Простые структуры данных

Определение. Тип данных — это множество значений и набор операций с ними.

Операции связаны с типами, а не наоборот. При выполнении операции
необходимо обеспечить, чтобы ее операнды и результат отвечали определенному типу. Пренебрежение этим правилом — распространенная ошибка программирования.

1 Базовых типы данных: целые числа (integer); числа с плавающей точкой (real, float); символы (char).

2 Фундаментальность массива, как структуры данных, заключается в их прямом соответствии системам памяти почти на всех компьютерах. Для извлечения содержимого слова из памяти машинный язык требует указания адреса. Таким образом, всю память компьютера можно рассматривать как массив, где адреса памяти соответствуют индексам. Большинство процессоров машинного языка транслируют программы, использующие массивы, в эффективные программы на машинном языке, в которых осуществляется прямой доступ к памяти. Можно с уверенностью сказать, что доступ к массиву с помощью выражения a[i] требует небольшого количества машинных команд.

Пример. «Решето Эратосфена»

Цель программы заключается в присвоении элементам `a[i]` значения 1, если i простое число, и значения 0 в противном случае. Сначала значение 1 присваивается всем элементам массива. Затем присваивается значение 0 элементам, индексы которых не являются простыми числами (представляют собой произведения известных простых чисел). Если после этого некоторый элемент `a[i]` сохраняет значение 1, его индекс является простым числом. Напомним, что простым числом называется число, не имеющее других делителей, кроме единицы и самого себя.

```
Procedure Solve; {N, массив a – глобальные
переменные}
  Var i,j:Integer;
  Begin
   For i:=2 To N Do a[i]:=1;
   For i:=2 To N Do
     If (a[i]=1) Then Begin
      j:=i;
      While (j*i<N)Do Begin
       a[i*j]:=0;
       j:=j+1;
      End;
    End;
   For i:=2 To N Do
    If (a[i]=1) Then Write(i,'   ');
 End;
```

3 "Строка" (string) обозначает массив символов переменной длины, определяемый начальной точкой и символом завершения строки. Ценность строк в качестве низкоуровневых структур данных обусловлена двумя главными причинами. Во-первых, многие вычислительные приложения предусматривают обработку текстовых данных, которые могут представляться непосредственно строками. Во-вторых, многие вычислительные системы предоставляют прямой и эффективный доступ к байтам памяти, которые в точности соответствуют символам строк. Таким образом, в подавляющем большинстве случаев абстракция строк связывает потребности приложения с возможностями компьютера.

Справка по ссылочному типу данных (указатели)

Для объявления указателей (переменных ссылочного типа) используется специальный символ «^», после которого указывается тип динамической (базовой) переменной.

```
Type <имя_типа>=^ <базовый тип>;
Var <имя_переменной>: <имя_типа>; или
<имя_переменной>: ^ <базовый тип>;
```

Пример.

```
Type ss = ^Integer;
Var x, y: ss;{Указатели на переменные целого типа.}
a:  ^Real;{Указатель на переменную вещественного
типа.}
```

Зарезервированное слово `nil` обозначает константу ссылочного типа, которая ни на что не указывает. Выделение оперативной памяти (в «куче») для динамической переменной базового типа осуществляется с помощью оператора `New(x)`, где `x` определен как соответствующий указатель. Обращение к динамическим переменным выполняется по правилу: `<имя_переменной>^.` Например, `x^:=15` – в область памяти (два байта), адрес которой является значением указателя *x*, записывается 15. Оператор `Dispose(x)` освобождает память, занятую динамической переменной.

При этом значение указателя x становится неопределенным. Описанные действия показаны на рис. 3.1.

