Методические рекомендации по дисциплине

Лабораторная работа 1 - 4

Линейные программы, ветвления, циклы, массивы, строки, процедуры и функции

1. Составить программу расчета значения функции
$Z = \text{tg } x^3 - |2 \sin x^2 y + 7.8 \cos x| + 10$ при любых значениях x и y. Результат вывести в виде: при x= … и y=… z=…
2. Определить сумму квадратов цифр введенного 3-значного числа.
3. Из чисел A, B, C, D выбрать максимальное.
4. Определить, сколько среди заданных чисел A, B, C, D отрицательных.
5. Задана арифметическая прогрессия. 7,6; 6,3; … . Сколько членов прогрессии нужно сложить, чтобы полученная сумма стала <0.
6. Протабулировать функцию $y=x^3-1$ на интервале [-1,3] с шагом 0.2.
7. Малое предприятие в первый день работы выпустило P единиц товарной продукции. Каждый последующий день оно выпускало продукции на Q единиц больше, чем в предыдущий. Сколько дней потребуется предприятию, чтобы общее количество выпущенной продукции за все время работы впервые превысило запланированный объем?
8. Создать процедуру для нахождения корней квадратного уравнения по его коэффициентам **a,b,c.**
9. Найти минимальное из A, B, C, создав функцию выбора минимального из двух произвольных чисел.
10. Найти (a! + b!)/(a+b)!, создав функцию для вычисления факториала произвольного натурального числа.
11. Создать процедуру для вывода и нахождения суммы первых N членов арифметической прогрессии, заданной формулой $\mathbf{a_n=2n+1.}$
12. Задан одномерный массив A[1..20]. Найти минимальный элемент среди элементов массива с n-го по k-й (n и k вводятся с клавиатуры)
13. Задан одномерный массив A[1..20]. Просуммировать все отрицательные элементы, стоящие на нечетных местах.
14. В двухмерном массиве хранится информация о количестве студентов в той или иной группе каждого курса института с первого по пятый (в первом столбце — информация о группах первого курса, во втором — второго и т.д.). На каждом курсе имеется 8 групп. Определить: а) среднее число студентов в одной группе на третьем курсе; б) количество студентов в самой большой группе на 1 курсе.
15. Вывести на экран матрицу 10x8, элементами которой являются целые случайные числа из интервала [19,49]. Определить минимальный элемент в каждом столбце и выбрать из них максимальный.
16. Напечатать самое длинное слово из заданного текста (слова разделены пробелами).
17. Преобразовать введенную строку так, чтобы сначала были расположены цифры, потом буквы.
18. Дано предложение. Все пробелы в нем заменить на символ "_"(создать соответствующую процедуру).

Лабораторная работа 5 - 6.

Основные этапы решения задач

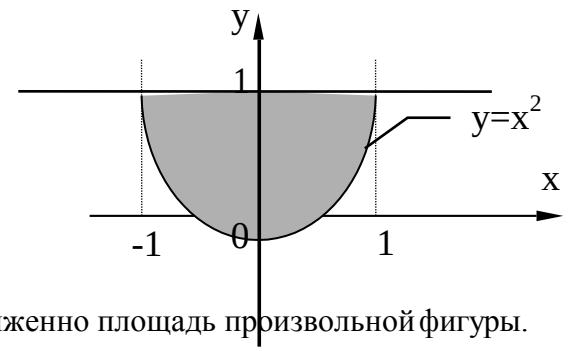
Решение задач с помощью ЭВМ включает в себя следующие основные этапы, часть из которых осуществляется без участия компьютера.

1. Постановка задачи.
2. Анализ, формализованное описание задачи, выбор модели.
3. Разработка алгоритма.
4. Программирование.
5. Отладка, тестирование и анализ полученных результатов.

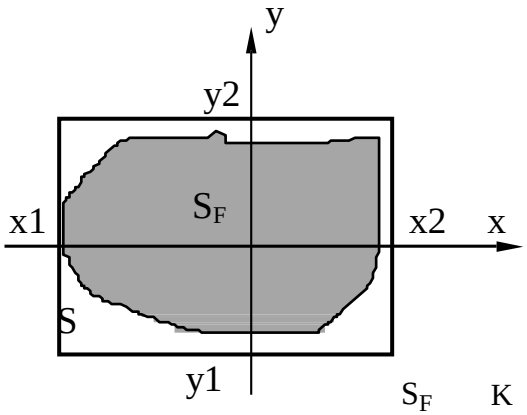
Пример 1. Вычисление площади произвольной фигуры методом

Монте- Карло.

1. Дано: фигура, ограниченная линиями
$y=x^2$ и $y=1$
Найти: площадь данной фигуры.



2. Метод Монте-Карло позволяет находить приближенно площадь произвольной фигуры.



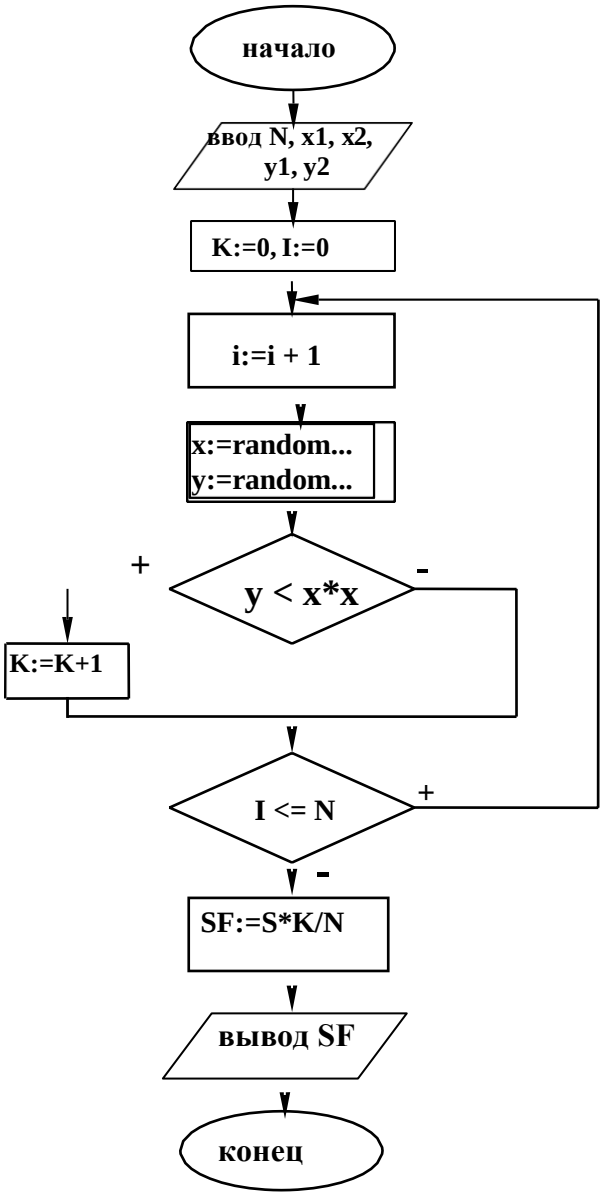
Для этого фигура помещается внутрь прямоугольника, образованного линиями x=x1, x= x2, y= y1, y=y2 (x1, x2, y1, y2 подбираются таким образом, чтобы фигура полностью лежала внутри полученного прямоугольника). Прямоугольник случайным образом заполняется N точками. Если внутрь фигуры F попали K точек, то отношение K/N приближенно равно отношению площадей S_F/S, где S_F – площадь фигуры, S – площадь прямоугольника.

$\frac{S_F}{S} = \frac{K}{N}$, отсюда $S_F = S*K/N$.

Для решения задачи необходимо задать: N - общее число точек, x1, x2, y1, y2- границы прямоугольника, содержащего фигуру. Вычислив площадь прямоугольника, содержащего фигуру S и подсчитав количество точек (x, y), попадающих внутрь фигуры, можно определить площадь фигуры.

В данном примере фигура, площадь которой необходимо найти, ограничена линиями $y=x^2$ и y=1 и помещена в прямоугольник, заданном границами x1= -1, x2= 1, y1=0, y2=1. Чтобы подсчитать количество точек K, принадлежащих фигуре, необходимо проверить условие $y>x^2$.

3.



4.

```
Program Pr1;
Var
   N, k, i: longint;
   x, y,x1,x2,y1,y2, S, SF: real;
BEGIN
 {ввод исходных данных}
 Write('N=');         Readln(N);
 Write('x1, x2='); Readln(x1, x2);
 Write('y1, y2='); Readln(y1, y2);
  k:=0;
 {площадь прямоугольника}
 S:=(x2-x1)*(y2-y1);
 Randomize;
 For i:=1 to N do

    {выбор случайной точки}
        x:=random*(x2-x1)+x1;

    {подсчет количества точек, попавших в фигуру}
        If y>x*x then k:=k+1;
 end;
 {площадь фигуры}
 SF:=S*k/N;
 Writeln('SF=', SF:3:3);
 Readln
END.
```

5. Для тестирования, необходимо подобрать такие исходные данные, которые позволят получить результат аналитическим путем и сравнить его с результатом выполнения программы. Например, рассмотрим 2 случая:

1) вычисление площади круга радиуса 1 (прогнозируемый результат можно получить по формуле $S=\pi R^2$);
2) вычисление площади криволинейной трапеции, ограниченной линиями y= x2 , x = 0, x = 1(прогнозируемый результат можно получить, вычислив определенный интеграл $\int_0^1 x^2 dx$)

Результаты тестирования представлены в таблице:

N теста	Исходные данные						Условие принадлежности точки (x,y) заданной фигуре	Результат	
	Линии, ограничивающие фигуру	N	x1	x2	y1	y2		прогнози-руемый	полученный

1	$x^2+y^2=1$	1000	-1	1	-1	1	$x^2+y^2<1$	3.142	3.128
		10000	-1	1	-1	1	$x^2+y^2<1$	3.142	3.133
2	$y= x^2$, $x= 0, x= 1$	1000	0	1	0	1	$y< x^2$	0.333	0.338
		10000	0	1	0	1	$y< x^2$	0.333	0.329

Пример 2. Определение суммы ряда с заданной точностью

Найти значение суммы $\frac{1}{2!}+\frac{1}{3!}+\frac{1}{4!}+\ldots$ с точностью E=0.00001 и определить количество слагаемых этой суммы.

1. Дано: сумма вида $\sum\frac{1}{k!}$, где k=2, 3, 4,… E=0.00001

Найти: n, $S=\sum_{k=1}^{n}\frac{1}{k!}$

2. Найти сумму с заданной точностью - это значит добавлять к сумме по одному слагаемому до тех пор, пока разница между полученным и предыдущим значением суммы не станет меньше точности E.

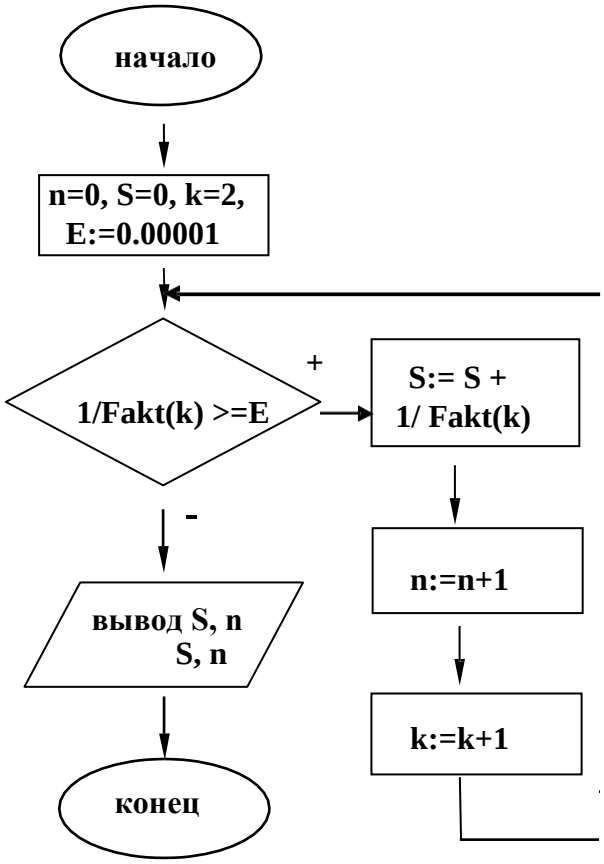
$$\sum_{k=1}^{n}\frac{1}{k!}-\sum_{k=1}^{n-1}\frac{1}{k!}<E$$

Разность между полученным и предыдущим значением суммы– это последнее слагаемое, которое было добавлено к сумме.

$$\sum_{k=1}^{n}\frac{1}{k!}-\sum_{k=1}^{n-1}\frac{1}{k!}=\frac{1}{n!}$$

Таким образом, если очередное слагаемое больше заданной точности ($\frac{1}{n!}>E$), то его следует добавить к сумме.

3.



4.

```
Program Pr2;
    Var S, E: real;
          n, k:  integer;
     Function Fakt(k: integer): real;
      Var i: integer; f: real;
       Begin
          f:=1;
          For i:=1 to k do
                   f:=f*i;
          Fakt:=f;
       End;
BEGIN
      S:=0; n:=0; k:=2;  E:=0.00001;
      While 1/Fakt(k)>=E do begin
              S:=S+1/Fakt(k);
              n:=n+1; k:=k+1;
      end;
      Writeln('S=', s:8:5, ' n=', n)
END.
```

5. Для тестирования программы необходимо подобрать такие исходные данные, которые позволят вручную вычислить значение суммы и сравнить его с данными программы. Например, рассмотрим 2 случая:

1) E=0.1. В этом случае искомая сумма будет содержать 2 слагаемых:

$S = \frac{1}{2!} + \frac{1}{3!}$, т.к. следующее слагаемое $\frac{1}{4!} < 0.1$

2) E=0.01. В этом случае искомая сумма будет содержать 3 слагаемых:

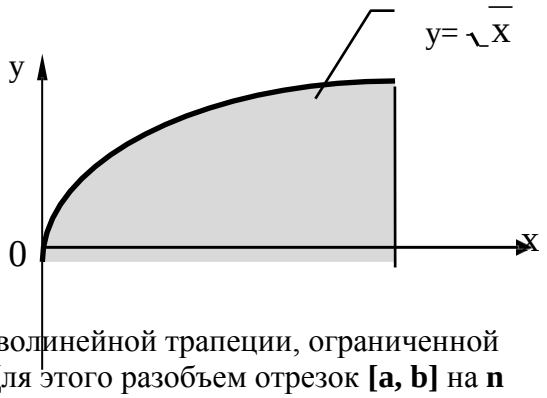
$S = \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!}$, т.к. следующее слагаемое $\frac{1}{5!} < 0.01$

Результаты тестирования представлены в таблице:

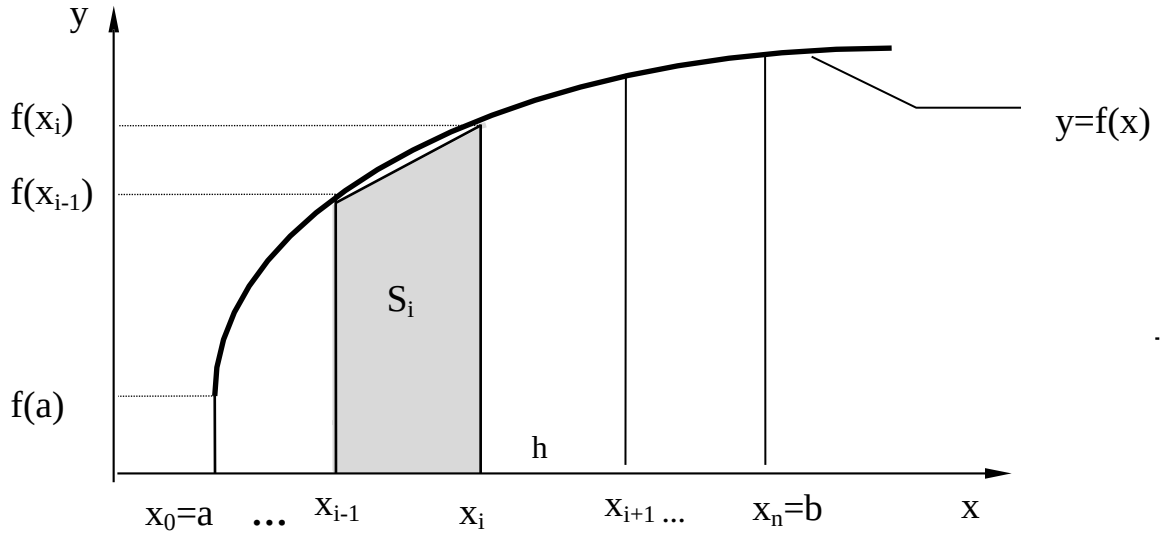
N теста	Исходные данные	Результат	
		прогнозируемый	полученный
1	E=0.1	S=0.66667 n=2	S=0.66667 n=2
2	E=0.01	S=0.70833 n=3	S=0.70833 n=3

Пример 3. Вычисление площади криволинейной трапеции

1. Дано: криволинейная трапеция,
ограниченная линиями $f(x)=\sqrt{x}$, x=0, x=4.
Найти: площадь криволинейной трапеции



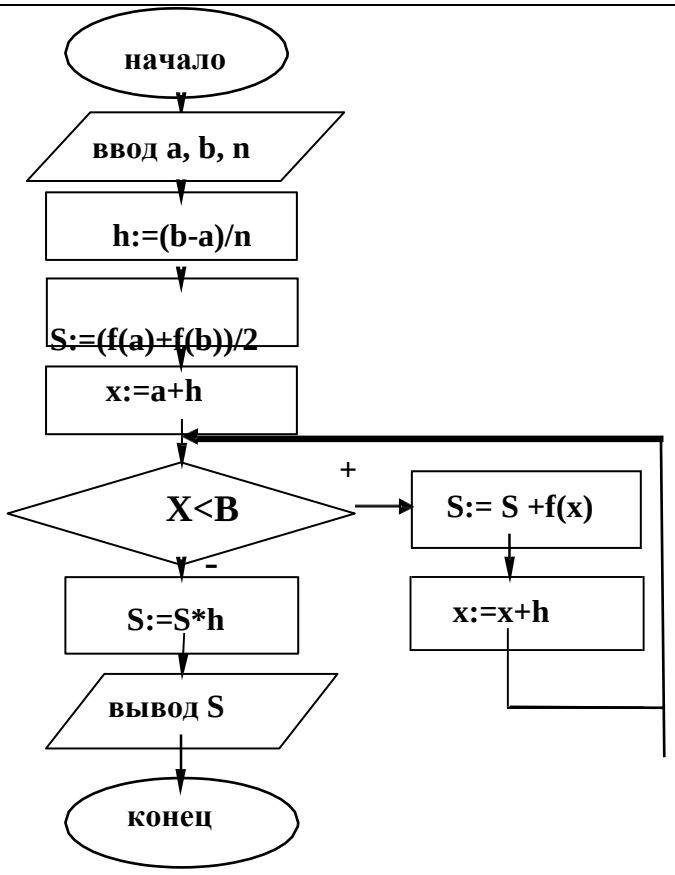
2. Можно найти приближенное значение площади криволинейной трапеции, ограниченной линиями **y=f(x), x=a, x=b**, используя метод трапеций. Для этого разобъем отрезок **[a, b]** на **n** равных частей и через точки разбиения проведем вертикальные прямые до пересечения с графиком функции **y=f(x).** Соединив точки пересечения отрезками, получим **n** прямоугольных трапеций с высотой $h=\frac{b-a}{n}$. Тогда $S_{ABCD}\approx\sum_{i=1}^{n}S_i$, где $S_i=\frac{f(x_{i-1})+f(x_i)}{2}h$. Чем больше **n**, тем точнее результат вычисления.



$$\sum_{i=1}^{n}S_i = h\left(\frac{f(x_0)}{2}+\frac{f(x_1)}{2}+\frac{f(x_1)}{2}+\frac{f(x_2)}{2}+\frac{f(x_2)}{2}+\ldots+\frac{f(x_{n-1})}{2}+\frac{f(x_n)}{2}\right)=$$

$$= h\left(\frac{f(x_0)}{2}+f(x_1)+f(x_2)+\ldots+f(x_{n-1})+\frac{f(x_n)}{2}\right)$$

3.



4.

```
Program Pr5;
Function F(x: real): real;
   Begin
      F:=sqrt(x);
   End;
Var a, b, h, x, s: real;
       n: integer;
BEGIN
    Write('a, b = '); Readln(a, b);
    Write('n = ');      Readln(n);
     h:=(b-a)/n;
     s:=(f(a)+f(b))/2;
     x:=a+h;
     While x<b do
         begin
               s:=s+F(x);    x:=x+h;
         end;
    s:=s*h;
   Writeln('s=', s:8:5);
   Readln
END.
```

5. Для тестирования программы необходимо подобрать такие исходные данные, которые позволят получить результат аналитическим путем и сравнить его с результатом выполнения программы. Например, рассмотрим 2 случая:

1) криволинейная трапеция, ограниченная линиями y= x^2, x=0, x=1. ($\int_0^1 x^2 dx$);

2) криволинейная трапеция, ограниченная линиями y= e^x, x=0, x=1. ($\int_0^1 e^x dx$);

Результаты тестирования представлены в таблице:

N теста	Исходные данные				Результат	
	y=f(x)	N	a	b	прогнозируемый	полученный
1	$y=x^2$	100	01	1	0.33333	0.33341
		1000	0	1	0.33333	0.33333
2	$y= e^x$,	100	0	1	1.71828	1.71830
		1000	0	1	1.71828	1.71828

Задания для самостоятельной работы

Вариант	Номер задачи
I	1a, 2a, 3a
II	1b, 2b, 3b

III	1c, 2c, 3c
IV	1d, 2d, 3d
V	1e, 2a, 3a
VI	1f, 2b, 3b
VII	1g, 2c, 3c
VIII	1h,2d, 3d
IX	1a, 2a, 3a
X	1f, 2b, 3b

1. Найти сумму ряда с точностью 0.0001, используя для вычисления знаменателя функцию:

a) $S = \frac{1}{1+2} + \frac{1}{1+2+3} + \ldots$

b) $S = \frac{2}{3!} + \frac{4}{5!} + \ldots$

c) $S = \frac{1}{1+3} + \frac{1}{1+3+5} + \ldots$

d) $S = \frac{1}{2+4} + \frac{1}{2+4+6} + \ldots$

e) $S = \frac{1}{1+2} + \frac{3}{1+2+3} + \ldots$

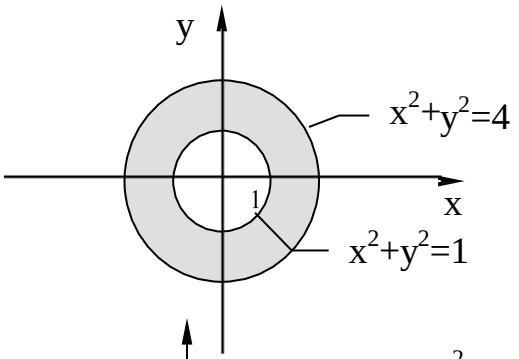
f) $S = \frac{1}{3!} + \frac{1}{5!} + \ldots$

g) $S = \frac{1}{1+3} + \frac{2}{1+3+5} + \ldots$

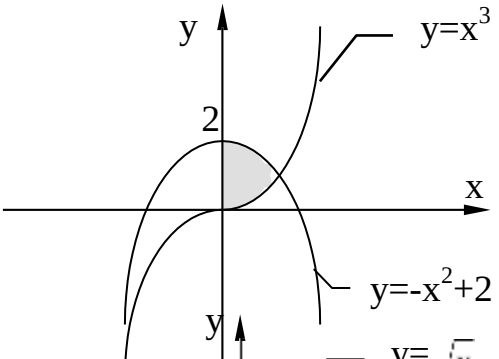
h) $S = \frac{1}{2+4} + \frac{3}{2+4+6} + \ldots$

2. Найти площадь фигуры методом Монте-Карло.

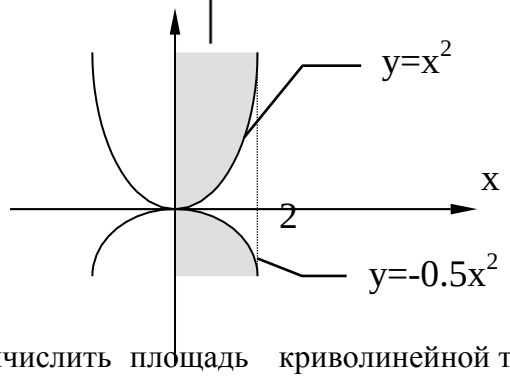
a)



b)



c)



d)

3. Вычислить площадь криволинейной трапеции, ограниченной линиями: y=f(x), x=a; x=b (использовать метод трапеций, разбивая отрезок [a, b] на n равных частей.

a) $f(x)=-(x-3)^2+4$, a=1, b=4, n=100, 200, 1000.

b) $f(x)=(x-4)^2$, a=1, b=4, n=100, 500, 1000.

c) $f(x)=\ln(x-1)$, a=2, b=4, n=100, 200, 1000.

d) $f(x)=|2\cos x+1|$, a=0, b=π. n=100, 500, 1000.

Лабораторная работа 7 - 8.
Файлы и записи

I. Файлы

1. Переписать из текстового файла f в файл g строки в перевернутом виде.

```
program example1;
    var
     f,g:text;
      m:array[1..100] of string;
     k,i:integer;
     st,st1:string;
  begin
     assign(f,'input.dat');
      reset(f);
       k:=0;
        while not eof(f) do
  begin
  readln(f,st);
    k:=k+1;
    st1:='';
      for i:= 1 to length(st) do st1:= st[i]+st1;
       m[k]:=st1;
end;
close(f);
     assign(g,'output.dat');
      rewrite(g);
     for i:= 1 to k do writeln(g,m[i]);
     close(g);
end.
```

Задания

1. Даны текстовые файлы f1 и f2. Переписать с сохранением порядка следования компоненты файла f1 в файл f2, а компоненты файла f2 в файл f1. Использовать вспомогательный файл h.
2. Дан текстовый файл f. Записать в файл g компоненты файла f в обратном порядке.
3. Даны текстовые файлы f и g. Записать в файл h сначала компоненты файла f, затем - компоненты файла g с сохранением порядка.
4. Дан файл f, компоненты которого являются целыми числами. Получить в файле g все компоненты файла f являющимися четными числами.
5. Дан файл f, компоненты которого являются целыми числами. Получить файл g, образованный из файла f исключением повторных вхождений одного и того же числа.
6. Дан файл f, компоненты которого являются целыми числами. Никакая из компонент файла не равна нулю. Файл f содержит столько же отрицательных чисел, сколько и положительных. Используя вспомогательный файл h, переписать компоненты файла f в файл g так, чтобы в файле g вначале шли положительные, затем отрицательные числа.
7. Дан файл f, компоненты которого являются целыми числами. Никакая из компонент файла не равна нулю. Числа в файле идут в следующем порядке: десять положительных, десять отрицательных, десять положительных, десять отрицательных и т.д. Переписать компоненты файла f в файл g так, чтобы в файле g числа шли в следующем порядке пять положительных, пять отрицательных, пять положительных, пять отрицательных и т.д.
8. Дан файл f, компоненты которого являются целыми числами. Записать в файл g наибольшее значение первых пяти компонент файла f, затем - следующих пяти компонент и т.д. Если в последней группе окажется менее пяти компонент, то последняя компонента файла g должна быть равна наибольшей из компонент файла f, образующих последнюю (неполную) группу.
9. Дан символьный файл f подсчитать число вхождений в файл каждой из букв 'a','b','c','d', 'e','f' и вывести результат в файл g в виде таблицы

a --> Na b --> Nb c --> Nc
d --> Nd e --> Ne f --> Nf
где Na, Nb, Nc, Nd, Ne, Nf - числа вхождений соответствующих букв.

10. Дан символьный файл f. Группы символов, разделенные пробелами (одним или несколькими) и не содержащие пробелов внутри себя, будем называть словами. Удалить из файла все однобуквенные слова и лишние пробелы. Результат записать в файл g.

