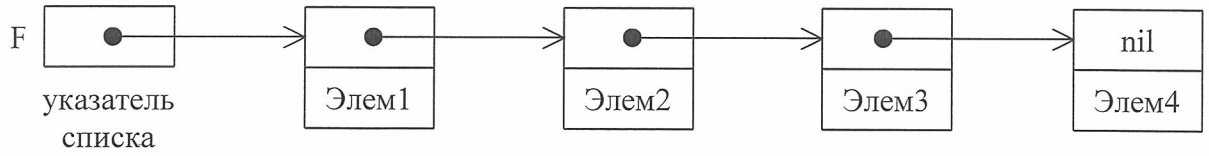
Программирование основных операций над списками

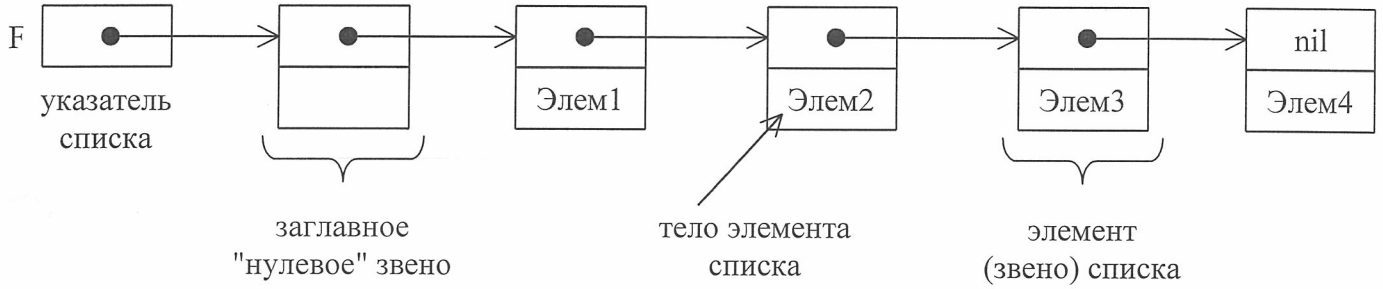
Прямой цепной список с однотипными элементами

Список без заглавного звена:



F — указатель списка (ссылка на первый элемент списка).

Список с заглавным звеном:



Заглавное звено — вспомогательное. Оно вводится для того, чтобы операции корректировки выполнялись одинаково для всех элементов списка. Иначе операция обработки первого элемента отличалась бы от операций над другими элементами, что усложнило бы программы обработки.

Такой список можно определить с помощью следующих описаний типов:

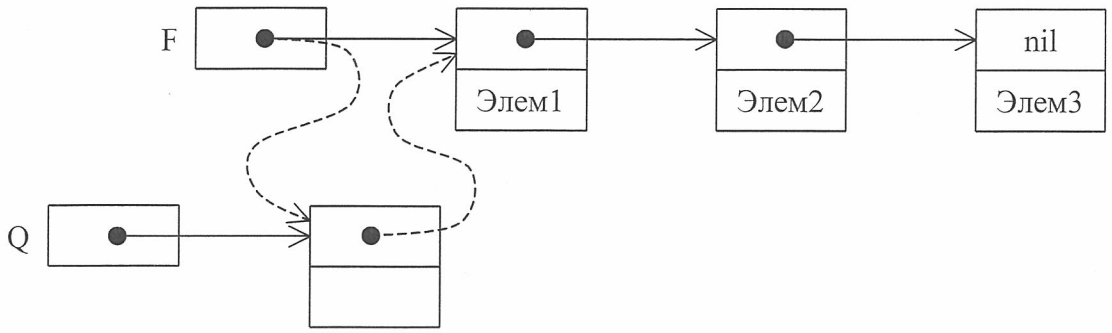
```
type
    TE = ...;              {тип тела элемента списка}
    LINK = ^TEL;           {тип: ссылка}
    TEL = record           {тип: элемент (звено) списка}
      NEXT : LINK;         {ссылка на следующий элемент}
      EL   : TE;           {тело элемента}
    end;
var F : LINK;              {указатель списка}
```

Рассмотрим простейшие операции.

Самое простое действие, которое можно выполнить с таким списком — это вставить в его начало некоторый элемент.