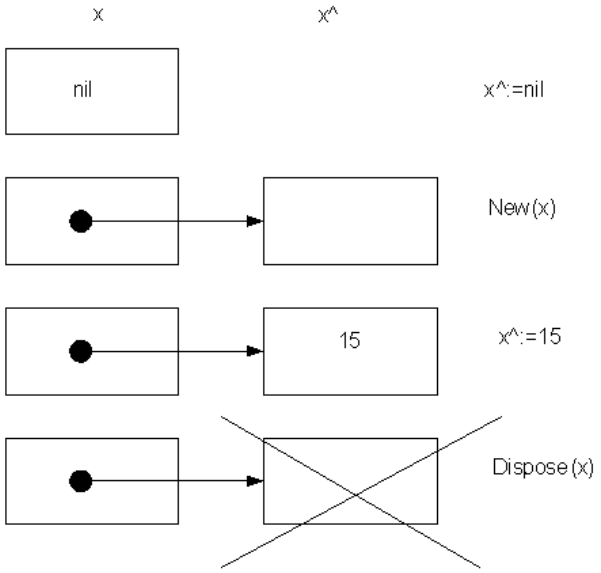

Рис. 3.1. Основные операторы для работы с указателями

4 Связные списки. Базовая структура данных, в которой каждый элемент содержит информацию, необходимую для получения следующего элемента. Основное преимущество связных списков перед массивами заключается в возможности эффективного изменения расположения элементов. За эту гибкость приходится жертвовать скоростью доступа к произвольному элементу списка, поскольку единственный способ получения элемента состоит в отслеживании связей от начала списка.

Определение. Связный список – это набор элементов, связи между которыми устанавливается с помощью указателей.

Обычно под связным списком подразумевается реализация последовательного расположения набора элементов. Начиная с некоторого элемента, мы считаем его первым элементом последовательности. Затем прослеживается его ссылка на другой элемент, который дает нам второй элемент последовательности и т.д. Поскольку список может быть циклическим, последовательность иногда представляется бесконечной. Чаще всего приходится иметь дело со списками, соответствующими простому последовательному расположению элементов, принимая одно из следующих соглашений для ссылки из последнего элемента:

- Это пустая (`nil`) ссылка, не указывающая на какой-либо элемент.
- Ссылка указывает на фиктивный элемент (`dummy node`).
- Ссылка указывает на первый элемент, что делает список циклическим.

В каждом случае отслеживание ссылок от первого элемента до последнего формирует последовательное расположение элементов. Массивы также задают последовательное расположение элементов, но оно реализуется косвенно, за счет позиции в массиве. (Массивы также поддерживают произвольный доступ по индексу, что невозможно для списков.)

Пример. На рис. 3.2 приведен пример линейного списка из четырех элементов, содержащего в поле данных целые числа 3, 5, 1, 9.

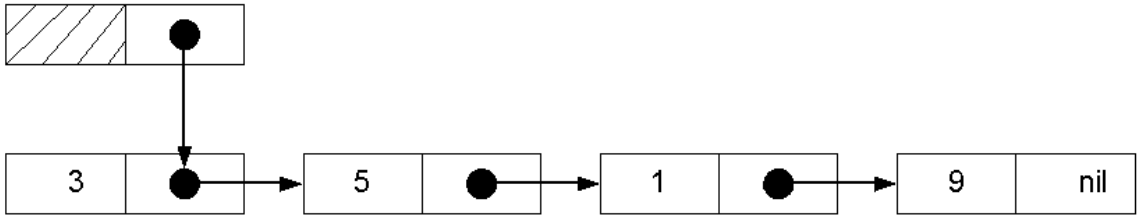

Рис. 3.2. Пример линейного списка

Описание элемента списка имеет вид:

```
Type pt=^elem;{Указатель на элемент списка.}
     elem=Record
      data:Integer;{Поле данных (ключ).}
      next:pt;{Указатель на следующий элемент
    списка.}
     End;
Var first:pt;{Указатель на первый элемент списка.}
```

Основные операции с элементами списка:

- вывод элементов списка;
- вставка элемента в список;
- удаление элемента из списка.

Вывод элементов списка, если он создан, – очевидная процедура. Элементы последовательно просматриваются друг за другом, начиная с первого. Вызов процедуры – `Print(first)`, где `first` – указатель на первый элемент списка.

```
Procedure Print(t:pt);
 Begin
  While t<>nil Do Begin
   Write(t^.data,' ');
   t:=t^.next;
  End;
 End;
```

Ее рекурсивная реализация:

```
Procedure Print(t:pt);
 Begin
  If t<>nil Then Begin
   Write(t^.data,' ');
   Print(t^.next);
  End;
 End;
```

Вставка элемента в список. Пусть дано значение указателя `p` на элемент списка, после которого требуется вставить элемент `y` (рис. 3.3). Список до вставки элемента показан на рис. 3.3а, после вставки – рис. 3.3б. Измененные значения указателей показаны пунктирной линией. Последовательность действий обозначена цифрами 1, 2, 3 и 4 и реализована в процедуре `Insert`.

```
Procedure Insert(p:pt; y:Integer);
  Var t:pt;
 Begin
  t:=p^.next; {1}
  New(p^.next); {2}
  p^.next^.data:=y; {3}
  p^.next^.next:=t; {4}
 End;
```