II. Записи

2. Пример: Даны два рациональных числа(дроби), опишите их, используя структуру данных запись (числитель, знаменатель). Привести их к несократимому виду, найди их сумму.

```
PROGRAM Example2;
      Type Tfraction = record
         Chisl: Integer;
          Znam: Word;
       End;
     Function nod(a,b:integer):integer;
      Var k:integer;
       Begin
         While a<>b do
         If a>b then a:= a-b else b:=b-a;
         nod:=a;
    End;
       Var
         x,y,s: Tfraction;
      n,p:integer;
    st: string;
    begin
      writeln('Введите два рациональных числа');
       write('x= '); readln(st); n:= pos('/',st);
        val(copy(st,1,n-1),x.chisl,p);
       val(copy(st,n+1,length(st)-n),x.znam,p);
       write('y= '); readln(st); n:= pos('/',st);
       val(copy(st,1,n-1),y.chisl,p);
         val(copy(st,n+1,length(st)-n),y.znam,p);
           {находим НОД для каждой дроби и сокращаем их}
         n:=nod(x.chisl,x.znam);
        x.Chisl:= x.chisl div n;
       x.znam:= x.znam div n;
       n:=nod(y.chisl,y.znam);
       y.Chisl:= y.chisl div n;
       y.znam:= y.znam div n;
     writeln('Сокращенные дроби:');
     writeln('X= ',x.chisl,'/',x.znam, ' Y=',y.chisl,'/',y.znam);
     if x.znam<>y.znam then
      begin
      s.Znam:=x.znam*y.znam;
      s.Chisl:=x.Chisl*y.Znam+y.Chisl*x.Znam;
      end
      else
      begin
      s.Znam:=x.znam;
     s.Chisl:=x.Chisl+y.Chisl;
     end;
     n:=nod(s.chisl,s.znam);
     s.Chisl:= s.chisl div n;
     s.znam:= s.znam div n;
```

```
 writeln('Сумма дробей:');
writeln('S= ',s.chisl,'/',s.znam);
END.
```

Задания

1. Багаж пассажира характеризуется количеством вещей и общим весом вещей. Сведения о багаже каждого пассажира представляют собой запись с двумя полями: одно поле целого типа (количество вещей) и одно - действительное (вес в килограммах).

a) Найти багаж, средний вес одной вещи в котором отличается не более, чем на 0.3 кг от общего среднего веса одной вещи.

b) Найти число пассажиров, имеющих более двух вещей и число пассажиров, количество вещей которых превосходит среднее число вещей.

c) Определить, имеются ли два пассажира, багажи которых совпадают по числу вещей и различаются по весу не более чем на 0,5 кг.

d) Выяснить, имеется ли пассажир, багаж которого превышает багаж каждого из остальных пассажиров и по числу вещей, и по весу.

e) Выяснить, имеется ли пассажир, багаж которого состоит из одной вещи весом менее 30 кг.

2. После поступления в ВУЗ о студентах ОЗО собрана информация: фамилия, нуждается ли в общежитии, стаж, работал ли учителем, что окончил, какой язык изучал. Составить программу, определяющую: 1) сколько человек нуждаются в общежитии; 2) списки студентов, проработавших 2 и более лет учителем; 3) списки окончивших педучилище; 4) списки языковых групп.

3. Описать, используя структуру данных запись, данные на учеников (фамилия, улица, дом, квартира). Составить программу, определяющую сколько учеников живет на улице Свердлова, списки учеников, живущих в доме номер 45.

4. В библиотеке для каждого заказывающего книгу читателя заполняется карточка: фамилия, дата заказа, дата выдачи книги. Определить: 1) самый маленький срок, за который нашли книгу; 2) сколько заказов было не удовлетворено; 3) кто чаще всего берет книги; 4) кому выдали книги 15.09.90; 5) сколько человек заказывали книги 25.04.90.

5. Описать, используя структуру данных запись, почтовую сортировку (город, улица, дом, квартира, кому, ценность). Составить программу, определяющую: 1) сколько посылок отправлено в г.Самару; 2) сколько и куда (список городов) отправлено посылок ценностью выше 10 рублей; 3) есть ли адреса куда отправлено более 1 посылки, если есть то сколько и кому.

6. Описать, используя структуру данных запись, завод (наименование станка, время простоя в месяц, время работы в месяц). Составить программу, определяющую общее время простоя на заводе, списки станков, не имеющих простоя, относительное время простоя всех и каждого станка (

7. В школе было три 9 класса, в августе каждый классный руководитель имел сведения о своих учениках: фамилия, куда поступал, поступил или нет. Определить сколько учеников хотели пойти в 10 класс, кто хотел поступать в училище и техникум, кто поступил в училище или техникум, сколько учеников будет учится в 10 классе, сколько необходимо создать 10 классов и по сколько человек.

8. На олимпиаде по информатике на школьников заполнялись анкеты: фамилия, номер школы, класс, занятое место. Напечатать: 1) списки школ, занявших призовые места; 2) какая из школ заняла больше всех призовых мест; 3) списки учеников занявших первое место, указать их класс.

9. В деканате хранится информация о зимней сессии на 1 курсе (фамилия, номер группы, оценка 1 по геометрии, оценка 2 по алгебре, оценка 3 по информатике). Составить программу, печатающую фамилии студентов, имеющих задолженность хотя бы по одному предмету, качество успеваемости, процент студентов, т.е. сдавших экзамены на 4 и 5, название предмета, который был сдан лучше всего, номера групп в порядке убывания средней успеваемости их студентов.

10. В отделе кадров студентов хранится следующая информация о каждом студенте: фамилия, имя, отчество, пол, возраст, курс. Составить программу которая печатает номер курса, на котором наибольший процент мужчин, самые распространенные мужские и женские

имена, фамилии в алфавитном порядке и инициалы всех студенток, отчество и возраст которых являются одновременно самыми распространенными.

Лабораторная работа 9-10
Модули

Всякий модуль Паскаля имеет следующую структуру:

Unit <имя_модуля>;
interface
<интерфейсная часть>;
implementation
< исполняемая часть >;
begin
<инициирующая часть>;
end.

Здесь UNIT – зарезервированное слово (единица); начинает заголовок модуля;
<имя_модуля> - имя модуля (правильный идентификатор);
INTERFACE – зарезервированное слово (интерфейс); начинает интерфейсную часть модуля;
IMPLEMENTATION – зарезервированное слово (выполнение); начинает исполняемую часть модуля;
BEGIN – зарезервированное слово; начинает инициирующую часть модуля; причем конструкция begin <инициирующая часть> необязательна;
END – зарезервированное слово – признак конца модуля.

Пример: создать модуль Паскаля, реализующий сложение и вычитание комплексных чисел с помощью процедур:

```
Unit complexn;
Interface
  type
  complex= record
  re, im: real;
  end;
procedure AddC (x, y: complex; var z: complex);
procedure SubC (x, y: complex; var z: complex);
const c: complex= (re: 0.1; im: -1);

implementation
procedure AddC;
begin
  z.re:= x.re + y.re;
  z.im:= x.im + y.im;
end; {AddC}
procedure SubC;
begin
  z.re:= x.re - y.re;
  z.im:= x.im - y.im;
end; {SubC}
end.
```

Текст этого модуля следует поместить в файл complexn.pas . Вы можете его откомпилировать, создав TPU -файл.
В следующей программе осуществляются арифметические операции над комплексными числами:

```
Program primer;
Uses complexn;
Var
  a,b,c: complex;
begin
  a.re:= 1; a.im:= 1;
  b.re:= 1; b.im:= 2;
  AddC(a, b, c);
  Writeln (‘ сложение :’, c.re: 5:1, c.im: 5:1, ‘i’);
  SubC (a, b, c);
  Writeln (‘ вычитание :’, c.re: 5:1, c.im: 5:1, ‘i’);
End.
```

После объявления Uses complexn программе стали доступны все объекты, объявленные в интерфейсной части модуля complexn . При необходимости можно переопределить любой из этих объектов, как произошло, например, с типизированной константой ***c*** , объявленной в модуле Паскаля. Переопределение объекта означает, что вновь объявленный объект «закрывает» ранее определенный в модуле одноименный объект. Чтобы получить доступ к «закрытому» объекту, нужно воспользоваться составным именем: перед именем объекта поставить имя модуля и точку. Например :

```
Writeln (complexn.c.re: 5: 1, complexn.c.im: 5: 1);
```

Этот оператор выведет на экран содержимое «закрытой» типизированной константы, объявленной в модуле Паскаля из предыдущего примера.

Задания

I. Реализовать в виде модуля набор подпрограмм для выполнения следующих операций над обыкновенными дробями вида *P/Q* (*P* — целое, *Q* — натуральное): 1) сложение; 2) вычитание; 3) умножение; 4) деление; 5) сокращение дроби; 6) возведение дроби в степень N (N — натуральное).

Используя этот модуль, решить задачи:

1. Дан массив A — массив обыкновенных дробей. Найти сумму всех дробей, ответ представить в виде несократимой дроби. Вычислить среднее арифметическое всех дробей, ответ представить в виде несократимой дроби.
2. Дан массив A — массив обыкновенных дробей. Отсортировать его в порядке возрастания.

II. Реализовать в виде модуля набор подпрограмм для выполнения следующих операций над комплексными числами: 1) сложение; 2) вычитание; 3) умножение; 4) деление; 5) вычисление модуля комплексного числа; 6) возведение комплексного числа в степень n (n — натуральное).

Комплексное число представить следующим типом:

```
Type Complex = Record
          R, M : Real; {действительная и мнимая часть числа}
          End;
```

Используя этот модуль, решить задачи:

1. Дан массив *A* — массив комплексных чисел. Получить массив *C*, элементами которого будут модули сумм рядом стоящих комплексных чисел.
2. Дан массив *A[M]* — массив комплексных чисел. Получить матрицу *B[N, M]*, каждая строка которой получается возведением в степень, равную номеру этой строки, соответствующих элементов данного массива *A*.

III. Реализовать в виде модуля набор подпрограмм для выполнения следующих операций с квадратными матрицами: 1) сложение двух матриц; 2) умножение одной матрицы на другую; 3) нахождение транспонированной матрицы; 4) вычисление определителя матрицы.

Матрицу описать следующим образом:

```
Const NMax = 10;
Type Matrica = Array [1..NMax, 1..Nmax] Of Real;
```

Используя этот модуль, решить следующие задачи:

1. Решить систему линейных уравнений *N*-го порядка (2<=*N*<=10) методом Крамера.

IV. Реализовать в виде модуля набор подпрограмм для выполнения следующих операций над векторами на плоскости: 1) сложение; 2) вычитание; 3) скалярное умножение векторов; 4) умножение вектора на число; 5) длина вектора.
Вектор представить следующим типом:
Type Vector = Record
X, Y : Real;
End;
Используя этот модуль, решить задачи:
1. Дан массив *A* — массив векторов. Отсортировать его в порядке убывания длин векторов.
2. С помощью датчика случайных чисел сгенерировать 2*N* целых чисел. *N* пар этих чисел задают *N* точек координатной плоскости. Вывести номера тройки точек, которые являются координатами вершин треугольника с наибольшим углом.

V. Реализовать в виде модуля набор подпрограмм для выполнения следующих операций над натуральными числами в *P*-ичной системе счисления (2<=*P*<=9): 1) сложение; 2) вычитание; 3) умножение; 4) деление; 5) перевод из десятичной системы счисления в *P*-ичную; 6) перевод из *P*-ичной системы счисления в десятичную; 7) логическая функция проверки правильности записи числа в *P*-ичной системе счисления; 8) функции, реализующие операции отношения (равно, не равно, больше или равно, меньше или равно, больше, меньше).
P-ичное число представить следующим типом:
Type
Chislo = Array [1..64] Of 0..8;
Используя этот модуль, решить задачи:
1. Возвести число в степень (основание и показатель степени записаны в *P*-ичной системе счисления). Ответ выдать в *P*-ичной и десятичной системах счисления.
2. Дан массив *A* — массив чисел, записанных в *P*-ичной системе счисления. Отсортировать его в порядке убывания. Ответ выдать в *P*-ичной и десятичной системах счисления.

VI. Реализовать в виде модуля набор подпрограмм для выполнения следующих операций над натуральными числами в шестнадцатеричной системе счисления: 1) сложение; 2) вычитание; 3) умножение; 4) деление; 5) перевод из двоичной системы счисления в шестнадцатеричную; 6) перевод из шестнадцатеричной системы счисления в десятичную; 7) функция проверки правильности записи числа в шестнадцатеричной системе счисления; 8) функции, реализующие операции отношения (равно, не равно, больше или равно, меньше или равно, больше, меньше).
Используя этот модуль, решить задачи:
1. Возвести число в степень (основание и показатель степени записаны в шестнадцатеричной системе счисления). Ответ выдать в шестнадцатеричной и десятичной системах счисления.
2. Дан массив *A* — массив чисел, записанных в шестнадцатеричной системе счисления. Отсортировать его в порядке убывания. Ответ выдать в шестнадцатеричной и десятичной системах счисления.

VII. Определим граф как набор точек, некоторые из которых соединены отрезками, подграф — граф, подмножество данного графа. Реализовать в виде модуля набор подпрограмм, определяющих: 1) число точек в графе; 2) число отрезков в графе; 3) число изолированных подграфов в графе (подграфов, не соединенных отрезками); 4) диаметр графа — длину максимальной незамкнутой линии в графе (длина каждого звена — единица); 5) граф — объединение двух графов; 6) подграф — пересечение двух графов; 7) подграф — дополнение данного графа до полного (графа с тем же количеством вершин, что и в заданном, и с линиями между любыми двумя вершинами); 8) число отрезков, выходящих из каждой вершины графа; 9) при запуске должны инициализироваться переменные: Full_Graph — полный граф с числом вершин NumberOfVertix, Null_Graph — граф без отрезков с числом вершин NumberOfVertix.
Граф представить как объект
Const NumberOfVertix = 50;
Type Graph = Array[1..NumberOfVertix, 1..NumberOfVertix] Of Boolean;

Используя модуль, решить задачу: найти все правильные графы из N вершин (граф правилен, если из всех вершин выходит равное количество отрезков).

VIII. Реализовать в виде модуля набор подпрограмм для работы с длинными целыми числами (числами, выходящими за диапазон допустимых значений любого целого типа): 1) сложение; 2) вычитание; 3) умножение; 4) нахождение частного и остатка от деления одного числа на другое; 5) функции, реализующие операции отношения (равно, не равно, больше или равно, меньше или равно, больше, меньше).
Длинное число представить следующим типом:

```
Type Tcifra = 0..9; Chislo = Array [1..1000] Of Tsifra;
```

Используя этот модуль, решить задачи:
1. Возвести число в степень (основание и показатель степени — длинные числа).
2. Дан массив длинных чисел. Упорядочить этот массив в порядке убывания.

IX. Реализовать в виде модуля набор подпрограмм для выполнения операций с многочленами от одной переменной (первый многочлен степени m, второй — степени n): 1) сложение; 2) вычитание; 3) умножение; 4) деление с остатком; 5) операции отношения (равно, не равно); 6) возведение в натуральную степень k одного из многочленов; 7) вычисление производной от многочлена; 8) вычисление значения в точке x_0.
Многочлен представить следующим типом:

```
Type Mnogochlen = Array [1..500] Of Integer;
```

Используя этот модуль, решить задачи:
1. Найти наибольший общий делитель многочленов $P(x)$ и $Q(x)$.
2. Вычислить: $P^s(x)-Q^r(x)$ (s, r — натуральные).

X. Реализовать в виде модуля набор подпрограмм для работы с длинными действительными числами (числами, выходящими за диапазон допустимых значений любого действительных типа или не представленных в памяти ЭВМ): 1) сложение; 2) вычитание; 3) умножение; 4) нахождение частного от деления одного числа на другое с заданным количеством знаков после запятой; 5) функции, реализующие операции отношения (равно, не равно, больше или равно, меньше или равно, больше, меньше); 6) тригонометрические функции, где аргументом и значениями являются длинные действительные числа (указание: использовать разложение соответствующей функции в ряд).
Длинное действительное число представить следующим типом:

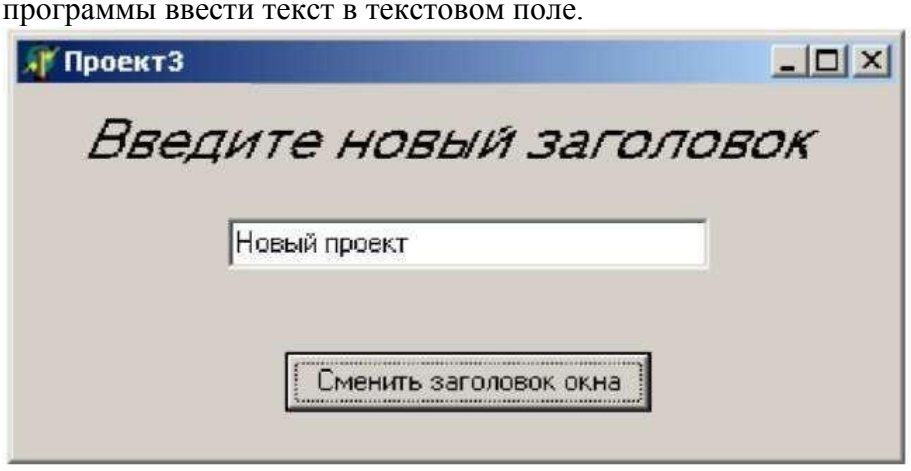
```
Type
Tsifra = 0..9;
Chislo = Array [1..1000] Of Tsifra;
LongReal = Record
Znak : 0..1; {0 - "плюс", 1 - "минус"}
Ts, Dr : Chislo {целая и дробная части}
End;
```

Используя этот модуль, решить задачи:
1. Возвести число в степень (основание — длинное действительное, показатель степени — длинное целое число).
2. Дан массив длинных действительных чисел. Упорядочить этот массив в порядке возрастания.

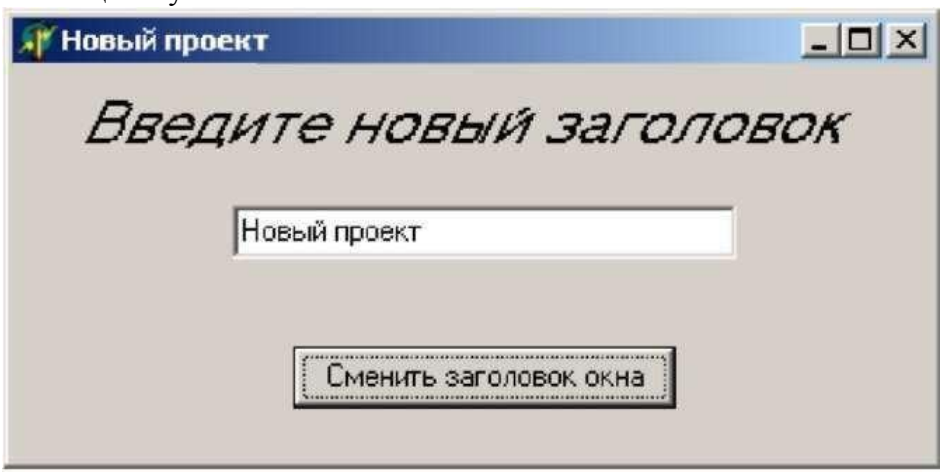
Лабораторнные работы 11 - 12
Основы работы в Lazarus

Задание №1

Цель работы - создать программу, выполняющую следующие действия: 1. После запуска программы ввести текст в текстовом поле.



2. По щелчку мышью на кнопке "Сменить заголовок окна" изменяется заголовок окна.



3. Ввести новый текст в текстовом поле.
4. Изменить название заголовка окна по нажатию клавиши Enter.
5. Для выхода из программы необходимо щелкнуть мышью на закрывающей кнопке в строке заголовка.

Описание плана разработки программы

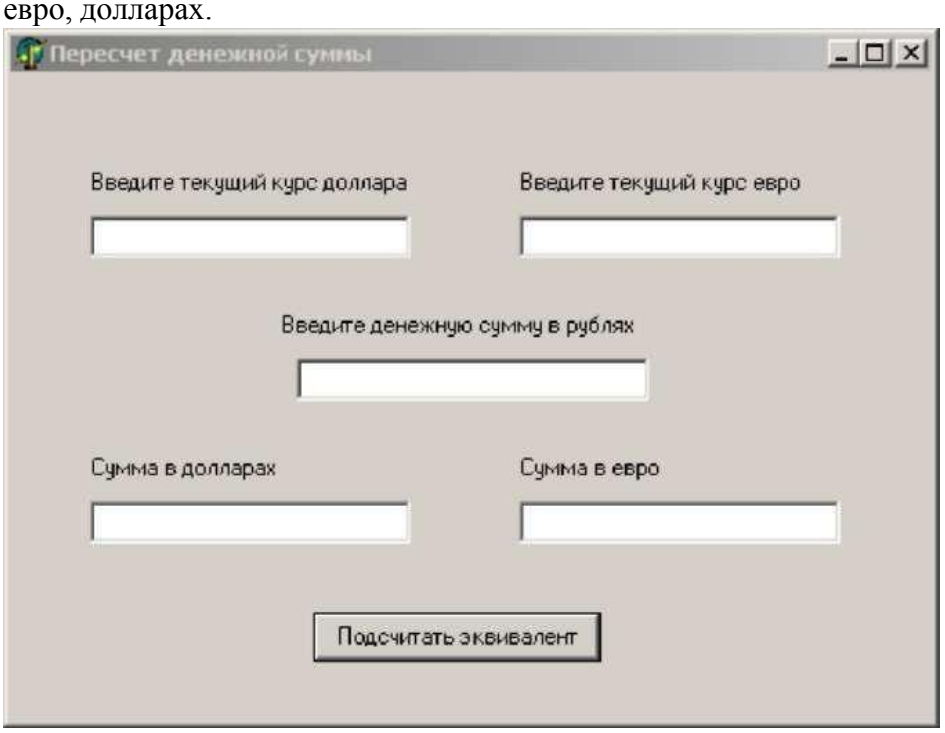
1. Открыть новый проект.
2. Разместить на форме экземпляры компонентов: метку Label, кнопку Button, текстовое поле Edit.
3. Выполнить следующие действия:

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
Forml	Properties	Caption	Установка имени формы "ПроектЗ"
Labell (Вкладка Standard)	Properties	Caption	Ввод текста надписи "Введите новый заголовок:"
Editl (Вкладка Standard)	Properties	Text	Очистить значение свойства Text
Buttonl (Вкладка Standard)	Properties	Caption	Установка имени кнопки "Сменить заголовок окна"
		Default	Выбрать в раскрывающемся списке значение True
	Events	OnClick	Form1.Caption := Edit1.Text;

4. Сохраните проект, запустите и протестируйте его.

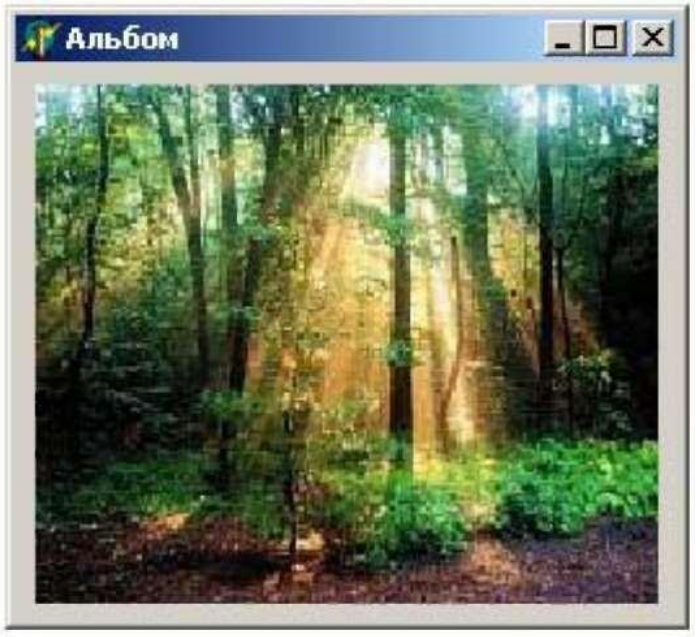
Задание для самостоятельного выполнения

Создать программу, выполняющую следующие действия:
1. После запуска программы отображаются: две строки для ввода текущих курсов для евро и доллара; строка для ввода денежной суммы в рублях; две строки для вывода эквивалента в евро, долларах.



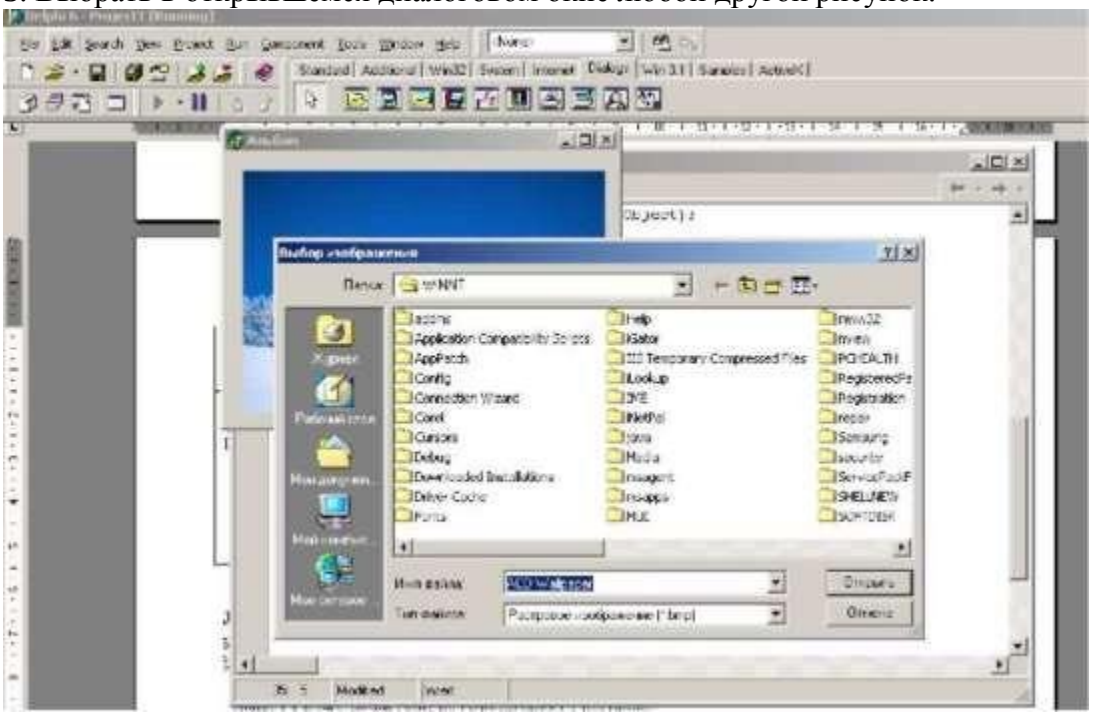
2. Ввести текущий курс для евро и доллара.
3. Ввести денежную сумму в рублях.
4. По щелчку мышью на кнопке "Подсчитать эквивалент" выводится денежная сумма в евро и долларах.
5. Ввести новый текущий курс для евро и доллара.
6. Ввести новую денежную сумму в рублях.
7. По щелчку мышью на кнопке "Подсчитать эквивалент" выводится новая денежная сумма в евро и долларах.
8. Для выхода из программы необходимо щелкнуть мышью на закрывающей кнопке в строке заголовка.

Задание №2
Цель работы - создать программу, выполняющую следующие действия: 1. После запуска программы в окне изображается рисунок.



2. По щелчку мышью на рисунке появляется диалоговое окно.

3. Выбрать в открывшемся диалоговом окне любой другой рисунок.



4. Для выхода из программы необходимо щелкнуть мышью на закрывающей кнопке в строке заголовка.

