# Лабораторная работа №7

# Динамические структуры данных

## 1. Цель работы:

1) Получить практические навыки работы с однонаправленными списками;

2) получить практические навыки работы с двунаправленными списками;

3) получить практические навыки работы с деревьями.

## 2. Краткие теоретические сведения

Во многих задачах требуется использовать данные, у которых конфигурация, размеры и состав могут меняться в процессе выполнения программы. Для их представления используют динамические информационные структуры. К таким структурам относят:

* однонаправленные списки;
* двунаправленные списки;
* бинарные деревья.

Они отличаются способом связи отдельных элементов и/или допустимыми операциями. Динамическая структура может занимать несмежные участки динамической памяти.

### 2.1. Однонаправленные списки

Наиболее простой динамической структурой является однонаправленный список, элементами которого служат объекты структурного типа (рис.).

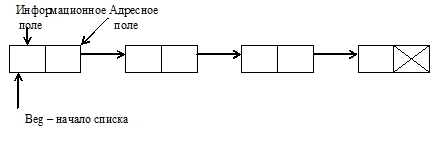


Рис.. Линейный однонаправленный список

Описание простейшего элемента такого списка выглядит следующим образом:

class имя\_типа

{

информационное поле;

адресное поле;

};

* информационное поле – это поле любого, ранее объявленного или стандартного, типа;
* адресное поле – это указатель на объект того же типа, что и определяемая структура, в него записывается адрес следующего элемента списка.

// простейшее определение элемента списка

class Point

{

public int data; //информационное поле

public Point next;//адресное поле

}

Информационных полей может быть несколько.

Для удобства работы с элементом списка добавим в класс Point конструктор без параметров, конструктор с параметрами и перегрузим метод ToString() для вывода информационного поля.

class Point

{

public int data;//информационное поле

public Point next;//адресное поле

public Point()//конструктор без параметров

{

data=0;

next=null;

}

public Point(int d)//конструктор с параметрами

{

data=d;

next=null;

}

public override string ToString()

{

return data+" ";

}

}

Для того, чтобы создать список, нужно создать сначала первый элемент списка, а затем в цикле добавить к нему остальные элементы. Добавление может выполняться как в начало, так и в конец списка.

//создание элемента списка

static Point MakePoint(int d)

{

Point p = new Point(d);

return p;

}

//добавление в начало однонаправленного списка

static Point MakeList(int size)

{

Random rnd = new Random();

int info = rnd.Next(0, 11);

Console.WriteLine("The element {0} is adding...", info);

Point beg = MakePoint(info);//создаем первый элемент

for (int i = 1; i < size; i++)

{

info = rnd.Next(0, 11);

Console.WriteLine("The element {0} is adding...", info);

//создаем элемент и добавляем в начало списка

Point p = MakePoint(info);

p.next = beg;

beg = p;

}

return beg;

}

//добавление в конец списка

static Point MakeListToEnd(int size)

{

Random rnd = new Random();

int info = rnd.Next(0, 11);

Console.WriteLine("The element {0} is adding...", info);

Point beg = MakePoint(info);//первый элемент

Point r = beg;//переменная хранит адрес конца списка

for (int i = 1; i < size; i++)

{

info = rnd.Next(0, 11);

Console.WriteLine("The element {0} is adding...",info);

//создаем элемент и добавляем в конец списка

Point p = MakePoint(info);

r.next = p;

r = p;

}

return beg;

}

Для обработки списка организуется цикл, в котором нужно переставлять указатель p с помощью оператора p=p.next на следующий элемент списка до тех пор, пока указатель p не станет равен 0, т. е. будет достигнут конец списка.

static void ShowList(Point beg)

{

//проверка наличия элементов в списке

if (beg == null) {

Console.WriteLine("The List is empty");

return;

}

Point p = beg;

while (p!=null)

{

Console.Write(p);

p = p.next;//переход к следующему элементу

}

Console.WriteLine();

}

В динамические структуры легко добавлять элементы и из них легко удалять элементы, т. к. для этого достаточно изменить значения адресных полей.