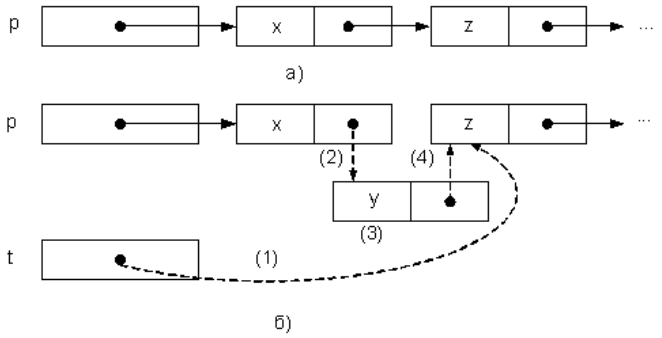

Рис. 3.3. Вставка элемента в список: а) до операции; б) после операции

Логически возможны три случая вставки – в начало, в середину и в конец списка. Для всех ли случаев применима процедура Insert? Очевидно, что во втором и третьем случаях «да». В третьем случае значением указателя является адрес последнего элемента списка, он имеет ссылку nil, и это значение переписывается в поле next вставляемого элемента. В первом случае процедура не работает. Требуется изменить значение глобальной переменной first, а мы не передаем в вызывающую логику изменное значение p. Если изменить параметры процедуры на Procedure Insert(Var p:pt; y:Integer), то проблема решена, но остается вопрос вставки первого элемента в пустой список (p=nil). В этом случае действие p^.next «выбрасывает» нас в неопределенную область памяти. В процедуре Ins_List решается эта проблема.

```
Procedure Ins_List(Var p:pt;y:Integer);
 Begin
  If p=nil Then Begin
        New(p);
        p^.data:=y;
        p^.next:=nil;
     End
    Else Insert(p,y);
 End;
```

Удаление элемента из списка. Действия при удалении элемента из списка сводятся к его поиску, а затем к переадресации от предшествующего удаляемому элементу, к элементу следующим за удаляемым (рис. 3.4а). Единственная сложность заключается в том, чтобы предусмотреть случай удаления первого элемента списка – изменяется значение переменной `first` (рис. 3.4б).

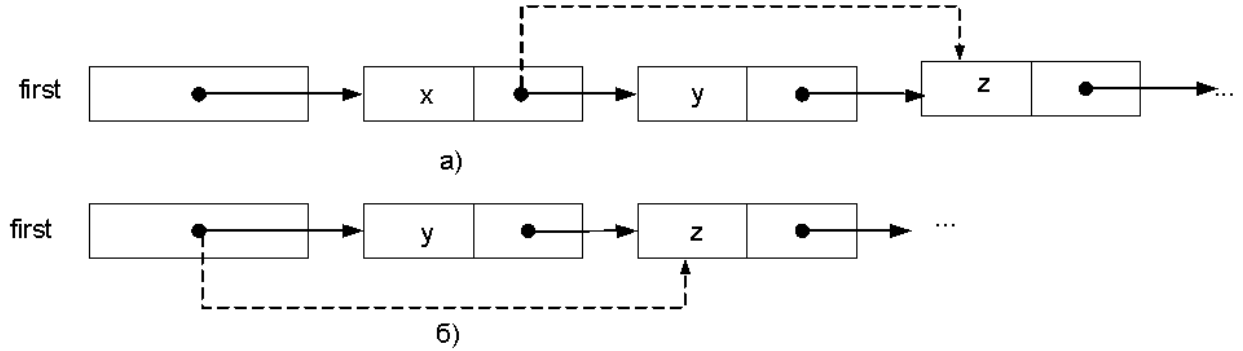

Рис. 3.4. Удаление элемента из списка

В процедуре Del_List из списка удаляются все элементы, равные y.

```
Procedure Del_List(Var first:pt; y:Integer);
 Var t,x,dx:pt;
 Begin
  t:=first;
  While t<>nil Do
   If t^.data=y Then
    If t=first Then Begin
      x:=first;first:=first^.next;
      Dispose(x);
      t:=first;
      End
     Else Begin
       x:=t; t:=t^.next; dx^.next:=t;
       Dispose(x);
      End
    Else Begin dx:=t;:=t^.next;End
 End;
```

Пример. Задача Иосифа.

