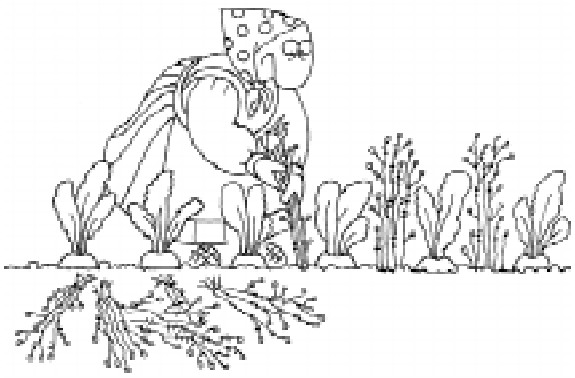
Дмитриева Марина Валерьевна

ПРОЦЕДУРЫ КАК ДАННЫЕ И ЗАДАЧА ОБХОДА СПИСКА

В статье рассматривается механизм итерации сложных структур данных на примере линейных списков. Суть механизма итерации заключается в отделении информации о внутреннем устройстве сложной структуры данных (в частности, списка) от процедур обработки его элементов. Рассматриваются процедурные типы данных, процедурные константы и переменные, обсуждаются возможности использования процедур в качестве параметров. Приведены рекурсивные алгоритмы работы со списками. Рассмотрен вариант представления списка с помощью массива.

ПРОЦЕДУРНЫЙ ТИП ДАННЫХ

В программировании часто встречается ситуация, когда в алгоритме используется результат вычисления некоторой функции, но сам алгоритм не зависит от того, какая именно функция вычисляется.



Задача обхода списка состоит в том, чтобы просмотреть все элементы списка, выполнив с каждым из них одно и то же действие...

В языке Turbo Pascal можно описать и использовать так называемый процедурный тип данных. Описание процедурного типа состоит из слова procedure или function, за которым в круглых скобках записывается список формальных параметров. Для функции после списка формальных параметров через двоеточие указывается тип функции.

После того, как определен процедурный тип, можно описать переменные процедурного типа. Переменным процедурного типа можно присваивать значения конкретных процедур и функций. Естественно, что процедурная переменная и та процедура, которая присваивается ей в качестве значения, должны иметь одинаковое число формальных параметров, совпадающих по типам. Если переменная имеет тип функции, то для функций, кроме формальных параметров, должны совпадать типы функций.

Те процедуры и функции, которые будут использоваться либо в качестве значений для процедурных переменных, либо в качестве фактических параметров при вызове процедур и функций, должны удовлетворять следующим правилам:

– должны компилироваться с ключом компилятора {$F+};

– не должны быть стандартными процедурами или функциями;

– не должны объявляться внутри других процедур или функций.

ОБХОД (ИТЕРАЦИЯ) СПИСКА

Под *обходом,* или *итерацией* сложной структуры понимается выполнение некоторого действия над всеми элементами этой структуры. Задача обхода списка состоит в том, чтобы просмотреть все элементы списка, выполнив с каждым из них одно и то же действие. Ранее была рассмотрена задача, в которой требовалось просмотреть список и поместить в стандартный файл вывода значения информационных полей элементов списка. Это задача обхода списка, обработка элемента списка состояла в помещении значения информационного поля в файл.

Предположим, требуется решить задачу: увеличить значения каждого информационного поля списка в 5 раз.

Решение этой задачи отличается от решения задачи вывода информационных полей списка только тем, что обработка элемента списка заключается в увеличении в 5 раз значения информационного поля списка.

Другими словами, вместо вызова процедуры **writeln(p^.info)** следует выполнить оператор присваивания

p^.info:=5*p^.info.

Воспользуемся процедурным типом данных для задания действия, которое требуется выполнить над информационным полем элемента списка. Будем считать, что определен тип данных **action**:

```
Type action= procedure (var z: elem_list)
```

Трактовать этот тип можно следующим образом: выполнить действия со значением типа **elem_list**.

ПРОЦЕДУРНЫЕ КОНСТАНТЫ И ПРОЦЕДУРНЫЕ ПЕРЕМЕННЫЕ

Опишем процедуру, которая обходит список и с каждым элементом выполняет действие **p** (листинг 1).

Листинг 1. Обход списка

```
procedure trav_list (t:list);
  var cur: list;
begin cur := t;
      while (cur <> nil) do
        begin p (cur^.info);
              cur := cur^.next
        end
end;
```

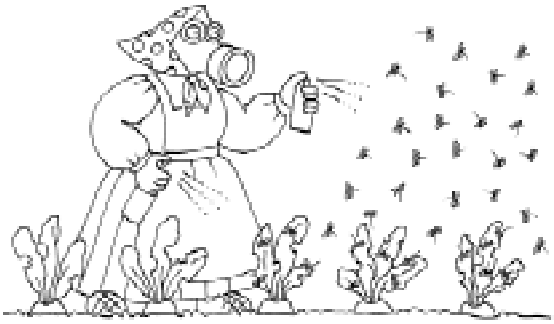
Такая процедура обхода не имеет самостоятельного смысла, пока не определено, какое действие выполняется над элементами списка. Конкретное действие должно быть задано до вызова процедуры обхода списка, например, с помощью присваивания процедурной переменной **p** некоторого значения. Процедурная переменная **p** является глобальной. Значение эта переменная должна получить в программе.