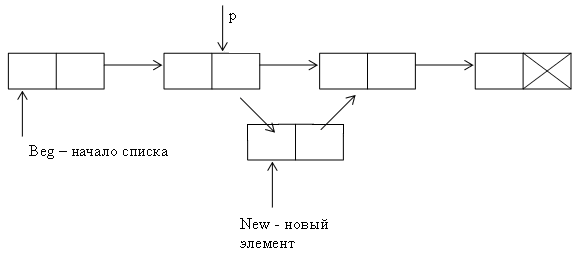


Рис. Добавление элемента в список

static Point AddPoint(Point beg, int number)

{

Random rnd = new Random();

int info = rnd.Next(10, 100);

Console.WriteLine("The element {0} is adding...", info);

//создаем новый элемент

Point NewPoint = MakePoint(info);

if (beg == null)//список пустой

{

beg = MakePoint(rnd.Next(10, 100));

return beg;

}

if (number == 1) //добавление в начало списка

{

NewPoint.next = beg;

beg = NewPoint;

return beg;

}

//вспом. переменная для прохода по списку

Point p = beg;

//идем по списку до нужного элемента

for (int i = 1; i < number-1 && p != null; i++)

p = p.next;

if (p == null)//элемент не найден

{

Console.WriteLine("Error! The size of List less than Number");

return beg;

}

//добавляем новый элемент

NewPoint.next = p.next;

p.next = NewPoint;

return beg;

}

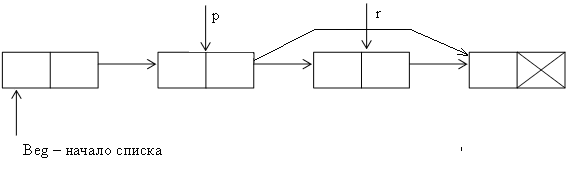


Рис. Удаление элемента из списка

static Point DelElement(Point beg, int number)

{

if (beg == null)//пустой список

{

Console.WriteLine("Error! The List is empty");

return null;

}

if (number == 1)//удаляем первый элемент

{

beg = beg.next;

return beg;

}

Point p = beg;

//ищем элемент для удаления и встаем на предыдущий

for (int i = 1; i < number - 1 && p != null; i++)

p = p.next;

if (p == null)//если элемент не найден

{

Console.WriteLine("Error! The size of List less than Number");

return beg;

}

//исключаем элемент из списка

p.next = p.next.next;

return beg;

}

### 2.1. Двунаправленные списки

Двунаправленный список имеет два адресных поля, которые указывают на следующий элемент списка и на предыдущий. Поэтому двигаться по такому списку можно как слева направо, так и справа налево.

//определение двунаправленного списка

class Point

{

public int data;

public Point next, //адрес следующего элемента

pred;//адрес предыдущего элемента

public Point()

{

data = 0;

next = null;

pred = null;

}

public Point(int d)

{

data = d;

next = null;

pred = null;

}

public override string ToString()

{

return data + " ";

}

}

//формирование двунаправленного списка

static Point MakeList(int size) //добавление в начало

{

Random rnd = new Random();

int info = rnd.Next(0, 11);

Console.WriteLine("The element {0} is adding...", info);

Point beg = MakePoint(info);

for (int i = 1; i < size; i++)

{

info = rnd.Next(0, 11);

Console.WriteLine("The element {0} is adding...", info);

Point p = MakePoint(info);

p.next = beg;

beg.pred = p;

beg = p;

}

return beg;

}

### 2.3. Бинарные деревья

Бинарное дерево – это динамическая структура данных, состоящая из узлов, ка­ждый из которых содержит, кроме данных, не более двух ссылок на различные бинарные деревья. На каждый узел имеется ровно одна ссылка.

Описать такую структуру можно следующим образом:

class Point2

{

public int data;

public Point left,//адрес левого поддерева

right;//адрес правого поддерева

public Point()

{

data = 0;

left = null;

right = null;

}

public Point(int d)

{

data = d;

left = null;

right = null;

}

public override string ToString()

{

return data + " ";

}

}

Начальный узел называется корнем дерева. Узел, не имеющий поддеревьев, называется листом. Исходящие узлы называются предками, входящие — потом­ками. Высота дерева определяется количеством уровней, на которых располага­ются его узлы.

