**[Программирование. Динамические списки Pascal-Паскаль](Программирование. Динамические списки Pascal-Паскальhttp://www.pascal.helpov.net › index › dynamic_lists_p)**

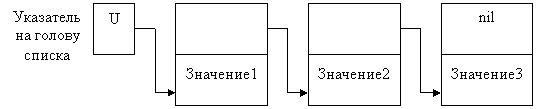
[http://www.pascal.helpov.net › index › dynamic\_lists\_p](Программирование. Динамические списки Pascal-Паскальhttp://www.pascal.helpov.net › index › dynamic_lists_p)

## Линейные списки (однонаправленные цепочки).

**Списком** называется структура данных, каждый элемент которой посредством указателя связывается со следующим элементом.

Каждый элемент связанного списка, во-первых, хранит какую-либо информацию, во-вторых, указывает на следующий за ним элемент. Так как элемент списка хранит разнотипные части (хранимая информация и указатель), то его естественно представить записью, в которой в одном поле располагается объект, а в другом – указатель на следующую запись такого же типа. Такая запись называется **звеном**, а структура из таких записей называется списком или цепочкой.

Лишь на самый первый элемент списка (голову) имеется отдельный указатель. Последний элемент списка никуда не указывает.



Пример описания узла списка:

**Type** ukazat= ^ S;  
   S= **record**  
      Inf: integer;  
      Next: ukazat;  
   **End**;

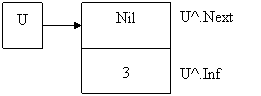
В Паскале существует основное правило: перед использованием какого-либо объекта он должен быть описан. Исключение сделано лишь для указателей, которые могут ссылаться на еще не объявленный тип.

Формирование списка

Чтобы список существовал, надо определить указатель на его начало.

**Пример описания списка**

**Type** ukazat= ^S;  
   S= **record**  
      Inf: integer;  
      Next: ukazat;  
   **End**;



Создадим первый элемент списка:

New (u); {выделяем место в памяти}  
u^. Next:= **nil**; {указатель пуст}  
u^. Inf:=3;

Продолжим формирование списка. Для этого нужно добавить элемент либо в конец списка, либо в голову.

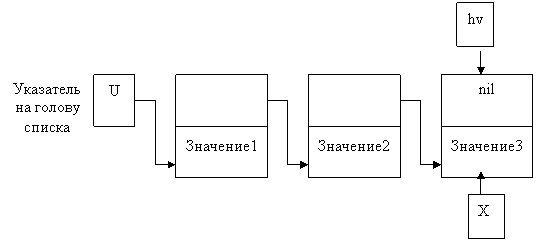
А) Добавим элемент в голову списка. Для этого необходимо выполнить последовательность действий:

* получить память для нового элемента;
* поместить туда информацию;
* присоединить элемент к голове списка.

**New**(x);  
Readln(x^.Inf); //В Delphi x^.Inf:=StrToInt(InputBox(‘Input’,’Input Inf’,’’));  
x^.Next:=u; // присоединить элемент к голове списка  
u:= x; //изменить указатель на голову списка

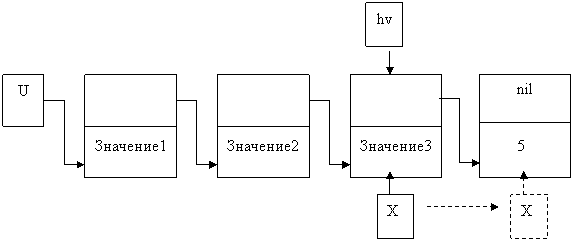
Б)Добавление элемента в конец списка. Для этого введем вспомогательную переменную, которая будет хранить адрес последнего элемента. Пусть это будет указатель с именем hv (хвост).

x:= hv;



New( x^. next); {выделяем память для следующего элемента}

x:=x^.next;  
x^.next:=nil;  
x^.inf:=5;  
hv:=x;