Приведем программу, в которой сначала строится линейный список, его значения выводятся в файл, затем значения информационных полей списка увеличиваются в пять раз, после чего опять осуществляется вывод значений в файл. Увеличение элементов списка и вывод значений информационных полей осуществляется при выполнении процедуры обхода списка. В листинге 2 содержится описание процедур вывода значения параметра в файл и увеличение значения параметра процедуры в пять раз.

Описанные процедуры будут использоваться в правых частях оператора присваивания, поэтому компилироваться они должны с ключом {$F+}. Можно сказать, что процедуры **out_info** и **mul5** являются процедурными константами (их значения нельзя изменить). Тип этих констант – **action**. Перед тем как использовать процедуру обхода с выводом значений в файл надо выполнить оператор присваивания: **p := out_info**.

Листинг 2. Описание процедур обработки элементов списка

```
{$F+}
procedure out_info (var z:elem_list);
  begin writeln (z,'')  end;
procedure mul5 (var z:elem_list);
  begin z := 5*z end;
{$F-}
```



Опишем процедуру обхода списка с двумя параметрами.

Если же требуется обойти список и увеличить значения всех элементов, то перед вызовом процедуры обхода следует выполнить присваивание процедурной переменной **p** значения другой процедурной константы: **p:= mul5**. В листинге 3 приведена программа решения задачи. Обратите внимание на использование процедурных констант и переменных, они выделены в тексте полужирным начертанием.

ПЕРЕДАЧА ПРОЦЕДУРЫ В КАЧЕСТВЕ ПАРАМЕТРА

Действие, которое требуется выполнить с каждым элементом списка, можно передать в процедуру обхода в качестве параметра процедурного типа. Опишем процедуру обхода списка с двумя параметрами. Первый параметр задает ссылку на первый элемент списка, который требуется обработать (обойти). Второй параметр задает

Листинг 3. Программа с процедурными константами и переменными

```
program p3;
type
   elem_list = integer; {тип элемента списка}
   list = ^elem;
   elem = record info: elem_list; next: list end;
   action= procedure (var z: elem_list);
var t: list; r: elem_list;
var p: action;
{добавление элемента в начало списка}
procedure add_beg (var t: list; r: elem_list);
  var q: list;
begin new (q); q^.info := r; q^.next := t; t := q end;
{обход списка, задание действия с помощью переменной}
procedure trav_list (t:list);
  var cur: list;
begin cur := t;
      while (cur <> nil) do
        begin p(cur^.info); cur := cur^.next  end
end;
{$F+}
procedure out_info (var z:elem_list);
  begin writeln (z,'')  end;
procedure mul5 (var z:elem_list);
  begin z := 5*z end;
{$F-}
begin t:= nil;  read (r); {прочли первое значение последовательности}
      while r <> 0 do
        begin add_beg(t,r); read (r) end; {список построен}
      writeln('Список:'); p := out_info; trav_list (t);
                                          writeln('конец списка');
      p := mul5; trav_list (t);
      writeln('Список:'); p := out_info; trav_list (t);
                                          writeln('конец списка');
end.
```

Листинг 4. Процедура обхода списка с параметром процедурного типа

```
procedure trav_list (t:list; p: action);
  var cur: list;
begin cur := t;{указатель на первый элемент списка}
      while (cur <> nil) do
        begin p(cur^.info);{обработка очередного элемента}
              cur := cur^.next  {переход к следующему элементу}
        end
end;
```

действие, которое требуется выполнить с каждым элементом списка. Второй параметр процедурного типа. Описание процедуры обхода списка приведено в листинге 4.

В программе для решения предыдущей задачи изменилась процедура обхода списка и раздел операторов программы. В листинге 5 приведен раздел операторов программы, полужирным начертанием выделены части текста, относящиеся к процедурным типам.

Может показаться, что общая процедура обхода не имеет смысла, поскольку про-

Может показаться, что общая процедура обхода не имеет смысла ...

Листинг 5. Обход списка с процедурой в качестве параметра

```
program p5;
...
action= procedure (var z: elem_list);
...
{процедура обхода списка}
procedure trav_list (t:list; p: action);
  var cur: list;
begin cur := t;
      while (cur <> nil) do
        begin p(cur^.info); cur := cur^.next end
end;
{Процедуры, используемые при обходе списка}
{$F+}
procedure out_info (var z:elem_list);
  begin writeln (z,'')  end;
procedure mul5 (var z:elem_list);

  begin z := 5*z end;
{$F-}
begin t:= nil; {список пуст}
      read (r); {чтение первого элемента}
     while r <> 0 do begin add_beg(t,r); read (r) end;{список построен}
      writeln('Список:'); trav_list(t,out_info);
                                          writeln('конец списка');
      trav_list(t,mul5);
      writeln('Список:'); trav_list (t,out_info);
                                           writeln('конец списка');
end.
```

цедуры, подобные рассмотренным, описать гораздо проще, не используя общего механизма итерации списка. Заметим, однако, что ни одна из процедур обработки элементов не использует знаний о структуре списка.

