Дмитриева Марина Валерьевна

ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ

СОЗДАНИЕ И ИСПОЛЬЗОВАНИЕ ЛИНЕЙНЫХ СПИСКОВ

Представление данных с помощью списков необходимо при решении задач обработки сложных структур данных: формул, деревьев, графов. Списки используются в задачах, связанных с формульными преобразованиями, задачах искусственного интеллекта, при представлении стеков, очередей и других структур данных. Многие языки программирования содержат средства, предоставляющие возможность работы со списками. Существуют языки, в которых и структуры данных представлены списком, и структуры управления являются списком. Примером такого языка является язык ЛИСП. В статье рассматриваются списки как абстрактные типы данных, обсуждаются способы представления списков в системе Turbo Pascal и приводятся основные алгоритмы обработки списков.

СТРУКТУРНЫЙ ТИП ДАННЫХ

Структурный тип данных используется для объединения в единый объект разнородной информации. Этим достигается возможность работы с объектом как с единым целым при описании, присваивании и передаче в качестве параметра в процедуру или функцию. С логической точки зрения очень важное свойство языка – возможность описывать объект сколь угодно сложной конструкции, и затем манипулировать с ним, как с целым.

Над структурным типом данных определена одна операция – выборка поля. Эта операция используется как для анализа объекта, так и для присваивания новых значений отдельным полям структуры.

Можно строить сколь угодно сложные динамически меняющиеся данные, используя механизм структур в качестве базового. При этом данные представляются в виде отдельных элементов, каждый из которых связывается с другими с помощью указателей. Отдельный элемент является структурным объектом, содержащий в качестве некоторых полей значение типа указателя на другие элементы.

Простейшим способом образования динамической структуры данных является связывание элементов в список. В этом случае каждый элемент содержит указатель на следующий элемент.

ЛИНЕЙНЫЕ ЦЕПНЫЕ СПИСКИ

В большинстве случаев существует несколько способов представления данных. Выбор способа представления зависит от того, как эти данные будут обрабатываться

...данные представляются в виде отдельных элементов, каждый из которых связывается с другими с помощью указателей...

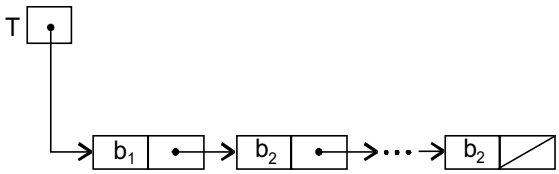


Рис. 1. Представление линейного списка

в программе. Данные представляются с помощью списка в случае, когда их обработка состоит в последовательном просмотре элементов, количество которых во время выполнения программы может меняться.

Списком будем называть конечное множество, каждый элемент которого содержит информацию о доступе к одному или нескольким элементам списка. Такое представление позволяет просматривать и обрабатывать элементы списка в определенной последовательности. Списки используются в задачах, связанных с формульными преобразованиями, в задачах искусственного интеллекта, при написании трансляторов и при решении других задач.

Простейший тип списка – это список, каждый элемент которого, кроме последнего элемента, связан с единственным (следующим) элементом. Будем называть такой список линейным. Существуют списки с более сложной структурой: симметричные, циклические, двусвязные и др.

Списки относятся к рекурсивным типам данных. Значения типа данных, которые называют рекурсивными, содержат одну или более компонент того же типа, что и все значение.

Списки удобно изображать с помощью прямоугольников и стрелок. Прямоугольники служат для обозначения элементов списка, стрелки – для связи между элементами. Перечеркнутое поле элемента списка означает отсутствие указателя на следующий элемент. Последовательность элементов **b1, b2, ..., bn** представлена на рис. 1 линейным списком. Поле, содержащее элемент списка (b_i), назовем информационным, поле для связи – ссылочным или указателем. Список задается указателем **T** на первый элемент списка.

ОПИСАНИЕ СПИСКОВ С ИСПОЛЬЗОВАНИЕМ УКАЗАТЕЛЕЙ

Гибкость динамических структур, в частности списков, состоит в том, что в любой момент времени можно добавить к списку новый элемент, или удалить любой элемент, освободив занимаемую память.

В языке Turbo Pascal предусмотрена возможность выделения памяти для размещения в ней различных данных, и освобождения памяти, когда данные уже обработаны. Такая возможность связана с определением в языке особых типов данных – указателей.

В языке Turbo Pascal элемент списка можно представить записью, одно из полей которой содержит информацию о следующем элементе. Это поле называется указателем или ссылкой на следующий элемент списка. Остальные поля (или поле) будем называть информационными полями.

