4 Подпрограммы

В практике программирования часто складываются ситуации, когда одну и ту же группу операторов, реализующих определенную цель, требуется повторить без изменений в нескольких местах программы. Для избавления от столь нерациональной траты времени была предложена концепция подпрограммы.

Подпрограмма – именованная, логически законченная группа операторов языка, которую можно вызвать для выполнения любое количество раз из различных мест программы. В языке Free Pascal существуют два вида подпрограмм: *процедуры* и *функции*. Главное *отличие процедуры от функции* заключается в том, что результатом исполнения операторов, составляющих тело функции, всегда является некоторое значение, поэтому функцию можно использовать непосредственно в выражениях, наряду с переменными и константами.

4.1 Общие сведения о подпрограммах. Локальные и глобальные переменные

Итак, *подпрограмма* – это поименованный набор описаний и операторов, выполняющих определенную задачу. Информация, передаваемая в подпрограмму для обработки, называется *параметрами*, а результат вычислений – *значениями*. Обращение к подпрограмме называют *вызовом*. Перед вызовом подпрограмма должны быть обязательно описана в разделе описаний. *Описание подпрограммы* состоит из заголовка и тела. В *заголовке* объявляется имя подпрограммы и в круглых скобках ее параметры, если они есть (для функции необходимо сообщить тип возвращаемого ею результата). *Тело подпрограммы* следует за заголовком и состоит из описаний и исполняемых операторов.

Любая подпрограмма может содержать описание других подпрограмм. Константы, переменные, типы данных могут быть объявлены как в основной программе, так и в подпрограммах различной степени вложенности. Переменные, константы и типы, объявленные в основной программе до определения подпрограмм, называются *глобальными*, они доступны всем функциям и процедурам. Переменные, константы и типы, описанные в какой-либо подпрограмме, доступны только в ней и называются *локальными*.

Для правильного определения области действия идентификаторов (переменных) необходимо придерживаться следующих правил:

- каждая переменная, константа или тип должны быть описаны перед использованием;
- областью действия переменной, константы или типа является та подпрограмма, в которой они описаны;
- все имена в пределах подпрограммы, в которой они объявлены, должны быть уникальными и не должны совпадать с именем самой подпрограммы;
- одноименные локальные и глобальные переменные – это разные переменные, обращение к таким переменным в подпрограмме трактуется как обращение к локальным переменным (глобальные переменные недоступны);
- при обращении к подпрограмме доступны объекты, которые объявлены в ней и до ее описания.

4.2 Формальные и фактические параметры. Передача параметров в подпрограмму

Обмен информацией между вызываемой и вызывающей функциями осуществляется с помощью механизма передачи параметров. Переменные, указанные в заголовке подпрограммы называются *формальными параметрами* или просто параметрами подпрограммы. Эти переменные могут использоваться внутри подпрограммы. Список переменных в операторе вызова подпрограммы – это *фактические параметры,* или *аргументы*.

Механизм передачи параметров обеспечивает обмен данных между формальными и фактическими параметрами, что позволяет выполнять подпрограмму с различными данными. Между фактическими параметрами в операторе вызова и формальными параметрами в заголовке подпрограммы устанавливается взаимно однозначное соответствие. *Количество, типы и порядок следования формальных и фактических параметров должны совпадать*.

Передача параметров выполняется следующим образом. Вычисляются выражения, стоящие на месте фактических параметров. В памяти выделяется место под формальные параметры в соответствии с их типами. Выполняется проверка типов и при их несоответствии выдается диагностическое сообщение. Если количество и типы формальных и фактических параметров совпадают, то начинает работать

механизм передачи данных между фактическими и формальными параметрами.

Формальные параметры процедуры можно разделить на два класса: *параметры-значения* и *параметры-переменные*.

При передаче данных через *параметры-значения* в подпрограмму передаются значения фактических параметров, и доступа к самим фактическим параметрам из подпрограммы нет.

При передаче данных *параметры-переменные* заменяют[51] формальные параметры, и, следовательно, в подпрограмме есть доступ к значениям фактических параметров. Любое изменение параметров-переменных в подпрограмме приводит к изменению соответствующих им формальных параметров. Следовательно, *входные данные следует передавать через параметры-значения*, для *передачи изменяемых в результате работы подпрограммы данных следует использовать параметры-переменные*.

От общетеоретических положений перейдем к практическому использованию подпрограмм при решении задач. Изучение подпрограмм начнем с процедур.

4.3 Процедуры

Описание процедуры имеет вид:

```
procedure имя_процедуры(формальные_параметры);
    label
        описание_меток;
    const
        описание_констант;
    type
        описание_типов;
    var
        описание_переменных;

    begin
        //Тело процедуры.
    end;
```

Начинается описание с *заголовка процедуры*, где `procedure` – ключевое слово языка, `имя_процедуры` – любой допустимый в языке Free Pasacal идентификатор, `формальные_параметры` – имена

51 Реально в подпрограмму передаются адреса фактических параметров.

формальных параметров и их типы, разделенные точкой с запятой. Рассмотрим примеры заголовков процедур с параметрами-значениями:

```
procedure name_1(r:real; i:integer; c:char);
```

Однотипные параметры могут быть перечислены через запятую:

```
procedure name_2(a,b:real; i,j,k:integer);
```

Список формальных параметров не обязателен и может отсутствовать:

```
procedure name_3;
```

Если в заголовке процедуры будут применяться *параметры-переменные*, то перед ними необходимо указывать служебное слово `var`, перед *параметрами-значениями* слово `var` отсутствует:

```
procedure name_4(x,y:real; var z:real);
                  //x, y – параметры-значения,
                  //z – параметр-переменная.
```

После заголовка идет *тело процедуры*, которое состоит из раздела описаний[52] (константы, типы, переменные, процедуры и функции, используемые в процедуре) и операторов языка, реализующих алгоритм процедуры.

Для обращения к процедуре необходимо использовать *оператор вызова*:

```
имя_процедуры(список_фактических_параметров);
```

Фактические параметры в списке оператора вызова отделяются друг от друга запятой:

```
a:=5.3; k:=2;
s:='a';
name_1(a, k, s);
```

Если в описании процедуры формальные параметры отсутствовали, то и при вызове их быть не должно:

```
name_3;
```

ЗАДАЧА 4.1. Найти действительные корни квадратного уравнения $ax^2+bx+c=0$.

Алгоритм решения этой задачи был подробно описан в задаче 3.3 (рис. 3.14). Однако там не была рассмотрена ситуация некорректного ввода значений коэффициентов. Например, если пользователь введет $a=0$, тот уравнение из квадратного превратится в линейное. Алгоритм

52 Раздел описаний в процедуре может отсутствовать, если в нем нет необходимости.

решения линейного уравнения тривиален: $x=-c/b$, при условии, что $b\neq0$. Чтобы не усложнять уже составленный алгоритм решения квадратного уравнения, запишем его в виде *подпрограммы-процедуры*. Далее приведен фрагмент программы с комментариями:

```
//Процедура для вычисления действительных
//корней квадратного уравнения.
procedure korni(a,b,c:real;var x1,x2:real;
                                  var pr:boolean);
 //Входные параметры процедуры
 //(параметры-значения):
 //a,b,c – коэффициенты квадратного уравнения;
 //Выходные параметры процедуры
 //(параметры-переменные):
 //x1,x2 – корни квадратного уравнения;
 //pr – логическая переменная,
 //принимает значение ложь,
//если в уравнении нет корней
 //и значение истина в противном случае.
 var
 d:real;
 begin
     d:=b*b-4*a*c;
     if d<0 then
         pr:=false
     else
     begin
         pr:=true;
         x1:=(-b+sqrt(d))/2/a;
         x2:=(-b-sqrt(d))/(2*a);
     end
 end;    //Конец подпрограммы
//Основная программа
var a_,b_,c_,x1_,x2_,x_:real; pr_:boolean;
begin
 write('a_:='); readln(a_);
 write('b_:='); readln(b_);
 write('c_:='); readln(c_);
 if a_=0 then   //Если a_=0, то уравнение
```

```
                    //квадратным не является.
 begin
     //Решение линейного уравнения bx+c=0.
     if b_<>0 then
     begin
         x_:=-c_/b_;
         writeln('x=',x_);
     end
     else
         writeln('Нет корней');
 end
 else       //Решение уравнения ax^2+bx+c=0.
 Begin
     //Вызов процедуры.
     korni(a_,b_,c_,x1_,x2_,pr_);
     if pr_=false then
         writeln('Нет корней')
     else
         writeln('x1=',x1_,' x2=',x2_);
 end;
end.
```

ЗАДАЧА 4.2. Вводится последовательность из N целых положительных чисел. В каждом числе найти наибольшую и наименьшую цифры.

Для решения задачи создадим процедуру max_min, результатом работы которой будут два значения: минимальная и максимальная цифры в заданном числе.

Текст программы:

```
//Процедура возвращает max наибольшую
//и min наименьшую цифры в числе M.
//В списке параметров:
//M параметр-значение (входной параметр),
//max и min параметры-переменные
//(выходные параметры).
procedure max_min(M:longint; var max:byte;
                                 var min:byte);
var  i: byte;
begin
```

```
 i:=1;
 while M div 10>0 do
 begin
     if i=1 then
     begin
//Предположим, что первая цифра является
         max:=M mod 10; //наибольшей или
         min:=M mod 10; //наименьшей.
         i:=i+1;
     end;
     //Поиск цифры больше max или меньше  min
     if M mod 10 > max then max:=M mod 10;
     if M mod 10 < min then min:=M mod 10;
     M:=M div 10;
 end;
end;
var
 X:longint; N,i,X_max, X_min:byte;
begin
 //Количество элементов в последовательности
 write('N='); readln(N);
 for i:=1 to N do
 begin
     //Элемент последовательности.
     write('X=');
     readln(X);
     if X>0 then //Если элемент положительный,
                 //то
     begin
         //вызов процедуры.
         max_min(X,X_max,X_min);
         //Печать результатов.
         writeln(' max=',X_max,' min=',X_min);
     end;
 end;
end.
```

4.4 Функции

Описание функции также состоит из заголовка и тела:

```
function имя_функции(формальные_параметры):тип;
 label
    описание_меток;
 const
    описание_констант;
 type
    описание_типов;
 var
    описание_переменных;
 begin
    //Тело функции.
 end;
```

Заголовок функции содержит: служебное слово `function`, любой допустимый в языке Free Pascal идентификатор - `имя_функции`; имена формальных параметров и их типы, разделенные точкой с запятой - `формальные_параметры`, тип возвращаемого функцией значения - `тип` (функции могут возвращать скалярные значения целого, вещественного, логического, символьного или ссылочного типа).