Прежде всего этот элемент типа TEL размещается в памяти, ссылка на него присваивается вспомогательной ссылочной переменной Q. После этого ссылкам присваиваются новые значения:

```
new(Q);
Q^.NEXT := F;
F:=Q;
```



Операция включения элемента в начало списка определяет, как можно построить такой список.

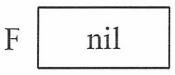
Формирование прямого цепного списка

Процесс формирования списка заключается в следующем: начиная с пустого списка, последовательно добавляются элементы в его начало.

Пусть нужно сформировать список из n элементов. Пусть для определённости TE=integer.

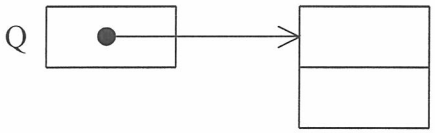
Для этого выполняются следующие шаги (изобразим графически):

1) Формируется пустой список:



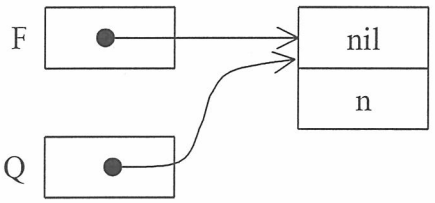

```
F := nil;
```

2) Размещается в памяти элемент списка:



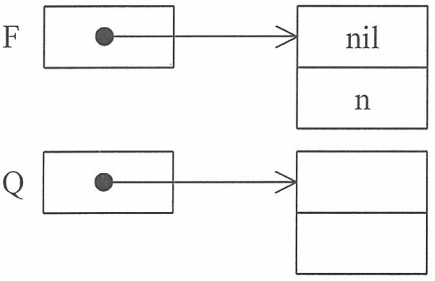

```
new(Q);
```

3) Добавляется элемент списка в его начало:



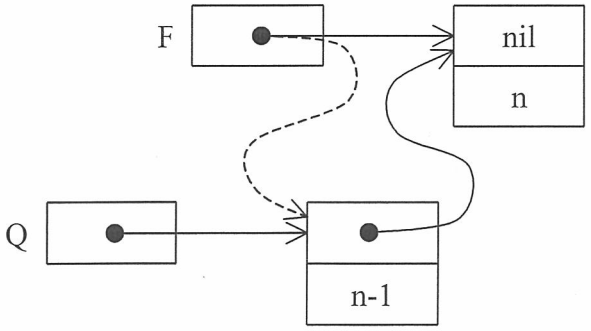

```
Q^.NEXT := F;
F := Q;
Q^.EL := n;
```

4) Размещается в памяти следующий элемент списка:




```
new(Q);
```

5) Добавляется новый элемент списка в его начало:



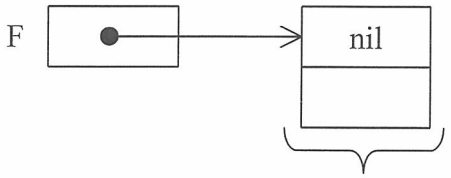

```
Q^.NEXT := F;
F := Q;
Q^.EL := n-1;
```

И так далее.

Таким образом, для формирования списка из n элементов можно использовать цикл:

```
F := nil;
while (n > 0) do begin
  new(Q);  Q^.NEXT := F;  F := Q;  Q^.EL := n;  n := n-1
end;
```

Далее будем рассматривать все списки с заглавным звеном. У пустого списка поле NEXT содержит ссылку nil.



— пустой список.

заглавное
"нулевое" звено

Поиск заданного элемента в списке

Алгоритм выполнения этой операции зададим в виде описания логической процедуры-функции, которая в качестве побочного эффекта определяет ссылку на первое вхождение заданного элемента в указанный список (ссылку на соответствующее звено цепочки).

