Лабораторная работа № 2

Односвязные списки

Цель работы: изучить и исследовать односвязные списки.

Указатели

Обычно переменная хранит некоторые данные. Однако помимо обычных, существуют переменные, которые ссылаются на другие переменные. Такие переменные называются указателями. Указатель — это переменная, значением которой является адрес другой переменной или структуры данных. Графически указатель может быть изображен так, как на рис. 1.

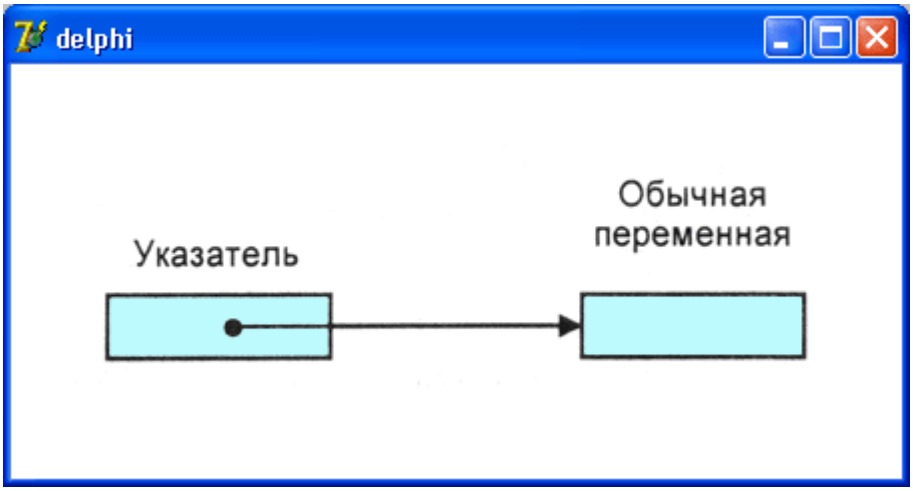


Рис. 1. Переменная-указатель

Указатель, как и любая другая переменная программы, должен быть объявлен в разделе объявления переменных. В общем виде объявление указателя выглядит следующим образом:

Имя: ^ Тил;

где:

- имя — имя переменной-указателя;
- Тип — тип переменной, на которую указывает переменная-указатель;

значок ^ показывает, что объявляемая переменная является указателем.

Приведем примеры объявления указателей:

p1: ^integer; p2: ^real;

В приведенном примере переменная p1 — это указатель на переменную типа integer, а p2 — указатель на переменную типа real.

Тип переменной, на которую ссылается указатель, называют типом указателя. Например, если в программе объявлен указатель p: ^integer, то говорят: ^p — указатель целого типа" или "p — это указатель на целое".

В начале работы программы переменная-указатель "ни на что не указывает". В этом случае говорят, что значение указателя равно NIL. Зарезервированное слово NIL соответствует значению указателя, который ни на что не указывает.

Идентификатор NIL можно использовать в инструкциях присваивания и в условиях. Например, если переменные p1 и p2 объявлены как указатели, то инструкция

p1 := NIL;

устанавливает значение переменной, а инструкция

ifp2 = **NILthen**ShowMessage('Указатель p2 не инициализирован!');

проверяет, инициализирован ли указатель p2.

Указателю можно присвоить значение — адрес переменной соответствующего типа (в тексте программы адрес переменной — это имя переменной, перед которым стоит оператор @). Ниже приведена инструкция, после выполнения которой переменная p будет содержать адрес переменной n.

p := @n;

Помимо адреса переменной, указателю можно присвоить значение другого указателя при условии, что они являются указателями на переменную одного типа. Например, если переменные p1 и p2 являются указателями типа integer, то в результате выполнения инструкции

p2 := p1;

переменные p1 и p2 указывают на одну и ту же переменную.

Указатель можно использовать для доступа к переменной, адрес которой содержит указатель. Например, если p указывает на переменную i, то в результате выполнения инструкции

p^ : = 5;

значение переменной i будет равно пяти. В приведенном примере значок ^показывает, что значение пять присваивается переменной, на которую указывает переменная-указатель.

Динамические переменные

Динамической переменной называется переменная, память для которой выделяется во время работы программы.

Выделение памяти для динамической переменной осуществляется вызовом процедуры new. У процедуры new один параметр — указатель на переменную того типа, память для которой надо выделить. Например, если p является указателем на тип real, то в результате выполнения процедуры new(p); будет выделена память для переменной типа real (создана переменная типа real), и переменная-указатель p будет содержать адрес памяти, выделенной для этой переменной.

У динамической переменной нет имени, поэтому обратиться к ней можно только при помощи указателя.

Процедура, использующая динамические переменные, перед завершением своей работы должна освободить занимаемую этими переменными память или, как говорят программисты, уничтожить динамические переменные. Для освобождения памяти, занимаемой динамической переменной, используется процедура Dispose, которая имеет один параметр — указатель на динамическую переменную.