Примеры описания функций:

```
function fun_1 (x:real):real;
function fun_2(a, b:integer):real;
```

Тело функции состоит из раздела описаний[53] (константы, типы, переменные, процедуры и функции, используемые в процедуре) и операторов языка, реализующих ее алгоритм. *В теле функции всегда должен быть хотя бы один оператор, присваивающий значение имени функции*.

Например:

```
function fun_2(a, b:integer):real;
 begin
    fun_2:=(a+b)/2;
 end;
```

Обращение к функции осуществляется по имени с указанием списка фактических параметров, разделенных запятой:

```
имя_функции (список_фактических_параметров);
```

53 Раздел описаний в функции может отсутствовать, если в нем нет необходимости.

Например:

```
y:=fun_1(1.28);
z:=fun_1(1.28)/2+fun_2(3,8);
```

ЗАДАЧА 4.3. Вводится последовательность из N целых чисел, найти среднее арифметическое совершенных чисел и среднее геометрическое простых чисел последовательности.

Напомним, что целое число называется простым, если оно делится нацело только на само себя и единицу. Подробно алгоритм определения простого числа описан в задаче 3.14 (рис. 3.33). Кроме простых чисел, в этой задаче фигурируют совершенные числа. Число называется совершенным, если сумма всех делителей, меньших его самого, равна этому числу. Алгоритм, с помощью которого можно определить делители числа, подробно рассмотрен в задаче 3.13 (рис. 3.32).

Для решения поставленной задачи понадобятся две функции:

•`prostoe` – определяет, является ли число простым, аргумент функции целое число `N`, функция возвращает `true` (истина), если число простое и `false` (ложь) – в противном случае;

•`soversh` – определяет, является ли число совершенным; входной параметр целое число `N`, функция возвращает `true` (истина), если число простое и `false` (ложь) – в противном случае.

Фрагмент программы с комментариями:

```
//Функция, которая определяет простое число.
function prostoe(N:word):boolean;
 var i:word;
 begin
     prostoe:=true;
     for i:=2 to N div 2 do
         if N mod i = 0 then
         begin
             prostoe:=false;
             break;
         end;
 end;
//Функция, которая определяет
//совершенное число.
function soversh(N:word):boolean;
 var i:word; S:word;
```

```
 begin
     soversh:=false;
     S:=0;
     for i:=1 to N div 2 do
         if N mod i =0 then S:=S+i;
         if S=N then soversh:=true;
 end;
var X:word; K,kol_p,kol_s,i:byte; Sum,Pro:real;
begin   //Начало основной программы.
 //Ввод количества элементов.
 write('K='); readln(K);
 Sum:=0; //Переменная для накапливания суммы.
 Pro:=1; //Переменная для вычисления произвед.
 kol_p:=0;      //Счетчик простых чисел.
 kol_s:=0;      //Счетчик совершенных чисел.
 for i:=1 to K do
 begin
     //Ввод элемента последовательности.
     Writeln('X='); readln(X);
     //Если число простое,
     if prostoe(X) then
     begin
         //выполнить операцию умножения,
         Pro:=Pro*X;
         //увеличить счетчик простых чисел.
         kol_p:=kol_p+1;
     end;
     //Если число совершенное,
     if soversh(X) then
     begin
         //выполнить операцию умножения,
         Sum:=Sum+X;
         //увеличить счетчик совершенных чисел.
         kol_s:=kol_s+1;
     end;
 end;
 //Если были найдены совершенные числа,
 if kol_s<> 0 then
```

```
begin
    //вычислить среднее арифметическое.
    Sum:=Sum/kol_s;
    writeln('Среднее арифметическое
                совершенных чисел ', Sum:5:2);
end
else   //иначе вывести сообщение:
    writeln('Совершенных чисел
                    в последовательности нет.');
//Если были найдены простые числа,
if kol_p<>0 then
begin
    //вычислить среднее геометрическое.
    Pro:= exp(1/kol_p*ln(Pro));
    writeln('Среднее геометрическое
                    простых чисел ',Pro:5:2);
end
else//иначе вывести сообщение:
    writeln('Простых чисел
                    в последовательности нет');
end.
```

Задача 4.4. Вводится последовательность целых чисел. 0 — конец последовательности. Определить, содержит ли последовательность хотя бы одно число-палиндром.

Палиндром — это число, симметричное относительно своей середины. Например, 123454321, 678876 — палиндромы. Чтобы определить, является ли число палиндромом нужно сравнивать первую и последнюю цифры, затем вторую и предпоследнюю и так далее. Если хотя бы в одной паре цифры не совпадут, то число палиндромом не является.

Для решения поставленной задачи понадобятся две функции:

- `cifra_kol` — определяет количество цифр в числе (подробно алгоритм описан в задаче 3.16);
- `palindrom` — возвращает значение истина, если переданное в нее число является палиндромом.

Текст программы с комментариями:

```
//Функция для вычисления количества
//цифр в числе M.
```

```
function cifra_kol(M:longint):byte;
 begin
     cifra_kol:=1;
     while M div 10 > 0 do
     begin
         cifra_kol:=cifra_kol+1;
         M:=M div 10;
     end;
 end;
//Функция возвращает значение истина,
//если число M,
//состоящее из kol цифр палиндром,
//и значение ложь в противном случае.
function palindrom(M:longint;kol:byte):boolean;
 var i:byte; j:longint;
begin
 j:=1;
//Возведение числа 10 в степень kol-1
//(разрядность числа).
 for i:=1 to kol-1 do
     j:=j*10;
 palindrom:=true;   //Пусть число - палиндром.
 for i:=1 to kol div 2 do
 begin
//Выделение старшего разряда M div j
//(первая цифра).
//Выделение младшего разряда  M mod 10
//(последняя цифра).
//Если первая и последняя цифры не совпадают,
     if M div j <> M mod 10 then
     begin
         //то число не палиндром.
         palindrom:=false;
         break;          //выход из цикла.
     end;
     //Изменение числа
         //Удаление 1-й цифры числа
     M:=M-(M div j)*j;
```

```
            //Удаление последней цифры числа.
        M:=M div 10;
            /Уменьшение разрядности.
        j:=j div 100;      /
        end;
    end;
    //Основная программа.
    var X:longint; pr:boolean;
    begin
     //Ввод элемента последовательности.
     write('X=');readln(X);
    //Пусть в последовательности нет палиндромов.
     pr:=false;
     while X<>0 do //Пока не ноль,
     begin
        if palindrom(X,cifra_kol(X)) then
        begin
            pr:=true;  //Найдено число палиндром,
            break;     //досрочный выход из цикла.
        End;
    //Ввод следующего элемента последовательности.
        write('X=');readln(X);
     end;
     if pr then writeln('Последовательность
                         содержит число-палиндром.')
     else writeln('Последовательность не содержит
палиндромов.');
    end.
```

4.5 Решение задач с использованием подпрограмм

В этом разделе мы рассмотрим задачи с несложными алгоритмами, но больше внимания уделим их интерфейсу в среде Lazarus.

ЗАДАЧА 4.5. Создать программу, которая автоматизирует процесс перевода градусной меры угла в радианную и наоборот, в зависимости от выбора пользователя. То есть пользователь должен выбрать, как он будет вводить угол, в радианах или в градусах. Введет в радианах, ответ получит в градусах и, соответственно, введет в градусах, ответ получит в радианах.

С точки зрения математика задача не вызывает сложности:

• чтобы найти радианную меру какого-нибудь угла по данной градусной его мере, нужно умножить число градусов на $\frac{\pi}{180}$, число минут - на $\frac{\pi}{180 \cdot 60}$, число секунд - на $\frac{\pi}{180 \cdot 60 \cdot 60}$ и найденные произведения сложить;

• чтобы найти градусную меру угла по заданной радианной нужно умножить число радиан на $\frac{180}{\pi}$; если из полученной дроби выделить целую часть, то получим градусы; если из числа полученного умножением оставшейся дробной части на 60, выделить целую часть получим минуты; секунды вычисляются аналогично из дробной части минут.

Для перевода угла из градусной меры в радианную создадим функцию

```
function gradus_radian(gradus,minuta,secunda:byte):real;
```

в которую будем передавать целочисленные значения градусов, минут и секунд. Результат работы функции – вещественное число, величина угла в радианах.

Задачу перевода из радианной меры в градусную решим, создав процедуру

```
procedure radian_gradus(radian:real;
                        var gradus,minuta,secunda:byte);
```

у которой один входной параметр – радианная мера угла и три выходных – градусы, минуты и секунды.

Разработаем интерфейс будущей программы в среде Lazarus. Создадим новый проект, установим свойства формы так, как показано в табл. 4.1, и разместим на ней компоненты в соответствии с рис. 4.2.

Таблица 4.1. Свойства формы

Свойство	Значение	Описание свойства
Caption	Величина угла	Заголовок формы
Height	265	Высота формы
Width	325	Ширина формы
BorderIcons.BiMaximize	false	Кнопка развертывания окна недоступна

Свойство	Значение	Описание свойства
BorderStyle	bdDialog	Стиль рамки – диалоговое окно, не изменяет размеры
Position	poScreenCenter	Окно появится в центре экрана

С компонентами Edit, Label и Button мы уже хорошо знакомы.

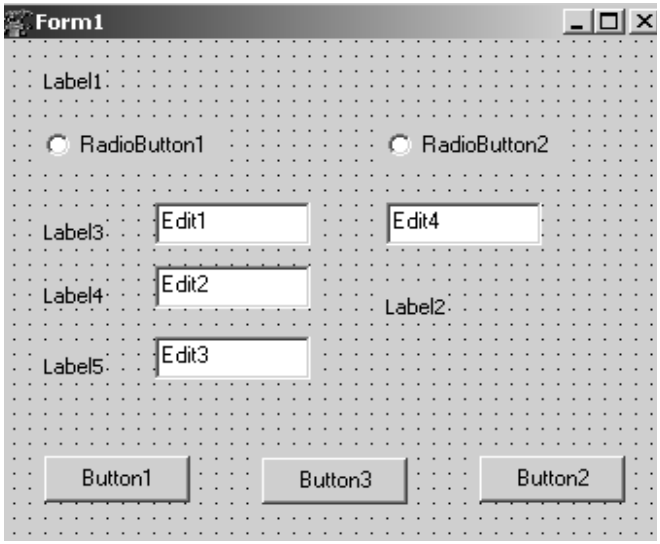

Рисунок 4.1: Настройка формы к задаче 4.5

Компонент RadioButton – это *переключатель*. Его используют для выбора одного из нескольких взаимоисключающих решений. Обычно на форму помещается, по меньшей мере, два таких компонента. Они могут иметь только два состояния, определяемых свойством Checked. Если у одного из компонентов это свойство истинно true, то во всех остальных ложно false. В данной задаче используется два компонента RadioButton1 и RadioButton2, предоставляя пользователю выбор: включен первый компонент – будет осуществлен перевод из градусной меры в радианную, включен второй – наоборот. Двойной щелчок по компоненту RadioButton1 приведет к созданию процедуры TForm1.RadioButton1Click обработки события «Щелчок мыши по кнопке переключателя». В тексте процедуры следует указать команды, которые будут выполняться, если пользователь включил или выключил компонент.