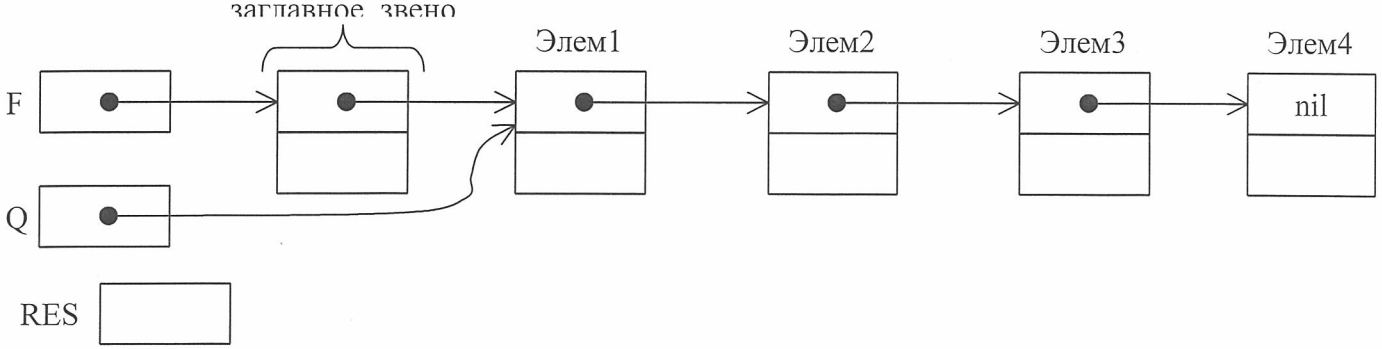
Алгоритм поиска следующий: будем просматривать — с использованием ссылок — последовательные звенья цепочки и сравнивать значение поля EL каждого звена с заданным элементом.

Этот процесс заканчивается в двух случаях:

1) очередное звено цепочки содержит заданный элемент; в этом случае функция должна принять значение true, а в качестве побочного эффекта выдаётся ссылка на это звено;
2) цепочка будет исчерпана (в поле NEXT очередного обработанного звена окажется ссылка nil); в этом случае функция должна принять значение false, а в качестве побочного эффекта будет выдавать значение nil.

Описание такой логической функции может иметь вид:

```
function Poisk(P : LINK; ELEM : TE; var RES : LINK) : boolean;
var Q : LINK;
begin
  Poisk := false;
  RES := nil;  Q := P^.NEXT;
  while ((Q <> nil) and (RES = nil)) do begin
    if (Q^.EL = ELEM) then begin
      Poisk := true;
      RES := Q
    end;
    Q := Q^.NEXT
  end
end;
```



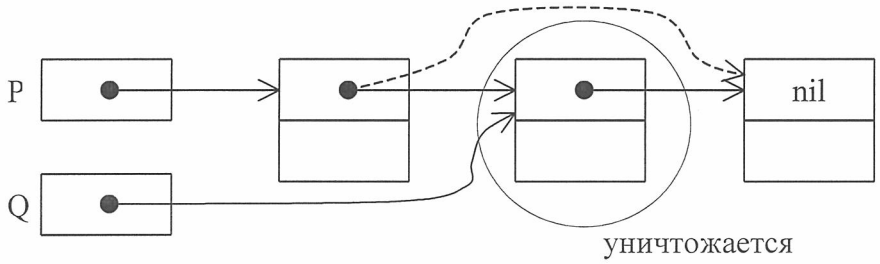
Замечание. За счёт повторных обращений к этой процедуре можно найти все вхождения в список заданного элемента — для нахождения его очередного вхождения достаточно снова обратиться к функции, задав в качестве первого фактического параметра ссылку на звено, которое содержало предыдущее вхождение заданного элемента. Этот процесс следует завершить, когда при очередном обращении к функции она возвратит значение, равное false.

Удаление из списка заданного элемента

Исключаемый элемент будем задавать с помощью ссылки на то звено, за которым следует этот элемент. (На практике наиболее типичен случай, когда надо исключить элемент, следующий за элементом, найденным с помощью рассмотренной ранее функции поиска.)

В описании процедуры удаления нужно предусмотреть контроль корректности задания фактического параметра: если фактический параметр указывает на последнее звено списка, то список остаётся без изменения. Кроме того, в описании процедуры необходимо предусмотреть уничтожение исключённого из цепочки звена с помощью процедуры dispose.

```
procedure Udal(P : LINK);
var Q : LINK;
begin
  if (P^.NEXT <> nil) then begin
    Q := P^.NEXT;
    P^.NEXT := P^.NEXT^.NEXT;
    dispose(Q);
  end
end;
```