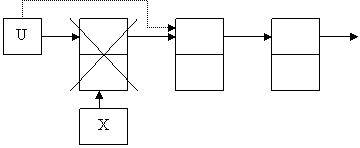


### Просмотр списка

**While** u<> **nil** **do //**цикл до тех пор, пока текущий указатель <> nil  
**Begin**  
 Writeln(u^.inf); //вывод информации на экран  
 u:=u^.next; //текущий указатель перемещается на следующий узел списка **end**;

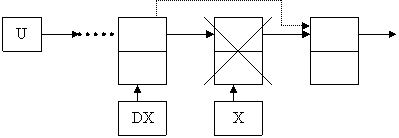
### Удаление элемента из списка

А)Удаление первого элемента. Для этого во вспомогательном указателе запомним первый элемент, указатель на голову списка переключим на следующий элемент списка и освободим область динамической памяти, на которую указывает вспомогательный указатель.



x:=u; //запомнить указатель на удаляемый узел  
u:=u^.next; //перевести указатель на голову списка на следующий после Х   
dispose(x);

Б) Удаление элемента из середины списка. Для этого нужно знать адреса удаляемого элемента и элемента, стоящего перед ним. Допустим, что digit – это значение удаляемого элемента.



x:=u; //текущий указатель переведён на голову списка  
**while** (x<> nil) **and** (x^.inf<>digit) **do**  
**begin**  
dx:=x; //запомнить указатель перед текущим узлом  
x:=x^.**next**; //перевести текущий указатель на следующий узел  
**end**;  
dx^.next:=x^.next; //предыдущий узел связать со следующим за х узлом  
dispose(x);

В)Удаление из конца списка. Для этого нужно найти предпоследний элемент.

x:=u; dx:=u;//инициализировать указатель на текущий узел и предшествующ.  
**while** x^.**next**<> nil **do //** поиск указателя нахвостовой элемент **begin**  
dx:=x; x:=x^.**next**;  
**end**;  
dx^.next:= nil; //теперь dx указывает на хвостовой элемент   
dispose(x);

**Прохождение списка**. Важно уметь перебирать элементы списка, выполняя над ними какую-либо операцию. Пусть необходимо найти сумму элементов списка.

summa:= 0;  
x:=u;  
**while** x<> nil **do**  
**begin**  
summa:=summa+x^.inf;  
x:=x^.**next**;  
**end**;

### Динамические объекты сложной структуры

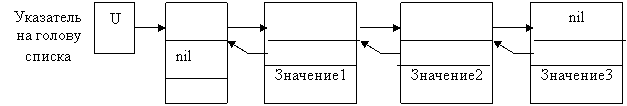
Использование однонаправленных списков при решении ряда задач может вызвать определенные трудности. Дело в том, что по однонаправленному списку можно двигаться только в одном направлении, от головы списка к последнему звену. Между тем нередко возникает необходимость произвести какую-либо операцию с элементом, предшествующим элементу с заданным свойством. Однако после нахождения элемента с данным свойством в однонаправленном списке у нас нет возможности получить удобный и быстрый способ доступа к предыдущему элементу.

Для устранения этого неудобства добавим в каждое звено списка еще одно поле, значением которого будет ссылка на предыдущее звено.

**Type**

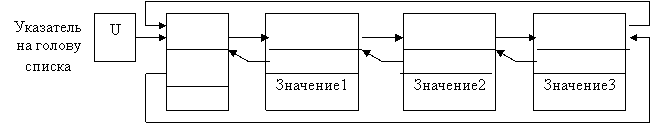
ukazat=^S;  
S= **record**  
 Inf:integer;  
 Next:ukazat;  
 Pred:ukazat;  
**End**;

Динамическая структура, состоящая из звеньев такого типа, называется **двунаправленным списком**, который схематично можно изобразить так:



Наличие в каждом звене двунаправленного списка ссылки как на следующее, так и на предыдущее звено позволяет от каждого звена двигаться по списку в любом направлении. По аналогии с однонаправленным списком здесь есть заглавное звено. В поле Pred этого звена фигурирует пустая ссылка nil, свидетельствующая, что у заглавного звена нет предыдущего (так же, как у последнего нет следующего).