Если дерево организовано таким образом, что для каждого узла все ключи его ле­вого поддерева меньше ключа этого узла, а все ключи его правого поддерева — больше, оно называется деревом поиска. Одинаковые ключи не допускаются. В дереве поиска можно найти элемент по ключу, двигаясь от корня и переходя на левое или правое поддерево в зависимости от значения ключа в каждом узле. Такой поиск гораздо эффективнее поиска по списку, поскольку время поиска определяется высотой дерева, а она пропорциональна двоичному логарифму ко­личества узлов.

В идеально сбалансированном дереве количество узлов справа и сле­ва отличается не более чем на единицу.

Линейный список можно представить как вырожденное бинарное дерево, в котором каждый узел имеет не более одной ссылки. Для списка среднее время поиска равно половине длины списка.

Деревья и списки являются рекурсивными структурами, т. к. каждое поддерево также является деревом. Таким образом, дерево можно определить как рекурсивную структуру, в которой каждый элемент является:

* либо пустой структурой;
* либо элементом, с которым связано конечное число поддеревьев.

Действия с рекурсивными структурами удобнее всего описываются с помощью рекурсивных алгоритмов.

#### 2.4.1. Обход дерева

Для того, чтобы выполнить определенную операцию над всеми узлами дерева, все узлы надо обойти. Такая задача называется обходом дерева. При обходе узлы должны посещаться в определенном порядке. Существуют три принципа упорядочивания. Рассмотрим дерево, представленное на рис.

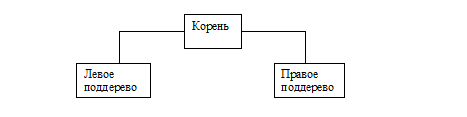


Рис. Бинарное дерево

На этом дереве можно определить три метода упорядочивания:

* Слева направо: Левое поддерево – Корень – Правое поддерево;
* Сверху вниз: Корень – Левое поддерево – Правое поддерево;
* Снизу вверх: Левое поддерево – Правое поддерево – Корень.

Эти три метода можно сформулировать в виде рекурсивных алгоритмов.

void Run(Point p)

//обход сверху вниз

{

if(p!=null)

{

<обработка p.data>

Run(p.left);//переход к левому поддереву

Run(p.right);//переход к правому поддереву

}

}

Если в качестве операции обработки узла поставить операцию вывода информационного поля, то мы получим функцию для печати дерева.

/\*рекурсивная функция для печати дерева по уровням с обходом слева направо\*/

static void ShowTree(Point p, int l)

{

if(p!=null)

{

ShowTree(p.left,l+3);//переход к левому поддереву

//формирование отступа

for (int i = 0; i < l; i++) Console.Write(" ");

Console.WriteLine(p.data);//печать узла

ShowTree(p.right,l+3);//переход к правому поддереву

}

}

#### 2.4.2. Формирование дерева

//построение дерева поиска

//формирование первого элемента дерева

static Point first(int d)

{

Point p=new Point(d);

return p;

}

//добавление элемента d в дерево поиска

static Point Add(Point root, int d)

{

Point p = root; //корень дерева

Point r = null;

//флаг для проверки существования элемента d в дереве

bool ok = false;

while (p!=null && !ok)

{

r = p;

//элемент уже существует

if (d == p.data) ok = true;

else

if (d < p.data) p = p.left; //пойти в левое поддерево

else p = p.right; //пойти в правое поддерево

}

if (ok) return p;//найдено, не добавляем

//создаем узел

Point NewPoint = new Point(d);//выделили память

// если d<r.key, то добавляем его в левое поддерево

if (d < r.data) r.left = NewPoint;

// если d>r.key, то добавляем его в правое поддерево

else r.right = NewPoint;

return NewPoint;

}

//построение идеально сбалансированного дерева

static Point IdealTree(int size, Point p)

{

Point r;

int nl, nr;

if(size==0){p=null;return p;}

nl=size/2;

nr=size-nl-1;

int number=Convert.ToInt32(Console.ReadLine());

r = new Point(d);

i++;

r.left=IdealTree(nl,r.left);

r.right=IdealTree(nr,r.right);

return r;