Например, если p — указатель на динамическую переменную, память для которой выделена инструкцией new(p), то инструкция dispose (p) освобождает занимаемую динамической переменной память.

Следующая процедура (ее текст приведен в листинге 1) демонстрирует создание, использование и уничтожение динамических переменных.

Листинг 1. Создание, использование и уничтожение динамических переменных

```
procedure TForm1.Button1Click(Sender: TObject);

var

p1,p2,p3: ^Integer; // указатели на переменные типа integer

begin

// создадим динамические переменные типа integer
```

```
// (выделим память для динамических переменных)

New(p1);

New(p2);

New(p3);

p1^ := 5;

p2^ := 3;

p3^ := p1^ + p2^;

ShowMessage('Сумма чисел равна ' + IntToStr(p3^));

// уничтожим динамические переменные

// (освободим память, занимаемую динамическими переменными)

Dispose(p1);

Dispose(p2);

Dispose(p3);

end;
```

В начале работы процедура создает три динамические переменные. Две переменные, на которые указывают p1 и p2, получают значение в результате выполнения инструкции присваивания. Значение третьей переменной вычисляется как сумма первых двух.

Списки

Указатели и динамические переменные позволяют создавать сложные динамические структуры данных, такие как списки и деревья.

Список можно изобразить графически (рис. 2).

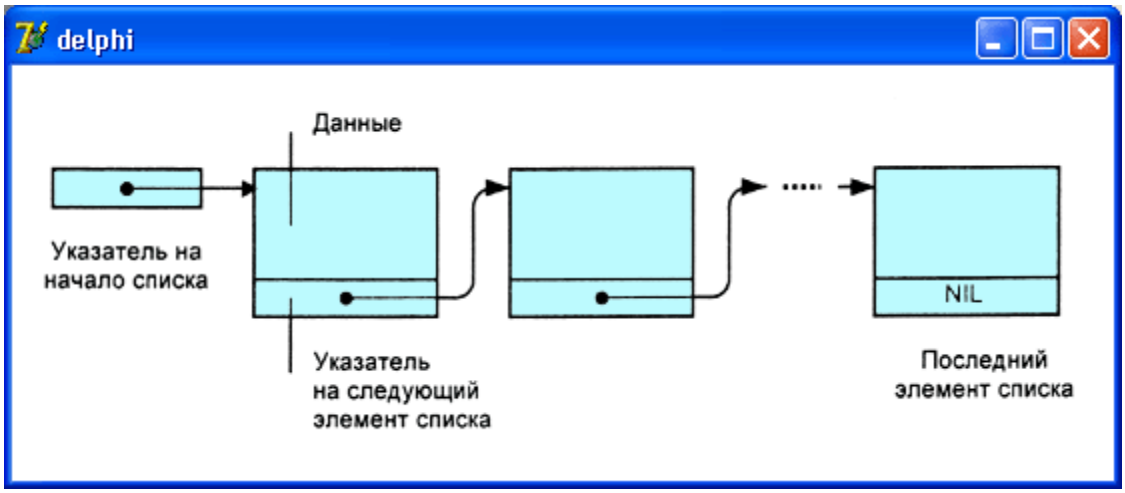


Рис. 2. Графическое изображение списка

Каждый элемент списка (узел) представляет собой запись, состоящую из двух частей. Первая часть — информационная. Вторая часть отвечает за связь со следующим и, возможно, с предыдущим элементом списка. Список, в котором обеспечивается связь только со следующим элементом, называется односвязным.

Для того чтобы программа могла использовать список, надо определить тип компонентов списка и переменную-указатель на первый элемент списка. Ниже приведен пример объявления компонента списка студентов:

```
type

TPStudent = ^TStudent; // указатель на переменную типа TStudent

// описание типа элемента списка

TStudent = record

surname: string[20]; // фамилия

name: string[20]; // имя

group: integer; // номергруппы

address: string[60]; // домашнийадрес

next: TPStudent; // указатель на следующий элемент списка

end;

var
```

head: TPStudent; // указатель на первый элемент списка

Добавлять данные можно в начало, в конец или в нужное место списка. Во всех этих случаях необходимо корректировать указатели. На рис. 3 изображен процесс добавления элементов в начало списка.

После добавления второго элемента в список head указывает на этот элемент.