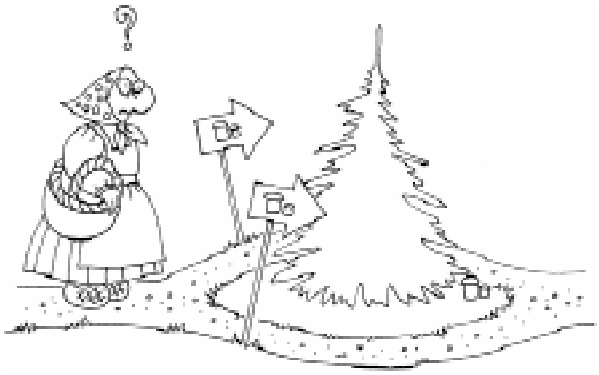
Если элемент списка имеет тип **T**, то тип значения, называемого ссылкой (указателем) на элемент типа **T** записывается **^T**. Естественно для типа **^T** можно ввести синоним **P** в разделе определения типов **Type P=^T**.

При описании процедур работы со списками будем считать, что определены структуры данных, представленные в листинге 1.

Информационные поля элементов, из которых строится список, задаются описа-

Листинг 1. Описание структур данных для работы с линейными списками

```
type
   elem_list = integer; {тип информационного поля}
   list = ^elem; (указатель на элемент списка)
   elem= record info: elem_list; (информационное поле списка)
                next: list {указатель на следующий элемент}
         end;
```



...оба указателя установлены ... на разные переменные, но значения этих переменных одинаковы.

телем **elem_list**. Заметим, что в определении **list=^elem** используется идентификатор **elem**, который еще не определен. В общем случае конструкция **^T** – это единственный описатель типа, который может содержать еще не определенный в программе идентификатор **T**.

В приведенном описании идентификатор **elem** является синонимом записи с двумя полями. Первое поле с выделителем **info** – информационное поле, его тип **elem_list**, второе поле с выделителем **next** – указатель, его тип – **list**.

В программе можно описать переменную **var L: list**. В этом случае говорят, что **L** является указателем на элемент типа **elem**. Переменная **L** может иметь значение **Nil**. Это означает, что указатель не установлен ни на какой элемент. Если значение переменной **L** отлично от **Nil**, то указатель и элемент, на который установлен указатель, можно изобразить так, как на рис. 2.

Переменная, на которую установлен указатель **L**, обозначается **L^**. В рассматриваемом случае **L^** имеет тип **elem**, поэтому получить доступ к полям записи можно,

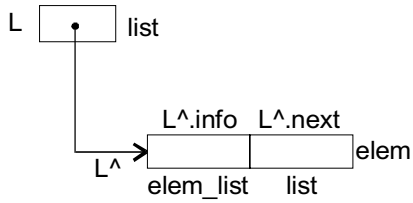


Рис. 2. Указатель на элемент списка

воспользовавшись выборками **L^.info** и **L^.next**.

Пусть в программе описаны переменные **P** и **Q**:

```
Var P, Q: list.
```

Предположим, что переменные получили некоторые значения. После выполнения оператора присваивания **P:=Q** оба указателя устанавливаются на одну и ту же переменную, на которую первоначально был установлен указатель **Q**.

После выполнения оператора присваивания **P^ := Q^** оба указателя установлены, вообще говоря, на разные переменные, но значения этих переменных одинаковы.

Предположим, что поля с выделителем **next** тех элементов, на которые установлены указатели **P** и **Q**, равны **nil** (см. рис. 3).

После выполнения оператора присваивания **P^next:= Q^** оба элемента будут связаны в список (рис. 4).

Тип выборки **P.^next** и переменной **Q** – **list**. После выполнения оператора присваивания **P.^next** и **Q** указывают на один и тот же элемент.

Указатели, ссылающиеся на объекты разных типов, сами являются объектами разных типов. Указатели одного и того же типа можно сравнивать с помощью операций равно (=) и не равно (<>).