}

## 3. Постановка задачи

1. Сформировать однонаправленный список, тип информационного поля указан в варианте.
2. Распечатать полученный список.
3. Выполнить обработку списка в соответствии с заданием.
4. Распечатать полученный список.
5. Удалить список из памяти.
6. Сформировать двунаправленный список, тип информационного поля указан в варианте.
7. Распечатать полученный список.
8. Выполнить обработку списка в соответствии с заданием.
9. Распечатать полученный список.
10. Удалить список из памяти.
11. Сформировать идеально сбалансированное бинарное дерево, тип информационного поля указан в варианте.
12. Распечатать полученное дерево.
13. Выполнить обработку дерева в соответствии с заданием, вывести полученный результат.
14. Преобразовать идеально сбалансированное дерево в дерево поиска.
15. Распечатать полученное дерево.

## 4. Варианты

|  |  |  |  |
| --- | --- | --- | --- |
| №  варианта | Однонаправленный | Двунаправленный | Бинарное дерево |
| 1 | Тип информационного поля int.  Удалить из списка все элементы с четными информационными полями. | Тип информационного поля string  Добавить в список элемент с заданным номером. | Тип информационного поля char.  Найти количество элементов с заданным ключом. |
| 2 | Тип информационного поля double.  Удалить из списка все элементы с четными номерами (2, 4, 6 и. т. д.). | Тип информационного поля string.  Добавить в список элементы с номерами 1, 3, 5 и т. д. | Тип информационного поля int.  Найти максимальный элемент в дереве. |
| 3 | Тип информационного поля int.  Удалить из списка первый элемент с четным информационным полем. | Тип информационного поля double.  Добавить в список элемент после элемента с заданным информационным полем. | Тип информационного поля string.  Найти количество листьев в дереве. |
| 4 | Тип информационного поля int.  Удалить из списка последний элемент с четным информационным полем. | Тип информационного поля string.  Добавить в список элемент с заданным номером. | Тип информационного поля double.  Найти минимальный элемент в дереве. |
| 5 | Тип информационного поля string.  Добавить в список элемент после элемента с заданным информационным полем. | Тип информационного поля int.  Удалить из списка все элементы с четными информационными полями. | Тип информационного поля char.  Найти высоту дерева. |
| 6 | Тип информационного поля string.  Добавить в список элемент с заданным номером. | Тип информационного поля double.  Удалить из списка все элементы с четными номерами (2, 4, 6 и. т. д.). | Тип информационного поля int.  Найти среднее арифметическое элементов дерева. |
| 7 | Тип информационного поля double.  Добавить в список после каждого элемента с отрицательным информационным полем элемент с информационным полем равным 0. | Тип информационного поля int.  Удалить из списка первый элемент с четным информационным полем. | Тип информационного поля string  Найти количество элементов дерева, начинающихся с заданного символа. |
| 8 | Тип информационного поля string  Добавить в список элементы с номерами 1, 3, 5 и т. д. | Тип информационного поля int.  Удалить из списка последний элемент с четным информационным полем. | Тип информационного поля char.  Найти количество элементов с заданным ключом. |
| 9 | Тип информационного поля int.  Удалить из списка все элементы с четными информационными полями. | Тип информационного поля string  Добавить в список элементы с номерами 1, 3, 5 и т. д. | Тип информационного поля double.  Найти максимальный элемент в дереве. |
| 10 | Тип информационного поля double.  Удалить из списка все элементы с четными номерами (2, 4, 6 и. т. д.). | Тип информационного поля string.  Добавить в список элемент после элемента с заданным информационным полем. | Тип информационного поля int  Найти количество листьев в дереве. |
| 11 | Тип информационного поля int.  Удалить из списка первый элемент с четным информационным полем. | Тип информационного поля string.  Добавить в список элемент с заданным номером. | Тип информационного поля double.  Найти минимальный элемент в дереве. |
| 12 | Тип информационного поля int.  Удалить из списка последний элемент с четным информационным полем. | Тип информационного поля string.  Добавить в список элемент с заданным номером. | Тип информационного поля char.  Найти высоту дерева. |
| 13 | Тип информационного поля string.  Добавить в список элемент после элемента с заданным информационным полем. | Тип информационного поля double.  Удалить из списка все элементы с отрицательными информационными полями. | Тип информационного поля int.  Найти среднее арифметическое элементов дерева. |
