Глава 2

Списки

> Все сплетено друг с другом... и едва ли найдется что-нибудь чуждое всему остальному. Ибо все объединено общим порядком и служит к украшению одного и того же мира.
>
> *Сенека*

2.1. Основные понятия о ссылочном типе данных (указателях)

> Пушка... Они заряжают пушку...
> Зачем? А! Они будут стрелять!
>
> *Из м/ф «Остров сокровищ»*

Обычные типы данных являются *статическими*. Область памяти для их размещения выделяется на стадии компиляции, и ее перераспределение на стадии выполнения программы не допускается.

Выделение же памяти для переменных уже на стадии выполнения программы возможно с использованием нового типа данных — *указателей* (*ссылок*). Значением указателя (переменной ссылочного типа) является *адрес области памяти* (первой ячейки) переменной заданного базового типа. Таким образом, здесь задается не значение переменной (величины), а адрес памяти, в которой находится переменная (*принцип косвенной адресации*). При этом для указателей область памяти выделяется статически (как обычно), а для переменных, на которые они указывают, — *динамически*, т. е. на стадии выполнения программы (поэтому они и называются *динамическими*). Для хранения динамических переменных выделяется специальная область памяти, называемая «*кучей*».

Работая с указателями, мы работаем с *адресами величин*, а не с их *именами*.

Структуры данных, сконструированные с использованием указателей, часто называют динамическими (по прин-

ципу их размещения в памяти). Отметим ряд преимуществ и недостатков динамических структур. При этом, будучи программистами, оценим их с точки зрения конкретной решаемой задачи.

Преимущества:

- разумное использование динамических структур данных приводит к сокращению объема памяти, необходимого для работы программы;
- динамические данные, в противоположность статическим и автоматически размещаемым данным, не требуют их объявления как данных фиксированного размера, причем в большинстве систем программирования для «кучи» выделяется достаточно большой объем памяти;
- целый ряд алгоритмов более эффективен при их реализации именно с использованием динамических структур. Например, вставка элемента на определенное место в массиве требует перемещения части других элементов массива, но при использовании динамических структур данных для вставки элемента в середину списка достаточно нескольких операторов присваивания.

Недостатки:

- алгоритмы на динамических структурах обычно более сложны, трудны для отладки по сравнению с аналогичными алгоритмами на статических данных;
- использование динамических структур данных требует дополнительных затрат памяти для хранения ссылок (так что в некоторых задачах объем памяти, отводимой для ссылок, превосходит объем памяти, выделяемой непосредственно для данных);
- существуют алгоритмы, реализация которых более эффективна именно на обычных данных. Например, в ряде задач индекс элемента в массиве можно просто вычислять, в то время как использование списковых структур требует обхода списка.

Однако вернемся к основной теме этого раздела.

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, оператор `x^:=15` записывает число 15 в область памяти (два байта), адрес которой является значением указателя `x`.

Оператор `Dispose(x)` освобождает память, занятую динамической переменной; при этом значение указателя `x` становится неопределенным.

Все описанные выше действия показаны на рис. 2.1.

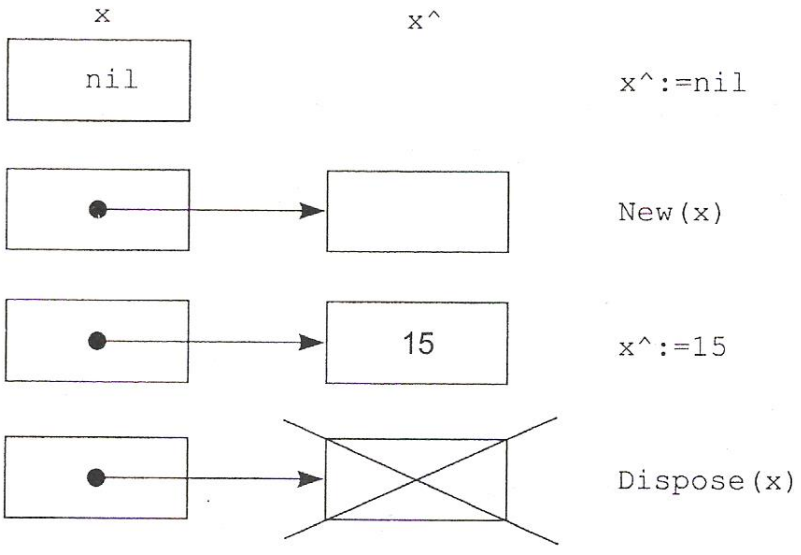

Рис. 2.1. Основные операторы для работы с указателями

2.2. Линейный список

— Анизотропное шоссе, — заявил Антон. Анка стояла к нему спиной. — Движение только в одну сторону.