Предположим, N человек решило выбрать главаря. Для этого они встали в круг и стали удалять каждого M-го человека в определенном направлении отсчета, смыкая ряды после каждого удаления. Задача состоит в определении, кто останется последним.

Номер выбираемого главаря является функцией от N и M, называемой функцией Иосифа. В более общем случае требуется выяснить порядок удаления людей. В примере, показанном на рис. 3.5, если N = 9 и M = 5, люди удаляются в порядке 5, 1, 7, 4, 3, 6, 9, 2, а 8-ой номер становится избранным главарем.

```
Procedure Solve; {N, M –
глобальные переменные}
 Var first,q,t:pt;{Тип данных pt
определен в вызывающей логике.}
     i:Integer;
 Begin
  New(first);
  first^.data:=N;
  q:=first;
  For i:=N-1 To 1 Do Begin
    New(t);
    t^.data:=i;
    t^.next:=first;
    first:=t;
   End;
  q^.next:=first;
  While (x<>x^.next)
   For i:=1 To M Do
x^.next:=x^.next^.next;
  Write(x^.data);
 End;
```

Реализация линейного списка с использованием массивов

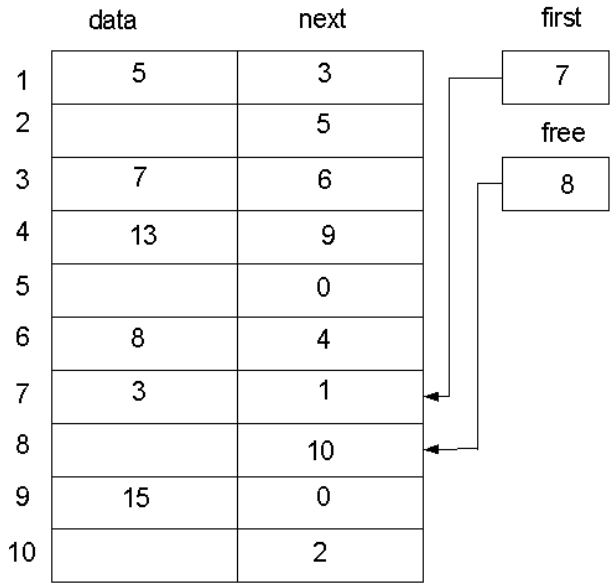

Рис. 3.6. Пример представления списка с использованием массивов

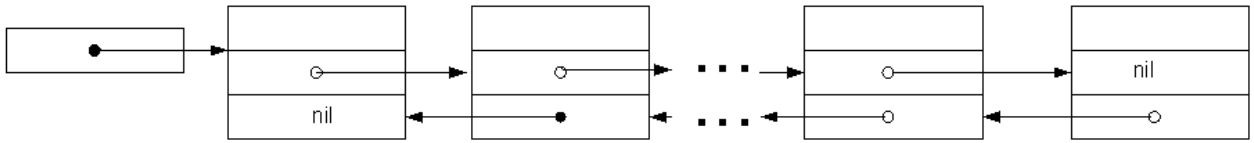

Рис. 3.7. Двусвязный список

Описание элемента списка в этом случае имеет вид:

```
Type pt=^elem;{Указатель на элемент
списка.}
     elem=Record
     data:Integer;{Поле данных (ключ).}
     next, prev:pt;{Указатели на
   следующий и предшествующий элементы
   списка.}
    End;
Var head:pt;{Указатель на первый элемент
списка.}
```

Лабораторно-практическая работа №3

1. Можно ли с помощью рассмотренного алгоритма («решета Эратосфена») найти все простые числа, меньшие 1000000000?

2. С помощью решета Эратосфена определить количество простых чисел, меньших N, для N= 10^3, 10^4, 10^5 и 10^6.

3. С помощью решета Эратосфена построить график зависимости N от количества простых чисел, меньших N, для значений N от 1 до 1000.

4. Имитация подбрасываний монеты. Если подбросить монету N раз, ожидается выпадение N/2 решек, но это число может быть любым в диапазоне от 0 до N. Необходимо выполнить эксперимент M раз (M и N задаются) и затем вывести гистограмму результатов эксперимента.