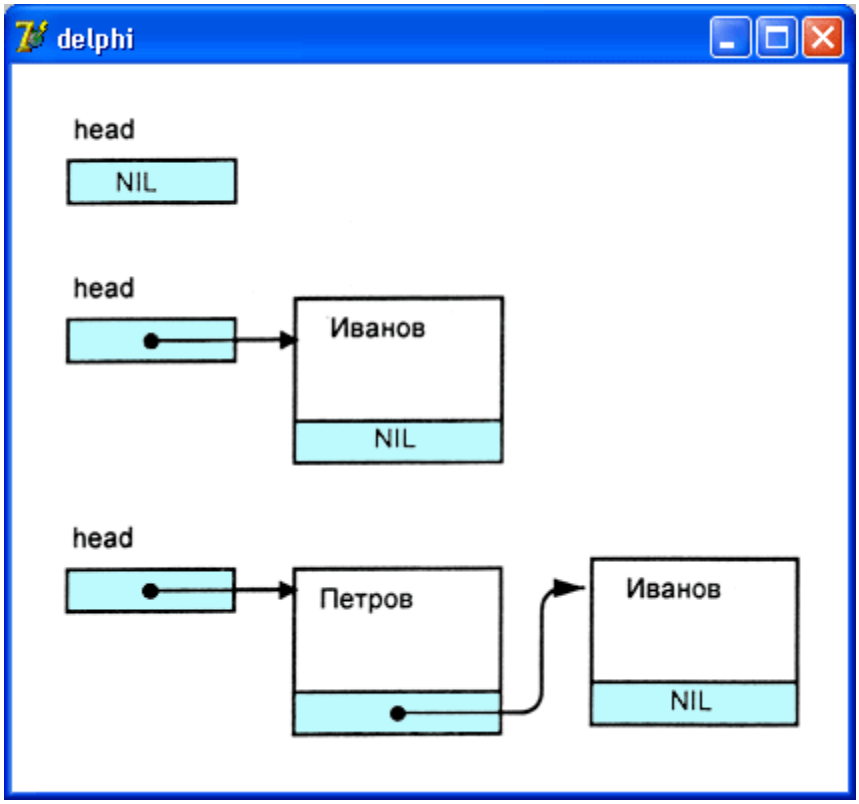


Рис. 3. Добавление элементов в список

Следующая программа (ее текст приведен в листинге 2) формирует список студентов, добавляя фамилии в начало списка. Данные вводятся в поля редактирования диалогового окна программы (рис. 4) и добавляются в список нажатием кнопки **Добавить** (button1).

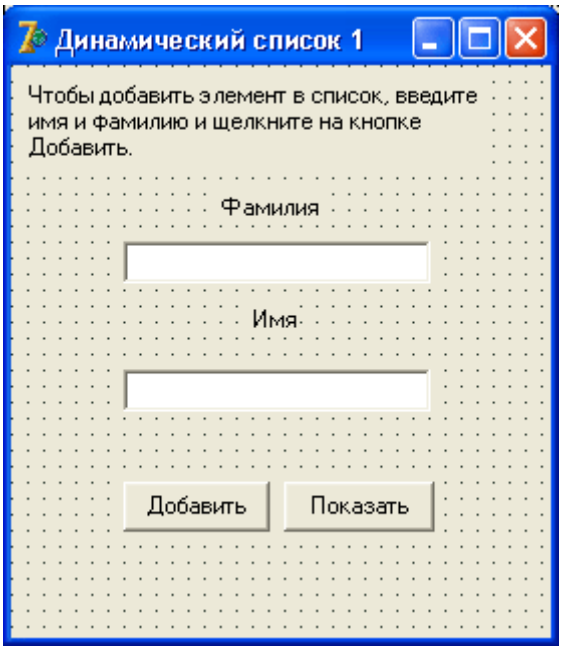


Рис. 4. Окно программы **Динамический список 1**

Листинг 2. Добавление элемента в начало динамического списка

```
unit Unit1;


interface


uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms,
  Dialogs, StdCtrls;


type
  TForm1 = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    Button1: TButton;
```

```
    Button2: TButton;
    Edit1: TEdit;
    Edit2: TEdit;
    Label3: TLabel;
procedure Button1Click(Sender: TObject);
procedure Button2Click(Sender: TObject);
private
{ Private declarations }
public
{ Public declarations }
end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

type
TPStudent=^TStudent; // указатель на тип TStudent
TStudent = record
f_name:string[20]; // фамилия
l_name: string[20]; // имя
next: TPStudent; // следующий элемент списка
```

```
end;
var
head: TPStudent; // начало (голова) списка
// добавить элемент в начало списка



procedure TForm1.Button1Click(Sender: TObject);
var
curr: TPStudent; // новый элемент списка
begin
new(curr); // выделить память для элемента списка
curr^.f_name := Edit1.Text;
curr^.l_name := Edit2.Text;
// добавление в начало списка
curr^.next := head; head := curr;
// очистить поля ввода
Edit1.text:=''; Edit2.text:= '';
end;

// вывестисписок
procedure TForm1.Button2Click(Sender: TObject);
var
curr: TPStudent; // текущий элемент списка
n: integer; // длина (кол-во элементов) списка
```

```
st: string; // строковое представление списка

begin

n := 0; st := '';

curr := head; // указатель на первый элемент списка

whilecurr<> NIL do begin

n := n + 1;

st := st + curr^.f_name + ' ' + curr^.l_name+#13;

curr := curr^.next;  // указатель на следующий элемент

end;

if n <> 0

then ShowMessage('Список:' + #13 + st)

else ShowMessage('В списке нет элементов.');

end;

end.
```

Добавление элемента в список выполняет процедура TForm1.Button1Click, которая создает динамическую переменную-запись, присваивает ее полям значения, соответствующие содержимому полей ввода диалогового окна, и корректирует значение указателя head.