— Мудры были предки, — задумчиво сказал Пашка. — Этак едешь-едешь километров двести, вдруг — хлоп! — «кирпич».

Аркадий и Борис Стругацкие

Списком называется структура данных, каждый элемент которой при помощи указателя связывается со следующим элементом. Из этого определения следует, что каждый элемент списка содержит, как минимум, поле ссылки на следующий элемент (назовем его `next`) и одно поле данных (назовем его `data` и для простоты будем считать, что оно имеет тип `Integer`), которое может иметь сложную структуру. Поле ссылки последнего элемента списка имеет значение `nil`. Указатель на начало списка (первый элемент) является значением отдельной переменной.

Пример. На рис. 2.2 показан линейный список из четырех элементов, содержащий в полях данных целые числа 3, 5, 1 и 9.

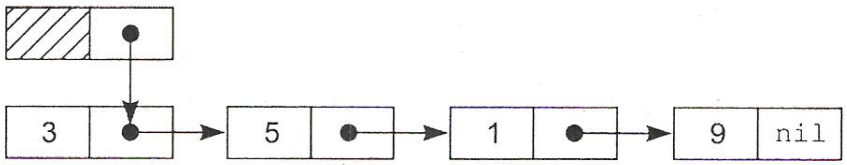

Рис. 2.2. Пример линейного списка

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}
     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) — это достаточно 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. 2.3). Список до вставки элемента `y` показан на рис. 2.3*а*, а после вставки — на рис. 2.3*б*. Изменен-

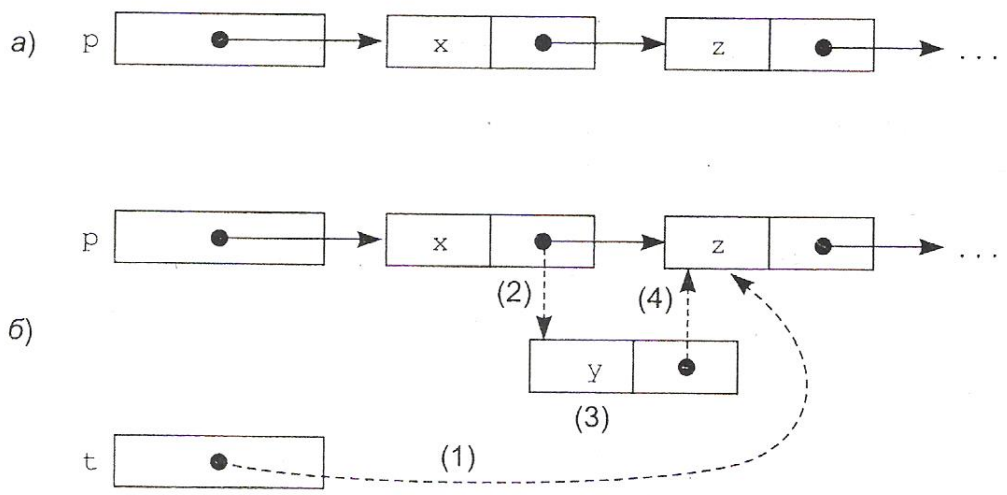

Рис. 2.3. Вставка элемента в список:
а) до операции; б) после операции

ные значения указателей показаны на рисунке 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;
```

Прежде всего здесь в рабочей переменной t запоминается значение указателя (поля next) элемента, после которого вставляется новый элемент. Вторым шагом выделяется область памяти для нового элемента, а ее адрес становится значением указателя элемента, после которого осуществляется вставка. На третьем шаге формируется поле data нового элемента, а на четвертом — в его поле next переписывается запомненное в t значение.

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. Причем 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 же случае эта процедура не работает! Ведь нам требуется изменить значение глобальной переменной first, а мы не передаем в вызывающую логику измененное значение p. Но если изменить параметры процедуры на