Нам уже известно, что свойства компонентов могут изменяться как в окне конструирования формы, так и непосредственно в программе. Если дважды щелкнуть по форме, вне размещенных на ней компонентов, то будет создана процедура TForm1.FormCreate обработки события открытия формы. На вкладке События инспектора

объектов это событие носит название `OnCreate`. В процедуре `TForm1.FormCreate` можно задать свойства всех компонентов на момент открытия формы.

Кнопки, расположенные на форме, несут следующую функциональную нагрузку:

•`Button1` запускает процесс перевода в зависимости от установок переключателей;

•`Button3` возвращает внешний вид формы в первоначальное состояние (до ввода и вывода данных);

•`Button2` – завершает процесс выполнения программы.

Текст программы с необходимыми комментариями приведен ниже. Результаты работы программы представлены на рис. 4.4 и рис. 4.5.

```
   unit Unit1;
   interface
   uses
   Windows, Messages, SysUtils, Variants, Classes,
Graphics, Controls, Forms, Dialogs, StdCtrls;
   type
   TForm1 = class(TForm)
   Button1: TButton;
   Label1: TLabel;
   RadioButton1: TRadioButton;
   RadioButton2: TRadioButton;
   Edit1: TEdit;
   Button2: TButton;
   Label2: TLabel;
   Edit2: TEdit;
   Edit3: TEdit;
   Label3: TLabel;
   Label4: TLabel;
   Label5: TLabel;
   Edit4: TEdit;
   Button3: TButton;
   procedure Button1Click(Sender: TObject);
   procedure RadioButton1Click(Sender: TObject);
   procedure RadioButton2Click(Sender: TObject);
   procedure FormCreate(Sender: TObject);
```

```
procedure Button2Click(Sender: TObject);
procedure Button3Click(Sender: TObject);
private
{ Private declarations }
public
{ Public declarations }
end;
var
Form1: TForm1;
implementation
{$R *.dfm}

//Щелчок по кнопке ВЫЧИСЛИТЬ.
procedure TForm1.Button1Click(Sender: TObject);
//Функция перевода данных из градусов
//в радианы.
function gradus_radian
            (gradus,minuta,secunda:byte):real;
 begin
    gradus_radian:=
           gradus*pi/180+minuta*pi/180/60+
                     secunda*pi/180/60/60;
 end;

 //Процедура перевода из радиан в градусы.
 procedure radian_gradus(radian:real;
            var gradus,minuta,secunda:byte);
 begin
    gradus:=trunc(radian*180/pi);
    minuta:= trunc((radian*180/pi-gradus)*60);
    secunda:=trunc(((radian*180/pi-
                  gradus)*60-minuta)*60);
 end;

var                      //Описание переменных.
grad,min,sec:byte;       //Градусная мера угла.
rad:real;                //Радианная мера угла.
```

```
//Контроль ввода.
kod_g,kod_m,kod_s,kod_r:integer;
begin
//Если первый переключатель вкл.
if RadioButton1.Checked then
begin
 Val(Edit1.Text,grad,kod_g);   //Ввод градусов.
 Val(Edit2.Text,min,kod_m);    //Ввод минут.
 Val(Edit3.Text,sec,kod_s);    //Ввод секунд.
 //Если ошибки при вводе не было, то
 if (kod_g=0) and (kod_m=0) and (kod_s=0) then
 begin
 //сделать видимым компонент Label2
 Label2.Visible:=true;
 //и вывести туда результат вычислений.
 //Вызов функции gradus_radian
 //перевода из градусов в радианы.
 Label2.Caption:='Величина угла ' +chr(13)+
 FloatToStrF(gradus_radian(grad,min,sec),
 ffFixed,8,6)+' радиан';
 end
 else
//Иначе выдать сообщение об ошибке при вводе.
 MessageDlg('Ошибка при вводе данных!',
                        MtWarning,[mbOk],0);
end;
//Если второй переключатель вкл.
if RadioButton2.Checked  then
begin
 //Ввод радианной меры угла.
 Val(Edit4.Text,rad,kod_r);
 //Если нет ошибок при вводе, то
 if (kod_r=0)then
 begin
 //сделать видимым компонент Label2
 Label2.Visible:=true;
//и вывести туда результат вычислений.
//Вызов процедуры перевода из радиан в градусы.
```

```
 radian_gradus(rad,grad,min,sec);
 Label2.Caption:='Величина угла' +chr(13)
            +IntToStr(grad)+' г '+IntToStr(min)
                  +' м '+IntToStr(sec)+' с';
 end
 else
//Иначе выдать сообщение об ошибке при вводе.
 MessageDlg('Ошибка при вводе данных!',
                             MtWarning,[mbOk],0);
end;
end;
//Щелчок по кнопке ВЫХОД.
procedure TForm1.Button2Click(Sender: TObject);
begin
 close;
end;
//Щелчок по кнопке ОЧИСТИТЬ.
procedure TForm1.Button3Click(Sender: TObject);
begin
 //Установка свойств компонентов
//в первоначальное состояние.
 Edit1.Text:='00';
 Edit2.Text:='00';
 Edit3.Text:='00';
 Edit4.Text:='00.000';
 Label1.Caption:='Введите значение';
 Label1.Font.Size:=10;
 Label3.Caption:='Градусы';
 Label4.Caption:='Минуты';
 Label5.Caption:='Секунды';
 Button1.Caption:='ВЫЧИСЛИТЬ';
 Button2.Caption:='ВЫХОД';
 Button3.Caption:='ОЧИСТИТЬ';
 Edit4.Enabled:=false;
 Label2.Visible:=false;
 RadioButton1.Checked:=true;
 RadioButton2.Checked:=false;
end;
```

```
//Обработка события открытие формы.
procedure TForm1.FormCreate(Sender: TObject);
begin
 //Установка свойств компонентов.
 Edit1.Text:='00';
 Edit2.Text:='00';
 Edit3.Text:='00';
 Edit4.Text:='00.000';
 Label1.Caption:='Введите значение';
 Label1.Font.Size:=10;
 Label3.Caption:='Градусы';
 Label4.Caption:='Минуты';
 Label5.Caption:='Секунды';
 Button1.Caption:='ВЫЧИСЛИТЬ';
 Button2.Caption:='ВЫХОД';
 Button3.Caption:='ОЧИСТИТЬ';
 Edit4.Enabled:=false;
 Label2.Visible:=false;
 RadioButton1.Checked:=true;
 RadioButton2.Checked:=false;
end;
//Обработка события щелчок
//по переключателю RadioButton1.
procedure Tform1.RadioButton1Click(
                              Sender: TObject);
begin
 if RadioButton1.Checked then
 begin
     Edit1.Enabled:=true;
     Edit2.Enabled:=true;
     Edit3.Enabled:=true;
     Label5.Enabled:=true;
     Label3.Enabled:=true;
     Label4.Enabled:=true;
     Edit4.Enabled:=false;
 end;
end;
```

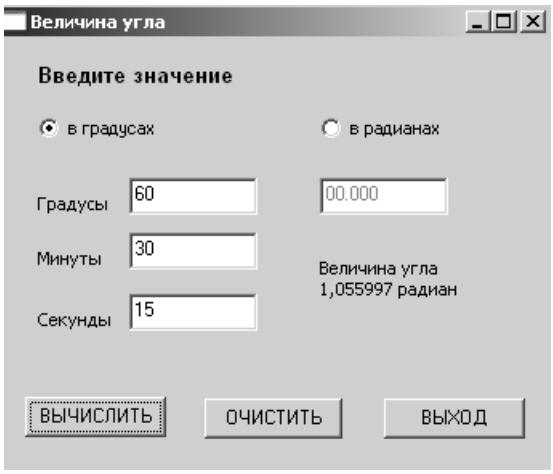

Рисунок 4.2: Перевод значений из градусной меры в радианную

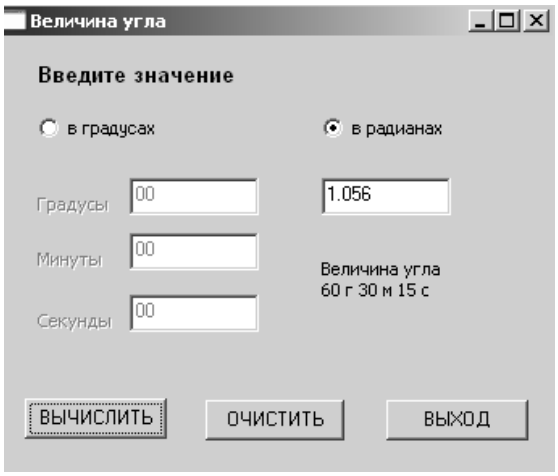

Рисунок 4.3: Перевод значений из радианной меры в градусную

```
//Обработка события щелчок
//по переключателю RadioButton2.
procedure Tform1.RadioButton2Click(
                              Sender: TObject);
 begin
 if RadioButton2.Checked then
 begin
     Edit4.Enabled:=true;
     Button1.Enabled:=true;
     Edit1.Enabled:=false;
     Edit2.Enabled:=false;
     Edit3.Enabled:=false;
     Label3.Enabled:=false;
     Label4.Enabled:=false;
     Label5.Enabled:=false;
 end;
end;
end.
```

ЗАДАЧА 4.6. Создать программу для решения уравнений:
•линейное $ax+b=0$;
•квадратное $ax^2+bx+c=0$;
•кубическое $ax^3+bx^2+cx+d=0$.