Описание плана разработки программы

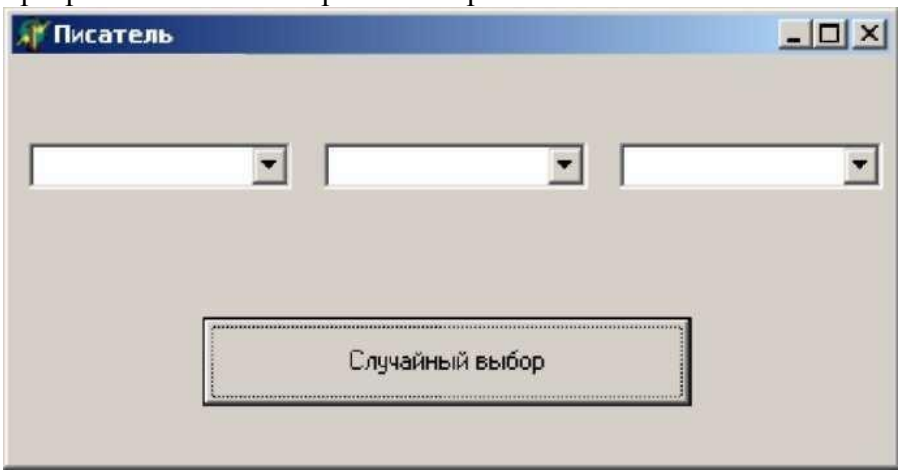
1. Открыть новый проект.
2. Разместить на форме экземпляры компонентов: панель Panel, рисунок Image, диалоговое окно OpenDialog.
3. Выполнить следующие действия:

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
Forml	Properties	Caption	Установка имени формы "Альбом"
Panell (Вкладка Standard)	Properties	Caption	Очистите значение свойства Caption
		BevelOuter	Выбрать в раскрывающемся списке значение bvLowered
		BevelInner	Выбрать в раскрывающемся списке значение bvNone
		BewelWidth	Присвоить значение 2
		Width	Присвоить значение 241
		Height	Присвоить значение 185
Imagel (Вкладка Additional)	Properties	Left	Присвоить значение 2
		Top	Присвоить значение 2
		Width	Присвоить значение 237
		Height	Присвоить значение 181
		Stretch	Включить свойство True
		Picture	С помощью кнопки-построителя открыть диалоговое окно Picture Editor (Редактор изображений). Щелкнуть на кнопке Load (Загрузить) - откроется диалоговое окно Load Picture (Загрузка рисунка). Открыть папку C:\Windows и выбрать файл Лєс.Ьгтф, щелкнуть на кнопке Открыть. Вернуться в окно Редактора изображений, щелкнуть на кнопке OK.
			Image1.Picture.LoadFromFile (OpenDialog1.FileName);

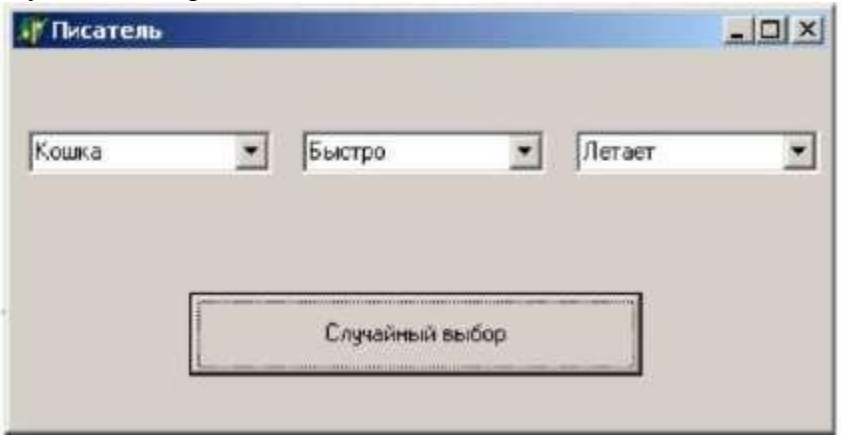
	Events	OnClick	OpenDialog1.Execute;
OpenDialogl (Вкладка Dialogs)	Properties	Title	Ввести текст: "Выбор изображения"
		FileName	Ввести полный путь доступа к файлу: CAWindowsYPec.bmp
		Filter	Ввести текст: Растровое изображение (*.bmp)|*.bmp
		DefaultExt	Присвоить свойству значение: .bmp
		Options	Подсвойству ofFileMustExit (Файл должен существовать) присвоить значение True (Да)

4. Сохраните проект, запустите и протестируйте его.
Листинг подпрограммы
procedure TForm1.Image1Click (Sender: TObject); begin
OpenDialog1.Execute;
Image1.Picture.LoadFromFile (OpenDialog1.FileName); end;

Задание №3
Цель работы - создать программу, выполняющую следующие действия: 1. После запуска программы в окне изображается три поля.



2. По щелчку мышью на кнопке "Случайный выбор" из трех слов составляется предложение случайным образом.



3. Для выхода из программы необходимо щелкнуть мышью на закрывающей кнопке в строке заголовка.
Описание плана разработки программы
1. Открыть новый проект.
2. Разместить на форме экземпляры компонентов: поле со списком ComboBox, командная кнопка Button.
3. Выполнить следующие действия:

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя	Действие

		события	
Forml ComboBoxl (Вкладка Standard)	Properties	Caption	Установка имени формы "Сочинитель"
	Events	OnCreate	ComboBox1.ItemIndex :=0; ComboBox2.ItemIndex :=0; ComboBox3.ItemIndex :=0;
	Properties	Style	Выберите значение cSDropDownList из раскрывающегося списка
		Items	Щелкните на кнопке построителя. Откроется окно String List Editor (Редактор списка строк). Ввести пункты списка по одному в каждую строчку, завершая ввод нажатием клавиши Enter. После того как список готов, щелкнуть на кнопке OK.
Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
ComboBox2 (Вкладка Standard)	Properties	Style	Выберите значение cSDropDownList из раскрывающегося списка
		Items	Щелкните на кнопке построителя. Откроется окно String List Editor (Редактор списка строк). Ввести пункты списка по одному в каждую строчку, завершая ввод нажатием клавиши Enter. После того как список готов, щелкнуть на кнопке OK.
ComboBox3 (Вкладка Standard)	Properties	Style	Выберите значение cSDropDownList из раскрывающегося списка
		Items	Щелкните на кнопке построителя. Откроется окно String List Editor (Редактор списка строк). Ввести пункты списка по одному в каждую строчку, завершая ввод нажатием клавиши Enter. После того как список готов, щелкнуть на кнопке OK.
Buttonl (Вкладка Standard)	Properties	Caption	Установка имени кнопки "Случайный выбор"
	Events	OnClick	ComboBox1.ItemIndex := Random(ComboBox1.ItemIndex.Count); ComboBox2.ItemIndex := Random(ComboBox2.ItemIndex.Count); ComboBox3.ItemIndex := Random(ComboBox3.ItemIndex.Count);

4. Сохраните проект, запустите и протестируйте его.

Список существительных	Список наречий	Список действий
Кошка	Быстро	Плавает
Змея	Высоко	Бегает
Кузнечик	Медленно	Летает
Дельфин	Сильно	Ползает
Черепаха	Хорошо	Прыгает
Ласточка	Плохо	Прячется

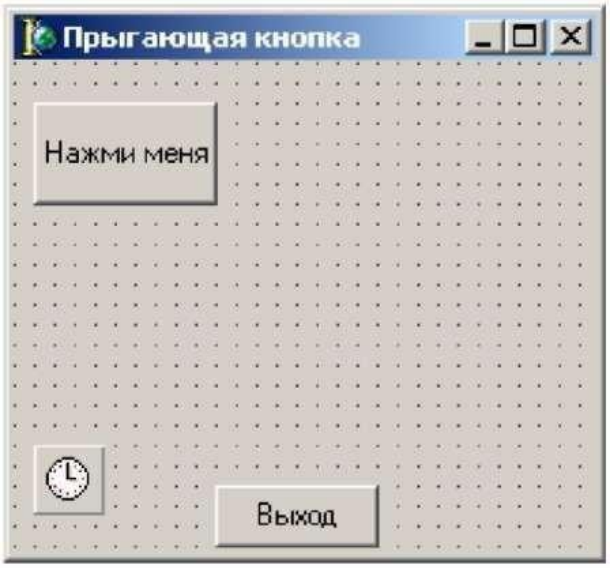
Листинг подпрограммы

```
procedure TFormLFonriCreate (Sender: TObject); begin
Randomize;
ComboBoxl.Itemlndex :=0; ComboBox2.ItemIndex :=0; ComboBox3.ItemIndex :=0; end;
procedure TForml.ButtonlClick (Sender: TObject); begin
ComboBoxl.ItemIndex := Random(ComboBoxl.ItemIndex.Count); ComboBox2.ItemIndex :=
Random(ComboBox2.ItemIndex.Count); ComboBox3.ItemIndex :=
Random(ComboBox3.ItemIndex.Count); end;
```

Задание №4

Цель работы - создать программу-игру, выполняющую следующие действия:
1. После запуска программы в окне изображается беспорядочно прыгающая кнопка.
2. Необходимо успеть щелкнуть по ней.
3. Кнопка перепрыгивает из одного места в другое по сигналу, полученному от таймера.
4. Для выхода из программы необходимо щелкнуть мышью на закрывающей кнопке в строке заголовка.

Описание плана разработки программы
1. Открыть новый проект.
2. Разместить на форме экземпляры компонентов: командная кнопка Button, таймер Timer.



3. Выполнить следующие действия:

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
Forml	Properties	Caption	Установка имени формы "Прыгающая кнопка"
		ClientWidth (Внутренняя ширина)	Присвоить значение 300
		ClientHeigth (Внутренняя высота)	Присвоить значение 200
		BorderStyle (тип границы)	Выбрать значение bsSingle (тонкая)
	Events	OnCreate	Randomize;
Buttonl (Вкладка Standard)	Properties	Caption	Ввести надпись "Нажми меня"
		TabStop	Присвоить значение False. Это свойство разрешает выбрать данный элемент управления клавишей Tab. Клавиатурой пользоваться запрещается.
		Visible	Присвоить значение False. Сначала кнопка невидима.
		Height	Присвоить значение 30
		Width	Присвоить значение 80
	Events	OnClick	Buttonl.Caption := 'Готово'; Buttonl.Enabled := False; Timerl.Enabled := False;

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие

Button2 (Вкладка Standard)	Properties	Caption	Ввести надпись "Выход"
		Default (по умолчанию)	Выбрать значение True
		Left (слева)	Присвоить значение 110
		Top (сверху)	Присвоить значение 160
		Width (ширина)	Присвоить значение 80
		Height (высота)	Присвоить значение 30
	Events	OnClick	Close;
Timerl (Вкладка System)	Properties	Interval (интервал)	Присвоить значение 500 (промежуток времени измеряется в миллисекундах)

	Events	Timer	var i: Integer; begin
			i:=Random(9);
			Button1.Visible := True;
			Button1.Top := 10 + 50 * (i div 3);
			Button1.Left := 10 + 100 * (i mod 3);
			end;

4. Сохраните проект, запустите и протестируйте его. Листинг подпрограммы

```
procedure TForm1.Button2Click (Sender: TObject); begin
Close; end;
procedure TForm1.Timer1Timer (Sender: TObject);
var i: Integer;
begin
i:=Random(9);
Button1.Visible := True;
Button1.Top := 10 + 50 * ( i div 3);
Button1.Left := 10 + 100 * ( i mod 3); end;
procedure TForm1.Button1Click (Sender: TObject); begin
Button1.Caption := 'Готово'; Button1.Enabled := False; Timer1.Enabled := False; end;
procedure TForm1.FormCreate (Sender: TObject); begin
Randomize; end;
```

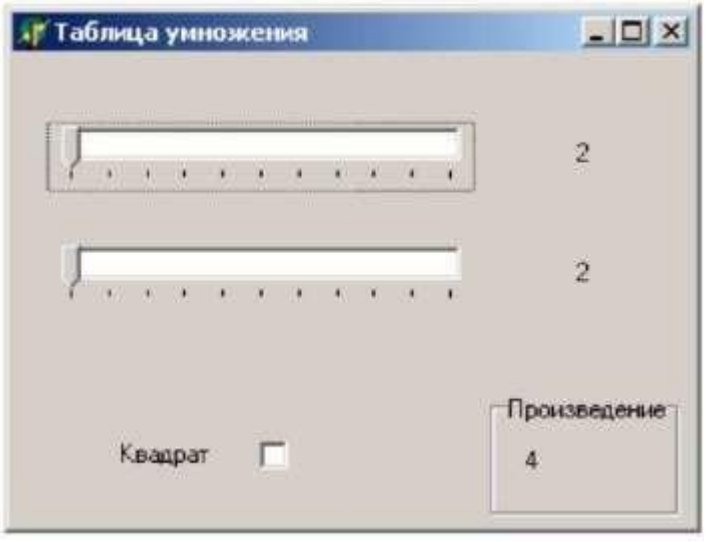
Задание для самостоятельного выполнения

1. Измените игру так, чтобы скорость можно было настраивать в процессе игры.
2. Создайте две кнопки: Медленнее и Быстрее. Щелчок на одной из них будет увеличивать или уменьшать значение свойства Timer1.Interval на 100 миллисекунд.

Задание №5

Цель работы - создать программу, выполняющую следующие действия:

1. После запуска программы в окне изображается два движка.
2. Необходимо выбрать два числовых значения и найти их произведение.
3. Если выбирается одно число, то находится его квадрат.



4. Для выхода из программы необходимо щелкнуть мышью на закрывающей кнопке в строке заголовка.

Описание плана разработки программы

1. Открыть новый проект.
2. Разместить на форме экземпляры компонентов: командная кнопка Button, движок TrackBar, рамка GroupBox, надпись Label, флажок CheckBox.
3. Выполнить следующие действия:

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
Forml	Properties	Caption	Установка имени формы "Таблица умножения"
TrackBarl (Вкладка Win32)	Properties	Min (Минимум)	Присвоить значение 2
		Max (Максимум)	Присвоить значение 99
		Position (Положение)	Присвоить значение 2
		LineSize (Малое изменение)	Присвоить значение 1
		PageSize (Постраничное изменение)	Присвоить значение 7
		Frequency (Частота засечек)	Присвоить значение 7
	Events	OnChange	Label1.Caption := IntToStr(TrackBar1.Position); Label3.Caption := IntToStr(TrackBar1.Position * TrackBar2.Position); if CheckBox1.Checked then TrackBar2.Position :=TrackBar1.Position;

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
TrackBar2 (Вкладка Win32)	Properties	Min (Минимум)	Присвоить значение 2
		Max (Максимум)	Присвоить значение 99
		Position (Положение)	Присвоить значение 2
		LineSize (Малое изменение)	Присвоить значение 1
		PageSize (Постраничное изменение)	Присвоить значение 7
		Frequency (Частота засечек)	Присвоить значение 7
	Events	OnChange	Label2.Caption := IntToStr(TrackBar2.Position); Label3.Caption := IntToStr(TrackBar1.Position * TrackBar2.Position); if CheckBox1.Checked then TrackBar1.Position := TrackBar2.Position;
GroupBoxl (Вкладка Standard)	Properties	Caption	Ввести подпись "Произведение"

Labell (Вкладка Standard)	Properties	AutoSize (Автоподбор)	Установить значение False
		Caption	Присвоить значение 2
		Alignment (Выравнивание)	Установить значение taRightJustify (Выравнивание по правому краю)
Label2 (Вкладка Standard)	Properties	AutoSize	Установить значение False
		Caption	Присвоить значение 2
		Alignment	Установить значение taRightJustify (Выравнивание по правому краю)
Label3 (Вкладка Standard)	Properties	AutoSize	Установить значение False
		Caption	Присвоить значение 4
		Alignment	Установить значение taRightJustify (Выравнивание по правому краю)
CheckBoxl (Вкладка Standard)	Properties	Caption	Ввести подпись "Квадрат"
		Alignment	Установить значение taLefttJustify (Выравнивание по левому краю)
	Events	OnClick	TrackBar2.Position := TrackBar1.Position;

4. Сохраните проект, запустите и протестируйте его.

Листинг подпрограммы

```
procedure TForm1.TrackBar1Change (Sender: TObject); begin
Label1.Caption := IntToStr(TrackBar1.Position);
Label3.Caption := IntToStr(TrackBar1.Position * TrackBar2.Position);
if CheckBox1.Checked then TrackBar2.Position := TrackBar1.Position;
end;
procedure TForm1.TrackBar2Change (Sender: TObject); begin
Label2.Caption := IntToStr(TrackBar2.Position); Label3.Caption := IntToStr(TrackBar1.Position *
TrackBar2.Position); if CheckBox1.Checked then TrackBar2.Position := TrackBar1.Position; end;
procedure TForm1.CheckBox1Click (Sender: TObject); begin
TrackBar2.Position := TrackBar1.Position; end;
```

Задание для самостоятельного выполнения

1. Изменить программу так, чтобы находить произведения не только двузначных, но и трехзначных чисел от 2 до 199.
2. Изменить программу так, чтобы находить сумму двух чисел.

Задание №6

Цель работы - создать программу, выполняющую следующие действия:

1. После запуска программы в окне изображается две полосы прокрутки. Вертикальная полоса будет управлять движением по вертикали, горизонтальная - по горизонтали.
2. Наводя указатель мыши на одну из двух фигур, можно выбирать, какая из этих фигур связана с полосами прокрутки.

3. Требуются дополнительные объекты, с помощью которых ограничивается область движения фигур в окне.
4. Если полоса прокрутки активная, то она должна реагировать на клавиши ВВЕХ, ВНИЗ, ВЛЕВО, ВПРАВО, PAGE UP, PAGE DOWN.
5. Для выхода из программы необходимо щелкнуть мышью на закрывающей кнопке в строке заголовка.
Описание плана разработки программы
1. Открыть новый проект.
2. Разместить на форме экземпляры компонентов: панель Panel, полоса прокрутки ScrollBar, фигура Shape.
3. Ввести дополнительную переменную логического типа num. Если она принимает значение True (Да), то текущей считается первая фигура. Значению False (Нет) соответствует вторая фигура. Эта переменная должна быть доступна во всех процедурах.
4. Выполнить следующие действия:

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
Forml	Properties	Caption	Установка имени формы "Перемещение фигур"
Panel (Вкладка Standard)	Properties	Height	Присвоить значение 161
		Width	Присвоить значение 161
		Caption	Оставить значение свойства пустым
ScrollBarl (Вкладка Standard)	Properties	Min	Присвоить значение 5
		Max	Присвоить значение 145
		Position	Присвоить значение 76
		SmallChange	Присвоить значение 2

Выделенный объект	Вкладка окна Object Inspector	Имя свойства/ имя события	Действие
		LargeChange (Большое изменение)	Присвоить значение 20
	Events	OnChange	if num then Shapel.Left := ScrollBarl.Position else Shape2.Left := ScrollBarl.Position
ScrollBar2 (Вкладка Standard)	Properties	Kind	Выбрать значение sbVertical. Горизонтальная полоса прокрутки станет вертикальной.
		Min	Присвоить значение 5
		Max	Присвоить значение 145
		Position	Присвоить значение 76
		SmallChange (Малое изменение)	Присвоить значение 2
		LargeChange	Присвоить значение 20
	Events	OnChange	if num then Shapel.Top := ScrollBar2.Position else Shape2.Top := ScrollBar2.Position
Shapel (Вкладка Additional)	Properties	Height	Присвоить значение 11
		Width	Присвоить значение 11
		Left	Присвоить значение 76
		Top	Присвоить значение 76
		Shape (Форма)	Выбрать значение stCircle (Круг)
		Brush (Кисть)	Выбрать для подсвойства Color (Цвет

			кисти) значение clAqua (голубой цвет)
	Events	OnMouseMove (При движении мыши)	Shape1.Brush.Color := clAgua; Shape1.Brush.Color := clFuchsia; Num := True; ScrollBar1.Position:= Shape1.Left; ScrollBar2.Position:= Shape1.Top;
Shape2 (Вкладка Additional)	Properties Events	Height	Присвоить значение 11
		Width	Присвоить значение 11
		Left	Присвоить значение 76
		Top	Присвоить значение 76
		Shape	Выбрать значение stSquare (Квадрат)
		Brush OnMouseMove	Выбрать для подсвойства Color (Цвет кисти) значение clFuchsia (фиолетовый цвет) Аналогично Shape2

5. Сохраните проект, запустите и протестируйте его. Листинг подпрограммы

```
procedure TForml.ScrollBarlChange (Sender: TObject); begin
if num then Shapel.Left := ScrollBarl.Position else Shape2.Left := ScrollBarl.Position
end;
procedure TForml.ScrollBar2Change (Sender: TObject); begin
if num then Shapel.Top := ScrollBar2.Position else Shape2.Top := ScrollBar2.Position
end;
procedure TForml.ShapelMouseMove
(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
Shapel.Brush.Color := clAgua; Shapel.Brush.Color := clFuchsia; Num := True;
ScrollBarl.Position:= Shapel.Left; ScrollBar2.Position:= Shapel.Top; end;
procedure TForml. Shape2MouseMove
(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
Shape2.Brush.Color := clFuchsia; Shape2.Brush.Color := clAgua; Num := False;
ScrollBarl.Position:= Shape2.Left; ScrollBar2.Position: = Shape2.Top; end;
procedure TForml. FormCreate (Sender: TObject); begin
num := True; end;
```

Лабораторнные работы 13 - 14
Объектно-ориентированное программирование

Задание *1.*

Знакомство со средой разработки . Lazarus Визуальное программирование, разработка простой формы, содержащей основные компоненты GUI-интерфейса:

- TPanel, TPageControl;
- TButton, TSpeedButton, TBitBtn;
- TLabel, TEdit, TRichEdit, TListBox, TcomboBox;
- TCheckBox, TRadioGroup.

1. Создать показанную на рисунках форму.
2. Обеспечить выполнение следующих функций:

Кнопка «Очистить все» - очищает содержимое всех элементов формы и приводит их в начальное состояние.

Кнопка «Записать» - строка из поля «Текст» записывается в список «Список 1» в зависимости от выбранного значения в выпадающем списке «Способ вставки» (в начало списка, в конец списка, перед текущей строкой, после текущей строки).

Кнопка ”>” – выделенная в списке 1 строка переносится в конец списка 2.

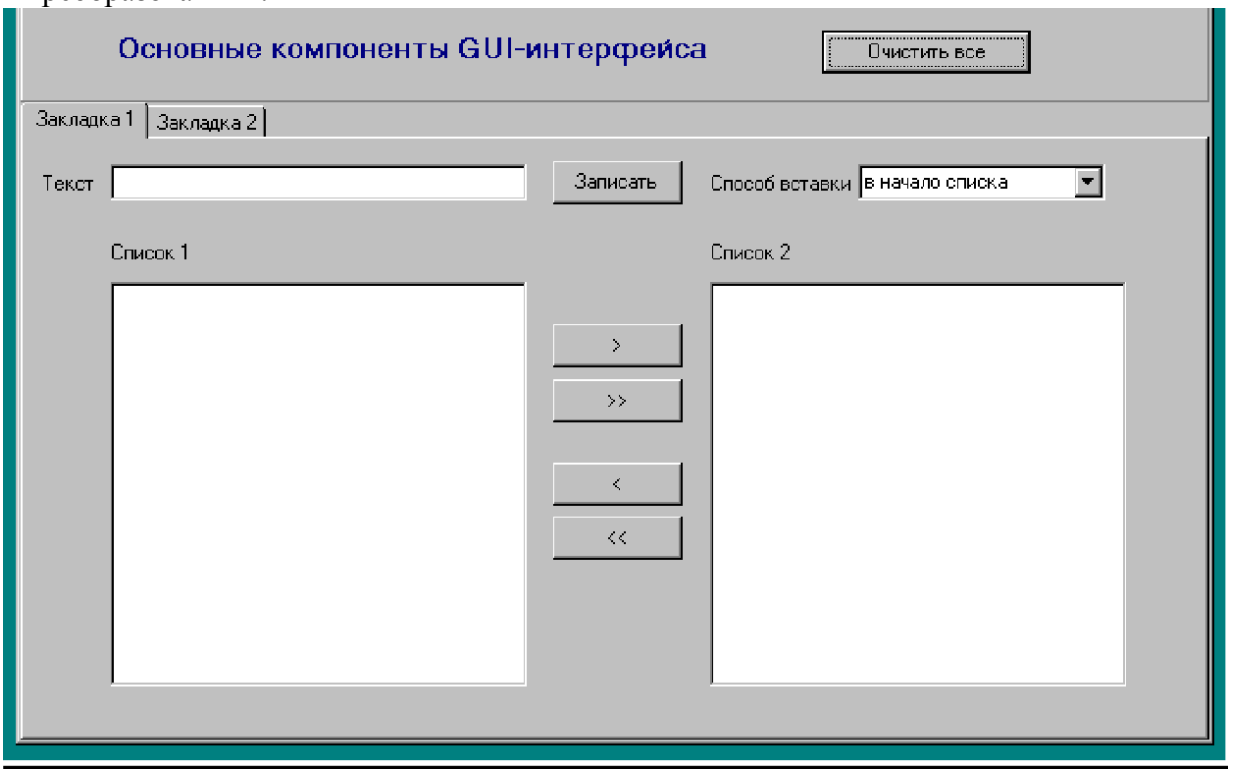
Кнопка ”<” – выделенная в списке 2 строка переносится в конец списка 1.

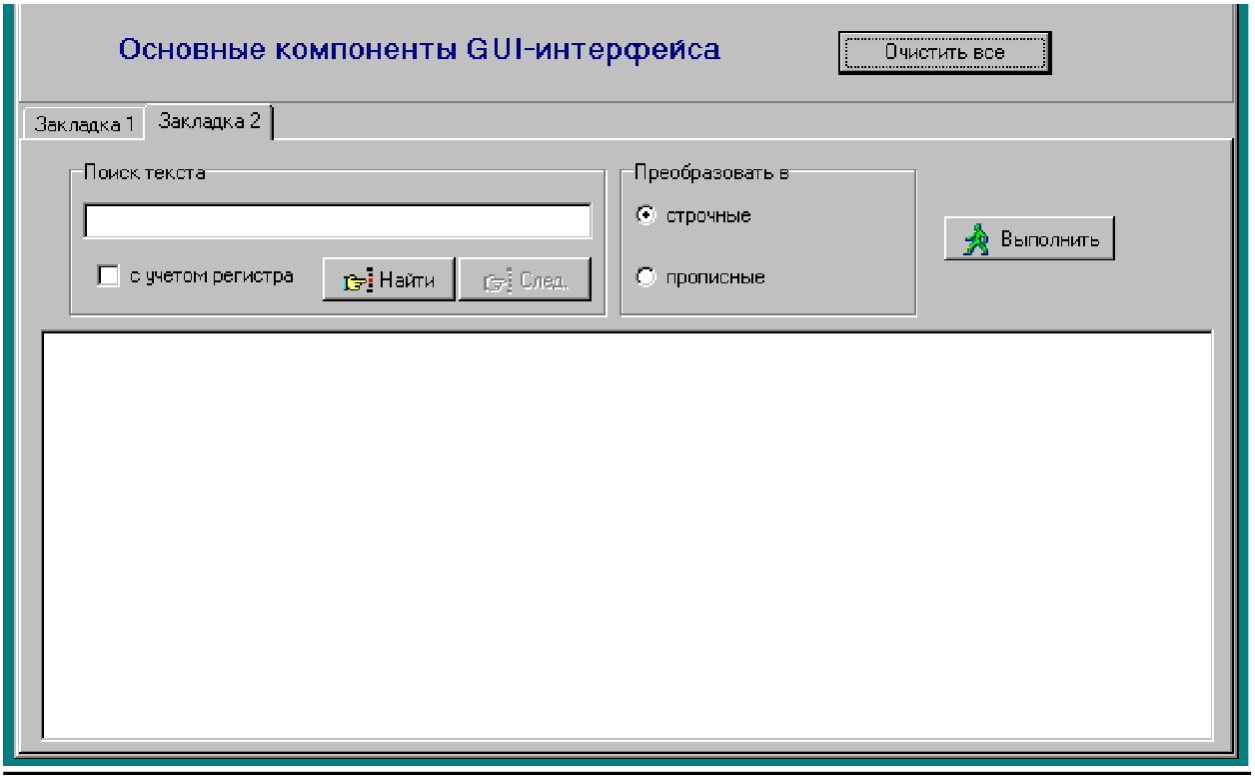
Кнопка “>>” – все строки из списка 1 переносятся в список 2.