Вывод списка выполняет процедура TForm1.Button2Click, которая запускается нажатием кнопки **Показать.** Для доступа к элементам списка используется указатель curr. Сначала он содержит адрес первого элемента списка. После того как первый элемент списка будет обработан, указателю curr присваивается значение поля next той записи, на которую указывает curr. В результате этого переменная curr содержит адрес второго элемента списка. Таким образом, указатель перемещается по списку. Процесс повторяется до тех пор, пока значение поля next текущего элемента списка (элемента, адрес которого содержит переменная curr) не окажется равно NIL.

Упорядоченный список

Как правило, списки упорядочены. Порядок следования элементов в списке определяется содержимым одного из полей. Например, список с информацией о людях обычно упорядочен по полю, содержащему фамилии.

Добавление элемента в список

Добавление элемента в список выполняется путем корректировки указателей. Для того чтобы добавить элемент в упорядоченный список, нужно сначала найти элемент, после которого требуется вставить новый. Затем следует скорректировать указатели. Указатель нового элемента нужно установить на тот элемент, на который указывает элемент, после которого добавляется новый. Указатель элемента, после которого добавляется новый элемент, установить на этот новый элемент (рис. 5).

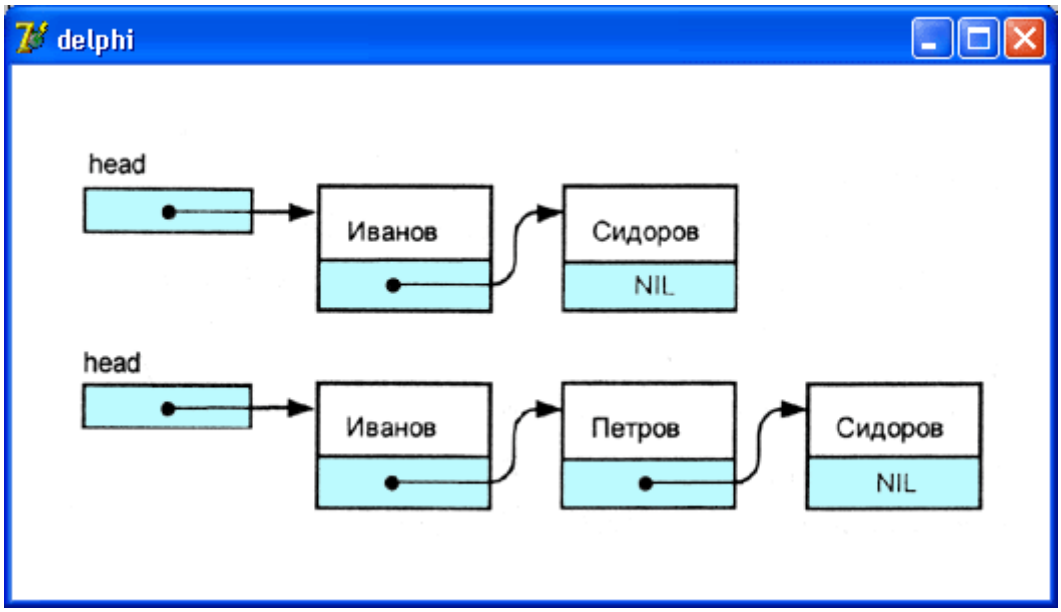


Рис. 5. Добавление элемента в упорядоченный список

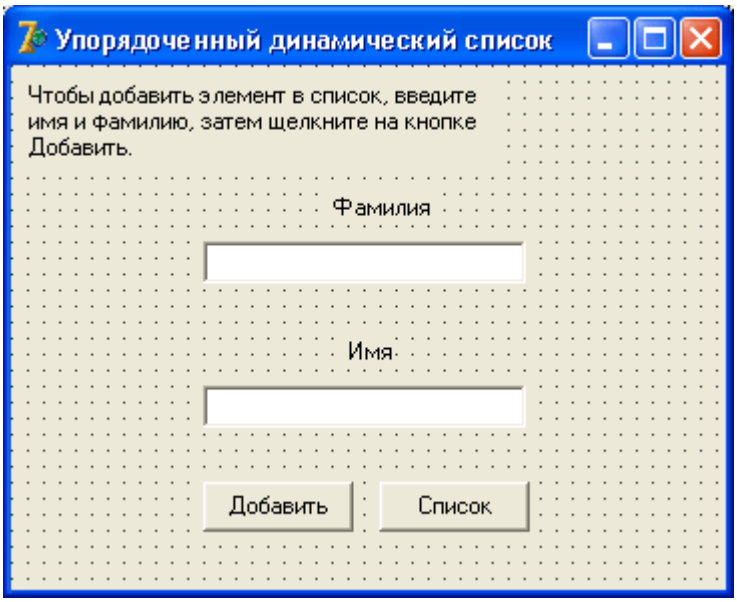


Рис. 6. Диалоговое окно программы **Упорядоченный динамический список 2**

Следующая программа (ее текст приведен в листинге 3, а диалоговое окно — на рис. 6) формирует список, упорядоченный по полю **Фамилия.** Данные вводятся в поля редактирования (Edit1 и Edit2) и нажатием кнопки **Добавить** (Button1) добавляются в список таким образом, что список всегда упорядочен по полю **Фамилия.**

Листинг3. Добавлениеэлементоввупорядоченныйсписок

```
unitUnit1;


interface


uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms,Dialogs, StdCtrls;


type

  TForm1 = class(TForm)

    Label1: TLabel;
```

```
    Label2: TLabel;
    Button1: TButton;
    Button2: TButton;
    Edit1: TEdit;
    Edit2: TEdit;
    Label3: TLabel;
procedure Button1Click(Sender: TObject);
procedure Button2Click(Sender: TObject);
procedureFormActivate(Sender: TObject);
private
{ Private declarations }
public
{ Public declarations }
end;
var
  Form1: TForm1;

implementation