Решение линейного уравнения тривиально $x=-b/a$, алгоритмы решения квадратного и кубического уравнений подробно рассмотрены в задачах 3.4 и 3.5.

Создадим новый проект (рис. 4.4). Свойства формы настроим по табл. 4.1, за исключением свойства `Caption`, которому присвоим значение **Решение уравнения**.

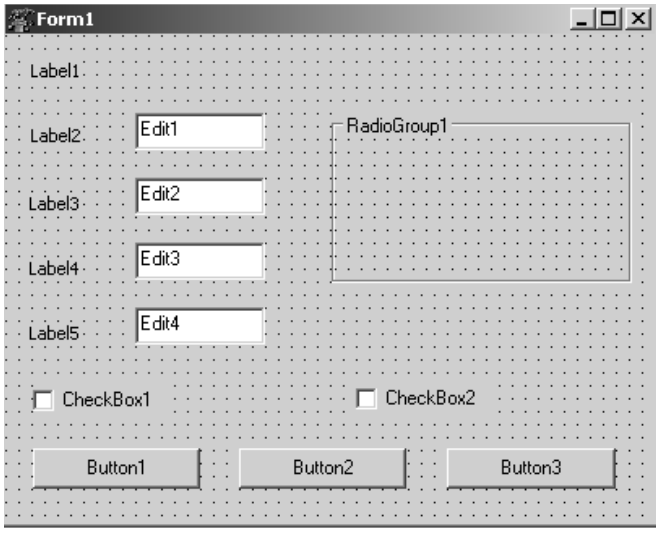

Рисунок 4.4: Процесс создания формы к задаче 4.6

Обратите внимание, что на форме появились не знакомые нам компоненты. Это `CheckBox` – флажок и `RadioGroup` – группа переключателей.

Компонент *флажок* `CheckBox` используется для того, чтобы пользователь мог включить или выключить значение какого-либо параметра. Установлен флажок или нет, определяет свойство `Checked` (`true`, `false`). В составе диалогового окна может быть несколько таких компонентов, причем состояние любого из них не зависит от состояния остальных.

Компонент *группа переключателей* `RadioGroup` объединяет в себе несколько переключателей. Каждый размещенный в нем переключатель помещается в специальный список `Items` и доступен по номеру, установленному в свойстве `ItemIndex`. После размещения на форме компонент пуст. Чтобы создать в нем хотя бы один переключатель, нужно выделить его, обратиться к инспектору объектов и выбрать свойство `Items` – *редактор списка*. Строки, набранные в редакторе, используются как поясняющие надписи справа от переключателей, а их количество определяет количество переключателей в группе. В нашем случае окно редактора списка будет иметь вид, как на рис. 4.5.

После создания компонента группа переключателей его свойство номер переключателя `ItemIndex` по умолчанию равно –1. Это означает, что ни один компонент в группе не установлен.

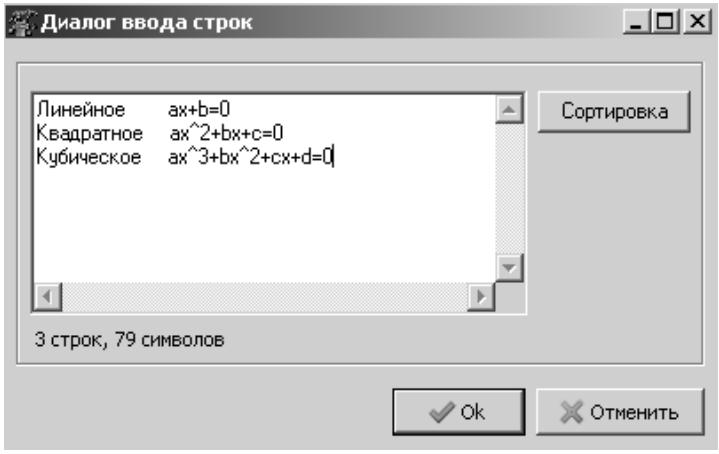

Рисунок 4.5: Окно редактора списка

Чтобы в момент появления компонента на экране один из переключателей был отмечен, нужно либо на этапе конструирования формы, либо программно присвоить свойству `ItemIndex` номер одного из элементов списка, учитывая, что нумерация начинается с нуля.