В программировании двунаправленные списки часто обобщают следующим образом: в качестве значения поля Next последнего звена принимают ссылку на заглавное звено, а в качестве значения поля Pred заглавного звена – ссылку на последнее звено:



Как видно, здесь список замыкается в своеобразное «кольцо»: двигаясь по ссылкам, можно от последнего звена переходить к заглавному звену, а при движении в обратном направлении – от заглавного звена переходить к последнему. Списки подобного рода называют **кольцевыми списками**.

Существуют различные методы использования динамических списков:

* **Стек** – особый вид списка, обращение к которому идет только через указатель на первый элемент. Если в стек нужно добавить элемент, то он добавляется впереди первого элемента, при этом указатель на начало стека переключается на новый элемент. Алгоритм работы со стеком характеризуется правилом: «последним пришел – первым вышел».
* **Очередь**– это вид списка, имеющего два указателя на первый и последний элемент цепочки. Новые элементы записываются вслед за последним, а выборка элементов идет с первого. Этот алгоритм типа «первым пришел – первым вышел».
* Возможно организовать списки с произвольным доступом к элементам. В этом случае необходим дополнительный указатель на текущий элемент.

**[Связные списки Программирование, уроки и примеры.](Связные списки Программирование, уроки и примеры.https://programm.ws › page)**

[https://programm.ws › page](Связные списки Программирование, уроки и примеры.https://programm.ws › page)

# Связные списки

Связный список — набор элементов, каждый из которых структурно состоит из двух частей — содержательной и связующей. Содержательная часть представляет собой собственно данные или указатель на область памяти, где эти данные  
содержатся. Связующая часть предназначена для связи элементов в списке и определяет очередность их следования. Благодаря связующей части в каждом из элементов становятся возможными «путешествия» по списку путем последовательных переходов от одного элемента к другому.  
В отличие от последовательного списка связный список относится к динамическим структурам данных. Для них характерны непредсказуемость в числе элементов, которое может быть нулевым, а может и ограничиваться объемом доступной памяти. Другая характеристика связного списка — необязательная физическая смежность элементов в памяти.  
Относительно связующей части различают следующие типы связных списков.

* Односвязный список — список, начало которого определяется указателем начала, а каждый элемент списка содержит указатель на следующий элемент. Последний элемент должен содержать признак конца списка или указатель на первый элемент списка. В последнем случае мы имеем одно-связный кольцевой список. Преимущество последнего состоит в том, что просмотр всех элементов списка можно начинать с любого элемента, а не только с головы. Более того, в случае кольцевого односвязного списка указателем его начала может быть любой элемент списка. Этот вид списка всегда линейный, так как однозначно задает порядок своего просмотра.
* Двусвязный список — список, связующая часть которого состоит из двух указателей — на предыдущий и последующий элементы. Начало и конец списка определяются отдельными указателями. Последние элементы дву-связного списка в каждом из направлений просмотра содержат признаки конца. Если замкнуть эти указатели друг на друга, то мы получим кольцевой двусвязный список. В этом случае потребность в указателе конца списка отпадает. Этот вид списка может быть как линейным (иметь одинаковый порядок просмотра в обоих направлениях), так и нелинейным (второй указатель каждого элемента списка задает порядок просмотра, который не является обратным порядку, устанавливаемому первым указателем).
* Многосвязный список — список, связующая часть элементов которого в общем случае содержит произвольное количество указателей. В этом виде списка каждый элемент входит в такое количество односвязных списков, сколько он имеет в себе полей связи.

В общем случае для связанных списков определены следующие операции:

* создание связного списка;
* включение элемента в связный список, в том числе и после (перед) определенным элементом;
* доступ к определенному элементу связного списка (поиск в списке);
* исключение элемента из связного списка;
* упорядочение (перестройка) связного списка;
* очистка связного списка;
* проверка объема связного списка (числа элементов в связном списке);
* объединение нескольких списков в один;
* разбиение одного списка на несколько;
* копирование списка;
* удаление связного списка.

Связные списки очень важны для представления различных сетевых структур, в частности деревьев, что и будет рассмотрено нами чуть ниже. Пока же рассмотрим работу с некоторыми из обозначенных нами типов связных списков на практических примерах.