{$R *.dfm}
type
TPStudent=^TStudent; // указательнатипTStudent
TStudent = record
f_name:string[20]; // фамилия
l_name: string[20]; // имя
```

```
next: TPStudent; // следующийэлементсписка
end;
var
head: TPStudent; // начало (голова) списка
// добавитьэлементвсписок
procedure TForm1.Button1Click(Sender: TObject);
var
node: TPStudent; // новыйузелсписка
curr: TPStudent; // текущийузелсписка
pre: TPStudent; // предыдущий, относительноcurr, узел
begin
new(node); // создание нового элемента списка
node^.f_name:=Edit1.Text; // фамилия
node^.l_name:=Edit2.Text; // имя
// добавление узла в список
// сначала найдем в списке подходящее место для узла
curr:=head;
pre:=NIL;
{ Внимание!
Если приведенное ниже условие заменить
на (node. f_name>curr^. f_name) and (curr<>NIL) ,
то при добавлении первого узла возникает ошибка времени
выполнения, т. к. curr = NIL и, следовательно,
переменной curr.^name нет!
В используемом варианте условия ошибка не возникает, т. к.
```

```
сначала проверяется условие (curr<>NIL), значение которого
FALSE, и второе условие в этом случае не проверяется.
}
while (curr<> NIL) and (node.f_name>curr^.f_name) do
begin
// введенное значение больше текущего
pre:= curr;
curr:=curr^.next; // кследующемуузлу
end;
ifpre = NILthen
begin
// новый узел в начало списка
node^.next:=head; head:=node;
end
else
begin
// новыйузелпосле pre, передcurr
node^.next:=pre^.next;
pre^.next:=node;
end;
Edit1.text:='';
Edit2.text:='';
Edit1.SetFocus;

end;
```

```
// вывестисписок
procedure TForm1.Button2Click(Sender: TObject);
var
curr: TPStudent; // текущий элемент списка
n: integer; // длина (кол-во элементов) списка
st: string; // строковое представление списка
begin
n := 0; st := '';
curr := head; // указатель на первый элемент списка
whilecurr<> NIL do begin
n := n + 1;
st := st + curr^.f_name + ' ' + curr^.l_name+#13;
curr := curr^.next;  // указатель на следующий элемент
end;
if n <> 0
thenShowMessage('Список:' + #13 + st)
elseShowMessage('В списке нет элементов.');
end;
// началоработыпрограммы
procedure TForm1.FormActivate(Sender: TObject);
begin
head:=NIL; // списокпустой
end;


end.
```

Процедура TForm1.Button1Click создает динамическую переменную-запись, присваивает ее полям значения, соответствующие содержимому полей ввода диалогового окна, находит подходящее место для узла и добавляет этот узел в список, корректируя при этом значение указателя узла next, после которого должен быть помещен новый узел.