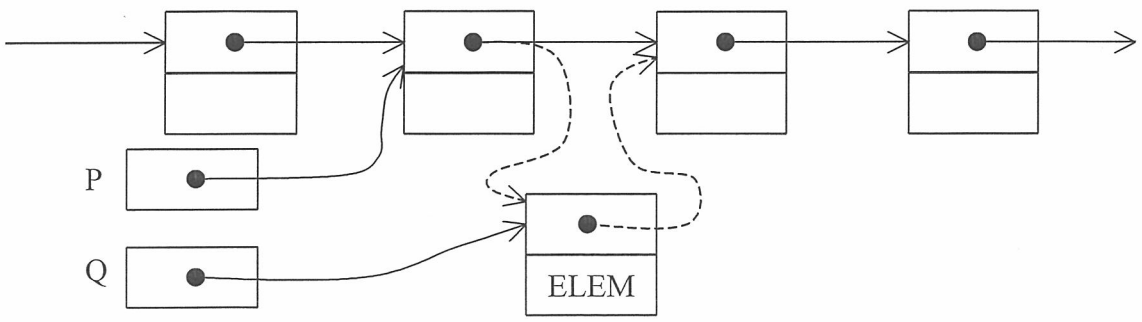
Вставка в список заданного элемента

Для вставки в список нового звена после заданного надо сделать следующие действия:

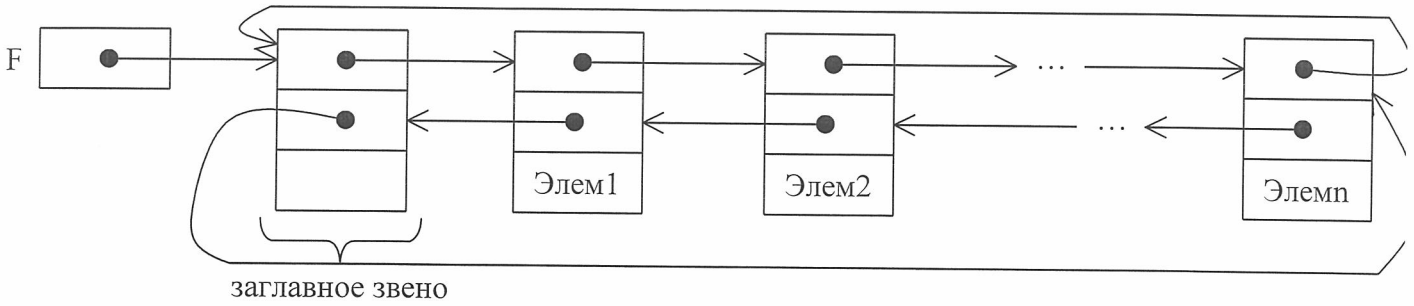
1) создать новый динамический объект, которым будет представлено вставляемое звено; этот объект должен иметь тип TEL, то есть запись;
2) в поле EL этой записи занести заданное значение тела элемента;
3) в поле NEXT этой записи занести ссылку, взятую из поля NEXT того звена, за которым должно следовать вставляемое звено;
4) в поле NEXT того звена, за которым должно следовать вставляемое звено, занести ссылку на это вставляемое звено.

Описание процедуры вставки:

```
procedure Vstavka(P : LINK; ELEM : TE);
var Q : LINK;
begin
  new(Q);
  Q^.EL := ELEM;
  Q^.NEXY := P^.NEXT;
  P^.NEXT := Q
end;
```



