9.8. Циклические списки

Оканчивающийся NULL-символом связанный список — это последователь­ ность узлов, которая начинается с головного узла и заканчивается узлом, поле указателя next которого имеет значение NULL. В разделе 9.1 разработана библиотека функций для сканирования такого списка и для вставки и удаления узлов. В этом разделе разрабатывается альтернативная модель, называемая циклическим связанным списком (circular linked list), которая упрощает разработку и кодирование алгоритмов последовательных списков. Многие профес­ сиональные программисты используют циклическую модель для реализации связанных списков.

Пустой циклический список содержит узел, который имеет неинициализи­ рованное поле данных. Этот узел называется заголовком (header) и первона­ чально указывает на самого себя. Роль заголовка — указывать на первый реальный узел в списке и, следовательно, на заголовок часто ссылаются как на узел-часовой (sentinel). В циклической модели связанного списка пустой список фактически содержит один узел, и указатель NULL никогда не исполь­ зуется. Мы приводим схему заголовка, используя угловые линии в качестве стоны узла.

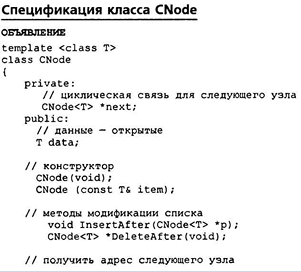


Заметьте, что для стандартного связанного списка и циклического связан­ ного списка тесты, определяющие, является ли список пустым, различны.



При добавлении узлов в список последний узел указывает на заголовочный узел. Мы можем представить циклический связанный список как браслет с заголовочным узлом, служащим застежкой. Заголовок связывает вместе реаль­ ные узлы в списке. 

В разделе 9.1 был описан класс Node и использовались его методы для создания связанных списков. В этом разделе мы объявляем класс Cnode, со­ здающий узлы для циклического списка. Этот класс предоставляет конструк­ тор по умолчанию, допускающий неинициализированное поле данных. Кон­ структор используется для создания заголовка.





Этот класс подобен классу Node в разделе 9.1. В действительности все члены этого класса имеют то же имя и те же функции. Детали открытых членов класса приводятся в следующем разделе, в котором описывается ре­ ализация класса. Класс CNode содержится в файле cnode.h.

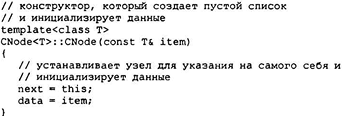
Реализация класса CNode

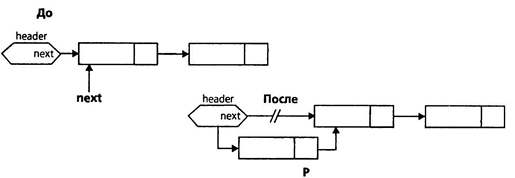
Конструкторы инициализируют узел его указанием на самого себя, поэтому каждый узел может служить в качестве заголовка для пустого списка. Ука­ затель на самого себя — это указатель this, и, следовательно, присваивание становится следующим:



Для конструктора по умолчанию поле data не инициализируется. Второй конструктор принимает параметр и использует его для инициализации поля data.

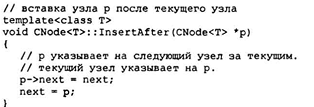
Никакой конструктор не требует параметра, определяющего начальное значение для поля next. Все необходимые изменения поля next выполняются с использованием методов InsertAfter и DeleteAfter.



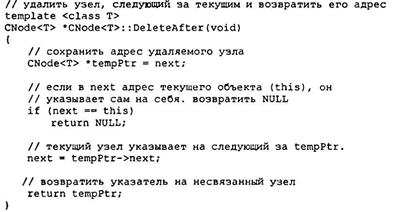


Операции класса CNode. Класс CNode предоставляет метод NextNode, ко­ торый используется для прохождения по списку. Подобно методу класса Node функция NextNode возвращает значение указателя next.

InsertAfter добавляет узел р непосредственно после текущего объекта. Для загрузки узла в голову списка не требуется никакого специального алгоритма, так как мы просто выполняем InsertAfter(header).



Удаление узла из списка выполняется методом DeleteAfter. DeleteAfter уда­ ляет узел, следующий непосредственно за текущим узлом, и затем возвращает указатель на удаленный узел. Если next равно this, то в списке нет никаких других узлов, и узел не должен удалять самого себя. В этом случае операция возвращает значение NULL.



АТД класса Циклический список

**Операция: Вставка нового узла**

* Условия на вход: Новый узел **newNode**, узел **node** после которого необходимо вставить новый узел.
* Данные на вход: Нет.
* Процесс:

1. Проверяем, что **newNode** не является **null**.
2. Присваиваем ссылке **newNode.Next** значение **node.Next**.
3. Присваиваем ссылке **node.Next** значение **newNode**.

* Выход: Нет.
* Постусловия: В циклический список вставлен новый узел **newNode** после узла **node**.

**Операция: Удаление следующего узла**

* Условия на вход: Узел **node**, удаляемый узел должен быть следующим после **node**.
* Данные на вход: Нет.
* Процесс:

1. Проверяем, что узел **node** имеет следующий узел.
2. Присваиваем ссылке **node.Next** значение **node.Next.Next**.

* Выход: Нет.
* Постусловия: Из циклического списка удален следующий узел, который был после узла **node**.

**Операция: Создание нового узла**

* Условия на вход: Значение **value** нового узла.
* Данные на вход: Нет.
* Процесс: Создается новый экземпляр класса **CyclicNode<T>** с заданным значением **value** и ссылкой на самого себя в качестве следующего узла.
* Выход: Новый экземпляр класса **CyclicNode<T>**.
* Постусловия: Создан новый узел с заданным значением.

**Операция: Создание циклического списка из массива значений**

* Условия на вход: Массив **values** значений элементов списка.
* Данные на вход: Нет.
* Процесс:

1. Если массив **values** пуст, то возвращаем **null**.
2. Создаем первый узел списка, используя первый элемент массива **values**.
3. Проходимся по оставшимся элементам массива **values** и вставляем новые узлы после последнего добавленного узла.
4. Возвращаем первый узел списка.

* Выход: Первый узел созданного циклического списка.
* Постусловия: Создан новый циклический список из элементов массива **values**.