Кнопка “<<” – все строки из списка 2 переносятся в список 1.

Кнопки «Найти», «След.» - выполняется поиск текста, набранного в поле «Поиск текста», в окне редактора, начиная с текущего положения курсора. Сравнение выполняется с учетом или без учета регистров символов в зависимости от установки поля «С учетом регистра». Если текст найден, то активной становится кнопка «След.». В противном случае – кнопка «Найти».

Кнопка «Выполнить» - все символы текста в окне редактора преобразуются в строчные или прописные буквы в зависимости от установленного значения компонента «Преобразовать в».





Компоненты общего назначения

TMemo, TRichEdit - ввод и отображение текста
Компоненты класса *TMemo* предназначены для ввода, редактирования и/или отображения достаточно длинного текста. Текст хранится в свойстве *Lines* класса *TStrings* и, таким образом, представляет собой пронумерованный набор строк (нумерация начинается с нуля). С помощью свойств и методов этого класса (Count, Add, Delete, Clear и т.д.) можно динамически формировать содержимое компонента. Многие свойства и методы этого класса аналогичны свойствам и методам класса TEdit. Свойство Text содержит отображаемый компонентом текст в виде одной длинной строки. В компонент можно загружать текст из файла с помощью метода LoadFromFile. Метод SaveToFile сохраняет текст из компонента в файле на диске.

TCheckBox - независимый переключатель
Независимый переключатель *TCheckBox* используется для того, чтобы пользователь мог указать свое решение типа *Да/Нет* или *Да/Нет/Не знаю* (в последнем случае в окошке компонента устанавливается флаг выбора, но само окошко закрашивается серым цветом). Это решение отражается в свойстве *State* компонента, доступном как для чтения, так и для записи. В составе диалогового окна может быть несколько компонентов *TCheckBox*. Состояние любого из них не зависит от состояния остальных, поэтому такие переключатели называются независимыми. Связанный с компонентом текст указывается в свойстве Caption.
Типичное использование компонента:

```
if CheckBoxl.Checked then
 . . . .
else
 . . . .
```

Или:

```
case CheckBoxl.State of
cbChecked     :...;
cbUnchecked :...;
cbGrayed      :...;
```

end;

TRadioGroup - группа зависимых переключателей

Компонент класса *TRadioGroup* представляет собой специальный контейнер, предназначенный для размещения зависимых переключателей класса *TRadioButton.* Каждый размещаемый в нем переключатель помещается в специальный список *Items* и доступен по индексу, что упрощает обслуживание группы.

Свойства компонента:

property Columns: Integer;	Определяет количество столбцов переключателей.
property ItemIndex: Integer;	Содержит индекс выбранного переключателя.
property Items: TStrings;	Содержит список строк с заголовками элементов. Добавление/удаление элементов достигается добавлением/удалением строк списка *Items*.

После размещения компонента на форме он пуст. Чтобы создать в нем хотя бы один переключатель, следует раскрыть редактор списка *Items* и ввести хотя бы одну строку: строки *Items* используются как поясняющие надписи справа от переключателей, а их количество определяет количество переключателей в группе. Замечу также, что после создания компонента его свойство *Itemlndex* по умолчанию имеет значение -1, это означает, что ни один переключатель в группе не выбран. Если в момент появления компонента на экране в каком-то переключателе выбор уже должен быть установлен, необходимо на этапе конструирования с помощью окна Инспектора Объектов или программно (например, в обработчике *OnActivate* формы) установить в свойство *Itemlndex* номер соответствующего переключателя (нумерация начинается с 0). Это же свойство позволяет программе проанализировать выбор пользователя, например:

case RadioGroupl.Itemlndex **of**
0: ...; *//Выбран 1-и переключатель*
1: ...; *//Выбран 2-й переключатель*
...
else
…. //Не выбран *ни один переключатель*
end;

TListBox - список выбора

Компонент класса *TListBox* представляет собой стандартный для *Windows* список выбора, с помощью которого пользователь может выбрать один или несколько элементов выбора. В компоненте предусмотрена возможность программной прорисовки элементов, поэтому список может содержать не только строки, но и произвольные изображения. Набор строк, показываемых в компоненте, хранится в свойстве Items. Создание элементов (опций) списка компонента реализуется с помощью методов его свойства Items – Add, Append, Insert или LoadFromFile.

TComboBox - раскрывающийся список выбора

Комбинированный список *TComboBox* представляет собой комбинацию списка *TListBox* и редактора *TEdit,* и поэтому большинство его свойств и методов заимствованы у этих компонентов. Существуют пять модификаций компонента, определяемые его свойством *Style:* csSimple, csDropDown, csDropDownList, csOwnerDrawFixed и csOwnerDrawVariable.

В первом случае список всегда раскрыт, в остальных он раскрывается после нажатия кнопки справа от редактора. В модификации csDropDownList редактор работает в режиме отображения выбора и его нельзя использовать для ввода новой строки (в других модификациях это возможно).

Свойство *DropDownCount* определяет количество элементов списка, появление которых еще не приводит к необходимости прокрутки списка. По умолчанию это свойство имеет значение 8: если в списке указано 9 и более элементов (т.е. больше, чем содержит *DropDownCount),* при его раскрытии к окну будет добавлена полоса прокрутки. Свойство *DroppedDown* определяет, раскрыт ли в данный момент список. Это свойство доступно также для записи, что позволяет программно управлять состоянием списка. Событие *OnDropDown* происходит при изменении состояния списка.
Наполнение списка ведется методами *Add, Append, Insert* и т.п. его свойства *Items* класса *TStrings*.

TPanel - панель
Компонент *TPanel* (панель) представляет собой контейнер общего назначения. В отличие от *TGroupBox* он не имеет заголовка и поэтому менее удобен для функционального группирования элементов. С другой стороны, его свойство *Caption* отображается в виде текстовой строки и может использоваться для вывода сообщений. Компоненты этого класса часто помещаются на форму для того, чтобы располагать вставленные в них дочерние компоненты вдоль одной из сторон окна независимо от изменения размеров этого окна.
Компонент имеет развитые средства создания различных эффектов трехмерности за счет использующихся в нем двух кромок - внешней и внутренней.
Свойства компонента:

TBorderStyle = bsNone . .bsSingle; property BorderStyle : TBorderStyle;	Определяет стиль рамки: *bsNone* - нет рамки; *bsSingle* - компонент по периметру обводится линией толщиной в 1 пиксель.

Для компонента объявлено событие *OnResize,* в обработчике которого программист может предусмотреть необходимую реакцию на изменение размеров компонента.

TPageControl - набор страниц с закладками
Компонент *TPageControl* может содержать несколько перекрывающих друг друга панелей класса *TTabSheet.* Каждая панель выбирается связанной с ней закладкой и может содержать свой набор помещенных на нее компонентов.
Чтобы на этапе конструирования добавить новую панель или выбрать ранее вставленную, щелкните по компоненту правой кнопкой мыши и выберите *New Page* (новая панель), *Next Page* (следующая панель) или *Previous Page* (предыдущая панель). Смена панелей идет циклически, т.е. после показа последней показывается первая и наоборот.

Примеры использования

1) Вставка строки в Список 1

```
 with ListBox1 do
  Case ComboBox1.ItemIndex of
   0 : begin // в начало списка
       Items.Insert(0,Edit1.Text);
       end;
   1 : begin // в конец списка
      Items.Add(Edit1.Text);
      end;
   2 : begin // перед текущей  строкой
       if ItemIndex<0 then ItemIndex:=0; // Если никакая строка не выделена
       Items.Insert(ItemIndex,Edit1.Text);
      end;
   3 : begin // после текущей строки
       if ItemIndex<0 then ItemIndex:=0; // Если никакая строка не выделена
       Items.Insert(ItemIndex+1,Edit1.Text);
```

```
    end;
  end;
```

2) Копирование строки из Списка 1 в Список2

```
 if ListBox1.Count=0 then begin // Список 1 пуст
 ShowMessage('Список 1 пуст!');
  exit;
 end;
 if ListBox1.ItemIndex<0 then begin // В Списке 1 нет выбранной строки
  ShowMessage(‘Выберите строку в Списке 1');
  exit;
 end;
// Добавление строки из Списка 1 в конец Списка 2
 ListBox2.Items.Add(ListBox1.Items[ListBox1.itemIndex]);
 ListBox2.ItemIndex:=ListBox2.Count-1; // Текущей становится вставленная строка

 // Запоминаем и изменяем позицию текущей строки в Списке 1, т.к. скопированная строка
// будет удалена
 i:=ListBox1.ItemIndex;
 if i=(ListBox1.Items.Count-1) then // Если скопирована последняя строка
   ListBox1.ItemIndex:=i-1
 else  ListBox1.ItemIndex:=i+1;
 ListBox1.Items.Delete(i); // Удаление скопированной строки из Списка 1
```

3) Преобразование в строчные и прописные буквы

```
  case RadioGroup1.ItemIndex of
   0 : RichEdit1.Text:=AnsiLowerCase(RichEdit1.Text);
   1 : RichEdit1.Text:=AnsiUpperCase(RichEdit1.Text);
  end;
```

4) Поиск текста

```
  with RichEdit1 do
  begin
    { Поиск начинается с текущей позиции курсора в тексте.
      Если курсора нет , то с перовй позиции }
    if SelLength > 0 then // Длина выделенного фрагмента текста
      StartPos := SelStart + SelLength
    else
      StartPos := 0;
    { SelStart – начало выделенного фрагмента в тексте
      StartPos – позиция, с которой начинается поиск
      ToEnd – количество символов, которые необходимо просмотреть}

    ToEnd := Length(Text) - StartPos;

    if CheckBox1.Checked then // с учетом регистра символов
     FoundAt := FindText(Edit2.Text, StartPos, ToEnd, [stMatchCase]) // метод поиска строки
    else
     FoundAt := FindText(Edit2.Text, StartPos, ToEnd, []);
     // FoundAt – позиция начала найденной строки в тексте
    if FoundAt <> -1 then
    begin
      // Выделение в тексте найденного фрагмента
      SetFocus;
      SelStart := FoundAt;
```

```
    SelLength := Length(Edit2.Text);
    // Активизация/деактивизация кнопок поиска
    BitBtn1.Enabled:=false;
    BitBtn3.Enabled:=true;
  end
  else begin
    ShowMessage('Строка не найдена. Задайте начальную позицию и повторите поиск.');
    // Активизация/деактивизация кнопок поиска
    BitBtn1.Enabled:=true;
    BitBtn3.Enabled:=false;
  end;
end;
```

5) Очистка элементов формы и приведение в исходное состояние

```
Edit1.Text:='';
Edit2.Text:='';
ComboBox1.ItemIndex:=0;
ListBox1.Items.Clear;
ListBox2.Items.Clear;
RichEdit1.Clear;
CheckBox1.Checked:=false;
RadioGroup1.ItemIndex:=0;
```

Задание *2.*

- Использование главного и выпадающих меню.
- Обработка исключительных ситуаций.
- Динамические массивы.

1. Разработать приложение «Калькулятор», который может выполнять операции +, -, /, * над целыми и вещественными числами, а также сохранять операнды и результаты в памяти, организованной по принципу стека.

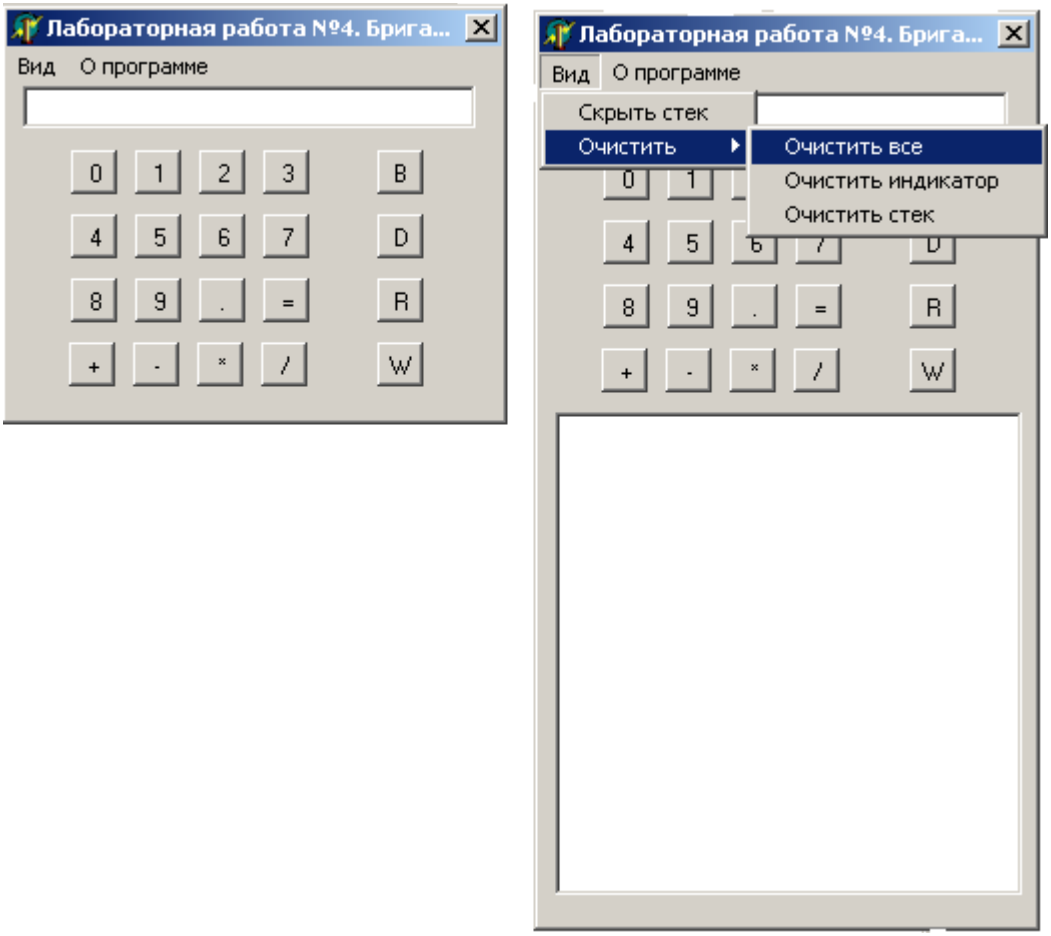
Кнопки

B – Backspace, удаляет символ слева от курсора
D – Delete, удаляет символ справа от курсора
R – Read, извлечение числа из стека
W – Write, запись числа в стек.

2. Для контроля над операциями использовать обработку исключительных ситуаций.
3. Для организации стека использовать динамический массив.
4. Для отображения содержимого стека использовать TListBox.
5. Программа должна иметь главное меню, содержащее пункты:
   Вид
   Показать/скрыть стек.
   Очистить.
   Очистить все
   Очистить индикатор
   Очистить стек

5.2. О программе (вызов модальной формы About).

6. Поле редактирования (индикатор) должно иметь выпадающее меню, содержащее пункты:
   Очисть
   Поместить в стек
   Загрузить из стека



Теоретическая часть

TMainMenu - главное меню формы (программы)

Компонент класса *TMainMenu* определяет главное меню формы. На форму можно поместить сколько угодно объектов этого класса, но отображаться в полосе меню в верхней части формы будет только тот из них, который указан в свойстве *Menu* формы. После установки компонента на форму необходимо создать его опции (пункты меню). Для этого следует дважды щелкнуть по компоненту левой кнопкой мыши, либо нажать на нем правую кнопку и выбрать пункт *Menu Designer* в появившемся вспомогательном меню.

Каждая опция главного меню может раскрываться в список подопций или содержать конечную команду. В названиях опций (свойство Caption) можно указать символ «&» перед тем символом, который определит клавишу быстрого выбора опции (в терминологии *Windows* такие клавиши называются акселераторами). Например, если опция *Файл* в строке *Caption* Инспектора Объектов содержит текст &Файл, то ее можно выбрать сочетанием клавиш *Alt+Ф*.

Свойства BitMap и ImageIndex позволяют связывать с опциями пиктограммы.

Если необходимо вставить разделительную черту, отделяющую группы подопций, то нужно очередной элемент меню назвать именем "-".

Для элемента меню определено единственное событие *OnClick,* которое возникает при щелчке на опции или при нажатии *Enter*, если в этот момент данная опция была выбрана (подсвечена).

TPopupMenu - вспомогательное (локальное) меню

Компоненты класса *TPopupMenu* используются для создания вспомогательных (локальных) меню, появляющихся после нажатия правой кнопки мыши. В отличие от главного меню вспомогательное меню может быть создано для любого оконного компонента. Чтобы связать щелчок правой кнопкой мыши на компоненте с раскрытием вспомогательного меню, в свойство *PopupMenu* компонента необходимо поместить имя компонента-меню.

Вспомогательное меню создается с помощью конструктора меню и процесс создания и свойства вспомогательного меню ничем не отличаются от *TMainMenu.*

Одномерные динамические массивы

Динамические массивы отличаются от обычных статических массивов тем, что в них не объявляется заранее длина — число элементов. Поэтому динамические массивы удобно использовать в приложениях, где объем обрабатываемых массивов заранее неизвестен и определяется в процессе выполнения в зависимости от действий пользователя или объема перерабатываемой информации.

Объявление динамического массива содержит только его имя и тип элементов — один из базовых типов. Например

var A: array of integer;

объявляет переменную A как динамический массив целых чисел.

При объявлении динамического массива место под него не отводится. Прежде, чем использовать массив, надо задать его размер процедурой **SetLength.** В качестве аргументов в нее передаются имя массива и целое значение, характеризующее число элементов. Например

SetLength(A,10);

выделяет для массива A место в памяти под 10 элементов и задает нулевые значения всех элементов.

Индексы динамического массива — всегда целые числа, начинающиеся с 0. Таким образом, в приведенном примере массив содержит элементы от **A[0]** до **A[9].**

Повторное применение **SetLength** к уже существующему в памяти массиву изменяет его размер. Если новое значение размера больше предыдущего, то все значения элементов сохраняются и просто в конце добавляются новые нулевые элементы. Если же новый размер меньше предыдущего, то массив усекается и в нем остаются значения первых элементов. Например, для приведенной ниже программы размерность и значения элементов массива при выполнении различных операторов показаны в комментариях к тексту.

```
var      A: array of integer;
N,i :    integer;
begin
N:=5;
SetLength(A,N); {массив A(0,0,0,0,0)}
for i:=0 to N do A[i]:=i+l; {массив A(1,2,3,4,5)}
N:=7;
SetLength(A,N); {массив A(1,2,3,4,5,0,0)}
N:=3;
SetLength(A,N); {массив A(1,2,3)}
N:=4;
SetLength(A,N); {массив A(1,2,3,0)}
end;
```

Усечение динамического массива лучше проводить функцией **Copy,** присваивая ее результат самому массиву. Например, оператор

A := Copy(A, 0, 3);

усекает динамический массив A, оставляя неизменными первые три его элемента.

Если динамический массив уже размещен в памяти, к переменной этого массива можно применять стандартные для массивов функции **Length** — длина, **High** — наибольшее значение индекса (очевидно, что всегда **High = Length** — 1) и **Low** — наименьшее значение индекса (всегда 0). Если массив имеет нулевую длину, то **High** возвращает -1, т.е. при этом получается, что **High < Low.**

Сама переменная динамического массива является указателем на начало массива. Если место под массив еще не выделено, значение переменной равно nil. Но его нельзя разыменовывать операцией ^, нельзя передать в процедуры **New** и **Dispose.**

Удалить из памяти динамический массив можно одним из следующих способов: присвоить ему значение nil, использовать функцию **Finalize** или установить нулевую длину. Таким образом, эквивалентны следующие операторы:

A := nil;
или
Finalize(A);
или
SetLength(A,0);
Если динамические массивы определены как переменные одного типа, например
var A,B: array of integer;
и размер массива A не меньше размера массива B или A = nil, то
возможно присваивание вида
B := A;
которое приводит к тому, что переменная B начинает указывать на тот же самый массив, что и A, т.е. получается как бы два псевдонима для одного массива. А содержимое массива B при этом теряется. В этом коренное различие присваивания статических и динамических массивов.
Если динамические массивы объявлены не как переменные одного типа, т.е.
var A: array of integer;
B: array of integer;
то присваивание
B := A;
вообще не допускается.
В операциях сравнения динамических массивов сравниваются только сами указатели, а не значения элементов массивов. Таким образом, выражение A = B вернет **true** только в случае, если A и B указывают на один и тот же массив. Выражение **A[0] = B[0]** сравнивает значения первых элементов двух массивов.
Динамические массивы могут передаваться в качестве параметров в те функции и процедуры, в описаниях которых параметр объявлен как массив базового типа без указания индекса, т. е. открытый массив. Например, функция
function CheckStrings(A: array of string): Boolean;
может работать в равной степени и со статическими, и с динамическими массивами.

Обработка исключений в блоках try ... except

При работе программы могут возникать различного рода ошибки: переполнение, деление на нуль, попытка открыть несуществующий файл и т.п. При возникновении таких исключительных ситуаций программа генерирует так называемое *исключение* и выполнение дальнейших вычислений в данном блоке прекращается. Исключение — это объект специального вида, характеризующий возникшую в программе исключительную ситуацию. Он может также содержать в виде параметров некоторую уточняющую информацию. Особенностью исключений является то, что это сугубо временные объекты. Как только они обработаны каким-то обработчиком, они разрушаются.
Если исключение не перехвачено нигде в программе, то оно обрабатывается методом **TApplication.HandleException.** Он обеспечивает стандартную реакцию программы на большинство исключений — выдачу пользователю краткой информации в окне сообщений и уничтожение экземпляра исключения.
Наиболее кардинальный путь борьбы с исключениями — обработка их с помощью блоков **try ... except.** Синтаксис этих блоков следующий:

```
try
    {Исполняемый код}
except
    {Код, исполняемый в случае генерации исключения}
end;
```

Операторы раздела **except** выполняются только в случае генерации исключения в операторах блока **try**. Таким образом, вы можете предпринять какие-то действия: известить пользователя о возникшей проблеме и подсказать ему пути ее решения, принять какие-то меры к

исправлению ошибки (например, при делении на нуль заслать в результат очень большое число соответствующего знака) и т.д.
Наиболее ценным является то, что в разделе **except** вы можете определить тип сгенерированного исключения и дифференцирование реагировать на различные исключительные ситуации. Это делается с помощью оператора:
on <класс исключения> do <оператор>;
Полная таблица классов исключений, которые вы можете использовать в этом операторе, приведена в Справочной системе.
Операторы **on...do** позволяют проводить выборочную обработку различных исключений.
Приведем пример такой обработки:

```
var A : shortint;
…
try
…
C := StrToInt(Editl.text);
A := B div C;
….
except
on EConvertError do
  MessageDlg('Вы ввели ошибочное число; повторите ввод',
mtWarning, [mbOk], 0);
on EDivByZero do
  MessageDlg('Вы ввели нуль; повторите
ввод',mtWarning,[mbOk],0);
on EIntOverflow do
   if (B*C) >= 0 then A := 32767 else A := -32767;
end;
```

Можно перехватывать не только отдельные исключения, но и группы родственных исключений. Дело в том, что, предопределенные в Lazarus исключения построены по иерархическому принципу. Базовым классом исключений является класс **Exception.** Другие исключения являются производными или непосредственно от него, или от промежуточных производных классов. Например, имеется класс исключений **EIntError**, производный от **Exception** и являющийся родительским для ряда исключений, связанных с ошибками целочисленных арифметических операций, например, для **EDivByZero и EIntOverflow.** Поэтому можно обработать сразу группу родственных исключений, используя, например, такой оператор:

```
on EIntError  do
MessageDlg('Ошибка целочисленной операции', mtWarning, [rnbOk] , 0);
```

Этот оператор перехватит исключения всех классов, порожденных от **EIntError.**
Помимо операторов **on,** обрабатывающих заданные типы исключений, в раздел **except** может быть включен оператор else, в котором выполняется обработка всех не перехваченных ранее исключений, т. е. происходит обработка по умолчанию. Например:

```
except
  on .. . ;
  on . . . ;
  else <обработчик всех не перехваченных ранее событий>;
end;
```

Конечно, оператор else должен идти последним, так как в противном случае все обработчики, записанные после него, работать не будут, поскольку все исключения уже будут перехвачены. Внутри обработчика по умолчанию можно получить доступ к объекту исключения, воспользовавшись функцией ExceptObject.

Следует указать на некоторую опасность применения else в этом контексте. Перехват всех исключений способен замаскировать какие-то непредвиденные ошибки в программе, что затруднит их поиск и снизит надежность работы.
В раздел except могут включаться или только операторы on, или только какие-то другие операторы. Смешение операторов on с другими не допускается.
Раздел except может включаться совместно с описанным ранее разделом finally. Например, можно организовать блоки try следующим образом:

```
try
…
  try
    …
   finally
     …
end;
except
  ….
end;
```

Последовательность обработки исключений

Блоки try...except могут быть вложенными явным или неявным образом. Примером неявной вложенности является блок try...except, в котором среди операторов раздела try имеются вызовы функций или процедур, которые имеют свои собственные блоки try...except. Рассмотрим последовательность обработки исключений в этих случаях. При генерации исключения сначала ищется соответствующий ему обработчик on в том блоке try...except, в котором создалась исключительная ситуация. Если соответствующий обработчик не найден, поиск ведется в обрамляющем блоке try...except (при наличии явным образом вложенных блоков) и т. д. Если в данной функции или процедуре обработчик не найден или вообще в ней отсутствуют блоки try...except, то по иск переходит на следующий уровень — в блок, из которого была вызвана данная функция или процедура. Этот поиск продолжается по всем уровням. И только если он закончился безрезультатно, выполняется стандартная обработка исключения, заключающаяся в выдаче пользователю сообщения о типе исключения.
Как только оператор on, соответствующий данному исключению, найден и выполнен, объект исключения разрушается и управление передается оператору, следующему за тем блоком **try...except,** в котором был осуществлен перехват.
Возможен также вариант, когда в самом обработчике исключения в процессе обработки возникла исключительная ситуация. В этом случае обработка прерывается, прежнее исключение разрушается и генерируется новое исключение. Его обработчик ищется в блоке **try...except,** внешнем по отношению к тому, в котором возникло новое исключение.

Для кнопок, соответствующих цифрам, целесообразно использовать общий обработчик события onClick, так как реакция программы на нажатие любой из этих кнопок должна быть одинаковой. Значение соответствующей цифры лучше всего хранить в свойстве Tag каждой кнопки. В обработчике события для определения нажатой кнопки использовать выражение (Sender as TSpeedButton).Tag
Sender – параметр процедуры обработки события, показывает, какой компонент получил событие и вызвал обработчик.

Задание 3.

Классы и модули

ТЕОРЕТИЧЕСКИЙ МАТЕРИАЛ

Модули и программы

Приложение Lazarus создается из файлов двух разных видов. Это один файл программы (*.dpr) и один или несколько модулей (*.pas).
Файл программы является основным – он связывает воедино все модули. Просмотреть содержимое файла программы можно, выполнив команду Project -> View Source. Как правило, код программы генерируется автоматически, поэтому многие начинающие программисты его не замечают.
Модули можно считать вторичными файлами, к которым обращается основная часть приложения – программа.