Двунаправленный кольцевой список



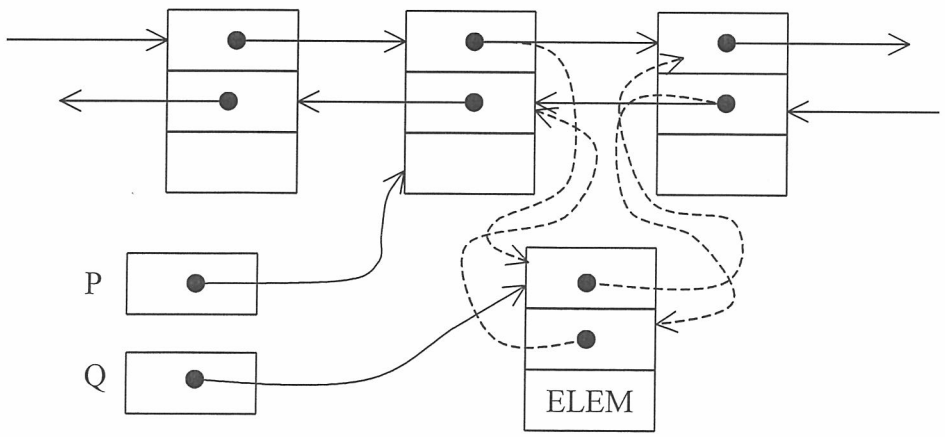
Такой список можно определить с помощью следующих описаний типов:

```
type
    TE = ...;              {тип тела элемента списка}
    LINK = ^TEL;           {тип: ссылка}
    TEL = record           {тип: элемент (звено) списка}
      NEXT : LINK;         {ссылка на следующий элемент}
      PRED : LINK;         {ссылка на предыдущий элемент}
      EL   : TE;           {тело элемента}
    end;
var F : LINK;              {указатель списка}
```

Вставка элемента в двунаправленный кольцевой список

Опишем процедуру с именем VSpisok, которая имеет два формальных параметра: один из них (ELEM) представляет вставляемый элемент (значение типа TE), а другой (P) — ссылку на звено, после которого необходимо вставить новый элемент.

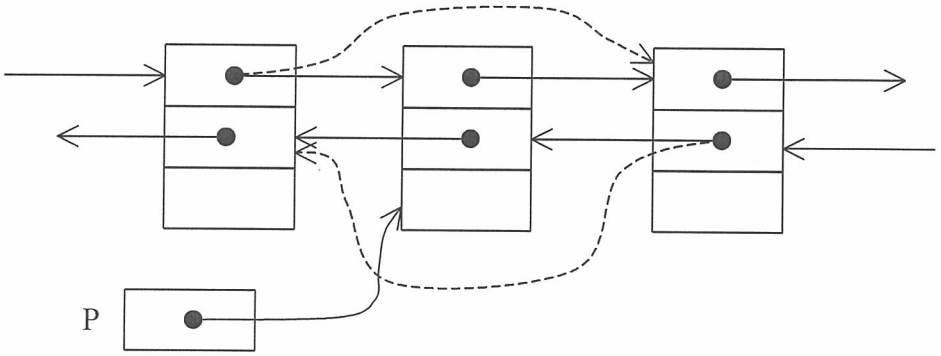
```
procedure VSpisok(ELEM : TE; P : LINK);
var Q : LINK;
begin
  {создание нового звена}
  new(Q);
  {формирование значений полей нового звена}
  Q^.EL := ELEM;
  Q^.NEXT := P^.NEXT;
  Q^.PRED := P^.NEXT^.PRED;
  {корректировка ссылок у соседних звеньев}
  P^.NEXT^.PRED := Q;
  P^.NEXT := Q
end;
```



Удаление элемента из двунаправленного кольцевого списка

Процедура удаления имеет единственный параметр, представляющий ссылку на удаляемое звено.

```
procedure IzSpiska(P : LINK);
begin
  {изменение поля PRED у следующего звена}
  P^.NEXT^.PRED := P^.PRED;
  {изменение поля NEXT у предыдущего звена}
  P^.PRED^.NEXT := P^.NEXT;
  {уничтожение удалённого элемента}
  dispose(P)
end;
```



Поиск элемента в двунаправленном кольцевом списке

Напишем один из возможных вариантов этой процедуры.

Опишем её в виде логической функции, которая в качестве побочного эффекта вырабатывает ссылку на звено списка, в котором находится искомый элемент.

Логическая функция использует три параметра:

1) формальный параметр F является указателем на список (ссылка на заглавное звено);
2) параметр ELEM представляет значение искомого элемента;
3) параметр RES представляет ссылочную переменную, которой в качестве побочного эффекта функции присваивается ссылка на первое по порядку звено, содержащее заданный элемент.