С остальными компонентами, размещенными на форме, пользователь уже знаком. На рис. 4.6 - 4.8 видно, как работает программа. Пользователю предоставляется возможность выбрать вид решаемого уравнения, ввести его коэффициенты и указать, какие решения — действительные или комплексные (если это возможно) — он хотел бы получить. Далее приведен текст программного модуля с комментариями:

```
unit Unit1;
interface
uses
Windows, Messages, SysUtils, Variants, Classes,
Graphics, Controls, Forms, Dialogs, StdCtrls,
                                        ExtCtrls;
type
TForm1 = class(TForm)
Label1: TLabel;
Label2: TLabel;
Label3: TLabel;
Label4: TLabel;
Label5: TLabel;
Edit1: TEdit;
Edit2: TEdit;
Edit3: TEdit;
Edit4: TEdit;
CheckBox1: TCheckBox;
CheckBox2: TCheckBox;
Button1: TButton;
```

```
Button2: TButton;
RadioGroup1: TRadioGroup;
Button3: TButton;
procedure FormCreate(Sender: TObject);
procedure Button2Click(Sender: TObject);
procedure Button3Click(Sender: TObject);
procedure Button1Click(Sender: TObject);
procedure RadioGroup1Click(Sender: TObject);
private
{ Private declarations }
public
{ Public declarations }
end;
var
Form1: TForm1;
implementation
{$R *.dfm}
//Щелчок по кнопке НАЙТИ КОРНИ.
procedure TForm1.Button1Click(Sender: TObject);
 //Решение линейного уравнения.
 //Параметры процедуры korni_1:
 //a,b — вещественные переменные,
 //коэффициенты уравнения;
 //x_ - строковая переменная,
 //решение уравнения в символьном виде.
 procedure korni_1(a,b:real;var x_:string);
     //x - решение уравнения в численном виде
     var x:real;
 begin
     x:=-b/a;
     x_:=FloatToStrF(x,ffFixed,5,2);
 end;
 //Решение квадратного уравнения.
 //Параметры процедуры korni_2:
 //a,b,c — вещественные переменные,
 //коэффициенты уравнения;
 //x1_,x2_ - строковые переменные,
 //решение уравнения в символьном виде;
```

```
//pr — целочисленная переменная,
//принимает значение, равное 1, если
//уравнение имеет действительные корни,
//и значение, равное 2, если корни мнимые.
procedure korni_2(a,b,c:real;
           var x1_,x2_:string;var pr:byte);
//x1,x2 - решение уравнения в численном виде,
//d — дискриминант.
    var d,x1,x2:real;
begin
    d:=b*b-4*a*c;
    if d<0 then
    begin
        x1:=-b/(2*a);
        x2:=sqrt(abs(d))/(2*a);
        x1_:= FloatToStrF(x1,ffFixed,5,2)+
        '+i*'+FloatToStrF(x2,ffFixed,5,2);
        x2_:= FloatToStrF(x1,ffFixed,5,2)+
        '-i*'+FloatToStrF(x2,ffFixed,5,2);
        pr:=2;     //Комплексные корни.
    end
    else
    begin
        x1:=(-b+sqrt(d))/2/a;
        x2:=(-b-sqrt(d))/(2*a);
        x1_:= FloatToStrF(x1,ffFixed,5,2);
        x2_:= FloatToStrF(x2,ffFixed,5,2);
        pr:=1;     //Действительные корни.
    end
end;
//Решение кубического уравнения.
//Параметры процедуры korni_3:
//a,b,c,d — вещественные переменные,
//коэффициенты уравнения;
//x1_,x2_,x3_ - строковые переменные,
//решение уравнения в символьном виде;
//pr — целочисленная переменная,
//принимает значение, равное 1, если
```

```
//уравнение имеет действительные корни,
//и значение, равное 2, если в уравнении
//один корень действительный и два мнимых.
procedure korni_3(a,b,c,d:real;
    var x1_,x2_,x3_:string;var pr:byte);
//x1,x2,x3 - решения в численном виде.
var r,s,t,p,q,ro,fi,u,v,x1,x2,x3,h,g:real;
begin
    r:=b/a;
    s:=c/a;
    t:=d/a;p:=(3*s-r*r)/3;
    q:=2*r*r*r/27-r*s/3+t;
    d:=(p/3)*sqr(p/3)+sqr(q/2);
    if d<0 then
    begin
        ro:=sqrt(-p*p*p/27);
        fi:=-q/(2*ro);
        fi:=pi/2-arctan(fi/sqrt(1-fi*fi));
//Вычисление действительных корней уравнения.
x1:=2*exp(1/3*ln(ro))*cos(fi/3)-r/3;
x2:=2*exp(1/3*ln(ro))*cos(fi/3+2*pi/3)-r/3;
x3:=2*exp(1/3*ln(ro))*cos(fi/3+4*pi/3)-r/3;
        x1_:=FloatToStrF(x1,ffFixed,5,2);
        x2_:=FloatToStrF(x2,ffFixed,5,2);
        x3_:=FloatToStrF(x3,ffFixed,5,2);
        pr:=1;  //Действительные корни.
    end
    else
    begin
        if -q/2+sqrt(d)>0 then
            u:=exp(1/3*ln(-q/2+sqrt(d)))
        else
            if -q/2+sqrt(d)<0 then
            u:=-exp(1/3*ln(abs(-q/2+sqrt(d))))
            else u:=0;
        if -q/2-sqrt(d)>0 then
            v:=exp(1/3*ln(-q/2-sqrt(d)))
        else
```

```
            if -q/2-sqrt(d)<0 then
            v:=-exp(1/3*ln(abs(-q/2-sqrt(d))))
            else v:=0;
        //Вычисление действительного корня.
        x1:=u+v-r/3;
        //Вычисление комплексных корней.
        h:=-(u+v)/2-r/3;
        g:=(u-v)/2*sqrt(3);
        x1_:=FloatToStrF(x1,ffFixed,5,2);
        x2_:=FloatToStrF(h,ffFixed,5,2)+
        '+i*'+FloatToStrF(g,ffFixed,5,2);
        x3_:=FloatToStrF(h,ffFixed,5,2)+
        '-i*'+FloatToStrF(g,ffFixed,5,2);
        pr:=2;  //1 действительный и
                //2 комплексных корня.
    end
 end;
//Обработка события
//щелчок по кнопке НАЙТИ КОРНИ.
var
    a_,b_,c_,d_:real;
    kod_a,kod_b,kod_c,kod_d:integer;
    _x,_x1,_x2,_x3:string; _pr:byte;
begin
 case RadioGroup1.ItemIndex of
 0:  //Пользователь выбрал 1-й переключатель.
 Begin  //Решение линейного уравнения.
    //Ввод исходных данных.
    val(Edit1.Text,a_,kod_a);
    val(Edit2.Text,b_,kod_b);
    //Ввод прошел успешно.
    if (kod_a=0) and (kod_b=0) then
    begin
        //1-й коэффициент не ноль.
        if a_<>0 then
        begin
            //Решение линейного уравнения.
            korni_1(a_,b_,_x);
```

```
            //Вывод найденного значения.
MessageDlg('Решение линейного уравнения x='+
                    _x,mtInformation,[mbOk],0);
        end
        //1-й коэффициент равен нулю,
        else
            //вывод соответствующего сообщения.
            MessageDlg('Нет корней!',
                    mtInformation,[mbOk],0);
    end
    else //Некорректный ввод данных.
        MessageDlg('Ошибка при вводе!',
                        mtInformation,[mbOk],0);
end;
1:  //Пользователь выбрал 2-й переключатель.
begin
    //Ввод исходных данных.
    val(Edit1.Text,a_,kod_a);
    val(Edit2.Text,b_,kod_b);
    val(Edit3.Text,c_,kod_c);
    //Ввод прошел успешно.
    if (kod_a=0) and (kod_b=0) and (kod_c=0)
    then begin
        //Первый коэффициент не ноль.
        if a_<>0 then
        begin
        //Решение квадратного уравнения.
            korni_2(a_,b_,c_,_x1,_x2,_pr);
//В переменной _pr содержится информация
//о типе корней:
//1 - действительные, 2 - комплексные.
//Оба флажка не установлены.
if (CheckBox1.Checked=false) and
         (CheckBox2.Checked=false) then
    MessageDlg('Выберите тип решения',
                mtInformation,[mbOk],0);
//Оба флажка установлены.
if CheckBox1.Checked and
```

```
                    CheckBox2.Checked then
    MessageDlg('Решение уравнения'+
                        chr(13)+' X1='+_x1+
    ' X2='+_x2,mtInformation,[mbOk],0);
//Установлен первый флажок,
//корни действительные.
if CheckBox1.Checked and
    (CheckBox2.Checked=false) and (_pr=1)
    then
        MessageDlg('Действительные корни'+
            chr(13)+' X1='+_x1+' X2='+_x2,
                mtInformation,[mbOk],0);
//Установлен второй флажок,
//корни действительные.
if (CheckBox1.Checked=false) and
    CheckBox2.Checked and (_pr=1) then
    MessageDlg('Уравнение имеет только
    действительные корни.',
                mtInformation,[mbOk],0);
//Установлен первый флажок, корни комплексные.
if CheckBox1.Checked and
    (CheckBox2.Checked=false) and (_pr=2) then
    MessageDlg('Действительных корней нет',
                        mtInformation,[mbOk],0);
//Установлен второй флажок, корни комплексные.
if (CheckBox1.Checked=false) and
    CheckBox2.Checked and (_pr=2) then
    MessageDlg('Комплексные корни уравнения'+
                chr(13)+' X1='+_x1+' X2='+_x2,
                    mtInformation,[mbOk],0);
end
else   //Первый коэффициент равен нулю.
MessageDlg('Первый коэффициент не равен 0!',
                        mtInformation,[mbOk],0);
end                 //Некорректный ввод данных.
else

MessageDlg('Ошибка при вводе коэффициентов!',
```

```
                        mtInformation,[mbOk],0);
end;
2:  //Пользователь выбрал 3-й переключатель.
Begin
    //Ввод исходных данных.
    val(Edit1.Text,a_,kod_a);
    val(Edit2.Text,b_,kod_b);
    val(Edit3.Text,c_,kod_c);
    val(Edit4.Text,d_,kod_d);
    //Ввод прошел успешно.
    if (kod_a=0) and (kod_b=0) and (kod_c=0)
    and (kod_d=0)then
    begin
    //Первый коэффициент не ноль.
    if a_<>0 then
        begin
//Решение кубического уравнения.
//В переменной _pr содержится информация
//о типе корней:
//1 - действительные,
//2 - один действительный и два комплексных.
korni_3(a_,b_,c_,d_,_x1,_x2,_x3,_pr);
//Оба флажка не установлены.
if (CheckBox1.Checked=false) and
(CheckBox2.Checked=false) then
MessageDlg('Выберите тип решения',
                  mtInformation,[mbOk],0);
//Оба флажка установлены.
if CheckBox1.Checked
    and CheckBox2.Checked then
    MessageDlg('Корни кубического уравнения'+
chr(13)+' X1='+_x1+    ' X2='+_x2+' X3='+_x3,
                  mtInformation,[mbOk],0);
    //Установлен первый флажок,
    //корни действительные.
    if CheckBox1.Checked and
    (CheckBox2.Checked=false) and (_pr=1) then
MessageDlg('Действительные корни уравнения'+
```

```
      chr(13)+' X1='+_x1+' X2='+_x2+' X3='+_x3,
                    mtInformation,[mbOk],0);
  //Установлен первый флажок, корни комплексные.
      if CheckBox1.Checked and
      (CheckBox2.Checked=false) and (_pr=2) then
  MessageDlg('Действительные корни уравнения'+
  chr(13)+' X1='+_x1,mtInformation,[mbOk],0);
  //Установлен второй флажок, корни комплексные.
      if (CheckBox1.Checked=false) and
          CheckBox2.Checked and (_pr=2) then
      MessageDlg('Комплексные корни уравнения'+
                 chr(13)+' X1='+_x2+' X2='+_x3,
                    mtInformation,[mbOk],0);
      //Установлен второй флажок,
      //корни действительные.
      if (CheckBox1.Checked=false) and
      CheckBox2.Checked and (_pr=1) then
      MessageDlg('Уравнение имеет только
                    действительные корни.',
                       mtInformation,[mbOk],0);
      end
      else
  MessageDlg('Первый коэффициент не равен 0!',
                       mtInformation,[mbOk],0);
      end
      else           //Некорректный ввод данных.
      MessageDlg('Ошибка при вводе !',
                       mtInformation,[mbOk],0);
  end;
  end;
  end;
 //Щелчок по кнопке ОЧИСТИТЬ.
 procedure TForm1.Button2Click(Sender: TObject);
 begin
  Label1.Caption:='Введите коэффициенты';
  Label2.Caption:='a=';
  Label3.Caption:='b=';
  Label4.Caption:='c=';
```

```
 Label5.Caption:='d=';
 Edit1.Text:='0.00';
 Edit2.Text:='0.00';
 Edit3.Text:='0.00';
 Edit4.Text:='0.00';
 Button1.Caption:='НАЙТИ КОРНИ';
 Button2.Caption:='ОЧИСТИТЬ';
 Button3.Caption:='ВЫХОД';
 CheckBox1.Caption:='Действительные корни';
 CheckBox2.Caption:='Комплексные корни';
 CheckBox1.Checked:=true;
 Label4.Enabled:=false;
 Label5.Enabled:=false;
 Edit3.Enabled:=false;
 Edit4.Enabled:=false;
 CheckBox2.Enabled:=false;
 RadioGroup1.ItemIndex:=0;
end;
//Щелчок по кнопке ВЫХОД.
procedure TForm1.Button3Click(Sender: TObject);
begin
 close;
end;
//Событие открытие формы.
procedure TForm1.FormCreate(Sender: TObject);
begin
 Label1.Caption:='Введите коэффициенты';
 Label2.Caption:='a=';
 Label3.Caption:='b=';
 Label4.Caption:='c=';
 Label5.Caption:='d=';
 Edit1.Text:='0.00';
 Edit2.Text:='0.00';
 Edit3.Text:='0.00';
 Edit4.Text:='0.00';
 Button1.Caption:='НАЙТИ КОРНИ';
 Button2.Caption:='ОЧИСТИТЬ';
 Button3.Caption:='ВЫХОД';
```

```
 CheckBox1.Caption:='Действительные корни';
 CheckBox2.Caption:='Комплексные корни';
 CheckBox1.Checked:=true;
 Label4.Enabled:=false;
 Label5.Enabled:=false;
 Edit3.Enabled:=false;
 Edit4.Enabled:=false;
 CheckBox2.Enabled:=false;
 RadioGroup1.ItemIndex:=0;
end;
//Выбор переключателя из группы.
procedure Tform1.RadioGroup1Click(
                              Sender: TObject);
begin
 case RadioGroup1.ItemIndex of
 0:          //Выбран первый из списка.
 begin
     Label2.Enabled:=true;
     Label3.Enabled:=true;
     Edit1.Enabled:=true;
     Edit2.Enabled:=true;
     Label2.Caption:='a=';
     Label3.Caption:='b=';
     Edit1.Text:='0.00';
     Edit2.Text:='0.00';
     Label4.Enabled:=false;
     Label5.Enabled:=false;
     Edit3.Enabled:=false;
     Edit4.Enabled:=false;
     CheckBox2.Enabled:=false;
 end;
 1:          //Выбран второй из списка.
 begin
     Label2.Enabled:=true;
     Label3.Enabled:=true;
     Label4.Enabled:=true;
     Edit1.Enabled:=true;
     Edit2.Enabled:=true;
```

```
        Edit3.Enabled:=true;
        Label2.Caption:='a=';
        Label3.Caption:='b=';
        Label4.Caption:='c=';
        Edit1.Text:='0.00';
        Edit2.Text:='0.00';
        Edit3.Text:='0.00';
        Label5.Enabled:=false;
        Edit4.Enabled:=false;
        CheckBox2.Enabled:=true;
    end;
    2:          //Выбран третий из списка.
    begin
        Label2.Enabled:=true;
        Label3.Enabled:=true;
        Label4.Enabled:=true;
        Label5.Enabled:=true;
        Edit1.Enabled:=true;
        Edit2.Enabled:=true;
        Edit3.Enabled:=true;
        Edit4.Enabled:=true;
        Label2.Caption:='a=';
        Label3.Caption:='b=';
        Label4.Caption:='c=';
        Label4.Caption:='d=';
        Edit1.Text:='0.00';
        Edit2.Text:='0.00';
        Edit3.Text:='0.00';
        Edit4.Text:='0.00';
        CheckBox2.Enabled:=true;
    end;
end;
end;end.
```