Модули

Приложения Lazarus интенсивно используют модули. За каждой формой скрывается соответствующий ей модуль. Однако модули не обязаны иметь соответствующие формы.
Модуль состоит из двух основных разделов - interface, где объявлено все, что доступно для других модулей (процедуры, функции, переменные) и раздела implementation с реальным кодом процедур и функций.
Кроме того, модуль может иметь два необязательных раздела: initialization с некоторым кодом запуска, который выполняется при загрузке в память программы, использующей данный модуль, и finalization, который выполняется при завершении программы.
В начале раздела interface может находиться предложение uses. Оно указывает к каким другим модулям мы должны получить доступ из раздела interface текущего модуля. Если же на другие модули необходимо сослаться из кода подпрограмм и методов, вы должны добавить новое предложение uses в начале раздела implementation.
В интерфейсе модуля можно объявить несколько различных элементов, в том числе процедуры, функции, глобальные переменные и типы данных. Также можно поместить в модуль класс.
Чтобы создать новый не относящийся к форме модуль, выберите команду File/New и отметьте на странице New появившегося окна Object Repository элемент Unit.

Классы и сокрытие информации

Класс может содержать сколько угодно данных и любое количество методов. Однако для соблюдения всех правил объектно-ориентированного подхода данные должны быть скрыты, или инкапсулированы, внутри использующего их класса. Использование метода для получения доступа к внутреннему представлению объекта уменьшает риск возникновения ошибочных ситуаций и позволяет автору класса модифицировать внутреннее представление в будущих версиях. В Object deal имеются две различные конструкции, подразумевающие инкапсуляцию, защиту и доступ к переменным: классы и модули. С классами связаны некоторые специальные ключевые слова – спецификаторы доступа:

- private – элементы интерфейса класса видны только в пределах модуля, в котором определен класс. Вне этого модуля private-элементы не видны и недоступны. Если в одном модуле создано несколько классов, то все они видят private-разделы друг друга.
- Раздел public не накладывает ограничений на область видимости перечисленных в нем полей, методов и свойств. Их можно вызывать в любом другом модуле программы.
- Раздел published не ограничивает область видимости, однако в нем перечислены свойства, которые должны быть доступны не только на этапе исполнения, но и на этапе конструирования программы. Раздел published используется при разработке компонентов. Без объявления раздел считается объявленным как published. Такой умалчиваемый раздел располагается в самом начале объявления класса любой формы и продолжается до первого объявленного раздела. В раздел published среда помещает описание вставляемых в форму компонентов. Сюда не нужно помещать собственные элементы или удалять элементы, вставленные средой.
- Раздел protected доступен только методам самого класса, а также любым потомкам, независимо от того, находятся ли они в том же модуле или нет.

Порядок следования разделов может быть любой, любой раздел может быть пустым.

Информация о типе на этапе выполнения

Правило языка Object Pascal о совместимости типов для классов-потомков позволяет вам использовать класс-потомок там, где ожидается класс - предок, обратное невозможно. Теперь предположим, что класс Dog содержит функцию Eat, которая отсутствует в классе Animal.
Если переменная MyAnimal ссылается на объект типа Dog, вызов этой функции должен быть разрешен. Но если вы попытаетесь вызвать эту функцию, а переменная ссылается на объект другого класса, возникнет ошибка. Поскольку компилятор не способен определить, будет ли значение правильным на этапе выполнения, то, делая явное приведение типов, мы рискуем вызвать опасную ошибку этапа выполнения программы. Для решения данной проблемы можно воспользоваться некоторыми подходами, основанными на системе RTTI.
Каждый объект знает свой тип и своего предка и может получить эту информацию с помощью операции is. Параметрами операции is являются объект и тип:
if MyAnimal is Dog then …
Выражение is становится истинным, только если в настоящее время объект MyAnimal имеет тип Dog или тип потомка от Dog. Другими словами, это выражение приобретает значение True, если вы можете без риска присвоить объект (MyAnimal) переменной заданного типа данных (Dog).
Такое прямое приведение можно выполнить так:

```
if MyAnimal is Dog then
  MyDog := Dog (MyAnimal) ;
```

То же действие можно выполнить напрямую с помощью другой операции RTTI – as. Мы можем написать так:

```
MyDog := MyAnimal as Dog ;
Text := MyDog. Eat ;
```

Если мы хотим вызвать функцию Eat, можно использовать и другую форму записи:

```
(MyAnimal as Dog) . Eat ;
```

Результатом выражения будет объект с типом данных класса Dog, поэтому вы можете применить к нему любой метод этого класса.
Приведение с операцией as отличается от традиционного приведения тем, что в случае несовместимости типа объекта с типом, к которому вы пытаетесь его привести, порождается исключение EInvalidCast. Чтобы избежать этого исключения, используйте операцию is и в случае успеха делайте простое приведение:

```
if MyAnimal is Dog then
(Dog (MyAnimal) ) . Eat ;
```

ЗАДАНИЕ

Создать программу, позволяющую выводить информацию о еде и голосе животных собака и кошка. Описание животных оформить в виде класса.

ВЫПОЛНЕНИЕ ЗАДАНИЯ

1) Создать новый проект (File/New Application).
2) Сохранить проект в папке Lab8 (Unit1.pas под новым именем Main.pas, а Project1.dpr под новым именем Lab.dpr).
3) Открыть модуль, не связанный с формой (File/New/Unit), и поместить в него три класса:

⧫ Класс Animal, который содержит в разделе public объявление конструктора Create и объявление метода-функции: Voice – звук, издаваемый животным. Тип результата возвращаемого функцией, – string. Метод Voice объявить виртуальным и абстрактным. В разделе private класса определить переменную Kind: string.

```
unit Unit1;

interface

type
 TAnimal=class
  private
    Kind:string;
  public
    constructor Create;
    function Voice: string; virtual; abstract;
  end;
```

⧫ Класс Dog объявить потомком класса Animal: TDog = class (TAnimal). В разделе public этого класса объявить конструктор и методы Voice и Eat. Метод Eat типа string объявить виртуальным (пища животного).

```
TDog=class(TAnimal)
 public
   constructor Create;
   function Voice: string; override;
   function Eat: string; virtual;
 end;
```

⧫ Класс Cat объявить потомком класса Animal: TCat = class (TAnimal). Раздел public класса содержит те же определения, что и соответствующий раздел класса Dog.

```
TCat=class(TAnimal)
 public
   constructor Create;
   function Voice: string; override;
   function Eat: string; virtual;
 end;
```

В реализациях конструктора каждого класса (раздел Implementation) переменной Kind присваивается имя соответствующего животного, например, для класса Animal:

```
implementation

constructor TAnimal.Create;
begin
 kind:='An Animal';
end;
```

Для конструкторов классов TDog и TCat код будет аналогичен (допишите его сами).

В реализациях методов Voice возвращается звук, издаваемый животным, например, для класса TDog (для класса TCat – аналогично):

```
function TDog.voice;
begin
 Result:='gav';
end;
```

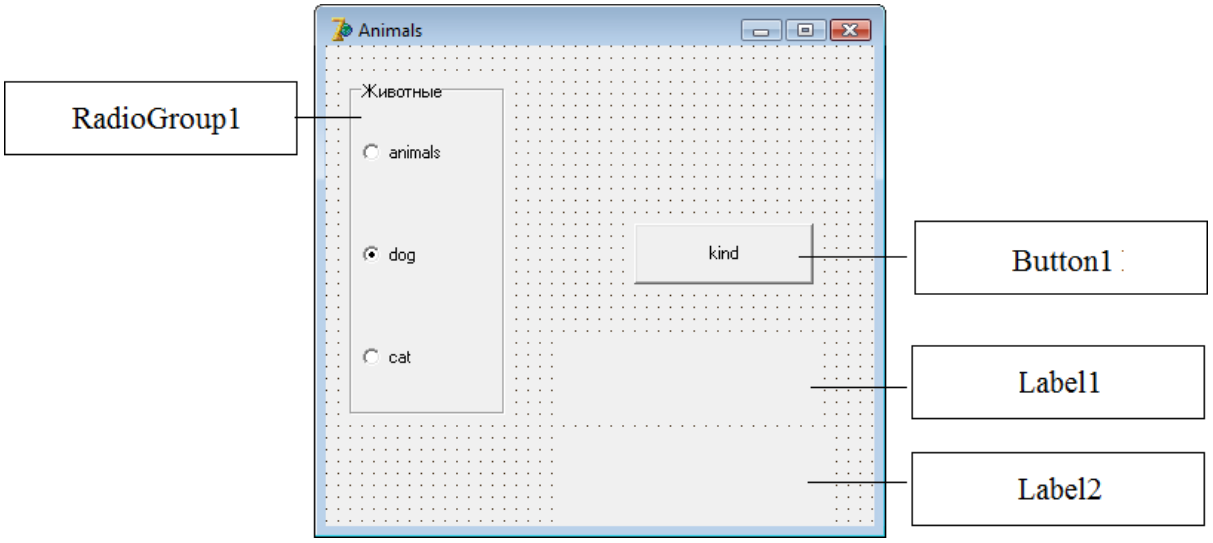
Переменная Result возвращает значение в функцию.
Поскольку в классе TAnimal метод Voice объявлен абстрактным, его реализация не требуется.

В реализациях методов Eat возвращается название пищи, которой питается соответствующее животное:

```
function TDog.eat;
begin
 Result:='кость';
end;
```

4) Сохранить созданный модуль.

5) На форме расположить компоненты, задать свойства аналогично рисунку (обратите внимание, что в RadioGroup второй пункт выбран по умолчанию – свойство ItemIndex):



Выбору одной из кнопок опций будет соответствовать выбор животного.
При нажатии кнопки команды надписи должны отобразить звук, издаваемый животным, и его пищу.

Определить в классе формы private-переменную MyAnimal:

```
type
  TForm1 = class(TForm)
    RadioGroup1: TRadioGroup;
    Button1: TButton;
    Label1: TLabel;
    Label2: TLabel;
    procedure FormCreate(Sender: TObject);
    procedure RadioGroup1Click(Sender: TObject);
    procedure Button1Click(Sender: TObject);
  private
    MyAnimal:TAnimal; { Private declarations }
  public
    { Public declarations }
  end;
```

6) Записать код для обработчика события OnCreate формы, где создается объект типа Dog, на который ссылается переменная MyAnimal:

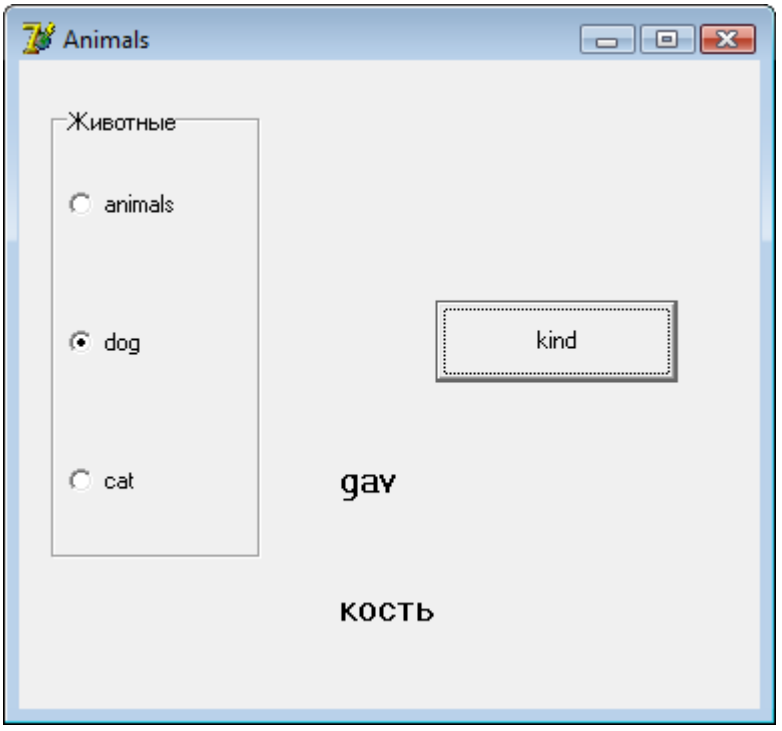
```
procedure TForm1.FormCreate(Sender: TObject);
begin
     MyAnimal := TDog.Create;
end;
```

7) В обработчике события OnClick компонента RadioGroup записать код, который удаляет текущий объект и создает новый в зависимости от выбранного пункта:

```
procedure TForm1.RadioGroup1Click(Sender: TObject);
begin
 MyAnimal.Free;
 Case RadioGroup1.ItemIndex of
  0: MyAnimal := TAnimal.Create;
  1: MyAnimal:=TDog.Create;
  2: MyAnimal:=TCat.Create;
 end;
end;
```

8) В обработчике события OnClick кнопки записать код, который будет помещать в надписи звук, издаваемый животным и его еду:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
 Label1.Caption := MyAnimal.Voice;
 if MyAnimal is TDog then Label2.Caption:=(TDog (MyAnimal) ) . Eat else
 if MyAnimal is TCat then Label2.Caption:=(TCat (MyAnimal) ) . Eat else
 Label2.Caption:='';
end;
```



Если вы все сделали правильно, при запуске приложения надписи будут отображать пищу и звук для Dog и Cat и приложение завершит работу по ошибке при выборе Animal.
Уберите ключевое слово abstract в объявлении метода Voice и реализуйте его, например, так:

```
function TAnimal.voice;
begin
 Result:='ay';
end;
```

Запустите приложение снова. Посмотрите, что изменилось в работе приложения. Объясните различия.

9) Попробуйте использовать метод Eat без приведения типов (без is).

10) Сохраните проект.

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ.

Доработайте проект следующим образом:

1) дополните каждый класс животных новой характеристикой – среда обитания.

2) разработайте два класса потомка от TAnimal, которые будут отображать особенности двух новых животных Fish (рыба) и Bird (птица).
3) доработайте форму так, чтобы можно было увидеть все характеристики разработанных классов.

Лабораторная работа № 15-16, МАТРИЦЫ

Постановка задачи

Создайте программу, которая в зависимости от величин N (количество строк) и M (количества столбцов) создает матрицу размером NxM. Программа предоставляет возможность заполнить матрицу с помощью случайных чисел или ввести значения вручную. Кроме этого можно подсчитать сумму элементов матрицы, определить максимальный и минимальный элементы матрицы.

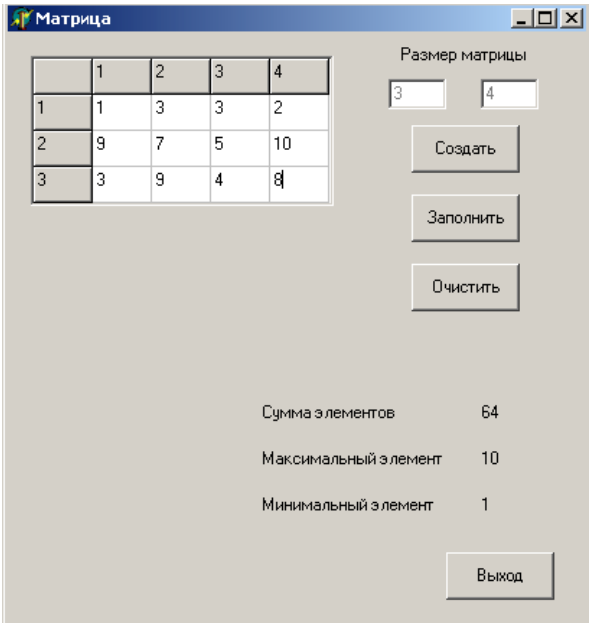


Рис.1

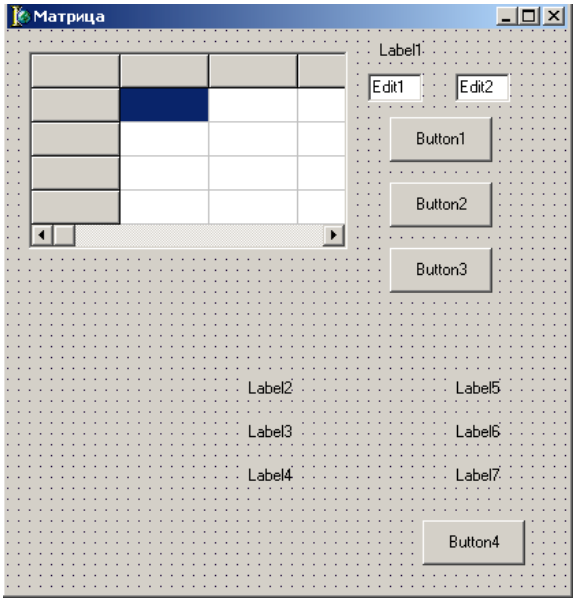


Рис.2

Новым в этой работе являются:

- компонент таблица строк **StringGrid** вкладки палитры компонентов **Additional**.

План разработки программы

1. Откройте новый проект.
2. Разместите в форме объекты в соответствии с рис.2.
3. Установите свойства компонент на вкладке **Object Inspector**:

Выделенный объект	Имя свойства	Значение
Label1	Caption	Размер матрицы
Edit1	Text	Удалить название объекта
Edit2	Text	Удалить название объекта
Button1	Caption	Создайте
Button2	Caption	Заполнить
Button3	Caption	Удалить название объекта
Label2	Caption	Сумма элементов
Label3	Caption	Максимальный элемент
Label4	Caption	Минимальный элемент
Label5	Caption	Удалить название объекта
Label6	Caption	Удалить название объекта

Label7	Caption	Удалить название объекта
Button4	Caption	Выход
StringGrid1	Name	MATR

4. Сохраните код программы и проект под именами, например, **Unit_M.pas** и **Pr_M.dpr**.

Немного теории

Компонент **StringGrid** (вкладка палитры компонентов **Additional**) предназначен для создания таблицы.

Свойству компонента **Cells** соответствует двухмерный массив ячеек, каждая из которых может содержать произвольный текст. Содержание ячейки массива, находящегося на пересечении столбца с номером **Col** и строки с номером **Row** определяется элементом **StringGrid 1.Cells [Col, Row]**. Это содержимое можно редактировать.

Двумерный массив состоит из двух частей: фиксированной и рабочей.

Фиксированная часть служит для показа заголовков столбцов (строк) и для ручного управления их размерами. Обычно фиксированная часть занимает крайний левый столбец и самую верхнюю строку таблицы. С помощью свойств **FixedCols** и **FixedRows** можно задать другое количество фиксированных столбцов и строк. Если свойства **FixedCols** и **FixedRows** имеют значение 0, то таблица не содержит фиксированной зоны.

Рабочая часть – это остальная часть таблицы. Она может содержать произвольное количество столбцов и строк, которое можно изменять в ходе выполнения программы. Если рабочая часть не умещается целиком в пределах окна компонента, то автоматически появляются полосы прокрутки. При прокрутке рабочей области фиксированная область не исчезает.

В таблице приведен перечень часто используемых свойств компонента **StringGrid.**

ColCount	Количество столбцов таблицы
RowCount	Количество строк таблицы
FixedCols	Количество столбцов фиксированной зоны
FixedRows	Количество строк фиксированной зоны
DefaultRowHeight	Высота строки таблицы
Height	Высота всей таблицы
DefaultColWidth	Ширина столбца таблицы
Width	Ширина всей таблицы
Options.goEditing	Признак редактирования содержимого ячеек таблицы. True – редактирование разрешено, False – запрещено
GridLineWidth	Ширина линий, ограничивающих ячейки таблицы
Font	Шрифт, используемый для отображения содержимого ячеек таблицы
Options .goEditing	Признак допустимости редактирования содержимого ячеек таблицы. True – редактирование разрешено, False – запрещено
Options .goTabs	Разрешает (True) или запрещает (False) использование клавиши «Tab» для перемещения курсора в следующую ячейку таблицы

5. Разместите в блоке реализации после слова **implementation** описание переменных:

```
Var
    N, M: Integer;     //N-количество строк, M-количество столбцов
    MATR_MAX,          //Значение максимального элемента таблицы
    MATR_MIN,          //Значение минимального элемента таблицы
    MATR_SUM: Integer;//Значение суммы элементов таблицы
```

6. Основная задача проекта – создать таблицу, размер которой будет определен после

заполнения полей **Edit1** и **Edit2**.
Создадим следующие процедуры обработки событий:

Выделенный объект	Имя событие	Текст процедуры

Button4 «Выход»	OnClick	procedure TForm1.Button4Click(Sender: TObject);
		begin Close;
		end;

Form1	OnCreat	procedure TForm1.FormCreate(Sender: TObject);
		MATR.Visible:=False;
		Button2.Enabled:=False; Button3.Enabled:=False;
		end;
		Комментарий
		При создании формы установим компонент **StringGrid** невидимым (новое имя этого компонента **MATR**), т.к. в начале неизвестно сколько же строк и столбцов в таблице. Кроме этого до определения размера таблицы установим недоступными кнопки «Заполнить» и «Очистить».

Button1 «Создать таблицу»	OnClick	procedure TForm1.Button1Click(Sender: TObject);
		Var I,J:Byte; begin
		If (Edit1.Text<>’’) and (Edit2.Text<>’’) Then begin
		Edit1.Enabled:=False; Edit2.Enabled:=False;
		Button1.Enabled:=False; Button2.Enabled:=True;
		Button3.Enabled:=True;
		
		N:=StrToInt(Edit1.Text); M:=StrToInt(Edit2.Text);
		
		MATR.DefaultColWidth:= 40; MATR.RowCount:=N+1;
		MATR.ColCount:=M+1;
		MATR.Height:=(MATR.DefaultRowHeight+2)*(N+1);
		MATR.Width:= (MATR.DefaultColWidth +2)*(M+1);
		MATR.Visible:=True;
		For I:=1 to N do MATR.Cells[0,I]:=IntToStr(I);
		For J:=1 to M do MATR.Cells[J,0]:=IntToStr(j);

		end; end; **Комментарий** Данная процедура будет выполняться только при условии, что введены размеры матрицы, т.е **Edit1.Text** и **Edit2.Text** не являются «пустым символом». В начале устанавливаются недоступными компоненты ввода, определяющие размер таблицы, и кнопка «Создать», а недоступные ранее кнопки «Заполнить» и «Очистить» становятся доступными. После этого переходим к формированию таблицы. Для этого устанавливаем значения переменных N и M, определяющих количество строк и столбцов таблицы – переводим из текстовой информации значения полей **Edit1** и **Edit2** в числовые значения. После этого программно задаем свойства компонента **StringGrid** (**MATR**) и устанавливаем его видимым. В конце заполняем фиксированную часть таблицы значениями номеров строк и столбцов.

Button2 «Заполнить таблицу»	OnClick	procedure TForm1.Button2Click(Sender: TObject);
		Var I,J:Byte; begin Randomize;
		For I:=1 to N do For J:=1 to M do
		MATR.Cells[J,I]:=IntToStr(Random(N*M));
		
		MATR_MAX:= StrToInt(MATR.Cells[1,1]); MATR_MIN:= StrToInt(MATR.Cells[1,1]); MATR_SUM:=0;
		For I:=1 to N do For J:=1 to M do
		begin MATR_SUM:=MATR_SUM+StrToInt(MATR.Cells[J,I]);
		IF StrToInt(MATR.Cells[J,I])>MATR_MAX Then
		MATR_MAX:=StrToInt(MATR.Cells[J,I]);
		IF StrToInt(MATR.Cells[J,I])<MATR_MIN Then
		MATR_MIN:=StrToInt(MATR.Cells[J,I]);
		end; Label5.Caption:=IntToStr(MATR_SUM);
		Label6.Caption:=IntToStr(MATR_MAX);
		Label7.Caption:=IntToStr(MATR_MIN); end;
		Комментарий
		В начале процедуры заполняются ячейки таблицы (MATR.Cells[J,I]) случайными числами в диапазоне от 0 до (N*M-1). Дальше определяются максимальное и минимальное числа и сумма элементов таблицы. В конце процедуры полученные значения выводятся на экран. Для этого используются компоненты **Label5**, **Label6**, **Label7**. Для заполнения ячеек таблицы как положительными, так и отрицательными

		числами, например, в диапазоне от -10 до 10 можно записать: MATR.Cells[J,I]:=IntToStr(Random(11))- 10;

Button3 «Очистить таблицу»		procedure TForm1.Button3Click(Sender: TObject);
		Var I,J:Byte; begin
		For I:=1 to N do For J:=1 to M do
		MATR.Cells[J,I]:= ’’;
		Label5.Caption:= ’’; Label6.Caption:= ’’; Label7.Caption:= ’’;
		Edit1.Text:= ’’;
		Edit2.Text:= ’’;
		
		Matr.Visible:=False; Button2.Enabled:=False;
		Button3.Enabled:=False; Button1.Enabled:=True;;
		Edit1.Enabled:=True; Edit2.Enabled:=True; end;
		Комментарий
		Все ячейки таблицы заполняется пустой строкой, а затем очищаются компоненты **Label5**, **Label6**, **Label7**, **Edit1.Text**, **Edit2.Text**. Компонент **StringGrid** (**MATR**) устанавливается невидимым, кнопки «Заполнить» и «Очистить» становятся недоступными, кнопка «Создать» и компоненты **Edit1** и **Edit2** – доступными.

7. Программа готова. Но если в поля ввода «Размерность матрицы» мы по ошибке вместо числа введем букву или какой-либо другой символ, то программа будет прервана. Для того чтобы это избежать создадим процедуры обработки события **KeyPress** для компонента **Edit1**.