На рис. 3.9 приведен результат. Каждые 10 выпаданий обозначаются одной звездочкой.

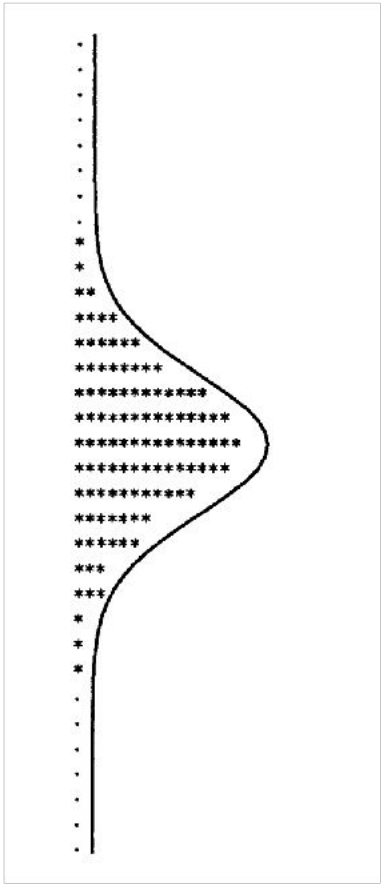

Рис. 3.9

Рис. 3.9 демонстрирует результат выполнения программы при N=32 и M= 1000. Имитируется 1000 экспериментов подбрасывания монеты по 32 раза. Количество выпаданий решки аппроксимируется нормальной функцией распределения, график которой показан поверх данных (его можно не приводить).

5. Исследуйте задачу Иосифа для различных значений n. Заполните табл. 3.1 для оставшихся значений n от 1 до 64 (t – номер оставшегося ребенка).

Таблица 3.1

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	…
t	1	1	3	1	3	5	7	1	3	5	7	9	11	13	15	1	3	…

Экспериментальным путем установите закономерность:

- t(1)=1 при n=1,
- t(2*n)=2*t(n)-1 при n≥1,
- t(2*n+1)=2*t(n)-1 при n≥1.

Если $n=2^m+q$, где 2^m – наибольшая степень 2, не превосходящая n, а q – разность $n-2^m$, то номер оставшегося ребенка вычисляется по формуле: $t(2^m+q)=2*q+1$ при $m\geq 0$ и $0\leq q<2^m$. Экспериментально проверьте правильность данной формулы и напишите версию программы, вычисляющей номер оставшегося участника игры без использования ссылочного типа данных. Сравните результаты.

6. В задаче об Иосифе удаляется каждый второй участник игры. Написать программу определения номеров двух последних оставшихся игроков.

7*. В задаче об Иосифе человек находится на месте i. Существует ли значение k (k-й удаляемый человек), такое, что этот человек останется последним в круге. Написать программу поиска значений k.

8. Список из элементов a_1, a_2, …, a_n (значения поля data) назовем упорядоченным по неубыванию, если выполняется следующее условие: $a_1 \le a_2 \le \ldots \le a_n$. Написать функцию проверки упорядоченности списка.

9. Верно ли, что следующая функция возвращает адрес (указатель) первого элемента списка, равного значению *x*.

```
Function Locate(t:pt;x:Integer):pt;
 Begin
  While (t<>nil) And (t^.data<>x) Do t:=t^.next;
  If t=nil Then Locate:=nil
   Else Locate:=t;
 End;
```

10. Список состоит не менее чем из пяти элементов, и мы вызываем Retrieve(first,x), где x – некоторое целое число. Каков результат работы функции? Что она делает?

```
Function Retrieve(t:pt;p:Integer):Integer; {Предполагаем,
что p<>0.}
 Var q:Integer;
 Begin
  q:=1;
  While (t<>nil) And (t^.data<>p) Do Begin
   q:=q+1;
   t:=t^.next;
  End;
  If t=nil Then Retrieve:=0
   Else Retrive:=q;
 End;
```

11. Определить, что делает функция Succ. При её вызове значение t равно first.

```
Function Succ(t:pt; p:Integer):pt;
 Var q:Integer;
 Begin
  q:=0;
  While (t<>nil) And (q<>p) Do t:=t^.next;
  If (t<>nil) And (p=q) Then Retrieve:=t^.next
   Else Retrive:=nil;
 End;
```

