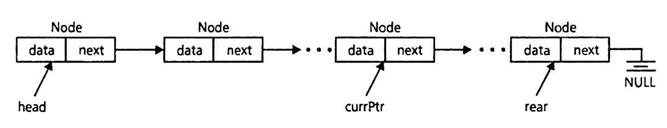
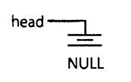
Узел (node) состоит из поля данных и указателя, обозначающего следую­ щий элемент в списке. Указатель — это соединитель, связывающий вместе отдельные узлы списка. 

Связанный список состоит из множества узлов, первый элемент которого (front), — это узел, на который указывает голова (head). Список связывает узлы вместе от первого до конца или хвоста (rear) списка. Хвост определяется как узел, чье поле указателя имеет значение NULL (0). Списочные прило­ жения проходят по узлам, следуя за каждым указателем на следующий узел.

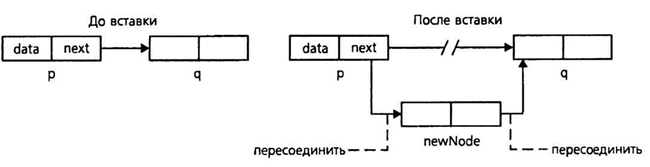
В любой точке сканирования на текущее положение ссылается указатель currPtr. Для списка без узлов head будет содержать значение NULL.



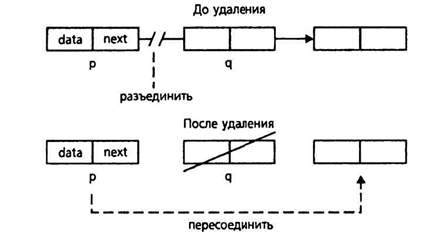
**Объявление типа Node**

Узел с его данными (data) и полями указателей (next) является строительным блоком для связанного списка. Структура узла имеет операции, которые инициализируют данные-члены, и методы управления указателями для до­ ступа к следующему узлу.

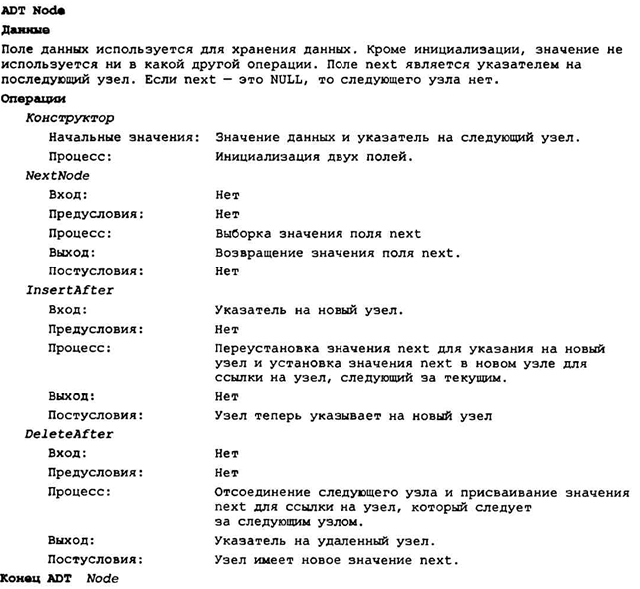
Далее приведен рисунок, иллюстрирующий базовые операции для обработки узлов. В любом данном узле р мы можем реализовать операцию IпsertAfter, которая присоединяет новый узел после текущего. Процесс начинается прерыванием соединения с последующим узлом q, вставкой newNode и восстановлением связей.



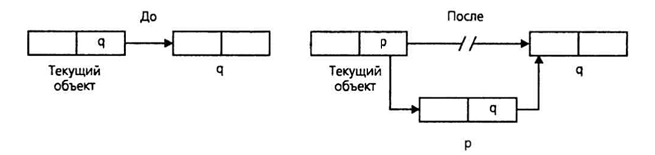
Аналогичный процесс описывает операцию DeleteAfter, которая удаляет узел, следующий за текущим. Мы отсоединяем р от следующего за ним узла q и затем соединяем р с узлом, следующим за q.



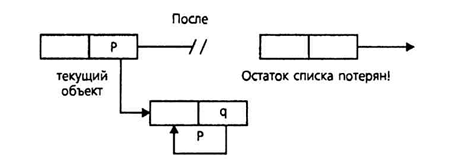
Структура узлов с операциями вставки и удаления описывает абстрактный тип данных. Для каждого узла эти операции относятся непосредственно к его последующему (следующему) узлу.



InsertAfter принимает узел р в качестве параметра и добавляет его в список в качестве следующего узла. Первоначально текущий объект указы­ вает на узел, адресом которого является q (значение в поле next). Алгоритм изменяет два указателя. Поле указателей р устанавливается на q, а полю указателей в текущем объекте присваивается значение р.



Порядок присваиваний указателей очень важен. Предположим, что операторы присваивания имеют обратный порядок.



DeleteAfter удаляет узел, который следует за текущим объектом и связывает его поле указателей со следующим узлом в списке. Если после теку­ щего объекта нет никакого узла (next == NULL), функция возвращает NULL.

Иначе, функция возвращает адрес удаленного узла для случая, если программисту необходимо освободить память этого узла. Алгоритм DeleteAfter сохраняет адрес следующего узла в tempPtr. Поле next узла tempPtr опрeделяет узел в списке, на который должен теперь указывать текущий объект.

Возвращается указатель на узел tempPtr. Процесс требует присваивания толь­ ко одного указателя.



Мой класс Node на C#:

public class Node<T>

{

// Значение узла

public T Value { get; set; }

// Ссылка на следующий узел

public Node<T> Next { get; set; }

// Конструктор для создания узла с указанным значением и ссылкой на следующий узел

public Node(T value, Node<T> next)

{

Value = value;

Next = next;

}

// Метод для вставки нового узла после текущего узла

public void InsertAfter(T value)

{

// Создаем новый узел со значением value и ссылкой на следующий узел текущего узла

Node<T> newNode = new Node<T>(value, Next);

// Обновляем ссылку на следующий узел текущего узла, чтобы она указывала на новый узел

Next = newNode;

}

// Метод для удаления следующего узла

public void DeleteAfter()

{

// Проверяем, что следующий узел не равен null (то есть, что он существует)

if (Next != null)

{

// Обновляем ссылку на следующий узел текущего узла, чтобы она указывала на узел, следующий за следующим узлом

Next = Next.Next;

}

}

}

class Program

{

static void Main()

{

// Создаем три узла

Node<string> firstNode = new Node<string>("A", null);

Node<string> secondNode = new Node<string>("B", null);

Node<string> thirdNode = new Node<string>("C", null);

// Связываем узлы в список

firstNode.Next = secondNode;

secondNode.Next = thirdNode;

// Выводим список в файл

using (StreamWriter writer = new StreamWriter("list.txt"))

{

Node<string> currentNode = firstNode;

while (currentNode != null)

{

writer.WriteLine(currentNode.Value);

currentNode = currentNode.Next;

}

}

// Удаляем список без потери ссылки на первый

firstNode.Next = null;

}

}