```
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key:Char);
begin
case Key of
  ’0’..’9’: ;
  #8:       ;
  #13: Edit2.SetFocus;
  else Key := Chr(0);
end;
```

Комментарий

В зависимости от нажатой клавиши программа выполнит следующие действия:

- нажата любая цифровая клавиша или клавиша «Back Space» (код клавиши #8)– программа ничего не изменит,
- нажата клавиша «Enter» (код клавиши #13) – курсор перейдет в окно редактора ввода **Edit2**,
- во всех остальных случаях – введенный символ будет удален (присвоение значения пустого символа Chr(0)).

Для компонента **Edit2** самостоятельно создайте процедуру, обрабатывающую ввод. Она будет отличаться тем, что при нажатии клавиши «Enter», курсор должен перейти на кнопку «Создать таблицу» (**Button1**).

8. Созданная программа не позволяет редактировать элементы таблицы. Внесем изменения в свойства компонента **StringGrid** (**MATR**). Для этого на вкладке **Object Inspector** свойству

Options.goEditing установим значение **True**. Процедура, обрабатывающая нажатие клавиши в поле компонента **StringGrid** (**MATR**) будет выглядеть так:

```
procedure TForm1.MATRKeyPress(Sender: TObject; var Key: Char);
Var I,J:Byte;
begin
 case Key of
  '0'..'9':   ;
  #8:         ;
  #13: if MATR.Col<MATR.ColCount-1 then MATR.Col:=MATR.Col+1;
  else Key:=Chr(0);
 end;
MATR_MAX:= StrToInt(MATR.Cells[1,1]);
MATR_MIN:= StrToInt(MATR.Cells[1,1]);
MATR_SUM:=0;
For I:=1 to N do
 For J:=1 to M do
  begin
  MATR_SUM:=MATR_SUM+StrToInt(MATR.Cells[J,I]);
  IF StrToInt(MATR.Cells[J,I])>MATR_MAXThen
MATR_MAX:=StrToInt(MATR.Cells[J,I]);
  IF StrToInt(MATR.Cells[J,I])<MATR_MINThen
MATR_MIN:=StrToInt(MATR.Cells[J,I]);
  end;
Label5.Caption:=IntToStr(MATR_SUM);
Label6.Caption:=IntToStr(MATR_MAX);
Label7.Caption:=IntToStr(MATR_MIN);

end;
```

Комментарий

Обработка нажатия допустимой клавиши рассмотрена в п.7. Отличие заключается в действии, которое произойдет, если нажата клавиша «Enter» - переход к следующей ячейке в данной строке, если эта ячейка не последняя в строке.

Дальше в процедуре идет подсчет максимального, минимального элемента и суммы элементов в таблице, после внесения изменений.

9. Сохраните проект окончательно, запустите и протестируйте его.

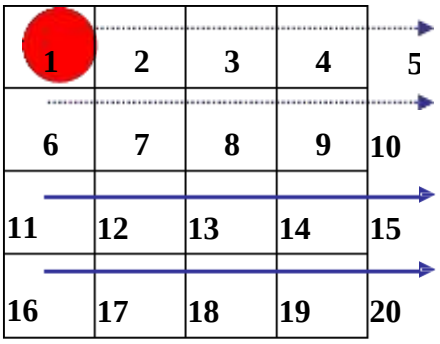
Задание для самостоятельного выполнения

1. Дополните программу, вставив блок определения суммы по каждому столбцу матрицы. *Подсказка*. Необходима еще одна таблица (компонент **StringGrid**), в которой будут находиться подсчитанные суммы по столбцам. Формировать эту таблицу нужно в момент формирования основной таблицы.

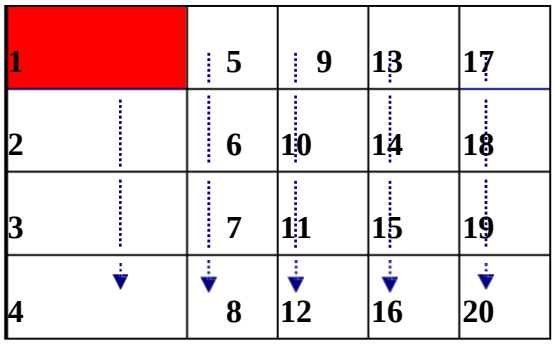
Продолжение работы

Доработать программу так, чтобы можно было бы заполнять матрицу различными способами:

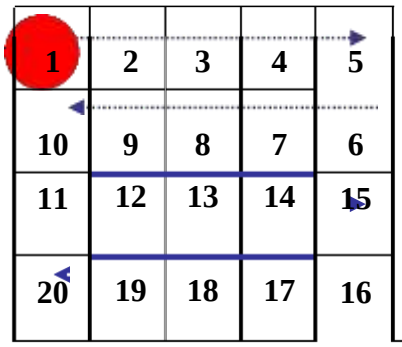
1 способ



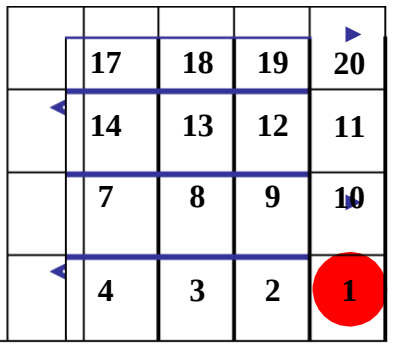
2 способ



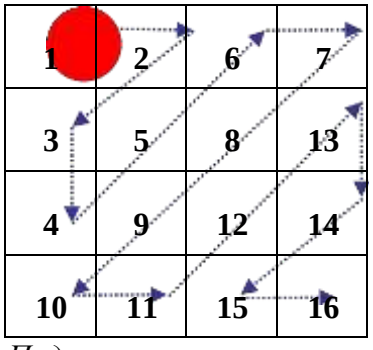
3 способ



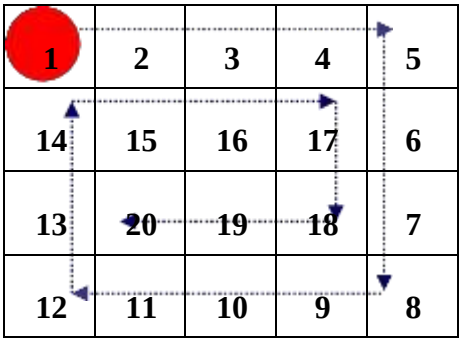
4 способ



5 способ



6 способ



Подсказка.

Удалите кнопку *Заполнить таблицу*, добавив отдельные процедуры для заполнения каждым способом.

Вначале необходимо внести изменения в раздел объявления:

```

```

Пример процедуры для заполнения с помощью случайных чилел может выглядеть так:

```
Procedure Z0 (Var MAS: T_MAS);
Var I,J:Byte;
begin
 Randomize;
 For I:=1 to N do
  For J:=1 to M do
   MAS[J,I]:=Random(N*M);
End.
```

Созданные вами процедуры должны располагаться в тексте программы сразу же после раздела объявления переменных.

Для выбора способа заполнения можно воспользоваться компонентой **ListBox.**

Для заполнения выделите объект **ListBox1**, найдите свойство **Items**, щелкните на кнопке с тремя точками, расположенной справа от него. В появившемся окне встроенного редактора **String List Editor** введите названия способов заполнения, каждый на новой строке. Например:

```
Случайными числами
По горизонтали
По вертикали
Змейкой слева
Змейкой справа
Зигзагом
Спиралью
```

Комментарий

а) Свойство **Items** содержит элементы списка.

б) Список может быть создан при создании формы или во время работы программы.

в) Свойство **ItemIndex** определяет номер элемента, выбранного из списка. Первый элемент имеет номер 0. Если не выбран ни один из элементов, то значение свойства **ItemIndex** равно – «-1».

Сохраните набранный текст в файле под именем ZAPOLN.txt. Для этого нажмите правую кнопку мыши и выберите режим **Save**. Для выхода из встроенного редактора щелкните на кнопке **OK**.

Комментарий

Просмотреть содержимое созданного текстового файла ZAPOLN.txt, можно с помощью любого текстового редактора, а также внести изменения в тестовый файл, не используя встроенный редактор Lazarus.

Выполните следующие действия:

Выделенный объект	Вкладка окна **Object Inspector**	Имя свойства/ Имя события	Значение/Действие

ListBox1	Events	OnKeyPress	if key=#13 then
			Case Listbox1.ItemIndex of
			0: Z0(MAS);
			1: Z1(MAS);
			2: Z2(MAS);
			3: Z3(MAS);
			4: Z4(MAS);
			5: Z5(MAS);
			6: Z6(MAS);
			end;
			For I:=1 to N do For J:=1 to M do
			MATR.Cells[J,I]:=IntToStr(MAS[I,J]);

Лабораторная работа №17-18, Угадай слово

Постановка задачи
Разработать программу, которая реализует логическую игру «Угадай слово», известную каждому с детства.

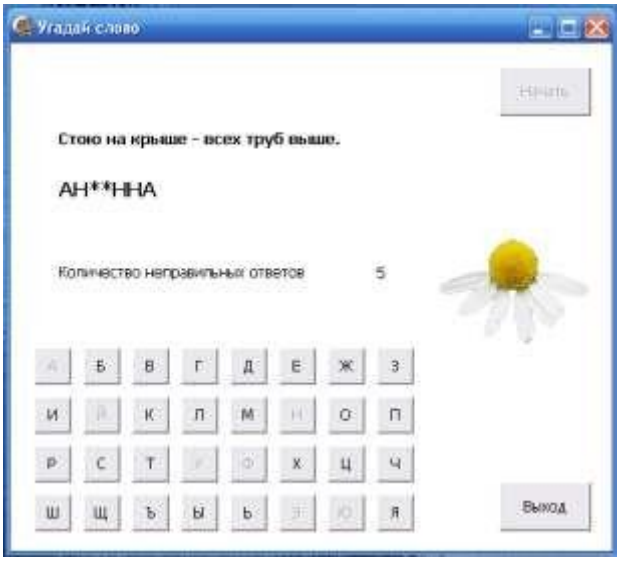


Рис.1

Правила игры

Ведущий загадывает слово, которое надо угадать, и записывает черточки, количество которых равно количеству букв в загаданном слове.
(Для подсказки можно использовать вопрос, ответом на который является загаданное слово.)
Игроку предоставляется право выбора букв по одной, при этом он может допустить только десять ошибок (попыток).
Игрок называет букву, которая на его взгляд может присутствовать в слове. Если выбранная игроком буква есть в загаданном слове, то ведущий записывает ее вместо черточки на своем месте. Если буква встречается в слове несколько раз, то ведущий «открывает» все эти буквы. Если буква названная игреком отсутствует в слове, то уменьшается количество предоставленных попыток.
Игра заканчивается, если слово угадано, или если исчерпаны все предоставленные попытки.

Новым в этой работе являются:

- создание компонент во время выполнения программы и обработка их событий,
- вывод иллюстраций.

Информационная постановка задачи

1. Информация о вопросах и ответах хранится в текстовом файле (Questions.txt). На каждый вопрос-ответ отводится две строки: в первой строке - вопрос, а во второй - ответ.
2. Информация из текстовых файлов в начале программы считывается в массив (AQ), из которого затем случайным образом выбираются тексты вопросов.
3. Выбранный вопрос выводится на экран, а ответ записывается в переменную STR_R.
4. Формируется угадываемое слово в виде черточек (переменная STR_N). Количество черточек соответствует количеству букв (например, STR_N='----- ').
5. Для выбора отгадываемых букв слова используются кнопки, которые создаются в ходе программы.
6. При нажатии на кнопку с буквой, которая есть в слове, эта буква появляется на нужном месте, т.е. меняется переменная STR_N (например, STR_N='-A-A--'). Если такой буквы нет, то уменьшается количество попыток (переменная ATTEMPT). Кнопка, содержащая нажатую букву, в дальнейшем она становится недоступной.
7. При угадывании слова (STR_R = STR_N) игроку предоставляется возможность начать новую игру или выйти из игры.
8. При использовании всех попыток (ATTEMPT=0) на экране появляется неугаданное слово и предоставляется возможность начать новую игру или выйти из игры.

План разработки программы Построение программы будем поэтапно. **Первый этап. Формирование формы экрана**

1. Откройте новый проект.
2. Разместите в форме экземпляры компонентов в соответствии с рис.2.

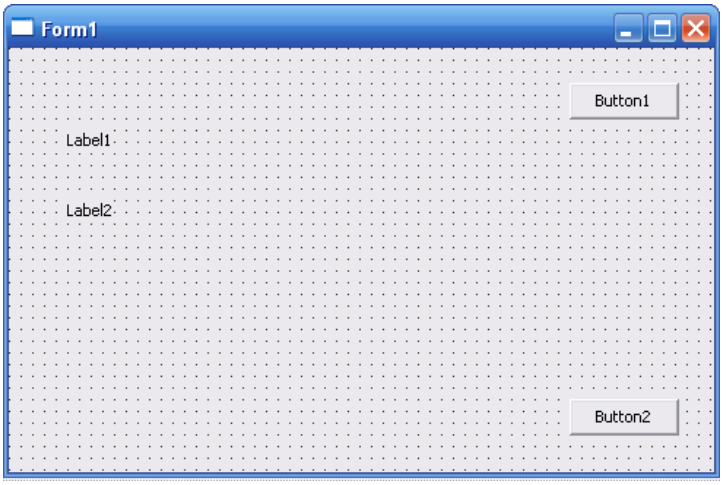


Рис.2

3. Выделите объект **Form1**, перейдите на вкладку **Events Инспектора объектов** (**Object**

Inspector), найдите событие **OnCreat**, справа от него дважды щелкнуть левой кнопкой мыши. Попав в код программы, надо написать следующий текст:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
// Формирование элементов формы
  With Form1 do
  begin
   Caption:='Угадай слово';
   Height:=450;
   Width:=500;
   Color:=clWhite;
  end;
  With Button1 do
  begin
    Caption:='Начать';
    Height:=40;
    Width:=75;
    Top:=20;
    Left:=400;
    Font.Size := 9;
```

```
  end;
  With Button2 do
  begin
    Caption:='Выход';
    Height:=40;
    Width:=75;
    Top:=360;
    Left:=400;
    Font.Size := 9;
  end;
  With Label1 do
  begin
    Caption:='';
    Height:=40;
    Width:=250;
    Top:=70;
    Left:=40;
    Font.Style:=[fsBold];
    Font.Size := 10;
  end;
  With Label2 do
  begin
    Caption:='';
    Height:=40;
    Width:=75;
    Top:=110;
    Left:=40;
    Font.Style:=[fsBold];
    Font.Size := 12;
  end;
 Read_File; //Процедура чтения информации из файла
end;
```

Пояснение

Для компонент **Form1**, **Button1**, **Button2**, **Label1**, **Label2** в тексте программы определены все необходимые свойства (размеры, местоположение и т.п.), поэтому при размещении компонент нет необходимости устанавливать свойства компонент с помощью вкладки **Properties** (Свойства) инспектора объектов.

4. Разместите в блоке реализации после слова **implementation** описание констант и переменных:

```
Const
 N=100;     //Максимальное количество записей в массиве
TYPE
 T_R = record       //Структура записи массива
         Que:string[250];
         Ans:string[30];
        end;
 R=array[1..N]of T R;


Var AQ:R;                        //Массив Вопросов и Ответов
    Quetions F:TextFile;         //Файловая переменная
    STR_R,STR_N:String[30]; //Отгадываемое слово
    KOL QUE:Integer;             //Количество вопросов в файле
```

5. Разместите после блока описания переменных процедуру чтения информации из файла и формирование массива вопросов **AQ**. К этой процедуре происходит обращение в конце процедуры **TForm1.FormCreate** (см.п.3).

6. Создадим процедуру, которая обрабатывает ситуацию нажатия кнопки **Button1** Начать игру). Для этого выделите объект **Button1**, перейдите на **вкладку Events Инспектора объектов** (**Object Inspector**), найдите событие **On Click**, справа от него дважды щелкнуть левой кнопкой мыши. Попав в код программы, надо написать следующий текст:

```
procedure TForm1.Button1Click(Sender: TObject); Var
I,NUMBER:integer;
begin
 Randomize;
 NUMBER:=Random(KOL_QUE-1)+1;
 Label1.Caption:=AQ[NUMBER].QUE;
 STR_R:=AQ[NUMBER].ANS; STR_N:='';
 for I := 1 to Length(STR_R) do
 STR_N:=STR_N+'*';
 Label2.Caption:=STR_N;
 Form1.Button1.Enabled:=False;
end;
```

Пояснение

Начало игры означает, что нужно выбрать вопрос, который затем выводиться на экран (**Label1**), и сформировать зашифрованное слово (**Label2**, **STR_N**), которое будет отгадывать игрок. Выбор вопроса из массива производится с помощью функции Random.

7. Самостоятельно добавьте событие обработки кнопки **Button2** (Выход).
8. Создайте текстовый файл **Quetions.txt**, записав его в ту же папку, где находится и программа. Текст файла может выглядеть так:

Стою на крыше - всех труб выше.

АНТЕННА

Всех на свете обшивает, а сама не надевает.

ИГОЛКА

Твой хвостик я в руке держал, ты полетел, я побежал.

ШАРИК

Золотое решето, черных домиков полно. ПОДСОЛНУХ Ах, не трогайте меня. Обожгу и без огня! КРАПИВА

9. Сохраните проект, запустите и протестируйте его. После запуска программы на экране видны две кнопки - **Начать** и **Выход**. Если нажать кнопку **Начать**, то появляется текст вопроса и зашифрованное слово ответа, при этом кнопка **Начать** становится недоступной. Проверьте, чтобы при каждом новом запуске программы у вас всегда выбирался новый текст ответа.

Второй этап. Создание компонент во время выполнения программы и обработка их событий

Для дальнейшей работой над проектом нам необходимо создать 33 кнопки, которые будут принимать значения букв русского алфавита, и обрабатывать ситуацию поиска выбранной буквы (нажатой кнопки). Эта задача не сложная, но очень утомительная - 33 раза повторить одни и теже операции. Как упростить этот процесс покажем на отдельном примере.

В данном примере по нажатию кнопки **Button1** создаются 32 кнопки, которые располагаются в три ряда. Свойства **Caption** этих кнопок принимают значения букв русского алфавита от «А» до «Я» (кроме буквы «Ё»). По нажатию каждой кнопки идет обращение к процедуре **TForm1.BtnClick**, которая определяет какая буква русского алфавита соответствует нажатой кнопке. Дальше идет обращение к процедуре **Poisk**, и нажатая кнопка становится недоступной. Поцедура **Poisk** ищет встречается ли выбранная буква (нажатая кнопка) в заданном слове **STR_N**, и если да, то заменяется в **STR_R** соответствующий символ на заданную букву.

1. Откройте новый проект.
2. Разместите в форме экземпляры компонентов **Label1** и **Button1**.
3. Для события **OnClick** компоненты **Button1** напишите такой текст:

```
procedure TForm1.Button1Click(Sender: TObject); var
 i: integer;
begin
for i:=1 to N do              // Цикл по созданию кнопок на форме
begin
  Btn:=TButton.Create(Form1);
  with Btn do
  begin            // Определение свойств создаваемых кнопок
    Parent := Form1; // Определение родителя,
                      //т.е. где создаются кнопки
    Caption := Chr(191+I);//Определение буквы по ее коду
    Height := 30;
    Width := 30;
    Top := 200+((I-1) div 11)*40; Left
    := ((I-1) mod 11 ) * 40 + 30;
    Font.Style:=[fsBold];
    OnClick := FORM1.BtnClick;      // Имя процедуры,
                        //обрабатывающей событие нажатия кнопки
  end;
```

4. Процедуру **TForm1.BtnClick** (обработка события нажатия созданных кнопок) мы создаем сами. Для этого вставьте текст этой процедуры в общий текст программы перед тем, как осуществляется обращение к процедуре **TForm1.BtnClick**.

```
procedure TForm1.BtnClick(Sender: TObject);
// Процедура обработки события нажатия созданных кнопок
Var STR_:String;
begin
STR_:= (Sender as TButton).Caption;
// Переменная Sender содержит имя
// объекта, которому соответствует данное событие
CHAR_:=STR_[1];
POISK(STR_R,STR_N,CHAR_); //Обращение процедуре поиска буквы
(Sender as TButton).Enabled:=False; //Кнопка буквы недоступна end;
```

5. В раздел описания процедур (перед **private**) добавьте строку описания заголовка процедуры:

```
procedure BtnClick(Sender: TObject);
```

6. Для того, чтобы наша программа заработала необходимо внести объявление переменных и описание процедуры **Poisk**.

```
 Const N=32;                  //Количество букв-кнопок
 Type STR_30=String[30];
 Var Btn: TButton;            // Переменной для создания кнопок
      STR_N,                  // Зашифрованное слово
      STR_R:STR_30;,          // Отгадываемое слово
      CHAR_:Char;             // Нажатая буква
    {$R *.dfm}


Procedure POISK(STR_R:STR_30;Var STR_N:STR_30;CHAR_:Char);
//Поиск буквы в слове и «открытие» ее
Var I:Integer;
       Flag:Boolean;         // Флаг найдена ли в слове нажатая буква
  begin
  Flag:=False;
  For I:=1 to Length(STR_R) do      // Поиск буквы в слове
   If STR_R[I]=CHAR_ then
     begin
      STR_N[I]:=CHAR_;
      Flag:=True;
     end;
  If Flag=True Form1.Label1.Caption:= STR_N;

  end;
```

7. Сохраните проект, запустите и протестируйте его. После запуска программы на экране видна только одна кнопка **Button1** и закодированное слово в виде звездочек. После нажатия на эту кнопку на экране появляется три ряда кнопок с буквами (см.рис.3) и кнопка **Button1** становится недоступной.

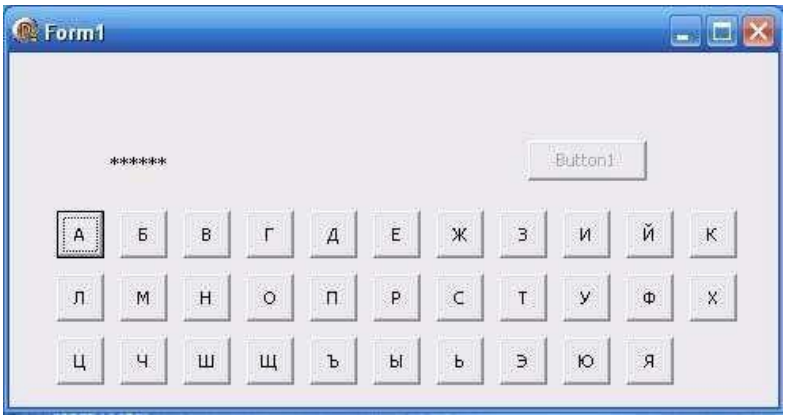


Рис.3

Нам известно слово, которое зашифровано - это слово «РОССИЯ». Протестируйте программу, введя буквы этого слова, и проверьте, чтобы они у вас «открылись» в зашифрованном слове.

Третий этап. Создание кнопок перед началом игры

Выполните этот этап самостоятельно, внеся изменения в основной текст программы, созданный на втором этапе. При создании кнопок расположите их не в три ряда, а в четыре (см.рис.4). Подумайте, как это сделать.

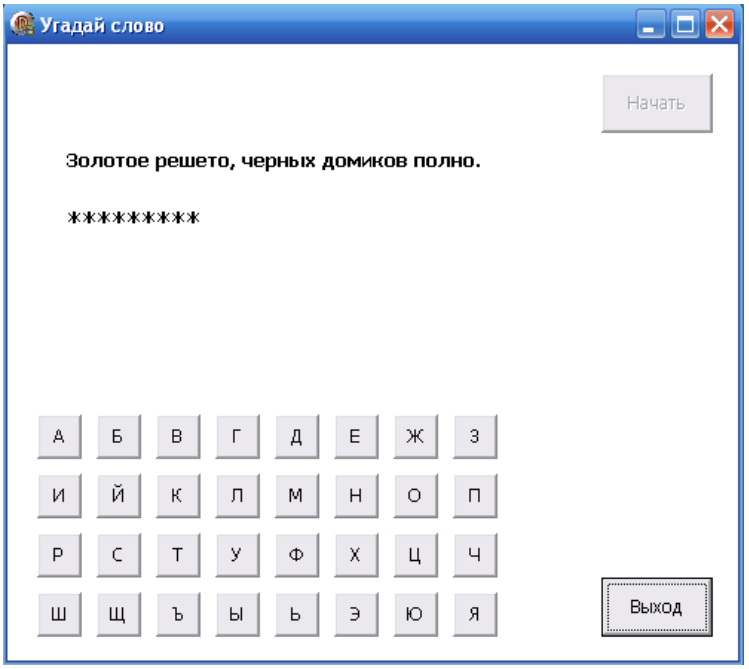


Рис.4

Четвертый этап. Анализ состояния игры

Необходимо внести изменения в программу, для определения закончена ли игра, т.е. угаданы все буквы в зашифрованном слове. Вместе с тем установим количество допустимых ошибок игрока.

1. Разместите в блоке реализации описание дополнительных констант и переменных:

```
Const
  Attempt=10; //Максимальное количество попыток
```

2. Разместите в форме экземпляры компонентов **Label3, Label4** (сообщение о количестве допущенных ошибок) и **Panel1** (сообщение о завершении игры).
3. Дополните текст процедуры **TForm1.FormCreate** описанием новых компонент:

```
With Label3 do
  begin
    Caption:='';
    Height:=40;
    Width:=75;
    Top:=180;
    Left:=40;
    Font.Size := 9;
  end;
  With Label4 do
  begin
    Caption:='';
    Height:=40;
    Width:=75; Top:=180;
    Left:=300;
    Font.Style:=[fsBold];
    Font.Size := 10;
  end;
  With Panel1 do
  begin
    Caption:='';
    Height:=45;
    Width:=185;
    Top:=15;
    Left:=110;
    Font.Style:=[fsBold];
    Font.Size:= 10;
    Visible:=False;
  end;
```

4. Вначале процедуры **TForm1.Button1Click** добавьте следующий текст:

```
 Form1.Panel1.Visible:=False;
 Label3.Caption:='Количество неправильных ответов';
 N_Attempt:=0;
 Label4.Caption:=IntToStr(N_Attempt);
```

5. Внесите изменения в текст процедуры поиска POISK для анализа результата. Процедура может выглядеть следующим образом:

```
Procedure POISK(STR_R:STR_30;Var STR_N:STR_30;CHAR_:Char);
//Поиск буквы в слове и «открытие» ее
Var I:Integer;
Flag:Boolean; // Флаг найдена ли в слове нажатая буква
begin
Flag:=False;
For I:=1 to Length(STR_R) do begin// Поиск буквы в слове
   If STR_R[I]=CHAR_ then
    begin
     STR_N[I]:=CHAR_;
     Flag:=True;
    end;
```

```
 end;
If Flag=False Then
 begin N_Attempt:=N_Attempt+1;
  Form1.Label4.Caption:=IntToStr(N_Attempt);
 end
 Else Form1.Label2.Caption:= STR_N;
If STR_N=STR_R then      //Игра окончена - отгадано все слово
 begin
 With Form1.Panel1 do
  begin
   Visible:=True;
   Font.Color:=clBlack;
   Caption:='Вы выиграли!';
  end; Form1.Button1.Enabled:=True;
  end;
 If N_Attempt=Attempt then
//Игра окончена - количество ошибок равно количеству попыток
 begin
  With Form1.Panel1 do
   begin
    Visible:=True;
    Font.Color:=clRed;
    Caption:='Вы проиграли...';
   end;
  Form1.Button1.Enabled:=True;
  end;
end;
```

6. Сохраните проект, запустите и протестируйте его.

Пятый этап. Вывод иллюстраций

Для того, чтобы программа была более привлекательной, добавим вывод измененных изображений в зависимости от количества допущенных ошибок. Ряд изменения изображений может выглядеть так:




0 ошибок 1 ошибка 2 ошибки 3 ошибки и т.д.

Осталось только добавить эти изображения в программу, но сначала создадим самостоятельную программу, которая будет выводить изображения в зависимости от нажатия кнопки.

Немного теории

Для вывода иллюстрации используется компонент **Image**, который находится на вкладке **Additional** палитры компонентов.

В таблице приведены основные свойства компонента **Image**.

Имя свойства	Значение
Name	Имя компонента
Picture	Свойство, являющееся объектом типа **Tbitmap**. Определяет выводимую картинку
Left	Расстояние от левого края формы до левой границы области картинки
Top	Расстояние от верхней границы формы до верхней границы области картинки
Height	Высота картинки
Width	Ширина картинки
Stretch	Признак автоматического сжатия или растяжения картинки таким образом, чтобы она была видна полностью в области, размер которой задан свойствами **Width** и **Height**
AutoSize	Признак автоматического изменения размера компонента в соответствии с реальным размером картинки. Если значение свойства **AutoSize** равно **True**, то при изменении значения свойства Picture автоматически меняется размер области вывода иллюстрации так, чтобы была видна вся картинка. Если значение свойства **AutoSize** равно **False**, а размер картинки превышает размер области, то отображается только часть картинки

Картинку, отображаемую в области **Image**, можно задать во время создания формы или во время работы программы:

- Во время создания формы картинка задается установкой значения свойства **Picture.**
- Во время работы программы - применением метода **LoadFromFile.** Например, для вывода иллюстрации, находящейся в файле **dog.bmp**:

Image1.Picture.LoadFromFile('dog.bmp');

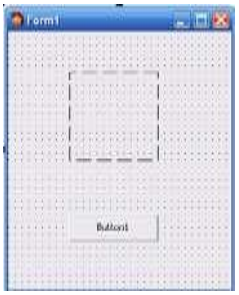
При проектировании формы можно жестко задать предельный размер иллюстрации. И если реальный размер иллюстрации превышает размер области, выделенной для ее вывода, то необходимо вычислить коэффициент масштабирования и установить максимально возможные, пропорциональные ширине и высоте иллюстрации, значения свойств **Width** и **Height** области вывода иллюстрации. А если размер иллюстрации меньше области вывода, то можно пропорционально увеличить картинку.

Реальные размеры иллюстрации, загруженной в область **Image**, можно получить из свойств: **Image1.Picture.Bitmap.Width Image1.Picture.Bitmap.Height.**

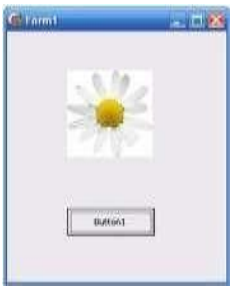
Пример программы вывода картинок

Программа позволяет последовательно выводить на экран при нажатии на кнопку **Button1** три картинки, находящиеся в файлах **1.bmp**, **2.bmp**, **3.bmp**. После последней картинки будет выведена опять первая - **1.bmp**. При выводе на экран размер картинки масштабируется в зависимости от заданного размера компанента **Image**.

Откройте новый проект. Разместите на форме экземпляры компонентов **Image** и Button.