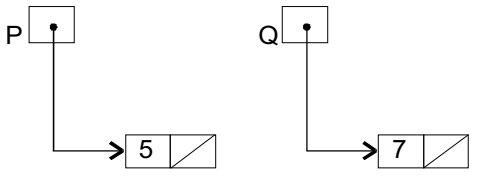


Рис. 3. Два элемента перед связыванием в список

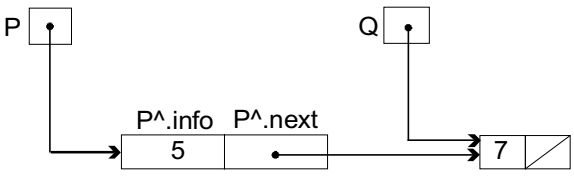


Рис. 4. Список из двух элементов

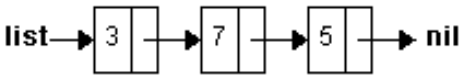


Рис. 5. Исходный список

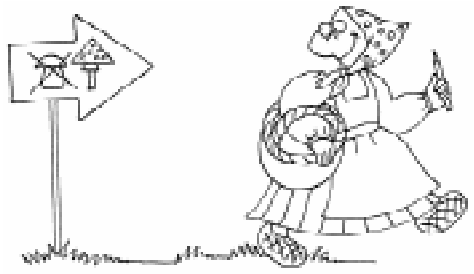
Стандартная процедура `New(p)`

Значение переменной типа указателя может измениться в результате выполнения стандартной процедуры **`New(p)`**, параметром которой в простейшем случае должна быть переменная типа указателя. При выполнении **`New(p)`** выделяется память под значение типа **`T`**, и указатель **`p`** устанавливается на новый, созданный объект типа **`T`** с неопределенными значениями полей.

ДОБАВЛЕНИЕ ЭЛЕМЕНТА В НАЧАЛО ЛИНЕЙНОГО СПИСКА

Для построения линейного списка нужно уметь добавлять элемент с заданным информационным полем в список. Элементы можно добавлять в начало списка, в конец списка, за определенным элементом или перед заданным элементом списка.

Для того чтобы добавить новый элемент в начало списка, требуется выполнить преобразования, которые мы проиллюстрируем с помощью рисунков. Пусть исходный список состоит из элементов с информационными полями 3, 7, 5. Вид исходного списка приведен на рис. 5



Значение переменной типа указателя может измениться

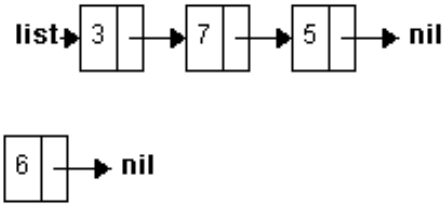


Рис. 6. Исходный список и новый элемент

Для вставки элемента с информационным полем 6 в начало списка требуется создать новый элемент с информационным полем 6. (рис. 6).

На рис. 7 указатель второго поля нового элемента устанавливается на первый элемент списка. В данный момент на первый элемент списка установлено два указателя.

На рис. 8 изменен указатель на первый элемент списка таким образом, чтобы он указывал на новый, только что созданный элемент списка.

Список после выполнения указанных действий приведен на рис. 9. Таким образом, новый элемент вставлен в начало списка.

Опишем процедуру добавления элемента в начало списка. Процедура имеет два параметра. Первым параметром является указатель на начало списка, в который добавляется элемент. Вторым параметром является значение, которое должно иметь информационное поле добавляемого элемента. При выполнении процедуры сначала производится запрос на выделение памяти под новый элемент списка. Информационным полем нового элемента становится вто-

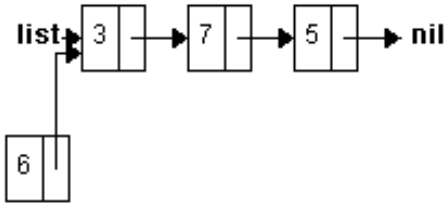


Рис. 7. Новый элемент вставляется перед первым элементом списка

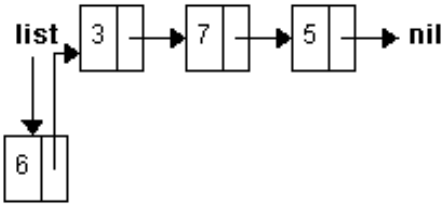
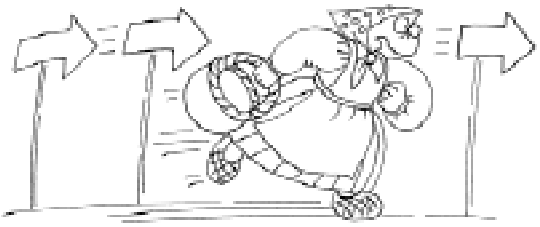


Рис. 8. Указатель на новый элемент списка



...чтобы узнать, из каких элементов состоит список, надо последовательно просмотреть элементы списка...

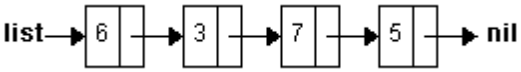


Рис. 9. Список после добавления элемента в начало

рой параметр процедуры. Оператор присваивания **p^next :=t** обеспечивает вставку нового элемента перед первым элементом исходного списка. Оператор присваивания **t:=p** изменит указатель списка на первый, только что созданный элемент. Текст процедуры приведен в листинге 2.