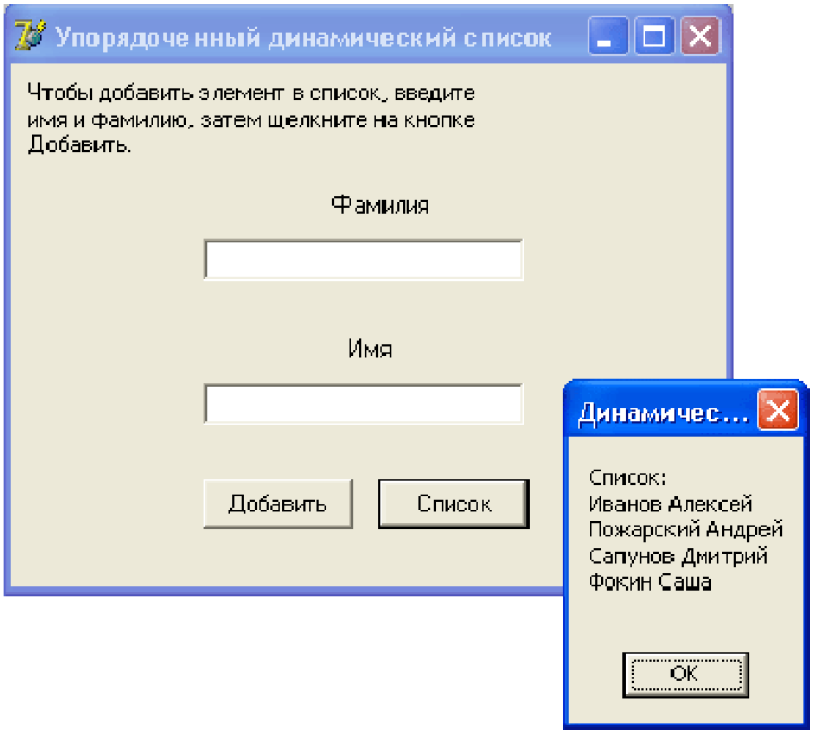


Рис. 7. Пример упорядоченного списка, сформированного программой

Вывод списка выполняет процедура TForm1.Button2Click, которая запускается нажатием кнопки**Показать.** После запуска программы и ввода нескольких фамилий, например, в такой последовательности: Иванов, Яковлев, Алексеев, Петров, список выглядит так, как показано на рис. 7.

Удаление элемента из списка

Для того чтобы удалить узел, необходимо скорректировать значение указателя узла, который находится перед удаляемым узлом (рис. 8).

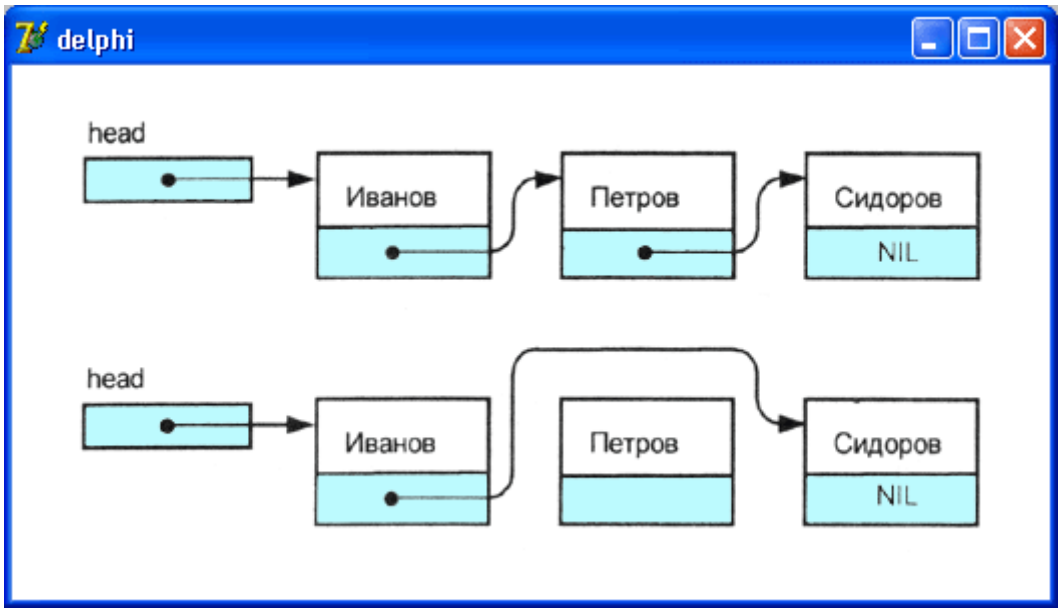


Рис. 8. Удаление элемента из списка

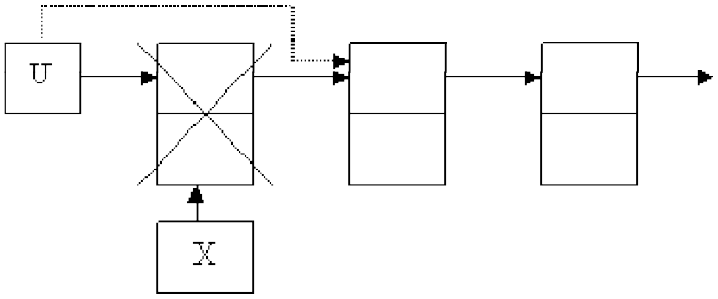
Поскольку узел является динамической переменной, то после исключения узла из списка занимаемая им память должна быть освобождена. Освобождение динамической памяти, или, как иногда говорят, "уничтожение переменной", выполняется вызовом процедуры dispose. У процедуры dispose один параметр — указатель на динамическую переменную. Память, занимаемая этой динамической переменной, должна быть освобождена. Например, в программе

```
var

p: ^integer;

begin

new(p);

{ инструкции программы }

dispose(p);

end
```

создается динамическая переменная p, а затем она уничтожается. Освободившуюся память смогут использовать другие переменные. Если этого не сделать, то, возможно, из-за недостатка свободной памяти в какой-то момент времени программа не сможет создать очередную динамическую переменную.