12. Определить, что делает функция Pred. При её вызове значение t равно first.

```
Function Pred(t:pt; p:Integer):pt;
 Var q:Integer;
     dt:pt;
 Begin
  q:=0;
  dx:=nil;
  While (t<>nil) And (q<>p) Do Begin
   dt:=t;
   t:=t^.next;
  End;
  If (t<>first) And (p=q) Then Retrieve:=dx
   Else Retrive:=nil;
 End;
```

13. Разработать:

- функцию, вычисляющую среднее арифметическое элементов непустого списка;
- рекурсивную функцию проверки наличия в списке заданного элемента;
- процедуру перестановки первого и последнего элементов непустого списка;
- процедуру вставки нового элемента перед (после) каждым вхождением заданного элемента;
- функцию проверки совпадения списков L_1 и L_2;
- функцию проверки вхождения списка L_1 в список L_2;
- процедуру переноса в конец непустого списка L его первого элемента;
- процедуру переноса в начало непустого списка его последнего элемента;
- процедуру копирования в список L за каждым вхождением заданного элемента всех элементов списка L_1;
- процедуру объединения двух упорядоченных по неубыванию списков L_1 и L_2 в один упорядоченный по неубыванию список путем построения нового списка L и изменением соответствующим образом ссылок в L_1 и L_2;
- функцию подсчета количества слов списка (поле Data имеет тип String), начинающихся и оканчивающихся одним и тем же символом.

14. Разработать процедуру удаления из списка L:

- второго элемента, если такой есть;
- всех элементов, равных x;
- первого отрицательного элемента, если такой есть;
- всех отрицательных элементов.

15. Разработать процедуру формирования списка L путем включения в него по одному разу элементов:

- входящих хотя бы в один из списков L_1 и L_2;
- входящих одновременно в оба списка L_1 и L_2;
- входящих в список L_1, но не входящих в список L_2;
- входящих в один из списков L_1 и L_2, но в то же время не входящих в другой из них.

16. Напишите процедуру вставки элемента в двусвязный список.

17. Приведите рисунок, поясняющий логику удаления элемента из списка во всех случаях: из начала, середины и конца списка. Напишите полный текст процедуры удаления.

18. Добавим в двусвязный список «фиктивный» элемент так, как это показано на рис. 3.10. Значением `prev` первого элемента и `next` последнего элемента является адрес «фиктивного» элемента. Упростятся ли в этом случае процедуры вставки и удаления из списка? Напишите их и сравните с процедурами из упражнений 1 и 2.

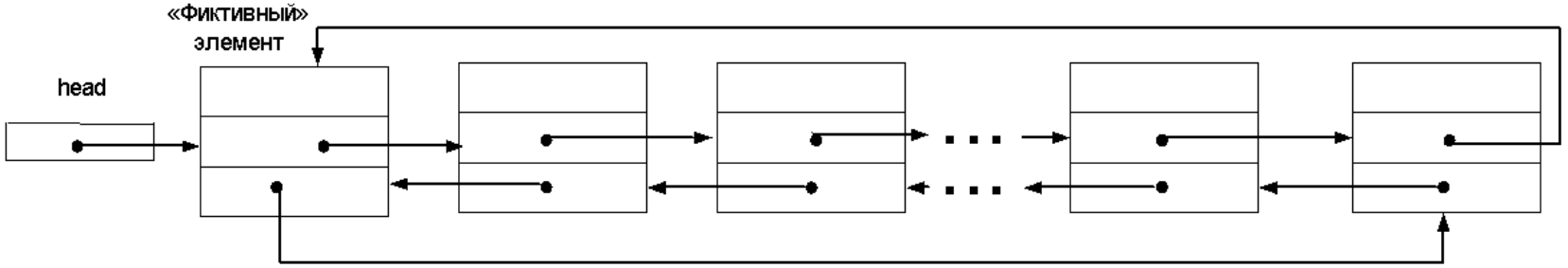

Рис. 3.10. Двусвязный список с «фиктивным» элементом

19. Напишите для двусвязного списка процедуры вставки и удаления перед заданным элементом списка.

20. Двоичное число $a_1, a_2, \ldots, a_n$, где a_i=0 или 1 (значение поле `data`), можно представить в виде двусвязного списка $a_1, a_2, \ldots, a_n$. Напишите процедуру прибавления единицы к двоичному числу.

21. Напишите процедуру слияния двух упорядоченных двусвязных списков в один (упорядоченный).