Исходная форма



Выполнение программы

```
unit Unit1;

interface

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

type
  TForm1 = class(TForm)
    Image1: TImage;
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender:TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation
Var
   iw,ih: integer;      // Первоначальный размер компонента Image
   N_Image: integer; // Номер выводимой картинки
   AFile: String;       // Имя картинки

{$R *.dfm}
// изменение размера области вывода иллюстрации пропорционально
// размеру иллюстрации
Procedure ScaleImage;
// Масштабирование изображения
var
    pw, ph : integer;         // Размер иллюстрации
    scaleX, scaleY : real; // Масштаб по X и Y scale
    : real;                   // Масштаб
begin
      // Иллюстрация уже загружена, получаем ее размеры
      pw := Form1.Image1.Picture.Width; ph
      := Form1.Image1.Picture.Height;
      scaleX := iw/pw;
      scaleY := ih/ph;
```

```
       // Выбераем наименьший коэффициент
       if   scaleX < scaleY then
            scale := scaleX
            else scale := scaleY;

       // Изменяем размер области вывода иллюстрации
Form1.Image1.Height := Round(Form1.Image1.Picture.Height*scale);
Form1.Image1.Width := Round(Form1.Image1.Picture.Width*scale);
// так как Strech = True и размер области пропорционален
// размеру картинки, то картинка масштабируется без искажений
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
 If N_Image = 4 Then N_Image:=1;
 // Формирование имени картинки
 AFile:=IntToStr(N_Image)+'.bmp';
 //Установка значения свойства Picture для вывода картинки
 // во время работы программы
 Form1.Image1.Picture.LoadFromFile(AFile);
 // Масштабирование картинки
 ScaleImage;
 N_Image:=N_Image+1;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
 Image1.AutoSize := False;
 Image1.Stretch := True;         // Разрешим масштабирование

   // Запомним первоначальный размер области вывода иллюстрации
   iw := Image1.Width; ih
   := image1.Height;

   N_Image:=1;    // Начальное значение номера выводимой картинки
end;
end.
```

Шестой этап. Заключительный

Самостоятельно внесите изменения в программу, добавив вывод картинок в зависимости от количества допущенных ошибок.

Протестируйте полученную программу.

Программа имеет еще один недостаток - после окончания игры (вывод панели с сообщением об оконачии игры), если нажимать кнопки с буквами, то программа может аварийно завершиться. Чтобы этого не было в начале процедуры Poisk выпоните проверку:

If (N_Attempt<Attempt) and (STR_N<>STR_R) then

Т.е. выполнять поиск нажатой буквы только в случае, если не исчерпаны все попытки и не отгадано все слово.

Листинг программы

```
unit Unit1;
interface uses
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls, ExtCtrls;

type
TForm1 = class(TForm) Label1: TLabel; Label2: TLabel; Button1: TButton; Button2: TButton;
Label3: TLabel; Label4: TLabel; Panel1: TPanel; Image1: TImage;
procedure FormCreate(Sender: TObject); procedure Button1Click(Sender: TObject); procedure
BtnClick(Sender: TObject); procedure Button2Click(Sender: TObject);
private
{ Private declarations } public
{ Public declarations } end;

var
Form1: TForm1; implementation
Const
ABC=32;              //Количество букв
N=100;               //Максимальное количество записей в массиве
Attempt=10; //Максимальное количество попыток
TYPE
T_R = record              //Структура записи массива Que:string[250]; //Вопрос Ans:string[30];
                          //Ответ
end;
R=array[1..N]of T_R;            //Массив Вопросов и Ответов
STR_30= String[30];

Var AQ:R;
Quetions_F:TextFile; //Файловая переменная STR_R,STR_N:STR_30; //Отгадываемое слово
CHAR_:Char;                    //Буква для поиска KOL_QUE:Integer;       //Количество
вопросов в файле Btn:TButton;
N_Attempt:Integer;             //Количество ошибок
Iw,Ih: integer;                // первоначальный размер компонента Image Afile:string; //Имя
файла-картинки

{$R *.dfm}

procedure Read_File;
```

```
//Чтение информации из файла и запись в массив
Var KOL, I:Integer; begin
Assignfile(Quetions_F,'Quetions.txt'); Reset(Quetions_F);
KOL:=1; I:=1;
While not Eof(Quetions_F) do begin
if (KOL mod 2)=1 then Readln (Quetions_F, AQ[I].QUE)
else begin
Readln(Quetions_F, AQ[I].ANS); Inc(I);
end;
Inc(KOL);
end; KOL_QUE:=I;
closefile(Quetions_F); end;

Procedure ScaleImage;
// Масштабирование изображения
// изменение размера области вывода иллюстрации пропорционально
// размеру иллюстрации
var
pw, ph : integer;                    // размер иллюстрации scaleX, scaleY : real; // масштаб по
X и Y scale : real;                  // масштаб
begin
// иллюстрация уже загружена
// получим ее размеры
pw := Form1.Image1.Picture.Width; ph := Form1.Image1.Picture.Height; scaleX := iw/pw;
scaleY := ih/ph;
// выберем наименьший коэффициент
if                scaleX < scaleY
then scale := scaleX else scale := scaleY;
// изменим размер области вывода иллюстрации Form1.Image1.Height :=
Round(Form1.Image1.Picture.Height*scale); Form1.Image1.Width :=
Round(Form1.Image1.Picture.Width*scale);
// так как Strech = True и размер области пропорционален
// размеру картинки, то картинка масштабируется без искажений
end;

Procedure POISK(STR_R:STR_30;Var STR_N:STR_30;CHAR_:Char);
//Поиск буквы в слове и «открытие» ее
Var I:Integer;
Flag:Boolean; // Флаг найдена ли в слове нажатая буква
begin
If (N_Attempt<Attempt) and (STR_N<>STR_R) then begin
Flag:=False;
For I:=1 to Length(STR_R) do begin// Поиск буквы в слове
If STR_R[I]=CHAR_ then begin
STR_N[I]:=CHAR_;
Flag:=True;
```

```
end; end;
If Flag=False Then
begin N_Attempt:=N_Attempt+1; Form1.Label4.Caption:=IntToStr(N_Attempt);
AFile:=IntToStr(N_Attempt+1)+ '.bmp'; Form1.Image1.Picture.LoadFromFile(AFile); ScaleImage;
end
Else Form1.Label2.Caption:= STR_N;
If STR_N=STR_R then begin
With Form1.Panel1 do begin
Visible:=True; Font.Color:=clBlack; Caption:='Вы выиграли!';
end; Form1.Button1.Enabled:=True; end;
If N_Attempt=Attempt then begin
With Form1.Panel1 do begin
Visible:=True; Font.Color:=clRed; Caption:='Вы проиграли...'; end;
Form1.Button1.Enabled:=True; end;
end; end;

procedure TForm1.BtnClick(Sender: TObject);
// Процедура обработки события нажатия созданных кнопок
Var STR_:String; begin
STR_:= (Sender as TButton).Caption;
// Переменная Sender содержит имя
// объекта, которому соответствует данное событие
CHAR_:=STR_[1]; POISK(STR_R,STR_N,CHAR_);
(Sender as TButton).Enabled:=False; end;
procedure TForm1.Button1Click(Sender: TObject); Var I,NUMBER:integer;
begin Form1.Panel1.Visible:=False;
Label3.Caption:='Количество неправильных ответов'; N_Attempt:=0;
Label4.Caption:=IntToStr(N_Attempt); Randomize;
NUMBER:=Random(KOL_QUE-1)+1;
Label1.Caption:=AQ[NUMBER].QUE; STR_R:=AQ[NUMBER].ANS; STR_N:='';
for I := 1 to Length(STR_R) do STR_N:=STR_N+'*';
Label2.Caption:=STR_N;
```

```
Form1.Button1.Enabled:=False;
for i:=1 to ABC do // Цикл по созданию кнопок на форме
begin Btn:=TButton.Create(Form1); with Btn do
begin // Свойства создаваемых кнопок
Parent := Form1; Caption := Chr(191+I); Height := 30;
Width := 30;
Top := 250+((I-1) div 8)*40;
Left := ((I-1) mod 8 ) * 40 + 20;
Font.Size := 9;
OnClick := FORM1.BtnClick; end;
end;
AFile:=IntToStr(N_Attempt+1)+'.bmp'; Form1.Image1.Picture.LoadFromFile(AFile); ScaleImage;
end;
procedure TForm1.Button2Click(Sender: TObject); begin
Close; end;
procedure TForm1.FormCreate(Sender: TObject); begin
With Form1 do begin
Caption:='Угадай слово'; Height:=450;
Width:=500; Color:=clWhite;
end;
With Button1 do begin
Caption:='Начать'; Height:=40; Width:=75; Top:=20; Left:=400; Font.Size := 9;
end;
With Button2 do begin
Caption:='Выход'; Height:=40; Width:=75; Top:=360; Left:=400; Font.Size := 9;
end;
With Label1 do begin
Caption:=''; Height:=40; Width:=250; Top:=70;
Left:=40; Font.Style:=[fsBold];
```

```
Font.Size := 10; end;
With Label2 do begin
Caption:=''; Height:=40; Width:=75; Top:=110;
Left:=40; Font.Style:=[fsBold]; Font.Size := 12;
end;
With Label3 do begin
Caption:=''; Height:=40; Width:=75; Top:=180;
Left:=40; Font.Size := 9;
end;
With Label4 do begin
Caption:=''; Height:=40; Width:=75; Top:=180; Left:=300;
// Font.Style:=[fsBold]; Font.Size := 10;
end;
With Panel1 do begin
Caption:=''; Height:=45; Width:=185; Top:=15; Left:=110;
Font.Style:=[fsBold]; Font.Size := 10; Visible:=False;
end;
With Image1 do begin
AutoSize := False;
Stretch := True; // разрешим масштабирование
// запомним первоначальный размер области вывода иллюстрации
Iw := Width; Ih := Height; Height:=140; Width:=140; Top:=120; Left:=350;
end; Read_File;
end; end.
```

Лабораторная работа №19-20, ПРОГРАММИРУЕМЫЙ ТЕСТ

1. Постановка задачи

Написать программу тестирования, которая удовлетворяет следующим требованиям:

1. Количество вопросов теста может варьировать от 5 до 20.
2. Для каждого вопроса должно быть предусмотрено четыре варианта ответов.
3. В результате тестирования должна быть выставлена оценка (неудовлетворительно, удовлетворительно, хорошо, отлично) в зависимости от количества правильных ответов.
4. Вопросы и ответы теста должны находиться в файле.
5. В программе должна быть заблокирована возможность возврата к предыдущему вопросу.

Новыми в этой программе являются:

- использование типизированных файлов,
- построение форм в ходе выполнения программы,
- использование компонента **SpinEdit** (редактор числа) со страницы палитры компонентов **Samples**.

2. Информационная постановка задачи

В условии задачи сказано, что для хранения используется файл. В Lazarus поддерживает три разных подхода для работы с файлами, для разного типа информации желательно использовать наиболее подходящий подход.

Немного теории

Текстовые файлы – в файле находится текстовая информация (набор строк из символов). Текстовая информация состоит не обязательно из букв, в ней могут содержаться любые символы из таблицы ASCII.

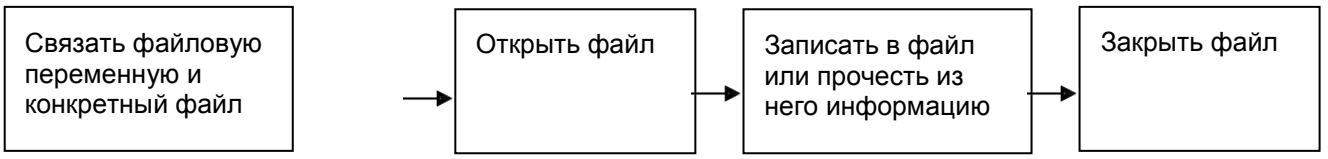
Типизированные файлы – в файле находится информация любого рода. Но структура такой информации обязательно должна повторяться. Название типизированный файл получил из-за того, что в нем хранится однотипная информация. Одна строка типа называется записью типизированного файла.

Примечание

Типизированные файлы очень часто используются для описания каких-либо списков информации, например, телефонных справочников. Каждая запись данного списка может состоять, например, из номера телефон абонента, имя, адрес.

Нетипизированные файлы – файл содержит последовательность байт без какой- либо структуры. Например, это может быть закодированный или сжатый блок информации. При работе с таким файлом вся информация рассматривается как набор отдельных кусков по N байт, где значение N задается программистом в диапазоне от 1 до 65535.

В процессе работы для любого файла нужно выполнять последовательность следующих действий:



Менять местами пункты нельзя.
Для каждого файла есть понятие – текущее положение в файле, т.е. это то положение, в котором на данный момент идет обработка информации. Например, в типизированном файле мы можем прочесть первую запись, затем вторую и так далее. В таком случае текущим положением в файле сначала будет первая запись, потом вторая, затем третья.

Структура информации

В нашей задаче единицей информации будет являться вопрос, который в свою очередь будет состоять из текста вопроса, четырех вариантов ответа и четырех указателей правильности ответа. Таким образом, можно ввести тип, описывающий вопрос.

```
Type TTEST = Record
TEXT : String [250];// Текст Вопроса
OTV_1 : String [100];// Вариант ответа № 1 OTV_2 : String [100];// Вариант ответа № 2 OTV_3 : String [100];// Вариант ответа № 3 OTV_4 : String [100];// Вариант ответа № 4
REZ_1 : Byte  // Признак правильности ответа № 1 REZ_2  : Byte  // Признак правильности ответа № 2 REZ_3 : Byte // Признак правильности ответа № 3 REZ_4 : Byte // Признак правильности ответа № 4
                                                                    end;
```

TEXT, **OTV_1**, **OTV_2**, **OTV_3**, **OTV_4** содержат текстовую информацию, а **REZ_1**, **REZ_2**, **REZ_3**, **REZ_4** будут содержать числовую информацию: 0 – ответ неправильный, 1 – ответ правильный. Можно было бы для «Признака правильности ответа» ввести тип *Boolean*, но мы ввели числовое значение, т.к. в ходе выполнения программы будем суммировать признаки правильности для ответов, которые выбирает тестируемый. В результате мы получим количество правильных ответов.

В предложенную структуру можно внести изменения, которые позволят в дальнейшем сократить текст программы. Для этого определим тип массива для данных «Вариант ответа» и «Признак правильности ответа». Тогда описание типа будет выглядеть:

```
Type TTEST = Record
TEXT     : String [250];                                    // Текст Вопроса
OTV      : Array [1..4]of String [100];// Варианты ответов
REZ      : Array [1..4]of Byte        // Правильный ответ
                                                                    end;
```

Непроизвольно мы подошли к тому, что при работе будем использовать типизированный файл. Для работы программы тестирования нам будет нужна еще информация о названии теста и количестве вопросов в тесте. Количество вопросов в тесте (количество записей в файле) можно узнать с помощью функции **FileSize**. А вот название теста необходимо поместить в файл. Поэтому информация в файле будет располагаться следующим образом:

Первая запись содержит название теста, все последующие записи – информацию о вопросах. В результате количество записей в файле будет на одну больше, чем количество вопросов в тесте.
Для формирования теста мы будем использовать массив:

```
Const KOL = 21;                                    // Max количество вопросов
Var       TEST_: Array [1..KOL] of TTEST;          // Массив ТЕСТА
```

Описание файла
При объявлении файловой переменной типизированных файлов указывается тип данных, содержащихся в файле.

```
Var FFile: File of TTEST;                          // Описание файла
```

Для чтения или записи файлов такого типа возможно использование только операторы **Read** и **Write**. **ReadLn** и **WriteLn** использовать нельзя, т.к. данные в файле это не строки текста, т.е. нет и линий.
Данные в типизированном файле хранятся во внутреннем машинном виде. Поэтом создавать файл и читать информацию из него можно только с помощью программы, которая «знает» структуру данных. Для этого мы создадим две программ. Первая будет отвечать за создание теста, а вторая – тестирование.

3. Программа создание теста

Откройте новый проект и разместите в форме компоненты в соответствии с рис.29.

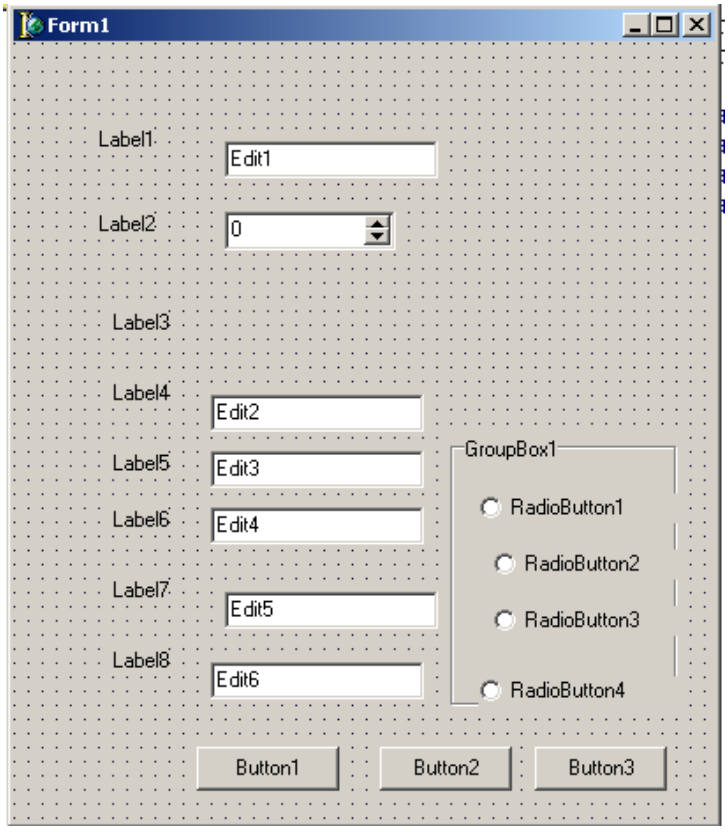


Рис.29

Нам понадобятся следующие компоненты:

№ п/п	Наименование компонента	Страница	Для чего предназначен
1	Label1	Standard	Вывод текста «Название теста»
2	Label2	Standard	Вывод текста «Количество вопросов» в тесте
3	Label3	Standard	Вывод текста «Вопрос №»
4	Label4	Standard	Вывод текста «Текст» вопроса
5	Label5	Standard	Вывод текста «Ответ № 1»
6	Label6	Standard	Вывод текста «Ответ № 2»
7	Label7	Standard	Вывод текста «Ответ № 3»
8	Label8	Standard	Вывод текста «Ответ № 4»
9	Edit1	Standard	Окно ввода названия теста
10	Edit2	Standard	Окно ввода текста вопроса
11	Edit3	Standard	Окно ввода варианта ответа № 1
12	Edit4	Standard	Окно ввода варианта ответа № 2
13	Edit5	Standard	Окно ввода варианта ответа № 3
14	Edit6	Standard	Окно ввода варианта ответа № 4
15	SpinEdit1	Samples	Выбор количества вопросов в *Тесте*. **Примечание** *SpinEdit* – редактор числа. Обеспечивает отображение и редактирование целого числа. Этот компонент содержит две кнопки со стрелками, используемые для увеличения или уменьшения значения числа. Диапазон значений числа задается свойствами *Min* и *Max*. Эти свойства определяют, соответственно, минимально и максимально возможные значения числа.
16	GroupBox1	Standard	Панель *GroupBox* позволяет объединить компоненты *RadioButton* для того, чтобы кнопки взаимодействовали друг с другом в группе (может быть нажата только одна кнопка из четырех).
17	RadioButton1	Standard	Выбор правильного ответа
18	RadioButton2	Standard	
19	RadioButton3	Standard	
20	RadioButton4	Standard	
21	Button1	Standard	Кнопка перехода на предыдущий вопрос
22	Button2	Standard	Кнопка перехода на следующий вопрос
23	Button3	Standard	Запись готового теста на диск (заполнены все вопросы)

Сохраните проект. В последующем сохраняйте проект и проверяйте его работоспособность после каждого изменения.

Не стремитесь располагать компоненты на свои места. Это мы сделаем в программе при создании формы. Для этого выделите объект **Form1,** перейдите на вкладку **Events Инспектора объектов** (**Object Inspector**), найдите событие **OnCreat**, справа от него дважды щелкнуть левой кнопкой мыши. Попав в код программы, надо написать следующий код:

```
Var Left_N : Integer;                    // Отступ слева верхней части
Top_N              : Integer;            // Отступ сверху
Left_NN: Integer;                        // Отступ слева для раздела ответов
Top_NN : Integer;// Отступ сверху для RadioButton K, I :Integer;
. . .
Left_N:=30; Top_N:=50; Left_NN:=60;

// Формирование элементов формы

Form1.Width:=740; Form1.Height:=540; Form1.Caption:='Создание теста';

Label1.Left:=Left_N; Label1.Top:=Top_N; Label1.Font.Style:=[fsBold]; Label1.Font.Size:=10;
Label1.Caption:='Название теста';

Edit1.Text:=''; Edit1.Top:=Top_N;
Edit1.Left:=Left_N+Label1.Width+10; Edit1.Width:=300;

Top_N:=Top_N+40;

Label2.Left:=Left_N; Label2.Top:=Top_N; Label2.Font.Style:=[fsBold]; Label2.Font.Size:=10;
Label2.Caption:='Количество вопросов';

SpinEdit1.Left:=Left_N+Label2.Width+10; SpinEdit1.Top:=Top_N; SpinEdit1.MinValue:=5;
SpinEdit1.MaxValue:=20; SpinEdit1.Text:='5'; SpinEdit1.Width:=40;

Top_N:=Top_N+60;

Label3.Left:=Left_N; Label3.Top:=Top_N; Label3.Font.Style:=[fsBold]; Label3.Font.Size:=10;
Label3.Caption:='Вопрос № 1';

Top_N:=Top_N+40; Label4.Left:=Left_NN;
```

Мы установили свойства компонентам **Label1**, **Edit1**, **Label2**, **SpinEdit1**, **Label3**, **Label4**, **Edit2**, **GroupBox1**. Все они обладают разными свойствами. Но компоненты, которые описывают 4 варианта ответов, можно сгруппировать – **Label** + **Edit** + **RadioButton** (для каждого ответа). Компоненты каждой строки будут отличаться только значением свойства **Top** и своими номерами. Компоненты распределятся по строкам следующим образом:
1-ая строка - Label5, Edit3, RadioButton1 2-ая строка – Label6, Edit4, RadioButton2 3-ая строка – Label7, Edit5, RadioButton3 4-ая строка – Label8, Edit6, RadioButton4
Поэтому возникает вопрос: «Каким образом можно перечислять в программе имена компонентов последовательно, например, переходя от **Label5** к **Label6**, потом к **Label7**?»
Это можно сделать, если воспользоваться методом **FindComponent**. Он возвращает указатель экземпляра компоненты, о которой известно. Допустим, что форма содержит экземпляр компоненты **TLabel** с именем **Label2**. Чтобы получить указатель на экземпляр **Label2** и изменить свойство **Caption**, можно записать:
K:=2;
TLabel(FindComponent('Label'+IntToStr(K))).Caption:= 'МЕТКА';
Следовательно, для формирования строк вариантов ответов в программу мы можем добавить текст:

```
//  Формирование раздела ОТВЕТОВ
K:=2; Top_NN:= -20;
For I:=1 To 4 do
  begin
  Top_N:=Top_N+40; Top_NN:=Top_NN+40;

  TLabel(FindComponent('Label'+IntToStr(K+I+2))).Left:=Left_NN;
  TLabel(FindComponent('Label'+IntToStr(K+I+2))).Top:=Top_N;
  TLabel(FindComponent('Label'+IntToStr(K+I+2))).Font.Size:=9;
  TLabel(FindComponent('Label'+IntToStr(K+I+2))).Caption:='Ответ №'
+IntToStr(I);
```

```
  TEdit(FindComponent('Edit'+IntToStr(K+I))).Text:='';
  TEdit(FindComponent('Edit'+IntToStr(K+I))).Top:=Top_N;
  TEdit(FindComponent('Edit'+IntToStr(K+I))).Left:=200;
  TEdit(FindComponent('Edit'+IntToStr(K+I))).Width:=400;

 TRadioButton(FindComponent('RadioButton'+IntToStr(I))).Left:=40;

TRadioButton(FindComponent('RadioButton'+IntToStr(I))).Top:=Top_NN;

TRadioButton(FindComponent('RadioButton'+IntToStr(I))).Caption:='';
 TRadioButton(FindComponent('RadioButton'+IntToStr(I))).Width:=10;

  end;
```

И в конце добавляем блок формирования трех кнопок.

```
//  Формирование нижней части формы

Top_N:=Top_N+70;

Button1.Caption:='Предыдущая';
Button1.Left:=Left_NN+100;
Button1.Top:=Top_N;
Button1.Width:=100;
Button1.Height:=30;
Button1.Enabled:=False;

Button2.Caption:='Следующая';
Button2.Left:=Left_NN+250;
Button2.Top:=Top_N;
Button2.Width:=100;
Button2.Height:=30;

Button3.Caption:='Записать';
Button3.Left:=Left_NN+500;
Button3.Top:=Top_N;
Button3.Width:=100;
Button3.Height:=30;
Button3.Enabled:=False;

STEP:=1;   //Номер вопроса = 1
```

В результате, после запуска проекта, вы увидите форму, представленную на рис.30.

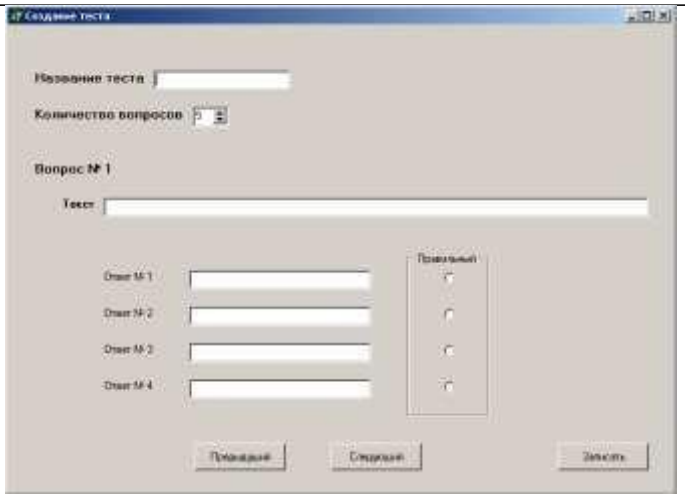


Рис.30

Наша задача – сформировать файл тестовых вопросов, который потом можно использовать для проведения тестирования. Для этого необходимо подумать о том, как будут организованы данные. Из общего вида **Form1** можно сделать вывод, что каждый вопрос состоит из текста вопроса, четырех вариантов ответов и четырех вариантов признаков правильности ответа. Для удобства воспользуемся типом данных **Запись** (**Record**). С одной стороны, запись можно рассматривать как единое целое, с другой стороны – как набор отдельных элементов разного типа. Для нашей задачи подойдет следующее описание типа:

```
Type TTEST = Record
     TEXT       : String [250];                // Текст Вопроса
     OTV        : Array [1..4]of String [40]; // Варианты ответов
     REZ        : Array [1..4]of Byte         // Правильный ответ
        end;
```

Решить нашу задачу можно следующим образом:

1. Ввести название теста и количество вопросов.
2. Ввести все вопросы с вариантами ответов, записывая их в массив.

Примечание.

Это позволит просматривать (вперед, назад) вопросы, вносить изменения до записи их в файл.

3. Сохранить все сформированные вопросы в файле.

Для этого наш массив будет иметь тип записи **TTEST**. В раздел описания включим описание файла и описание массива.

```
Const KOL = 21;                              // Max количество
вопросов
Var FFile: File of TTEST;                    // Описание файла
    TEST : Array [1..KOL] of TTEST;          // Массив ТЕСТА
```

При построении массива первый элемент массива (**TEST_[1]**) будет содержать информацию о тесте: **TEST_[1].TEXT** – название теста

Начиная со второго элемента массива, будет располагаться непосредственно информация о вопросах теста.