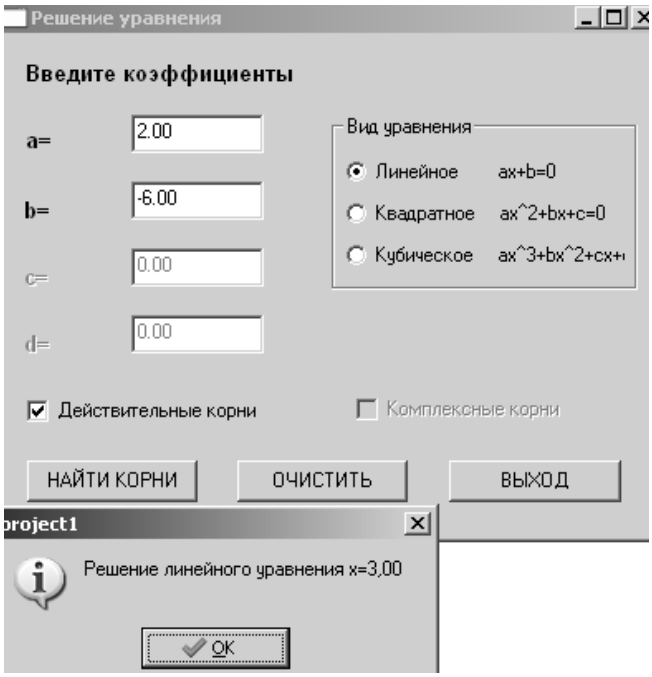

Рисунок 4.6: Решение линейного уравнения

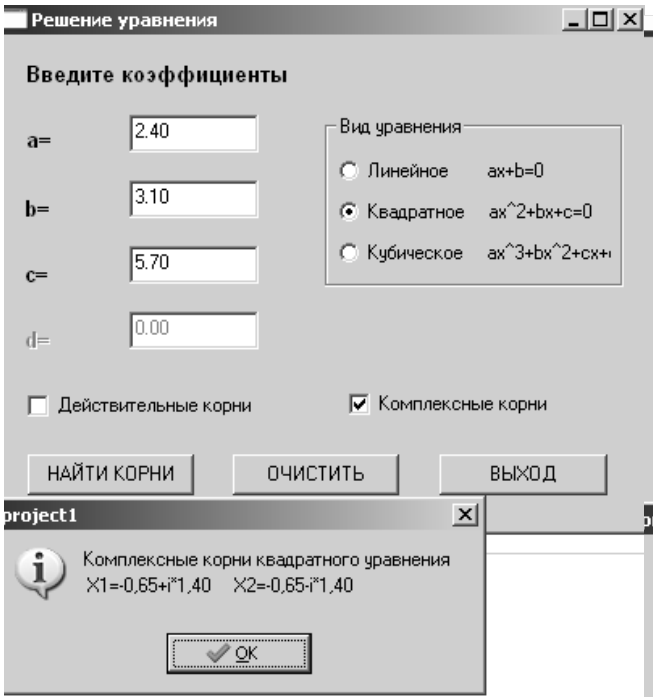

Рисунок 4.7: Решение квадратного уравнения

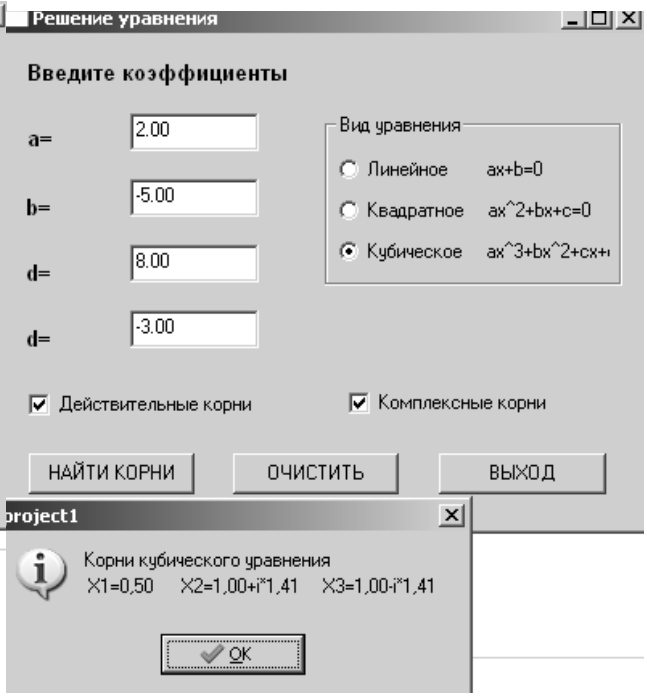

Рисунок 4.8: Решение кубического уравнения

4.6 Рекурсивные функции

Под *рекурсией* в программировании понимают подпрограмму, которая вызывает сама себя. Рекурсивные функции чаще всего используют для компактной реализации рекурсивных алгоритмов. Классиче-

скими рекурсивными алгоритмами могут быть возведение числа в целую положительную степень, вычисление факториала. С другой стороны, любой рекурсивный алгоритм можно реализовать без применения рекурсий. Достоинством рекурсии является компактная запись, а недостатком расход памяти на повторные вызовы функций и передачу параметров, кроме того, существует опасность переполнения памяти.

В рекурсивной функции необходимо обязательно предусмотреть завершение рекурсивного вызова. Иначе функция никогда не завершит свою работу.

Рассмотрим применение рекурсии на примерах.

ЗАДАЧА 4.7. Вычислить факториал числа *n*.

Вычисление факториала подробно рассмотрено в задаче 3.10 (рис. 3.29). Для решения этой задачи с применением рекурсии создадим функцию `factorial`, алгоритм которой представлен на рис. 4.9.

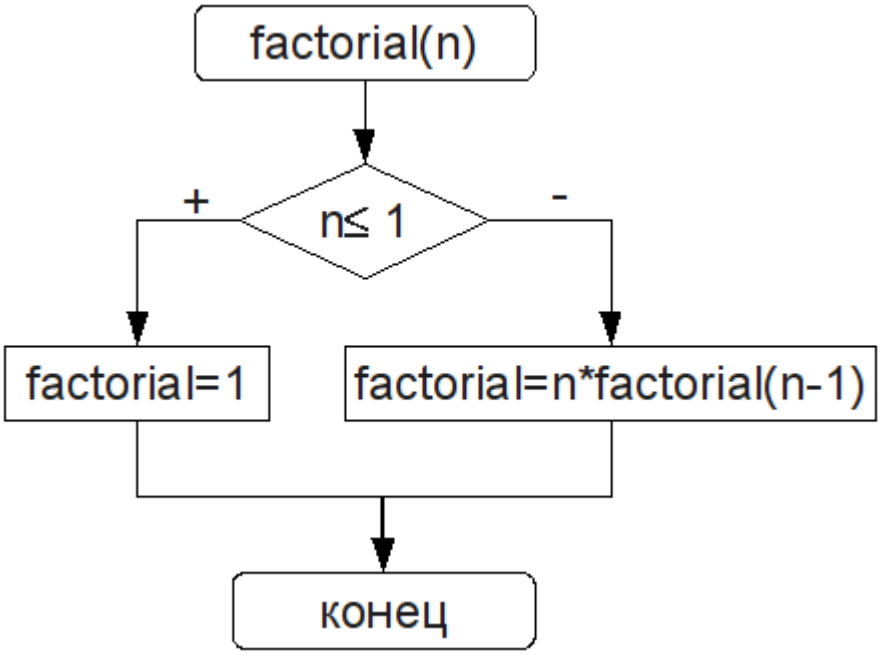

Рисунок 4.9: Алгоритм вычисления факториала

Текст подпрограммы с применением рекурсии:

```
function factorial(n:word):longint;
begin
 if n<=1 then   factorial:=1
 else   factorial:=n*factorial(n-1)
end;
var i:integer;
begin
```

```
 write('i=');
 read(i);
 write(i,'!=',factorial(i));
end.
```

ЗАДАЧА 4.8. Вычислить *n*-ю степень числа *a* (*n* – целое число)

Результатом возведения числа *a* в целую степень *n* является умножение этого числа на себя *n* раз. Но это утверждение верно только для положительных значений *n*. Если *n* принимает отрицательные значения, то $a^{-n}=\frac{1}{a^n}$. В случае если *n*=0, то $a^0=1$.

Для решения задачи создадим рекурсивную функцию `stepen`, алгоритм которой представлен на рис. 4.10.

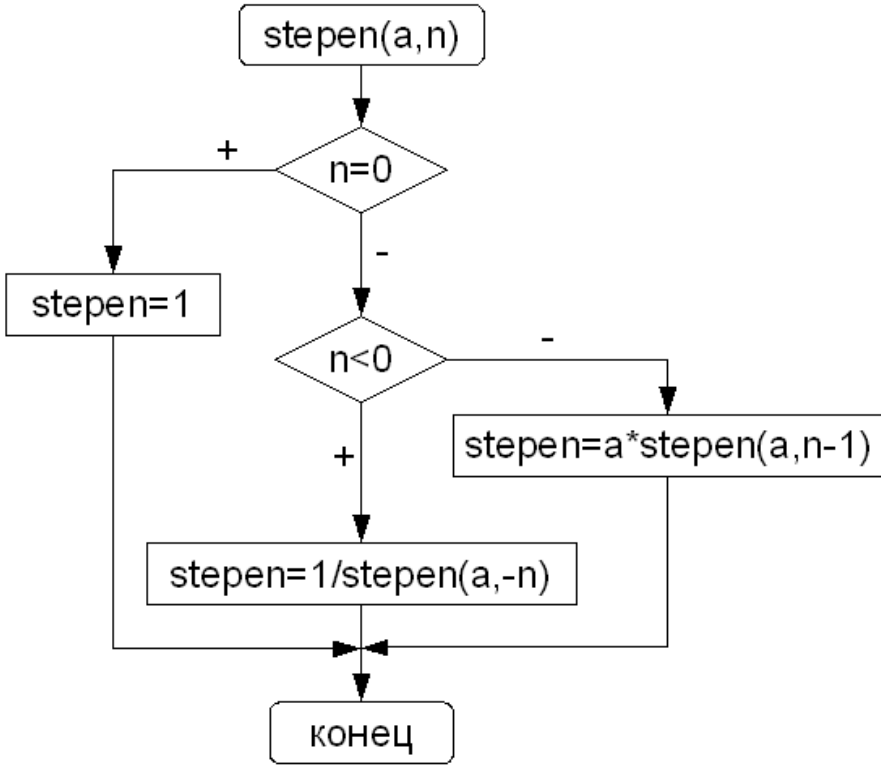

Рисунок 4.10: Рекурсивный алгоритм вычисления степени числа

Фрагмент программы с применением рекурсии:

```
function stepen(a:real;n:word):real;
begin
 if n=0 then stepen:=1
 else
     if n<0 then
         stepen:=1/stepen(a,-n)
```

```
    else
        stepen:=a*stepen(a,n-1);
end;
var
 x:real;k:word;
begin
 writeln('x='); readln(x);
 writeln('k='); readln(k);
 writeln(x:5:2,'^',k,'=',stepen(x,k):5:2);
end.
```

ЗАДАЧА 4.9. Вычислить *n*-е число Фибоначчи.