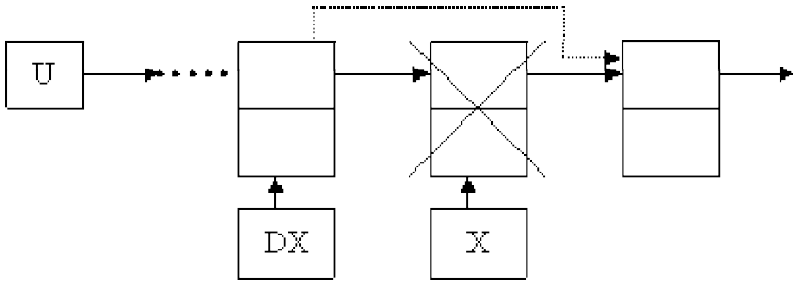
Алгоритм удаления первого элемента списка отличается от алгоритма удаления элемента из середины списка.

А)Удаление первого элемента. Для этого во вспомогательном указателе запомним первый элемент, указатель на голову списка переключим на следующий элемент списка и освободим область динамической памяти, на которую указывает вспомогательный указатель.




```
x:= u;
u:= u^.next;
dispose(x);
```

Б) Удаление элемента из середины списка. Для этого нужно знать адреса удаляемого элемента и элемента, стоящего перед ним. Допустим, что digit – это значение удаляемого элемента.




```
x:= u;
while ( x<> nil) and ( x^. inf<> digit) do
begin
dx:= x;
x:= x^.next;
end;
dx^.next:= x^.next;
dispose(x);
```

Следующая программа позволяет добавлять и удалять узлы упорядоченного списка. Диалоговое окно программы приведено на рис. 9.

Процедуры добавления узла в список и вывода списка, а также объявление типа узла списка ничем не отличаются от соответствующих процедур рассмотренной ранее программы **Упорядоченный динамический список** 2.

Удаление узла из списка выполняет процедура TForm1.Button3Click, которая запускается нажатием кнопки**Удалить** (Button3). Текст процедуры приведен в листинге 4.