```
Procedure Insert(Var p:pt; y:Integer)
```

то проблема будет решена, хотя и останется вопрос вставки первого элемента в пустой список (p = nil): в этом случае действие p^.next «выбрасывает» нас в неопределенную область памяти. В процедуре же Ins_List решается и 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); {Параметр p - глобальный
      в процедуре Insert. Четыре оператора этой
      процедуры лучше просто "прописать" здесь}
  End;
```

Удаление элемента из списка. Действия при удалении элемента из списка сводятся к его поиску, а затем — к переадресации от элемента, предшествующего удаляемому, к элементу, следующему за удаляемым (рис. 2.4*а*). Единственная сложность здесь заключается в том, чтобы предусмотреть случай удаления первого элемента списка, когда изменяется значение переменной `first` (рис. 2.4*б*).

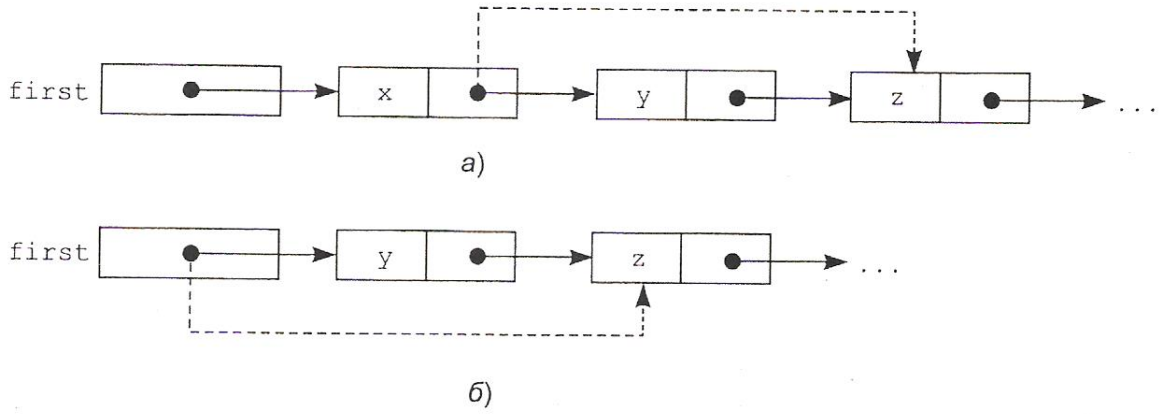

Рис. 2.4. Удаление элемента из списка

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; {Переменная цикла}
    While t<>nil Do {Пока список не просмотрен}
      If t^.data=y Then {Есть совпадение}
        If t=first Then Begin
         {Удаляем первый элемент списка}
         x:=first;
         {Запоминаем, ибо "кучу" засорять не следует}
```

```
        first:=first^.next; {Изменяем значение
        указателя на первый элемент списка}
        Dispose(x); {Освобождаем место в "куче"}
        t:=first;
        {Переменная цикла изменила свое значение}
        End
      Else Begin
        x:=t;
        {Запоминаем адрес удаляемого элемента}
        t:=t^.next;
        dx^.next:=t; {Удаление элемента не должно
        нарушать структуру списка}
        Dispose(x);
      End
    Else Begin
      dx:=t; {Переход к следующему элементу списка.
      Адрес текущего элемента списка запоминается
      в переменной dx}
      t:=t^.next;
    End
End;
```

Упражнения

1. Список из элементов $a_1, a_2, \ldots, a_n$ (значения поля data) назовем упорядоченным по неубыванию, если выполняется следующее условие: $a_1 \leqslant a_2 \leqslant \ldots \leqslant a_n$. Напишите функцию проверки упорядоченности такого списка.

2. 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;
```

3. Список состоит не менее чем из пяти элементов. Мы вызываем функцию Retrieve(first,x), где x — некото-