Если нулевой элемент последовательности равен нулю, первый – единице, а каждый последующий представляет собой сумму двух предыдущих, то это последовательность чисел Фибоначчи (0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...).

Алгоритм рекурсивной функции `fibonachi` изображен на рис. 4.11.

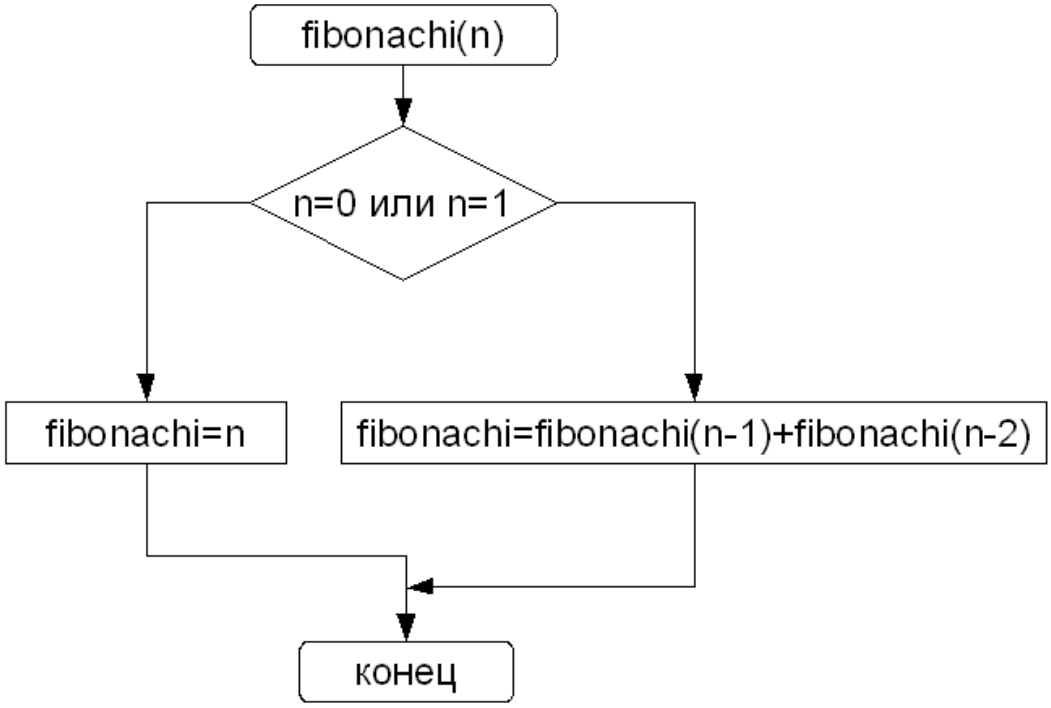

Рисунок 4.11: Рекурсивный алгоритм вычисления числа Фибоначчи

Текст подпрограммы:

```
function fibonachi(n:word):word;
begin
 if (n=0)or(n=1) then  fibonachi:=n
```

```
 else fibonachi:=fibonachi(n-1)+fibonachi(n-2);
end;
var x:word;
begin
 write('Введите номер числа Фибоначчи x=');
 readln(x);
 writeln(x,' -е число Фибоначчи = ',
                                  fibonachi(x));
end.
```

4.7 Особенности работы с подпрограммами

Рассмотрим еще несколько способов *передачи параметров* в подпрограмму.

4.7.1 Параметры-константы

Параметр-константа указывается в заголовке подпрограммы подобно параметру-значению, но перед его именем записывается зарезервированное слово `const`, действие которого распространяется до ближайшей точки с запятой. Параметр-константа передается в подпрограмму как параметр-переменная, но компилятор блокирует любые присваивания параметру-константе нового значения в теле подпрограммы.

В языке Free Pascal можно использовать параметры-переменные и параметры-константы без указания типа, и тогда фактический параметр может быть переменной любого типа, а ответственность за правильность использования того или иного параметра возлагается на программиста.

4.7.2 Процедурные типы

Процедурные типы были разработаны как средство передачи функций и процедур в качестве фактических параметров обращения к другим процедурам и функциям. Для объявления процедурного типа используется заголовок подпрограммы, в котором пропущено ее имя. Существует два процедурных типа: *тип-процедура* и *тип-функция*:

```
type
тип_процедура=procedure (формальные_параметры);
тип_функция=function (формальные_параметры):тип;
```

Кроме того, в программе могут быть объявлены переменные процедурного типа:

```
var
 имя_переменной_1: тип_процедура;
 имя_переменной_2: тип_функция;
```

Например:

```
type
 Fun1=function (x,y:real):real;
 Fun2=function:string;
 Proc1=procedure (x,y:real; var c,z:real);
 Proc2=procedure ();
Var
 F1,f2: Fun1;
 p1,p2: Proc1;
 mass:array [1..5] of Proc2;
```

Обращаться к переменным процедурного типа следует по их адресу[54]:

```
@имя_переменной
```

Рассмотрим механизм передачи подпрограмм в качестве параметра на примере следующей задачи.

ЗАДАЧА 4.10. Создать программу, которая выводит на экран таблицу значений функций $f(x)$ и $g(x)$.

Вычисление и вывод значений осуществляются с помощью функции `VivodFunc`. Ее параметры:

- `a` и `b` — границы интервала изменения аргумента функции;
- `n` — количество точек, на которые будет разбит интервал (`a`, `b`);
- `ff` – имя функции.

```
type    {Описание процедурного типа func.}
 func=function(x:real):real;
function VivodFunc (a,b:real;
                    N:word;ff:func):integer;
 var
     x,y,hx:real;
     f:text;
 begin
     {Шаг изменения переменной x.}
     hx:=(b-a)/N;
     x:=a;
```

54 Операция взятия адреса (п. 2.4.4)

```
        while(x<=b) do
        begin
            y:=ff(x);
            writeln('x=',x:5:2,' y=',y:7:2);
            x:=x+hx;
        end;
    end;
   {Определение функций f и g с описателем far.}
   function f(x:real):real;far;
    begin
        f:=exp(sin(x))*cos(x);
    end;
   function g(x:real):real;far;
    begin
        g:=exp(cos(x))*sin(x);
    end;
   begin
   {Вызов Vivodfunc с функцией f в качестве пара-
метра.}
        VivodFunc(0,1,7,@f);
        writeln;
   {Вызов Vivodfunc с функцией g в качестве пара-
метра.}
        VivodFunc(0,2,8,@g);
   end.
```

В языке Free Pascal есть возможность использования в качестве формальных параметров *массивов функций*.

Модифицируем задачу 4.10. Программа, представленная далее, выводит на экран таблицу значений нескольких функций с помощью функции `VivodFunc`. Здесь в роли параметров функции выступают:

- интервал (`a`, `b`);
- количество узлов `n` интервал (`a`, `b`);
- массив функций `ff`, в которых необходимо вычислить значения;
- количество функций `m` в массиве `ff`.

```
type
 func=function(x:real):real;
{В процедуру VivodFunc передается}
{входной параметр ff -
```

```
                        открытый массив функций[55].}
function VivodFunc(a,b:real;N:word;
            ff:array of func; m:word):integer;
 var
     x,y,hx:real;
     i:integer;
 begin
     hx:=(b-a)/N;
     {Цикл по всем функциям.}
     for i:=0 to m-1 do
     begin
         x:=a;
         while(x<=b) do
         begin
 {Вычисление значения i-й функции в точке x.}
             y:=ff[i](x);
             writeln('x=',x:5:2,' y=',y:7:2);
             x:=x+hx;
         end;
         writeln;
     end;
 end;
function f(x:real):real;far;
 begin
     f:=exp(sin(x))*cos(x);
 end;
function g(x:real):real;far;
 begin
     g:=exp(cos(x))*sin(x);
 end;
{Описание массива восьми функций fff.}
var fff:array[1..8] of func;
 begin
     {Запись реальных функций в массив fff.}
     fff[1]:=@f;  fff[2]:=@g;
     VivodFunc(0,1,7,fff,2);{Вызов процедуры.}
 end.
```

55 Подробнее о передаче массива в подпрограмму см. п. 5.10

Несколько подпрограмм можно объединять в модуль и затем использовать в других программах. Рассмотрим процесс создания модуля более подробно.

4.8 Разработка модулей

Модуль – это автономная программная единица, включающая в себя различные компоненты: константы, переменные, типы, процедуры и функции. Мы использовали модули Lazarus при разработке визуальных приложений. Теперь рассмотрим, как создавать личный модуль.

Модуль имеет следующую структуру:

```
UNIT имя модуля;
INTERFACE
 интерфейсная часть
IMPLEMENTATION
 исполняемая часть
BEGIN
 инициирующая часть
END.
```

Заголовок модуля состоит из служебного слова `UNIT` и следующего за ним имени. Причем имя модуля должно совпадать с именем файла, в котором он хранится. Модуль `EDIT` должен храниться в файле `edit.pas`.

Интерфейсная часть начинается служебным словом `INTERFACE`, за которым находятся объявления всех глобальных объектов модуля: типов, констант, переменных и подпрограмм. Эти объекты будут доступны всем модулям и программам, вызывающим данный модуль.

Исполняемая часть начинается служебным словом `IMPLEMENTATION` и содержит описания подпрограмм, объявленных в интерфейсной части. Здесь же могут объявляться локальные объекты, которые используются только в интерфейсной части и остаются недоступными программам и модулям, вызывающим данный модуль.

В инициирующей части размещаются исполняемые операторы, содержащие некоторый фрагмент программы. Эти программы исполняются до передачи управления основной программе и обычно используются для подготовки ее работы. Инициирующая часть может отсутствовать вместе с начинающим ее словом `begin`.