| 14 | Тип информационного поля string.  Добавить в список элемент с заданным номером. | Тип информационного поля int.  Удалить из списка последний элемент с четным информационным полем. | Тип информационного поля char.  Найти количество элементов с заданным ключом. |
| 15 | Тип информационного поля double.  Добавить в список после каждого элемента с отрицательным информационным полем элемент с информационным полем равным 0. | Тип информационного поля int.  Удалить из списка все элементы с четными номерами (2, 4, 6 и. т. д.). | Тип информационного поля string.  Найти количество элементов дерева, начинающихся с заданного символа |
| 16 | Тип информационного поля string.  Добавить в список элементы с номерами 1, 3, 5 и т. д. | Тип информационного поля int.  Удалить из списка первый элемент с четным информационным полем. | Тип информационного поля int.  Найти максимальный элемент в дереве. |
| 17 | Тип информационного поля int.  Удалить из списка все элементы с четными информационными полями. | Тип информационного поля string.  Добавить в список элемент с заданным номером. | Тип информационного поля double.  Найти количество листьев в дереве. |
| 18 | Тип информационного поля double.  Удалить из списка все элементы с четными номерами (2, 4, 6 и. т. д.). | Тип информационного поля string.  Добавить в список элемент с заданным номером. | Тип информационного поля int.  Найти минимальный элемент в дереве. |
| 19 | Тип информационного поля int.  Удалить из списка первый элемент с четным информационным полем. | Тип информационного поля string.  Добавить в список элементы с номерами 1, 3, 5 и т. д. | Тип информационного поля char.  Найти высоту дерева. |
| 20 | Тип информационного поля int.  Удалить из списка последний элемент с четным информационным полем. | Тип информационного поля string.  Добавить в список элемент после элемента с заданным информационным полем. | Тип информационного поля double.  Найти среднее арифметическое элементов дерева. |
| 21 | Тип информационного поля string.  Добавить в список элемент после элемента с заданным информационным полем. | Тип информационного поля int.  Удалить из списка все элементы с четными информационными полями. | Тип информационного поля string.  Найти количество элементов дерева, начинающихся с заданного символа. |
| 22 | Тип информационного поля string.  Добавить в список элемент с заданным номером. | Тип информационного поля double.  Удалить из списка все элементы с четными номерами (2, 4, 6 и. т. д.). | Тип информационного поля char.  Найти количество элементов с заданным ключом. |
| 23 | Тип информационного поля double.  Добавить в список после каждого элемента с отрицательным информационным полем элемент с информационным полем равным 0. | Тип информационного поля int.  Удалить из списка первый элемент с четным информационным полем. | Тип информационного поля string.  Найти количество листьев в дереве. |
| 24 | Тип информационного поля string.  Добавить в список элементы с номерами 1, 3, 5 и т. д. | Тип информационного поля int.  Удалить из списка последний элемент с четным информационным полем. | Тип информационного поля double.  Найти максимальный элемент в дереве. |
| 25 | Тип информационного поля int.  Удалить из списка все элементы с четными информационными полями. | Тип информационного поля string.  Добавить в список элементы с номерами 1, 3, 5 и т. д. | Тип информационного поля double.  Найти минимальный элемент в дереве. |

## 5. Методические указания

1. Для формирования элементов списков/дерева написать отдельные функции.
2. Для формирования списков/дерева, удаления добавления элементов, поиска заданных элементов написать отдельные функции.
3. В функции Main() должны быть размещены только описания переменных и обращения к соответствующим функциям.
4. Если в списке/дереве отсутствуют элементы, соответствующие критерию поиска (например, при удалении элемента с номером k, k больше, чем количество элементов в списке), должно быть выведено сообщение о том, что требуемые элементы не найдены.
5. Интерфейс реализовать с помощью текстового меню.
6. Предусмотреть проверку правильности ввода соответствующих типов данных.

## 6. Содержание отчета

1. Постановка задачи (общая и для конкретного варианта).
2. Анализ задачи.
3. Проектирование функций (вход, выход, описание решаемой задачи для основного потока и исключительных ситуаций).
4. Определения функций для реализации поставленных задач.
5. Определение функции Мain().
6. Тесты.