two\_Node newNode = new two\_Node(newElement);

 //создаем объект

 Current.Prev.Next = newNode;

 newNode.Prev = Current.Prev;

 Current.Prev = newNode;

 newNode.Next = Current; } }

 public void Push\_Front(int newElement)

 { two\_Node newNode = new two\_Node(newElement);

 if (First == null)

 { First = Last = newNode; }

 else

 { newNode.Next = First;

 First = newNode;

 //First и newNode указывают на один и тот же объект

 newNode.Next.Prev = First; }

 Cnt++; }

 public two\_Node Pop\_Front()

 { if (First == null)

 { throw new InvalidOperationException(); }

 else

 { two\_Node temp = First;

 if (First.Next != null)

 { First.Next.Prev = null; }

 First = First.Next; Cnt--;

 return temp; } }

 public void Push\_Back(int newElement)

 { two\_Node newNode = new two\_Node(newElement);

 if (First == null)

 { First = Last = newNode; }

 else

 { Last.Next = newNode;

 newNode.Prev = Last;

 Last = newNode; }

 Cnt++; }

 public two\_Node Pop\_Back()

 { if (Last == null)

 { throw new InvalidOperationException(); }

 else

 { two\_Node temp = Last;

 if (Last.Prev != null)

 {Last.Prev.Next = null; }

 Last = Last.Prev;

 Cnt--;

 return temp; } }

 public void ClearList() //полностью очистить список

 { while (!isEmpty())

 { Pop\_Front(); } }

 public void DeleteElement(int index)

 { //удалить элемент по индексу

 if (index < 1 || index > Cnt)

 { throw new InvalidOperationException(); }

 else if (index == 1)

 { Pop\_Front(); }

 else if (index == Cnt)

 { Pop\_Back(); }

 else

 { int count = 1; Current = First;

 while (Current != null && count != index)

 { Current = Current.Next; count++; }

 Current.Prev.Next = Current.Next;

 Current.Next.Prev = Current.Prev; } }

 public two\_Node FindNode(int Data) //найти Node и вернуть его

 { Current = First;

 while (Current != null)

 {Current = Current.Next; }

 return Current; }

 public int GetIndex(int Data) //достать индекс по значению элемента

 { Current = First;

 int index = 1;

 while (Current != null)

 { Current = Current.Next;

 index++; }

 return index; } }