В качестве примера рассмотрим модуль работы с комплексными числа, в котором будут представлены подпрограммы, реализующие основные операции[56] с комплексными числами:

·`procedure sum(a,b:complex;var c:complex)` — процедура сложения двух комплексных чисел `a` и `b`, результат возвращается в переменной `c`;

·`procedure razn(a,b:complex;var c:complex)` — процедура вычитания двух комплексных чисел `a` и `b`, результат возвращается в переменной `c`;

·`procedure umn(a,b:complex;var c:complex)` — процедура умножения двух комплексных чисел `a` и `b`, результат возвращается в переменной `c`;

·`procedure delenie(a,b:complex;var c:complex)` — процедура деления двух комплексных чисел `a` и `b`, результат возвращается в переменной `c`;

Кроме того, в модуле будет процедура вывода комплексного числа на экран `procedure vivod(a:complex)`.

Текст модуля приведен ниже.

```
UNIT compl;
INTERFACE
type
complex=record
x:real;
y:real;
end;
procedure sum(a,b:complex;var c:complex);
procedure razn(a,b:complex;var c:complex);
procedure umn(a,b:complex;var c:complex);
procedure delenie(a,b:complex;var c:complex);
procedure vivod(a:complex);
IMPLEMENTATION
procedure sum(a,b:complex;var c:complex);
begin
```

56 Суммой двух чисел a+bi и c+di является число (a+c) + (b+d)i, разностью этих чисел является число (a−c) + (b−d)i, произведением — число (ac−bd) + (bc+ ad)i, частным число - $\left(\frac{ab+bd}{c^2+d^2}\right)+\left(\frac{bc+ad}{c^2+d^2}\right)i$.

```
c.x:=a.x+b.x;
c.y:=a.y+b.y;
end;
procedure razn(a,b:complex;var c:complex);
begin
c.x:=a.x-b.x;
c.y:=a.y-b.y;
end;
procedure umn(a,b:complex;var c:complex);
begin
c.x:=a.x*b.x-a.y*b.y;
c.y:=a.y*b.x+a.x*b.y;
end;
procedure delenie(a,b:complex;var c:complex);
begin
c.x:=(a.x*b.x+a.y*b.y)/(b.x*b.x+b.y*b.y);
c.y:=(a.y*b.x-a.x*b.y)/(b.x*b.x+b.y*b.y);
end;
procedure vivod(a:complex);
begin
if a.y>=0 then
    writeln(a.x:1:3,'+',a.y:1:3,'i')
else
    writeln(a.x:1:3,'-',-a.y:1:3,'i')
end;
end.
```

Для работы с модулем запустим Geany, введем текст модуля, сохраним его под именем `compl.pas`. После этого выполним команду **Собрать**. В каталоге, где сохранен модуль, в результате компиляции появятся два файла — с расширением **.o** (в Linux и **.obj** в Windows), и с расширением **.ppu**. Для использования нужен именно файл с расширением **.o**. Теперь скомпилированный файл модуля с расширением **.o** должен находится либо в том же каталоге, что и программа, которая будет его вызывать, либо в том, где находятся стандартные модули вашего компилятора.

Теперь можно написать программу, использующую этот модуль. Текст этой консольной программы приведен ниже.

```
uses compl;
```

```
var g,h,e:complex;
BEGIN
 writeln('Введите первое комплексное число');
 read(g.x,g.y);
 vivod(g);
 writeln('Введите второе комплексное число');
 read(h.x,h.y);
 vivod(g);
 sum(g,h,e);
 writeln('Сумма чисел');
 vivod(e);
 razn(g,h,e);
 writeln('Разность чисел');
 vivod(e);
 umn(g,h,e);
 writeln('Произведение чисел');
 vivod(e);
 delenie(g,h,e);
 writeln('Частное чисел');
 vivod(e);
end.
```

Результаты работы программы представлены на рис. 4.12.

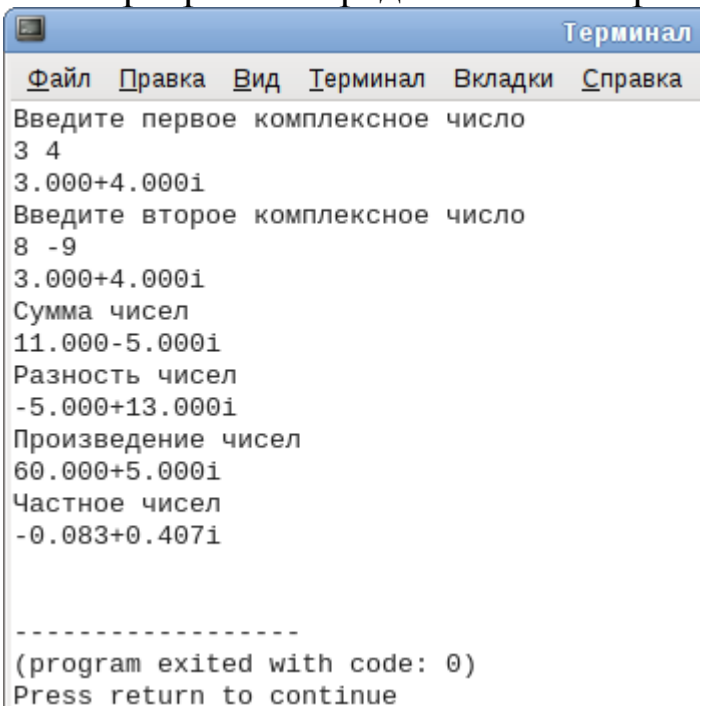

Рисунок 4.12: Операции с комплексными числами

4.9 Задачи для самостоятельного решения

Напишите программу, используя процедуры и функции.

1. Вводится последовательность целых чисел, 0 – конец последовательности. Определить, содержит ли последовательность хотя бы одно число, сумма цифр в котором равна их количеству. Создать процедуру, которая возвращает сумму и количество цифр в числе.

2. Вводится последовательность целых чисел, 0 – конец последовательности. Определить, содержит ли последовательность хотя бы одно совершенное число. Для определения совершенного числа создать функцию.

3. Вводится последовательность из N целых положительных элементов. Определить, содержит ли последовательность хотя бы одно простое число. Для определения простого числа создать функцию.

4. Вводится последовательность из N целых положительных элементов. Посчитать количество чисел палиндромов. Для определения палиндрома создать функцию.

5. Вводится последовательность из N целых положительных элементов. Подсчитать количество совершенных чисел в последовательности. Для определения совершенного числа создать функцию.

6. Поступает последовательность целых положительных чисел, 0 – конец последовательности. Определить, в каком из чисел больше всего делителей. Для подсчета делителей числа использовать функцию.

7. Поступает последовательность целых положительных чисел, 0 – конец последовательности. Определить, в каком из чисел больше всего цифр. Для подсчета количества цифр числа использовать функцию.

8. Вывести на экран значения функции $f(x)=x-2e^x$ и ее первой производной $f'(x)$, в диапазоне от -5 до 5, с шагом 0.5. Для вычисления значений $f(x)$ и $f'(x)$ создать функции.

9. Вводится последовательность из N целых положительных элементов. Найти число с минимальным количеством цифр. Для определения количества цифр в числе использовать функцию.

10. Вводится последовательность из N целых элементов. Для всех положительных элементов последовательности вычислить значение факториала и вывести его на печать. Вычисление факториала оформить в виде функции.

11. Поступает последовательность целых положительных чисел, 0 – конец последовательности. Вывести на экран все числа последовательности, не являющиеся простыми, и их делители. Определение простого числа оформить в виде функций.

12. Вводится последовательность из N целых элементов. Вывести на экран все числа последовательности, являющиеся совершенными, и их делители. Определение совершенного числа оформить в виде функций.

13. Поступает последовательность целых положительных чисел, 0 – конец последовательности. Найти среднее арифметическое простых чисел в этой последовательности. Определение простого числа оформить в виде функций.

14. В последовательности из N целых положительных элементов найти число с наибольшим количеством нулей в своем представлении. Составить функцию для подсчета нулей в числе.

15. В последовательности из N целых положительных элементов найти сумму всех палиндромов. Для определения палиндрома создать функцию.

16. Поступает последовательность целых положительных чисел, 0 – конец последовательности. Посчитать количество элементов последовательности, имеющих в своем представлении цифру 0. Создать процедуру, возвращающую значение истина, если в числе есть нули, и ложь в противном случае.

17. Поступает последовательность целых положительных чисел, 0 – конец последовательности. Для каждого числа найти количество нулей и единиц. Создать процедуру, которая возвращает количество нулей и единиц в заданном числе.

18. Вводится последовательность из N целых элементов. Для каждого элемента последовательности найти среднее значение его цифр. Создать функцию для расчета среднего значения цифр в числе.

19. Поступает последовательность целых положительных чисел, 0 – конец последовательности. Определить количество цифр и наименьшую цифру для каждого числа последовательности. Написать процедуру, которая для заданного числа возвращает два параметра: количество цифр в нем и наименьшую цифру.

20. Вводится последовательность из N целых элементов. Для каждого элемента последовательности вывести на экран количество цифр и количество делителей. Написать процедуру, которая рассчитывает оба параметра.

21. Поступает последовательность целых положительных чисел, 0 – конец последовательности. Записать каждое число последовательности в обратном порядке. Например, 12345 → 54321. Создать функцию для преобразования числа.

22. Поступает последовательность целых положительных чисел, 0 – конец последовательности. Для каждого элемента последовательности вывести на экран количество цифр в числе и наибольшую цифру. Написать процедуру, которая возвращает количество цифр и наибольшую цифру заданного числа.

23. Вводится последовательность из N целых положительных элементов. Для простых элементов последовательности определить сумму цифр. Написать процедуру, которая проверяет, является ли число простым, и вычисляет сумму цифр в нем. Если число простым не является, то процедура выдает соответствующее сообщение.

24. Поступает последовательность целых положительных чисел, 0 – конец последовательности. Для каждого числа определить сумму и количество цифр в числе. Написать процедуру для подсчета суммы и количества цифр в числе.

25. Вывести на экран значения функций $f(x)=\sqrt{(5x^3)}\cdot\sin(x^2)$ и $g(x)=\begin{cases} e^{\frac{x}{4}}, если\ x\geq 0 \\ \sqrt[5]{x^2}, если\ x<0 \end{cases}$ в диапазоне от *a* до *b,* с шагом *h*. Для вычисления значений *f*(*x*) и g(*x*) создать функции.