Смысл процедуры итерации состоит именно в отделении информации о внутреннем устройстве списка от процедур обработки элементов.

РЕКУРСИВНЫЕ АЛГОРИТМЫ РАБОТЫ СО СПИСКОМ

Списки относятся к рекурсивным типам данных. Для рекурсивных типов данных естественно применять рекурсивные методы обработки. Приведем решение двух задач с помощью рекурсивных процедур.

Рассмотрим задачу обхода списка. Если список пуст (тривиальный случай), то делать ничего не надо, осуществляется выход из блока. Если список не пуст, то обрабатывается первый элемент списка. Далее осуществляется обход списка без первого элемента (рекурсивный вызов).

При решении задачи поиска элемента в списке поступаем следующим образом. Если список пуст, то он не содержит ни одного элемента, следовательно, в нем нет элемента, совпадающего с образцом. Если список не пуст, то сравнивается первый элемент с образцом. Если они равны, то задача решена. Если же элементы различны, то результат поиска будет зависеть от того, найден или нет нужный элемент в «хвосте» списка. В листинге 6 приведено описание рекурсивных процедур обхода списка и поиска элемента в списке.

ПРЕДСТАВЛЕНИЕ СПИСКА С ПОМОЩЬЮ МАССИВА

Список можно представить как массив записей. Первое поле записи соответствует информационному полю элемента списка, второе поле содержит индекс следующего элемента списка. Такое представление используется в случаях, в частности, когда в языке не определены данные типа указателей. Можно воспользоваться представлением списка с помощью массива и в случае, когда список требуется сохранить, например, в текстовом или типизированном файле.

Если список пуст ...,
то делать ничего не надо ...

Листинг 6. Рекурсивные обход списка и поиск элемента в списке

```
{обход списка, рекурсивный вариант}
procedure trav_list (t: list; p: action);
begin if t = nil then exit;
      p (t^.info);
      trav_list (t^.next, p)
end;
{поиск элемента в списке, рекурсивный вариант}
function sear_rec (t: list; r: elem_list): boolean;
  begin if t=nil then sear_rec := false
        else
          begin if t^.info = r
                then sear_rec := true
                else sear_rec := sear_rec (t^.next, r)
          end
  end
```

Для обработки списка требуется знать индекс первого элемента списка, и удобно использовать индекс последнего занятого под элементы списка элемента массива. В листинге 7 рассмотрена программа, при выполнении которой по последовательности чисел формируется два списка (один содержит лишь положительные элементы, другой лишь отрицательные). При обходе списка значения информационных полей выводятся в стандартный файл вывода.

ЗАДАЧИ

1. Опишите процедуру, определяющую число элементов в списке при обходе списка. Используйте процедурный тип данных.

Листинг 7. Программа построения и обхода списка, представленного массивом

```
program p16_28;
const max_elem = 20; {максимальное число элементов списка}
type
   elem_list = integer; {тип элемента списка}
   elem = record info: elem_list; next: byte end;
   list = array[1..max_elem] of elem;
   action = procedure (var r: elem_list);
var t1,t2: list; r: elem_list;
    ind1, ind2: byte;
    first1, first2: byte;
{добавление элемента в начало списка}
procedure add_beg (var t: list; var first, ind: byte; r: elem_list);
begin inc(ind);
      t[ind].info := r;
      t[ind].next := first;
      first := ind
end;
{обход списка}
procedure trav_info (var t: list; first, ind: byte; p: action);
  var cur: byte;
begin cur := first;      while (cur <> 0) do
        begin p (t[cur].info);
              cur := t[cur].next
        end
end;
{$F+}
procedure out (var z: elem_list);
begin write(z,' * ') end;
{$F-}
begin ind1:= 0; ind2:= 0; first1 := 0; first2 := 0;
      read (r);
      while r <> 0 do
        begin
          if  r > 0 then  add_beg(t1, first1,ind1, r)
                    else  add_beg(t2, first2,ind2, r);
          read (r)
        end;
      writeln('начало списка:'); trav_info(t1, ind1, first1, out);
      writeln('конец списка');
      writeln('начало списка:'); trav_info(t2, ind2, first2, out);
      writeln('конец списка');
end.
```

2. Опишите процедуру, определяющую максимальный элемент списка при обходе списка. Используйте процедурный тип данных.

3. Опишите рекурсивную процедуру сортировки линейного списка.

4. Опишите рекурсивную процедуру, которая освобождает память, занятую элементами списка.

5. Два полинома степени n и m представляются в виде списка. Информационное поле элемента списка содержит коэффициент и степень. Напишите процедуры, которые строят полином и вычисляют сумму, разность, произведение заданных полиномов. Результирующий полином также представляется списком. Списки должны быть представлены с помощью массивов.

© Наши авторы, 2007
Our authors, 2007

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