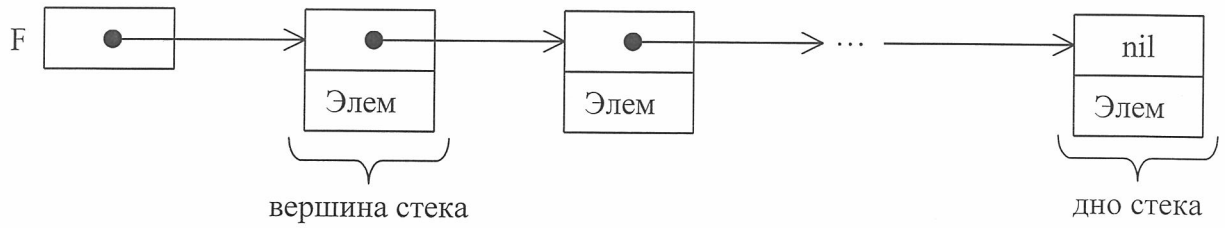
```
function Poisk(F : LINK; ELEM : TE; var RES : LINK) : boolean;
var P,Q : LINK;
    B : boolean;
begin
  B := false; P := F; RES := nil; Q := P^.NEXT;
  while ((P <> Q) and (not B)) do begin
    if (Q^.EL = ELEM) then begin
      B := true;
      RES := Q
    end;
    Q := Q^.NEXT
  end;
  Poisk := B
end;
```

Операции над стеком

Представим стек в виде динамической цепочки звеньев.

Вершиной стека является первое звено цепочки. (Поскольку в стеке доступна только его вершина, то заглавное звено в цепочке становится излишним.)

В этом случае значением указателя стека является ссылка на вершину стека.
Дно стека имеет ссылку nil.
Схематично стек можно представить следующим образом:



Стек задается следующим описанием:

```
type
    TE = …; {тип элементов стека}
LINK = ^TEL; {тип: ссылка}
    TEL = record {тип: звено стека}
      NEXT : LINK; {ссылка на следующий элемент}
      EL : TE {элемент стека}
    end;
var F : LINK; {указатель стека}
```

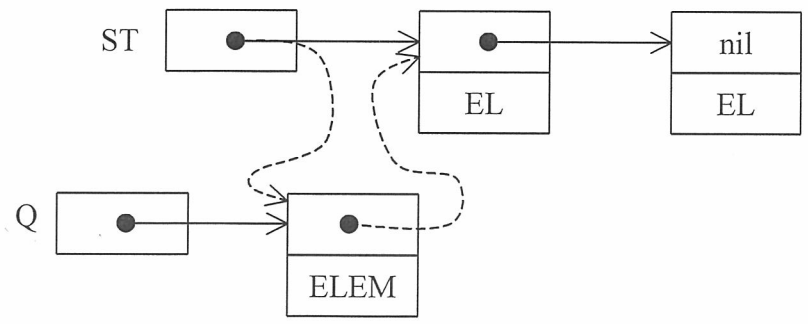
Если стек пуст, то значением указателя F является ссылка nil.

К началу использования стека его необходимо сделать пустым, что обеспечивается оператором присваивания F:=nil;

Занесение элементов в стек

Так как при решении задачи может использоваться несколько различных стеков, то процедура занесения в стек должна иметь два параметра: один из них задает нужный стек, а другой - заносимое в него значение.

```
procedure VStek(var ST : LINK; ELEM : TE);
var Q : LINK;
begin
  {создание нового звена}
  new(Q);
  Q^.EL := ELEM;
  {созданное звено сделать вершиной стека}
  Q^.NEXT := ST; ST := Q;
end;
```



Выбор элемента из стека

При выполнении этой операции элемент, находящийся в вершине стека, должен быть присвоен в качестве значения некоторой переменной, а звено, в котором был представлен этот элемент, должно быть исключено из стека.

В описании процедуры необходимо предусмотреть контроль корректности использования стека: если стек оказался пустым должно выдаваться диагностическое сообщение: "ПОПЫТКА ВЫБОРА ИЗ ПУСТОГО СТЕКА".

```
procedure IzSteka(var ST: LINK; var A:TE);
var Q: LINK;
begin
  {проверка, не пуст ли стек}
  if (ST = nil) then begin
    writeln;
    writeln('ПОПЫТКА ВЫБОРА ИЗ ПУСТОГО СТЕКА')
  end
  else begin
    {выбор элемента из вершины}
    A := ST^.EL;
    {заполнение ссылки на старую вершину}
    Q := ST;
    {исключение и уничтожение использованного звена}
    ST := ST^.NEXT; dispose(Q)
  end
end;
```