**Двунаправленные списки**

Для объединения элементов в множество наиболее широко используют **связанный линейный список**. Как видно из названия, структура имеет вид «линии», состоящей из связанных элементов. Линейные списки более гибкие и универсальные чем другие структуры (к примеру деревья), поэтому их чаще используют для решения задач.

В структуре объекта списка обязательно присутствует специальное поле с ссылкой на **следующий** объект, которое содержит адрес следующего элемента. И как следствие для поиска элемента в списках используются отношения элементов. Чтобы обратиться к какому-либо элементу нужно пройтись по всем элементам списка – сначала к первому, потом ко второму, третьему…и так пока не найдется заданный элемент.

**Двунаправленный линейный список** имеет некоторое преимущество перед обычным «классическим» списком, это наличие у его объекта дополнительного поля. Оно содержит ссылку с адресом на **предыдущий** элемент. Такая модифицированная структура списка позволяет обращаться не только к следующему элементу, но и предыдущему, что дает возможность программисту полностью оперировать структурой, т.е организовывать вставку элемента в любое место списка.

Стоит отметить, что двунаправленные списки могут быть реализованы только с помощью механизма динамического распределения памяти (динамическая, адресная или связная реализация).

Работа с двунаправленными линейными списками дает некоторые преимущества, приведенных ниже:

* экономия оперативной памяти. Структура не приводит к лишним затратам, т.к не требует выделения непрерывной области памяти
* уменьшение временных ресурсов на выполнение операций добавления и удаления элементов.

Недостатки у структур с двунаправленной списочной реализации все же есть:

* ведет к некоторым появлениям ошибок при выполнении программы. Списки имеют более сложную программную реализацию, т.к реализация происходит на основе адресных переменных
* замедление работы программы при многократном динамическом выделении памяти.

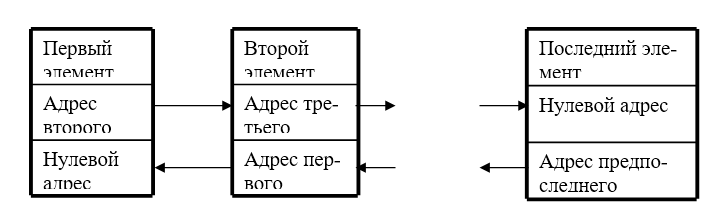
У двунаправленных списков есть некоторые разновидности реализации.

Различают списки **кольцевые и разомкнутые**. Они различаются наличием ссылок на первый элемент у последнего элемента и на последний элемент у первого.

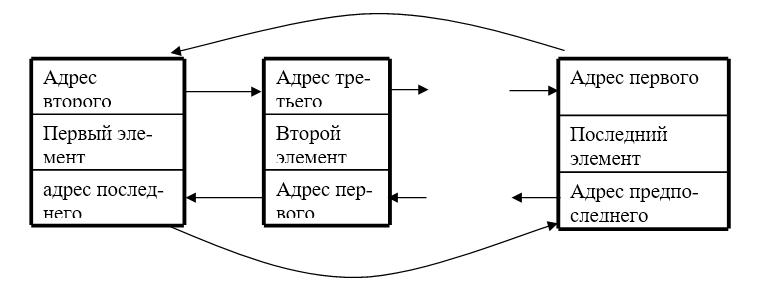
Также есть списки **с заголовочным элементом и без заголовка**. Из названия разновидности понятно, что разница между списками заключается в наличии заголовка у первого элемента.

Комбинация этих видов образуют четыре возможных реализации двунаправленных списков:

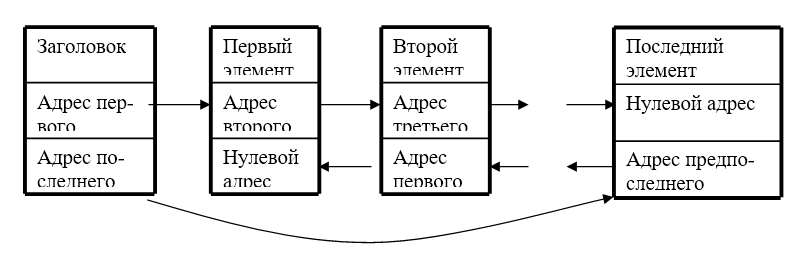
1. **Разомкнутый** список **без** **заголовочного элемента**: поле **next** у последнего элемента и поле **prev** у первого содержат пустые адреса



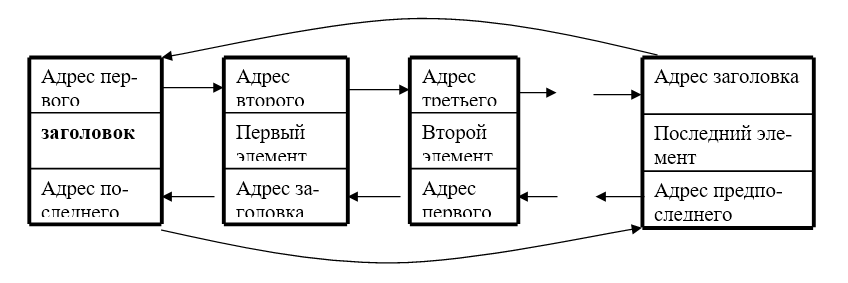
1. **Кольцевой (замкнутый)** список **без** **заголовка**: поле **next** у последнего элемента содержит адрес **первого** элемента, поле **prev** у первого – адрес **последнего** (нулевые адресные поля **НЕ** используются!)