В процедуру **TForm1.FormCreate** необходимо добавить еще один блок подготовки массива **TEST_** к работе.

```
For I:=1 to KOL do
  begin
  TEST_[I].TEXT:='';
  for K:=1 to 4 do begin
    TEST_[I].OTV[K]:='';
    TEST_[I].REZ[K]:=0
    end;
  end;
```

Создадим процедуру, которая обрабатывает ситуацию нажатия кнопки **Button3** (запись созданного массива в файл). Для этого выделите объект **Button3**, перейдите на **вкладку Events Инспектора объектов** (**Object Inspector**), найдите событие **On Click**, справа от него дважды щелкнуть левой кнопкой мыши. Попав в код программы, надо написать следующий код:

```
procedure TForm1.Button3Click(Sender: TObject);
Var I    :Byte;
    STROK:TTEST;
begin
AssignFile(FFILE,'TEST.txt');
Rewrite(FFILE);
Button1.Enabled:=False;     //Недоступны все три кнопки
Button2.Enabled:=False;
Button3.Enabled:=False;

 For I:=1 to  STEP_N+1 do
  begin
   STROK:=TEST_[I];
   Write(FFILE, STROK)
  end;
 CloseFile(FFILE)
end;
```

В представленной процедуре файловой переменной **FFILE** ставится в соответствие имя файла, т.е. массив мы запишем в файл **TEST.txt**. Затем открывается файл (в нашем случае он создается) для записи. В файл переписываются все элементы массива **TEST_**. Их количество соответствует введенному параметру «Количество вопросов в тесте» плюс 1, т.к. добавлен один элемент (первый), который содержит название теста.
Устанавливаются недоступными кнопки «Предыдущая», «Следующая», «Запись», а затем массив записывается в файл. В конце файл закрывается.

Перейдем к созданию процедуры обработки **Button2** («Следующая» – переход к следующей записи). Для этого выделите объект **Button2**, перейдите на вкладку **Events Инспектора объекто***в* (**Object Inspector**), найдите событие **On Click**, справа от него дважды щелкнуть левой кнопкой мыши. Разберем код процедуры по частям.

Часть № 1

```
procedure TForm1.Button2Click(Sender: TObject);
Var I:Byte;
begin
If (RadioButton1.Checked=False) and
   (RadioButton2.Checked=False) and
   (RadioButton3.Checked=False) and
   (RadioButton4.Checked=False)
Then
 begin
 ShowMessage('Не выбран правильный ответ');
 Exit
 end;
```

В начале процедуры необходимо проверить выбран ли правильный ответ (должна быть нажата одна из кнопок **RadioButton**). Если значение свойства **Checked y RadioButton** равно **False**, то будет выдано сообщение «Не выбран правильный ответ» и процедура закончит свою работу.

Часть № 2

```
If STEP=1 Then
 begin
  TEST_[1].TEXT:=Edit1.Text;
  Edit1.Enabled:=False;
  STEP_N:= StrtoInt(SpinEdit1.Text);   // Количество вопросов в
тесте
  SpinEdit1.Enabled:=False;
  STEP:=STEP+1;
 end;
```

Этот кусок процедуры необходим, т.к. в массиве элементы с индексом 1 содержат информацию о названии теста. Поэтому, если у нас первый вопрос (**STEP**=1), то осуществляется запись в массив информации о тесте и устанавливаются объекты **Edit1** и **SpinEdit1** недоступными. Переходим к формированию вопросов.

Часть № 3

```
TEST_[STEP].TEXT:=Edit2.Text;

For I:=1 to 4 do
 begin
TEST_[STEP].OTV[I]:=TEdit(FindComponent('Edit'+IntToStr(I+2))).Text;
If TRadioButton(FindComponent('RadioButton'+IntToStr(I))).Checked= True
Then TEST_[STEP].REZ[I]:= 1;
 end;
```

В третьей части элементам массива с индексом **STEP** присваивается значение соответствующих полей. Если нажата **RadioButton**, то соответствующему элементу массива **REZ[I]** (*Результат ответа*) присваивается значение 1.

Часть № 4

В заключительной части сначала устанавливается доступной кнопка **Button1** («Предыдущая»), т.к. уже есть одна запись и мы, после выполнения процедуры, переходим на вторую. Дальше идет проверка – является ли наша запись последняя, т.е. номер текущей записи (**STEP**) больше количества вопросов в тесте (**STEP_N**). Если мы записали в массив все вопросы, то кнопка **Button2** («Следующая») устанавливается недоступной, а кнопка **Button3** («Запись») – доступной.

1. Если не достигли конца теста, то идет подготовка к следующему вопросу: увеличивается переменная **STEP** и свойствам компонент экрана присваиваются значения следующей записи массива.
2. Если массив не заполнен до конца, то это будут пустые записи.
3. Если массив был заполнен, и мы переходили от одной записи к другой с помощью кнопок **Button1** («Предыдущая») и **Button2** («Следующая»), то на экран будут выведены значения соответствующей записи.

Перейдем к созданию процедуры обработки **Button1** («Предыдущая» – переход к предыдущей записи). Для этого выделите объект **Button1**, перейдите на вкладку **Events Инспектора объектов** (**Object Inspector**), найдите событие **On Click**, справа от него дважды щелкнуть левой кнопкой мыши. Попав в код программы, надо написать следующий код:

```
procedure TForm1.Button1Click(Sender: TObject);
Var I: Byte;
begin
STEP:=STEP-1;
Label3.Caption:= 'Вопрос № '+IntToStr(STEP-1);
Edit2.Text:=TEST_[STEP].TEXT;
For I:=1 to 4 do
 begin
TEdit(FindComponent('Edit'+IntToStr(I+2))).Text:=TEST_[STEP].OTV[I];
If TEST_[STEP].REZ[I]= 1 Then
```

```
    TRadioButton(FindComponent('RadioButton'+IntToStr(I))).Checked:=    True
    Else
    TRadioButton(FindComponent('RadioButton'+IntToStr(I))).Checked:=    False
 end;
If STEP=2 then
 Button1.Enabled:=False;
Button2.Enabled:=True;
end;
```

В начале переходим к предыдущему вопросу (уменьшаем значение переменной **STEP**), а затем выводим на экран содержимое вопроса. В конце процедуры проверяем, если вопрос является первым (номер строки массива **STEP**=2), то делаем недоступной кнопку *Предыдущая* и в любом случае должна быть доступна кнопка *Следующая*.

Сохраните проект окончательно, запустите и протестируйте его.

Задание

Внесите изменение в программу так, чтобы можно было бы просматривать уже созданные тесты.

4. Программа тестирования

Программа тестирования будет состоять из двух форм: титульной формы, на которой осуществляется выбор теста из справочника тестов (текстовый файл) и форма формы непосредственно теста.

В зависимости от правильности ответов на вопросы теста, подсчитывается результат и выставляется оценка.

Создание титульной формы

Титульная форма будет выполнять следующие функции:

- Поиск справочника тестов (текстовый файл **STEST.txt**), чтение информации о тестах-файлах и вывод перечня тестов в компонент **RadioGroup1**.
- В зависимости от выбранного тест-файла, формирование рабочего файла тестов.
- Вызов формы непосредственного тестирования.
- Удаление рабочего файла тестов при окончании работы программы.

Справочник тестов – текстовый файл, который можно создать в любом текстовом редакторе. Каждая строка файла содержит наименование теста. Может иметь, например, такое содержание:

```
Информация и информационные процессы
Архитектура компьютера
Измерение информации
Основы алгебры логики
```

Каждому тесту соответствует файл-тест, который имеет имя **TEST_N.txt**, где **N** – номер строки в файле.

Откройте новый проект и разместите в форме компоненты в соответствии с рис.31.

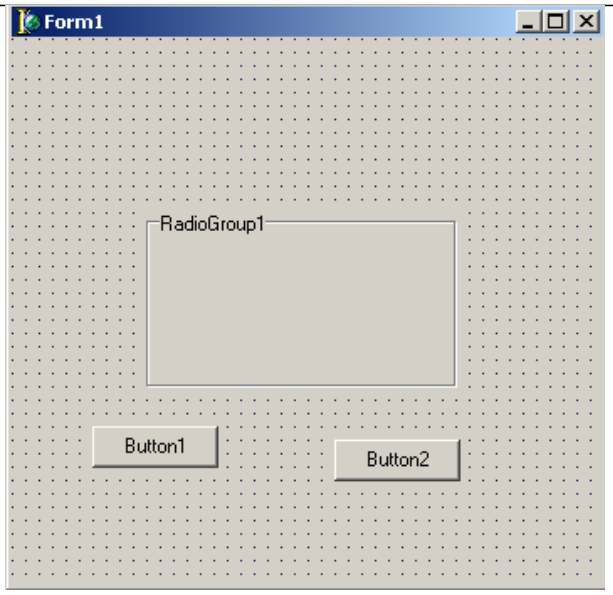


Рис.31

№ п/п	Наименование компонента	Страница	Для чего предназначен
1	RadioGroup1	Standard	Перечень возможных тестов (из справочника тестов)
2	Button1	Standard	Кнопка для перехода к тестированию после выбора теста
3	Button2	Standard	Кнопка выхода из программы тестирования

В разделе **implementation** разместите описание типа и данных, которые мы будем использовать при работе.

```
Type TTEST = Record
TEXT          : String [250];                    // Текст вопроса
OTV           : Array [1..4]of String [40]; // Варианты ответов
REZ           : Array [1..4]of Byte              // Правильный ответ
                                        end;
Var SFILE : TextFile;                 // Справочник тестов
FFile: File of TTEST; // Файл тестов
FF_R : File of TTEST; // Рабочий файл тестов
KOL           : Byte;                 // Количество тестов в справочнике тестов
TEST_: TTEST;                         // Строка записи файла
```

Выделите объект **Form1**, перейдите на вкладку **Events Инспектора объектов** (**Object Inspector**), найдите событие **OnCreat**, справа от него дважды щелкнуть левой кнопкой мыши. Попав в код программы, надо написать следующий код:

```
procedure TForm1.FormCreate(Sender: TObject);
Var I      : Byte;
    SF     : String;
    T_TEXT : Array [1..10] of String[50];
    Left_N : Integer;    // отступ слева
    Top_N  : Integer;    // отступ справа
begin
```

```
KOL:=1;
AssignFile(SFILE,'STEST.txt');
{$I-} Reset(SFILE); {$I+}
If IOResult = 0 then
 begin
  Repeat
    ReadLn(SFILE,SF);
    T_TEXT[KOL]:=SF;
    KOL:=KOL+1;
  until (Eof(SFILE)) or (KOL>10);
  CloseFile(SFILE);

  Left_N:=30;
  Top_N:=50;

  Form1.Caption:='Тестирование';
  Form1.Width:=400;
  Form1.Height:=Top_N+ KOL*25 + 120;

  RadioGroup1.Top:=Top_N;
  RadioGroup1.Left:=Left_N;
  RadioGroup1.Width:= 350;
  RadioGroup1.Height:= KOL*25;
  RadioGroup1.Caption:= ' Выбери тест ';
  For I:=1 to KOL-1 do
   RadioGroup1.Items.Add(T_TEXT[I]);
  RadioGroup1.Font.Size:=11;

  Button1.Caption:='Тест';
  Button1.Left:= Left_N;
  Button1.Top:=Top_N+ KOL*25+40;
  Button1.Width:= 100;
  Button1.Height:=30;

  Button2.Caption:='Выход';
  Button2.Left:= Left_N+ 250;
  Button2.Top:=Top_N+ KOL*25+40;
  Button2.Width:= 100;
  Button2.Height:=30;
 end
  else
   ShowMessage('Отсутствует справочник тестов');
end;
```

Немного теории

Директива компилятора {$I+}/{$I-} включает или выключает автоматическую проверку ошибок ввода/вывода. Когда директива включена и возникает ошибка ввода/вывода, то выполнение программы завершается с сообщением о произошедшей ошибке. Но можно отключить аварийное завершение программы при возникновении ошибки. В этом случае программист берет на себя обработку аварийной ситуации. Для того, чтобы узнать произошла ли ошибка при открытии файла, можно воспользоваться функцией *IOResult*. Она возвращает код отличный от нуля, если произошла ошибка и ноль, если операция выполнилась успешно.

Типичной ошибкой ввода/вывода является отсутствие файла. По умолчанию директива компилятора включена – {$I+}.

В начале процедуры открывается файл справочника тестов **STEST.txt**.
Если файл существует, то информация из него переписывается в массив **T_TEXT** и подсчитывается количество записей (**KOL**).
В программе установлено ограничение на количество элементов массива **T_TEXT** – не более 10. По желанию это можно изменить. После устанавливаются свойства компоненты форма.
Список значений свойства **Items** компоненты **RadioGroup1** будем формировать в ходе программы в зависимости от количества элементов в массиве справочника тестов. Для этого к свойству **Items** применяем метод **Add**:
For I:=1 to 4 do RadioGroup1.Items.Add(T_TEXT[I]);
Если файл справочника тестов отсутствует, то будет выдано сообщение *Отсутствует справочник тестов*.
В результате форма будет выглядеть так же как на рис.32.

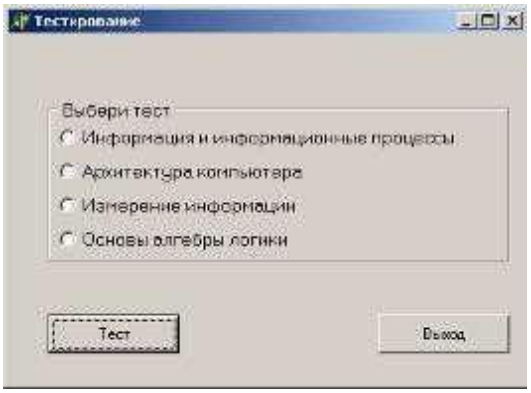


Рис.32

Создайте новую форму **Form2** и сохраните под именем **Unit2.pas**. Подсоедините созданную форму к титульной форме. Для этого в **Unit1.pas** в разделе **interface** в строку списка подсоединенных модулей добавьте **Form2**.

Создадим процедуру, которая обрабатывает ситуацию нажатия кнопки **Button1** (*Тест*). Для этого выделите объект **Button1**, перейдите на вкладку **Events Инспектора объектов** (**Object Inspector**), найдите событие **On Click**, справа от него дважды щелкнуть левой кнопкой мыши. Попав в код программы, надо написать следующий код:

```
procedure TForm1.Button1Click(Sender: TObject);
Var N:Byte;
begin
N:=RadioGroup1.ItemIndex+1;
AssignFile(FFILE,'TEST_'+IntToStr(N)+'.txt'); // Тест с номером
N
{$I-} Reset(FFILE); {$I+}
If IOResult = 0 then
 begin
  AssignFile(FF_R, 'TEST_R.txt');   // Рабочий файл тестов
  Rewrite(FF_R);
  Repeat
    Read(FFILE,TEST_);
```

В начале процедуры формируется имя теста в зависимости от номера выбранной строки компонента **RadioGroup1**.

Примечание

Индекс первого переключателя равен 0, а массив справочника содержит индекс первого элемента равный 1.

Дальше открывается выбранный файл-тест. Если данный файл отсутствует, то выдается сообщение «Отсутствует файл тестов». Если при открытии файла-теста не возникло ошибки, то информация из него, переписывается в рабочий файл тестов (**TEST_R.txt**). В завершении открывается форма как модальная. Это означает, что управление передается новой форме, и пользователь не может передать фокус другой форме данного приложения до тех пор, пока он не закроет модальную форму.

Создадим процедуру, которая обрабатывает ситуацию нажатия кнопки **Button2** («Выход»). Для этого выделите объект **Button2**, перейдите на вкладку **Events Инспектора объектов** (**Object Inspector**), найдите событие **On Click**, справа от него дважды щелкнуть левой кнопкой мыши. Попав в код программы, надо написать следующий код:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
Close;
end;
```

Создание формы тестирования

Сделайте активной **Form2** и разместите на ней компоненты в соответствии с рис.33.

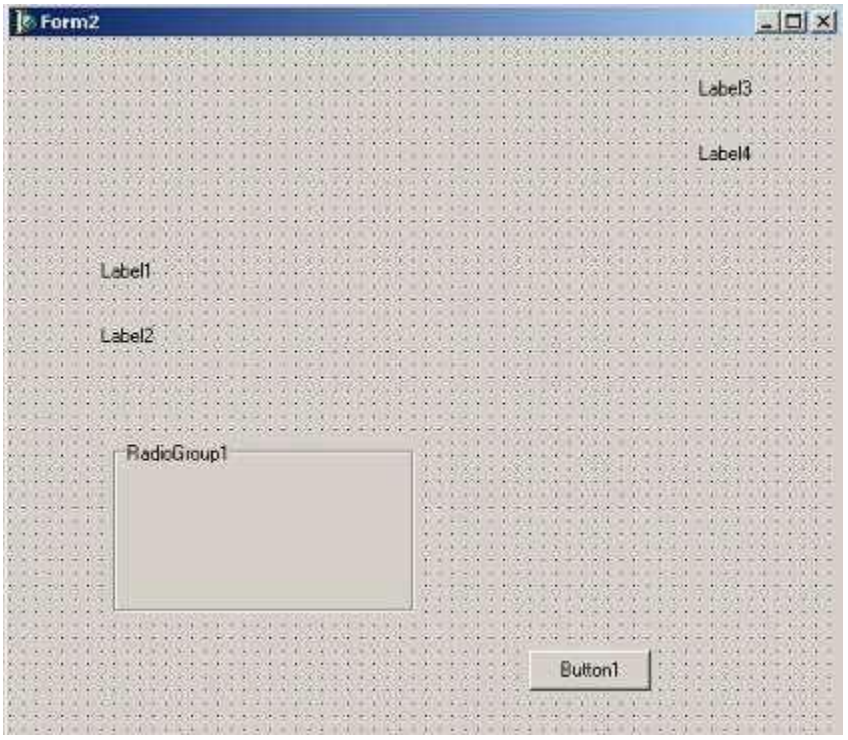


Рис.33

№ п/п	Наименование компонента	Страница	Для чего предназначен
1	Label1	Standard	Вывод текста «Вопрос № …»
2	Label2	Standard	Вывод текста «Текст» вопроса
3	Label3	Standard	Вывод текста «Всего вопросов …»
4	Label4	Standard	Вывод текста «Правильных ответов …»
5	RadioGroup1	Standard	Вывод вариантов ответов
6	Button1	Standard	Кнопка для перехода к следующему вопросу, показу результата и завершения тестирования

В разделе **implementation** разместите описание типа и данных, которые мы будем использовать при работе.

```
Type TTEST = Record
     TEXT      : String [250];                // Текст вопроса
     OTV       : Array [1..4]of String [100];// Варианты ответов
     REZ       : Array [1..4]of Byte          // Правильный ответ
        end;
Var FF_R : File of TTEST; // Рабочий файл тестов
    KOL  : Byte;          // Количество вопросов в тесте
    KOL_N: Byte;          // Текущий номер вопроса
    TEST_: TTEST;         // Строка записи файла
    REZ_N: Byte;          // Количество правильных ответов
```

Для объекта **Form2**, перейдите на вкладку **Events Инспектора объектов** (**Object Inspector**), найдите событие **On Activate**, справа от него дважды щелкнуть левой кнопкой мыши. Попав в код программы, надо написать следующий код:

```
procedure TForm2.FormActivate(Sender: TObject);
Var I : Byte;
begin
AssignFile(FF_R,'TEST_R.txt');
Reset(FF_R);
Read(FF_R,TEST_);
KOL:= FileSize(FF_R)-1;
KOL_N:=1;
REZ_N:=0;

Label1.Visible:=True;
RadioGroup1.Visible:=True;

Form2.Caption:=TEST_.TEXT;
Form2.Height:=420;
Form2.Width:=850;

Label3.Caption:='Всего вопросов - '+IntToStr(KOL);
Label3.Top:=20;
Label3.Left:= 700;
Label3.Font.Size := 8;

Label4.Caption:='Правильных ответов - '+IntToStr(REZ_N);
Label4.Top:=40;
```

```
Label4.Left:= 700;
Label4.Font.Size := 8;
Label4.Font.Color:=clBlack;

Button1.Caption:='Следующий';
Button1.Top:=350;
Button1.Left:=720;
Button1.Width:= 100;
Button1.Height:=30;

Read(FF_R,TEST_);

Label1.Caption:='Вопрос № '+IntToStr(KOL_N);
Label1.Font.Style := [fsBold];
Label1.Font.Size := 11;
Label1.Height:=50;
Label1.Width:=400;
Label1.Left:=40;
Label1.Top:=70;

Label2.Caption:=TEST_.TEXT;
Label2.Font.Size := 12;
Label2.Height:=50;
Label2.Width:=500;
Label2.Left:=60;
Label2.Top:=100;
Label2.WordWrap := True;
Label2.AutoSize := False;

RadioGroup1.Top:=160;
RadioGroup1.Left:=60;
RadioGroup1.Width:= 750;
RadioGroup1.Height:= 150;
RadioGroup1.Font.Size:=11;
RadioGroup1.Caption:= ' Выбери ответ ';
RadioGroup1.Items.Clear;
 For I:=1 to 4 do
  RadioGroup1.Items.Add(TEST_.OTV[I]);

end;
```

Работа данной процедуры начинается с того, что открывается рабочий файл тестов **TEST_R.txt** и считывается первая запись, которая содержит название теста. С помощью функции **FileSize** определяем количество записей в файле, а количество вопросов будет на единицу меньше.
Дальше настраиваем компоненты **Form2**, **Label3**, **Label4**, **Button1**. Читаем из рабочего файла тестов еще одну запись – первый вопрос. После этого настраиваем компоненты **Label1**, **Label2**, **RadioGroup1**. При описании компонента **RadioGroup1** мы добавляем RadioGroup1.Items.Clear;
Эта строка позволяет нам очистить список значений свойства **Items** компоненты **RadioGroup1**, который формируется в ходе программы. Применение метода **Clear** нам необходимо при повторном тестировании – очистить список от предыдущих вариантов ответов.

В результате форма будет выглядеть так же как на рис.34.

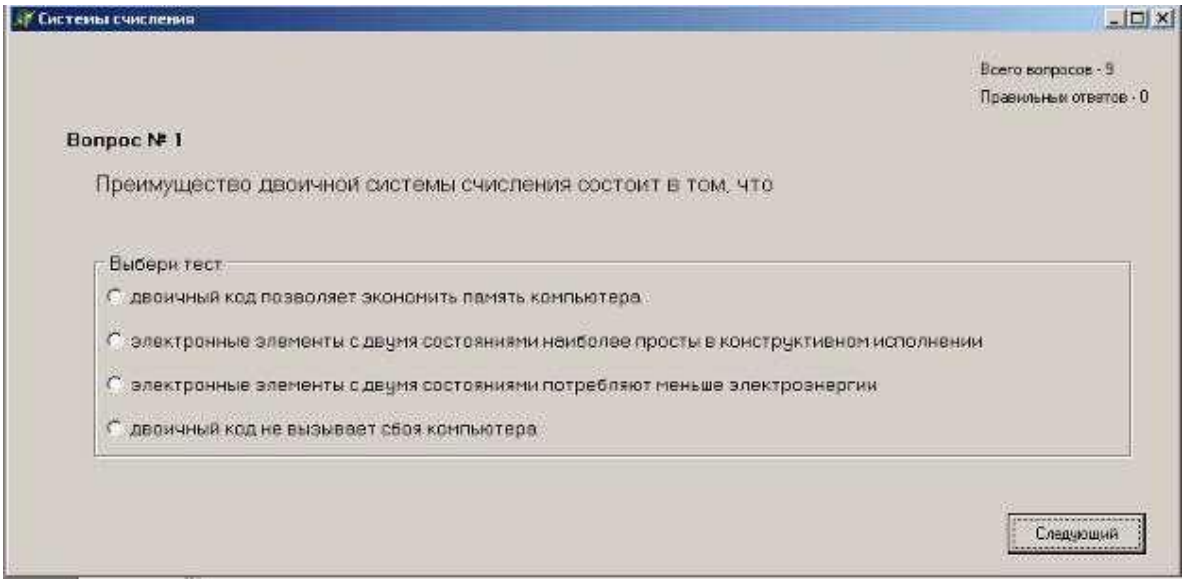


Рис.34

Создадим процедуру, которая обрабатывает ситуацию нажатия кнопки **Button1**. Для этого выделите объект **Button1**, перейдите на вкладку **Events Инспектора объектов** (**Object Inspector**), найдите событие **On Click**, справа от него дважды щелкнуть левой кнопкой мыши. Попав в код программы, надо написать следующий код:

Часть № 1

```
procedure TForm2.Button1Click(Sender: TObject);
Var N, I   :Byte;
    OZENKA : Byte;
begin

If Button1.Caption= 'Результат' Then
 begin
  OZENKA:=Round(REZ_N/KOL*5);
  Button1.Caption:= 'Выход';
  Label1.Visible:=False;
  RadioGroup1.Visible:=False;

  Label2.Font.Size := 12;
  Label2.Left:=250;
  Label2.Top:=150;
  Label2.Caption:='Количество правильных ответов - '+IntToStr(REZ_N)+
  ' из '+IntToStr(KOL);

  Label3.Font.Size := 14;
  Label3.Font.Style := [fsBold];
  Label3.Left:=300;
  Label3.Top:=200;
  Label3.Caption:='Оценка';

  Label4.Font.Size := 14;
  Label4.Left:=350;
  Label4.Top:=200;
  Case OZENKA of
```

```
1..2: Label4.Font.Color:=clPurple;
3          : Label4.Font.Color:=clMaroon;
4          : Label4.Font.Color:=clGreen;
5          : Label4.Font.Color:=clNavy; end; Label4.Caption:=IntToStr(OZENKA);
end else
If Button1.Caption= 'Выход' Then begin
Close; CloseFile(FF_R); end
else begin
N:=RadioGroup1.ItemIndex+1; REZ_N:=REZ_N+ TEST_.REZ[N]; KOL_N:=KOL_N+1;

If KOL_N>KOL Then begin
Label4.Caption:='Правильных ответов - '+IntToStr(REZ_N); Button1.Caption:= 'Результат'
end
else begin
Label4.Caption:='Правильных ответов - '+IntToStr(REZ_N); Read(FF_R,TEST_);
Label1.Caption:='Вопрос № '+IntToStr(KOL_N); Label2.Caption:=TEST_.TEXT;
RadioGroup1.Items.Clear;
For I:=1 to 4 do RadioGroup1.Items.Add(TEST_.OTV[I]);
end; end;
end;
```

В ходе программы кнопка **Button1** может иметь надписи «Следующая», «Результат», «Выход».

Если кнопка имеет надпись «Результат» и вы кликнули на ней, то в этом случае определяется значение переменной **OZENKA** по пяти бальной системе в зависимости от количества правильных ответов **REZ_N** и общего количества вопросов **KOL**. Затем устанавливается надпись кнопки «Выход», формируется новое изображение формы, выводится итоговая оценка, делаются невидимыми компоненты **Label1** и **RadioGroup1** (поэтому в процедуре **TForm2.FormActivate** устанавливаются данные компоненты видимыми).
Если кнопка имеет надпись «Выход» и вы кликнули на ней, то в этом случае закрывается рабочий файл тестов и закрывается форма.
Если кнопка имеет надпись «Следующий» и вы кликнули на ней, то в этом случае определяется какой был выбран ответ на вопрос, результат складывается с переменной **REZ_N**. Увеличивает значение переменной **KOL_N** (номер текущего вопроса). Если новый текущий номер больше количества вопросов, то изменяется надпись на кнопке («Результат»), в противном случае переходим к чтению нового вопроса из рабочего файла тестов и формируем вывод нового вопроса.
Все, программа готова. Осталось только ее протестировать.

Задание для самостоятельного выполнения

	Задание
1	Написать программу, которая осуществляет конвертирование информации из текстового файла в типизированный файл, для файла с тестовыми заданиями.

2	Предусмотреть возможность выбора случайным образом тестового задания из общего списка заданий.
3	Создать файл, в котором накапливается информация о прохождении тестирования. Информация, которая может находиться в нем следующая: фамилия, имя, класс, номер вопрос, номер ответа учащегося, количество правильных ответов, количество неправильных ответов, время начала тестирования, время окончания тестирования и т.п.