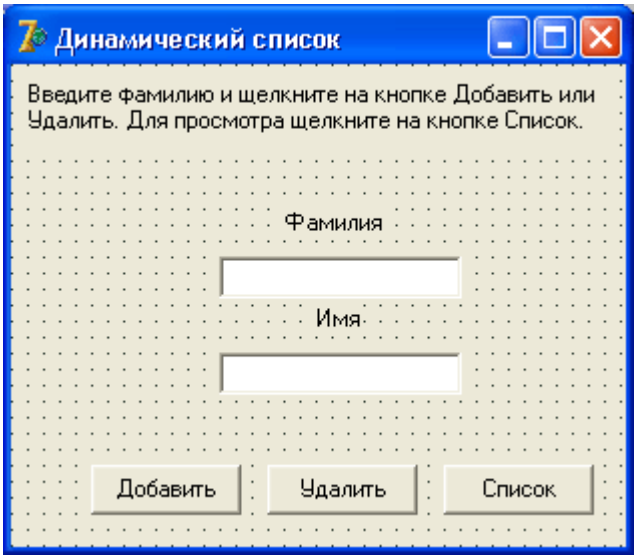


Рис. 9. Окно программы **Динамический список**

Листинг 4. Удалениеузлаизсписка

```
unitUnit1;


interface


uses

  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms, Dialogs, StdCtrls;


type

  TForm1 = class(TForm)

    Label1: TLabel;

    Label2: TLabel;

    Button1: TButton;
```

```
    Button2: TButton;
    Edit1: TEdit;
    Edit2: TEdit;
    Label3: TLabel;
    Button3: TButton;
procedure Button1Click(Sender: TObject);
procedure Button2Click(Sender: TObject);
procedureFormActivate(Sender: TObject);
procedure Button3Click(Sender: TObject);
private
{ Private declarations }
public
{ Public declarations }
end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

type
TPStudent=^TStudent; // указательнатипTStudent
TStudent = record
```

```
f_name:string[20]; // фамилия
l_name: string[20]; // имя
next: TPStudent; // следующийэлементсписка
end;
var
head: TPStudent; // начало (голова) списка

// добавитьэлементвсписок
procedure TForm1.Button1Click(Sender: TObject);
var
node: TPStudent; // новыйузелсписка
curr: TPStudent; // текущийузелсписка
pre: TPStudent; // предыдущий, относительноcurr, узел
begin
new(node); // создание нового элемента списка
node^.f_name:=Edit1.Text; // фамилия
node^.l_name:=Edit2.Text; // имя
// добавление узла в список
// сначала найдем в списке подходящее место для узла
curr:=head;
pre:=NIL;
{ Внимание!
Если приведенное ниже условие заменить
на (node. f_name>curr^. f_name) and (curr<>NIL) ,
то при добавлении первого узла возникает ошибка времени
```

```
выполнения, т. к. curr = NIL и, следовательно,
переменной curr.^name нет!
В используемом варианте условия ошибка не возникает, т. к.
сначала проверяется условие (curr<> NIL), значение которого
FALSE, и второе условие в этом случае не проверяется.
}
while (curr<> NIL) and (node.f_name>curr^.f_name) do
begin
// введенное значение больше текущего
pre:= curr;
curr:=curr^.next; // к следующему узлу
end;
if pre = NIL then
begin
// новый узел в начало списка
node^. next:=head; head:=node;
end
else
begin
// новыйузелпослеpre, передcurr
node^.next:=pre^.next;
pre^.next:=node;
end;
Edit1.text:='';
Edit2.text:='';
```

```
Edit1.SetFocus;


end;


// вывестисписок
procedure TForm1.Button2Click(Sender: TObject);
var
curr: TPStudent; // текущий элемент списка
n: integer; // длина (кол-во элементов) списка
st: string; // строковое представление списка
begin
n := 0; st := '';
curr := head; // указатель на первый элемент списка
whilecurr<> NIL do begin
n := n + 1;
st := st + curr^.f_name + ' ' + curr^.l_name+#13;
curr := curr^.next;  // указатель на следующий элемент
end;
if n <> 0
thenShowMessage('Список:' + #13 + st)
elseShowMessage('В списке нет элементов.');



end;
```

```
// началоработыпрограммы
procedure TForm1.FormActivate(Sender: TObject);
begin
head:=NIL; // списокпустой
end;

// удалитьэлементизсписка
procedure TForm1.Button3Click(Sender: TObject);
var
curr: TPStudent; // текущийузелсписка
pre: TPStudent; // предыдущий, относительноcurr, узел
st: string; // строковое представление списка

begin
curr:=head;
pre:=NIL;

while (curr<> NIL) and (Edit1.Text <>curr^.f_name) do
begin
// поиск значение больше текущего
pre:= curr;
curr:=curr^.next; // к следующему узлу
end;

ifcurr = NIL then
```

```
begin
// искомое значение в списке не обнаружено
ShowMessage('В списке нет такого элемента.');
end
else
begin
st := st + curr^.f_name + ' ' + curr^.l_name+#13;
ShowMessage('Удаляемыйэлемент:' + #13 + st) ;

if pre = NIL then
begin
// удаляется узел из начала списка
curr:=head; head:=head^.next;
dispose(curr);
end
else
begin
// удаляетсянайденныйузел

pre^.next:=curr^.next;
dispose(curr);

end;
Edit1.text:='';
Edit2.text:='';
```

```
Edit1.SetFocus;

end;

end;


end.
```

Процедура просматривает список от начала, сравнивая содержимое полей текущего узла с содержимым полей ввода диалогового окна. Если их содержимое совпадает, то процедура удаляет его. Если узла, который хочет удалить пользователь, в списке нет, программа выводит сообщение об ошибке.

Задание.

1. Изучить односвязные списки.

2. Разработать программу для работы с односвязными списками (добавление элементов в односвязный список; добавление элементов в упорядоченный список; удаление элемента из упорядоченного списка).

3. Выполнить задание:

1. Убедиться, что список является упорядоченным. Если есть элементы нарушающие упорядоченность, то удалить их. (Использовать в качестве базового вариант программы **Листинг 2**)

2. Исключить из упорядоченного списка все элементы с заданным именем.

3. Исключить из упорядоченного списка все элементы с фамилиями, начинающимися с заданной буквы.

Контрольные вопросы

1. Что такое список?
2. В чем преимущества использования списков?

3. В чем недостатки использования списков?
4. В каких случаях удобно использовать списки?
5. Опишите структуру элемента списка.
6. Как создать пустой список?
7. Как создать новый элемент списка?
8. Как включить в начало списка новый элемент, на который ссылается указатель p?
9. Как удалить из списка 1-й элемент?
10. В чем преимущество упорядоченных списков.

Контрольные задания

1. Напишите фрагмент программы включения элемента в конец списка.
2. Разработайте программу для удаления из списка всех элементов, равных последнему.
3. Разработайте программу для замены на заданный идентификатор значения предпоследнего элемента списка.
4. Разработайте программу для замены на заданный идентификатор значения последнего элемента списка.
5. Разработайте программу для определения количество элементов в списке, следующих после заданного идентификатора.