1. **Разомкнутый** список **с** **заголовочным** **элементом** – поле **next** заголовка содержит адрес первого элемента, поле **prev** – адрес последнего, для хранения **неизменяемого** адреса заголовка необходима только одна указательная переменная



1. **Кольцевой список с заголовком:** комбинация второго и третьего вариантов



Наиболее часто встречающейся реализацией является кольцевой двунаправленный список с заголовком, поэтому приведем алгоритмы его основных операций.

**Инициализация пустого списка:**

1. Выделение памяти под заголовок (указатель pHead)

2. Занесение в адресные поля заголовка адреса самого заголовка:

pHead^.next := pHead; pHead^.prev := pHead;

Добавление первого элемента в пустой список:

1. Выделение памяти с использованием указателя pNew

2. Заполнение информационных полей

3. Заполнение обоих адресных полей с занесением в них адреса заголовка (pHead)

4. Изменение адресных полей заголовка с занесением в них адреса созданного элемента (pNew)

Проход по списку в обратном направлении

1. Объявление вспомогательного указателя pTemp и установка его в адрес последнего элемента списка:

pTemp := pHead^.prev;

2. Организация цикла по условию достижения первого элемента (внимание: нулевой адрес НЕ используется!)

while ( pTemp <> pHead) do < тело цикла >

3. В теле цикла обрабатываем текущий элемент и переходим к предыдущему:

pTemp := pTemp^.prev;

**Добавление нового элемента после заданного в непустой список:**

1. Поиск заданного элемента

2. Если элемент найден и адресуется указателем pTemp, то выделяем память для нового элемента, используя указатель pNew

3. Формируем поля нового элемента, в частности:

* в поле next заносится адрес следующего элемента (берется из поля next найденного элемента): pNew^.next := pTemp^.next;
* в поле prev заносится адрес предшествующего элемента, которым является найденный элемент: pNew^.prev := pTemp;

4. Изменяем адресное поле prev у элемента, который должен следовать за новым, на адрес нового элемента:

pNew^.next^.prev := pNew; // ниже представлено более подробно

5. Изменяем адресное поле next у найденного элемента на адрес нового элемента: pTemp^.next := pNew;

Комментарий к шагу 4. Следует изменить поле prev (ссылка на предыдущий элемент) у элемента, следующего за заданным - нужен адрес этого элемента. Адрес можно получить двумя способами:

* либо из поля next заданного элемента (оно пока еще не изменено) с помощью конструкции pTemp^.next
* либо из установленного на предыдущем шаге поля next у нового элемента (конструкция pNew^.next )

Тогда конструкции (pTemp^.next^) или (pNew^.next^) будут обозначать сам адресуемый элемент-запись, доступ к полям которого можно оформить стандартным образом как pTemp^.next^.prev или pNew^.next^.prev .

Также можноиспользовать промежуточный указатель на следующий элемент, например – с именем pNext. Это позволит заменить один оператор присваивания (шаг 4) двумя более простыми:

1. pNext := pNew^.next или pNext := pTemp ^.next; // получение адреса следующего эл-та

2. pNext^.prev := pNew;

**Алгоритм добавления**

1. Поиск заданного элемента

2. Если элемент найден и адресуется указателем pTemp, то выделяем память для нового элемента, используя указатель pNew

3. Формируем поля нового элемента, в частности:

* в поле next заносится адрес заданного элемента:

pNew^.next := pTemp;

* в поле prev заносится адрес предшествующего элемента (берется из поля prev найденного элемента)

pNew^.prev := pTemp^.prev;

4. Изменяем адресное поле prev у заданного элемента на адрес нового элемента: pTemp^.prev := pNew;

5. Изменяем адресное поле next у предшествующего элемента на адрес нового элемента: pNew^.prev^.next := pNew; (см. комментарий к предыдущей операции)

Удаление происходит путем изменения указателей у соседних элементов: правый указатель у левого соседа и левый указатель у его правого соседа.