ВЫВОД ЗНАЧЕНИЙ ИНФОРМАЦИОННЫХ ПОЛЕЙ СПИСКА

Для того чтобы узнать, из каких элементов состоит список, надо последовательно просмотреть элементы списка и вывести значения информационных полей. Опишем процедуру просмотра элементов списка. Значение информационного поля каждого элемента помещается в стандартный файл вывода. Переменная **cur** «пробегает» по списку, обрабатывая каждый из его элементов. Описание процедуры приведено в листинге 3.

ПОСТРОЕНИЕ ДВУХ СПИСКОВ ПО ПОСЛЕДОВАТЕЛЬНОСТИ ЧИСЕЛ

Рассмотрим решение следующей задачи. В стандартном файле располагается последовательность ненулевых целых чисел, число ноль считается признаком конца последовательности. Требуется сформировать две последовательности чисел. В одну последовательность включить положительные числа, во вторую – отрицательные.

При решении задачи будем использовать два списка. В первый список помещать положительные элементы, во второй – отрицательные. При просмотре входной последовательности анализируется очередное число и добавляется элемент списка с прочитанным информационным полем в один из списков.

Листинг 2. Добавление элемента в начало списка

```
procedure add_beg (var t: list; r: elem_list);
  var p: list;
begin new (p); {1}
      p^.info := r; {2}
      p^.next := t; {3}
      t := p {4}
end;
```

Листинг 3. Вывод информационных полей списка

```
procedure out_info (t:list);
  var cur: list;
begin cur := t;{указатель на начало списка}
      while (cur <> nil) do
         begin writeln (cur^.info);
                    {вывод информационного поля очередного элемента}
               cur := cur^.next {переход к следующему элементу списка}
        end
end;
```

Значение указателя p становится неопределенным.

После просмотра входной последовательности сформированы оба списка. Следующий этап – вывести значения информационных полей сначала одного списка, а затем другого. Для вывода элементов списка воспользуемся процедурой **out_info(p)**. Программа, решающая задачу, приведена в листинге 4. Тела процедур **add_beg** и **out_info** в тексте программы не приводятся.

Стандартная процедура dispose (p)

Процедура **dispose (p)**, где **p** – указатель, позволяет освободить область памяти, на которую указывает указатель **p**. В дальнейшем освобожденная память может быть вновь использована. Значение указателя **p** становится неопределенным.

ВОЗВРАЩЕНИЕ СИСТЕМЕ ПАМЯТИ, ЗАНИМАЕМОЙ СПИСКОМ

Опишем процедуру, которая возвращает память, занятую элементами списка. В листинге 5 располагается текст процедуры, в которой последовательно просматриваются элементы списка, и память, занимаемая элементами, возвращается системе.

В точке {1} указатель **p** устанавливается на элемент, который должен быть удален из списка. В точке {2} указатель **t** переносится на следующий элемент списка. В точке {3} память, занятая элементом списка, на который установлен указатель **p**, возвращается системе. После того как указатель **t** пробежит по всем элементам списка, память, занятая под элементы списка,

Листинг 4. Построение двух списков

```
program p_4;
type
   elem_list=integer;
   list=^elem;
   elem= record info: elem_list; next: list end;
var t1,t2: list; r: elem_list;
{добавление элемента в начало списка}
procedure add_beg (var t: list; r: elem_list);
  {тедо процедуры};
{вывод информационных полей списка}
procedure out_info (t:list);
   {тедо процедуры};
begin t1:= nil; t2:= nil;{списки пусты}
     read (r);{прочли первый элемент последовательности}
      while r <> 0 do
        begin
         if r > 0 then add_beg(t1,r) else add_beg(t2,r);
          read (r)
        end;
      writeln('Список с положительными элементами:');
      out_info(t1); writeln('конец списка');
      writeln('Список с отрицательными элементами:');
      out_info(t2); writeln('конец списка');
end.
```

Листинг 5. Возвращение системе памяти, занимаемой списком

```
Procedure del_list (var t: list);
  Var p: list;
Begin while t <> nil do
        Begin p := t; {1}
              t := t^.next; {2}
              dispose (p) {3}
        end
end
```

будет освобождена и возвращена системе (листинг 5).

ЗАДАНИЕ 1

1. Опишите процедуру добавления элемента в конец списка.
2. Опишите процедуру добавления элемента перед заданным элементом.
3. Опишите функцию определения числа элементов в списке.
4. Опишите функцию определения максимального элемента списка.
5. Опишите функцию, выдающую указатель на последний элемент списка с минимальным значением информационного поля.
6. Опишите функцию, определяющую количество минимальных элементов в линейном списке, информационные поля которого – целые числа.
7. Опишите функцию, значение которой истинно, если информационные поля всех элементов списка различны.

Дмитриева Марина Валерьевна,
доцент кафедры информатики математико-механического факультета Санкт-Петербургского государственного университета.