**Алгоритм удаления заданного элемента**

1. Проверка на пустоту

2. Поиск удаляемого элемента

3. Если элемент найден и адресуется указателем pTemp, то:

* изменяем адресное поле prev у следующего за ним элемента на адрес элемента, предшествующего удаляемому (этот адрес берем из поля prev удаляемого элемента):

pTemp^.next^.prev:= pTemp^.prev;

* изменяем адресное поле next у элемента, предшествующего удаляемому на адрес элемента, следующего за удаляемым (этот адрес берем из поля next удаляемого элемента):

pTemp^.prev^.next := pTemp^.next;

4. Обрабатываем удаляемый элемент, используя указатель pTemp

Подведем итоги в виде краткой характеристики каждой реализации двунаправленных списков.

**Разомкнутый список без заголовочного элемента:**

* указатель pFirst определяет положение первого элемента
* нулевое значение указателя pFirst означает пустоту списка
* отсутствие у первого и последнего элементов указателей на левого соседа и правого соседа соответственно создает некие проблемы (требует проверки)
* операции добавления и удаление перед и после крайних элементов списка вызывают особые ситуации.

Замкнутый (кольцевой) список без заголовочного элемента:

* указатель pFirst определяет положение первого элемента, но можно использовать указатель на текущий элемент (если позволяет задача)
* нулевое значение указателя pFirst означает пустоту списка
* pFirst изменяет свое значение при добавлении и удалении элементов
* адресные поля не содержат нулевые значения
* любой элемент списка имеет оба соседей

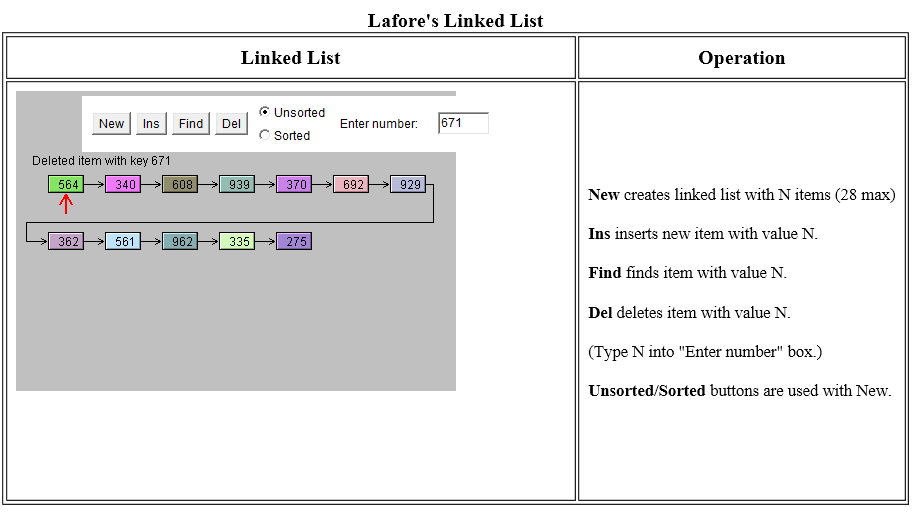
**Разомкнутый список с заголовком:**

* указатель pFirst содержит только адрес заголовка и никогда не изменяется
* у пустого списка есть только заголовочный элемент
* наличие заголовка помогает при операциях над первым элементом
* у последнего элемента отсутствует правый сосед

Более наглядная реализация двунаправленного списка представлена в [2].

С помощью программы applet, которая является разновидностью Java-программ, выполняющаяся в веб-обозревателе, представлена интерактивная демонстрация всевозможного оперирования списковыми структурами.

Пользователь прослеживает пошагово происходящие изменения со структурой. Представлены операции добавления, поиска и удаления заданных элементов.



**ТЗ**

В качестве среды разработки была выбрана платформа Delphi версии 7 фирмы Enterprise (Borland), так как она предоставляет наиболее широкие возможности для программирования приложений ОС Windows.

Delphi — это среда быстрой разработки, в которой в качестве языка программирования используется язык Delphi. Язык Delphi — императивный, структурированный, объектно-ориентированный язык программирования со строгой статической типизацией переменных, в основе которого лежит Object Pascal. Основная область использования — написание прикладного программного обеспечения.

Borland Delphi 7 Studio позволяет создавать самые различные программы: от простейших однооконных приложений до программ управления распределенными базами. В состав пакета включены разнообразные утилиты, обеспечивающие работу с базами данных, XML-документами, создание справочной системы, решение других задач. Отличительной особенностью седьмой версии является поддержка технологии .NET.

Основными преимуществами в сравнении с аналогичными средами разработки являются:

* быстрота разработки приложения;
* высокая производительность разработанного приложения;
* низкие требования разработанного приложения к ресурсам;

Borland Delphi представляет собой средство разработки приложений для Microsoft Windows. Delphi является мощным и простым в использовании инструментом для создания автономных программ.

Список литературы

1. Козин А.Н., Таренко Л.Б. - Структуры и алгоритмы обработки данных. Учебно-методическое пособие - 2017г.
2. Лафоре Роберт Структуры данных и алгоритмы Java (2013)
3. Вирт Н. - Алгоритмы и структуры данных (Классика программирования) - 2010г.