СОДЕРЖАНИЕ

1. Основные этапы решения задач 2
2. Структуры данных 13
2.1. Стек 13
2.2. Очередь 18
3. Рекурсивные алгоритмы 23
4. Поиск данных 30
5. СОРТИРОВКА ДАННЫХ 38
6. Построение графических изображений 45
6.1. Графика в PascalABC 45
6.2. Построение графиков функций в среде LASARUS 54
ЛИТЕРАТУРА 60

1. ОСНОВНЫЕ ЭТАПЫ РЕШЕНИЯ ЗАДАЧ

Краткие теоретические сведения.

Процесс решения задач на компьютере - это совместная деятельность человека и ЭВМ. Этот процесс можно представить в виде нескольких последовательных этапов. На долю человека приходятся этапы, связанные с творческой деятельностью — постановкой, алгоритмизацией, программированием задач и анализом результатов, а на долю компьютера — этапы обработки информации в соответствии с разработанным алгоритмом.

Первый этап - постановка задачи. На этом этапе участвует человек, хорошо представляющий предметную область задачи. Он должен четко определить цель задачи, дать словесное описание содержания задачи и предложить общий подход к ее решению.

Второй этап - математическое или информационное моделирование. Цель этого этапа - создать такую математическую модель решаемой задачи, которая может быть реализована в компьютере.

Третий этап - алгоритмизация задачи. На основе математического описания необходимо разработать алгоритм решения.

Используются различные способы записи алгоритмов. Наглядно представляется алгоритм языком блок-схем.

Четвертый этап — программирование. Программой называется план действий, подлежащих выполнению некоторым исполнителем, в качестве которого может выступать компьютер. Составление программы обеспечивает возможность выполнения алгоритма и соответственно поставленной задачи исполнителем-компьютером.

Пятый этап - ввод программы и исходных данных в ЭВМ. Программа и исходные данные вводятся в ЭВМ с клавиатуры с помощью редактора текстов, и для постоянного хранения осуществляется их запись на гибкий или жесткий магнитный диск.

Шестой этап - тестирование и отладка программы. На этом этапе происходят исполнение алгоритма с помощью ЭВМ, поиск и исключение ошибок. При этом программисту приходится выполнять работу по проверке работы программы, поиску и исключению ошибок, и поэтому для сложных программ этот этап часто требует гораздо больше времени и сил, чем написание первоначального текста программы. *Отладка программы* - сложный и нестандартный процесс. Исходный план отладки заключается в том, чтобы оттестировать программу на контрольных примерах.

Контрольные примеры стремятся выбрать так, чтобы при работе с ними программа прошла все основные пути блок-схемы алгоритма, поскольку на каждом из путей могут быть свои ошибки, а детализация плана зависит от того, как поведет себя программа на этих примерах: на одном она может зациклиться (т. е. бесконечно повторять одно и то же действие); на другом - дать явно неверный или бессмыс-

ленный результат и т. д. Сложные программы отлаживают отдельными фрагментами.

Седьмой этап - исполнение отлаженной программы и анализ результатов. На этом этапе программист запускает программу и задает исходные данные, требуемые по условию задачи.

Полученные в результате решения выходные данные анализируются постановщиком задачи, и на основании этого анализа вырабатываются соответствующие решения, рекомендации, выводы.

Возможно, что по итогам анализа результатов потребуются пересмотр самого подхода к решению задачи и возврат к первому этапу для повторного выполнения всех этапов с учетом приобретенного опыта. Таким образом, в процессе создания программы некоторые этапы будут повторяться до тех пор, пока мы получим алгоритм и программу, удовлетворяющие показанным выше свойствам.

<u>Пример 1. Определение суммы ряда с заданной точностью</u>

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.

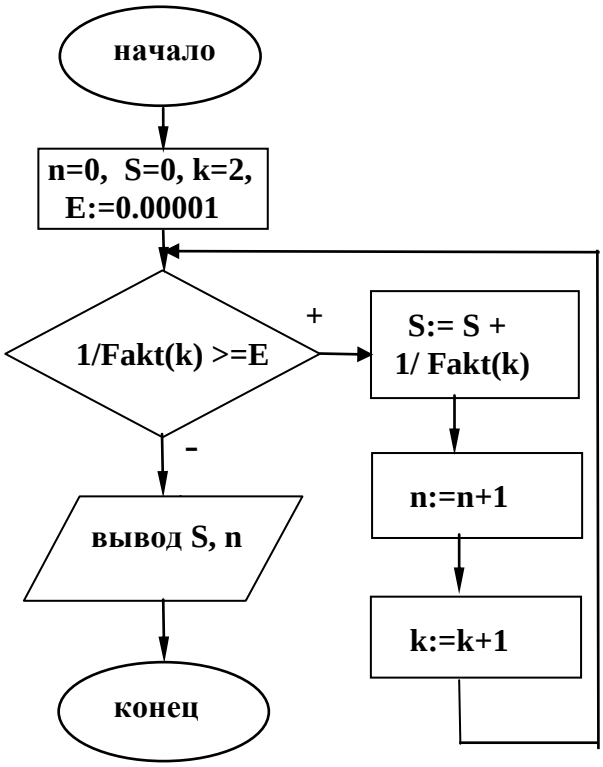
$$\sum_{k=1}^{n} \frac{1}{k!} - \sum_{k=1}^{n-1} \frac{1}{k!} < E$$

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:

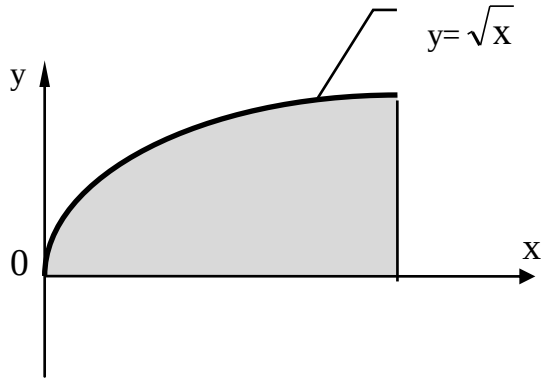
1) E=0.1. В этом случае искомая сумма будет содержать 2 слагаемых:
   $S = \frac{1}{2!} + \frac{1}{3!}$, т.к. следующее слагаемое $\frac{1}{4!} < 0.1$
2) E=0.01. В этом случае искомая сумма будет содержать 3 слагаемых:
   $S = \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!}$, т.к. следующее слагаемое $\frac{1}{5!} < 0.01$

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

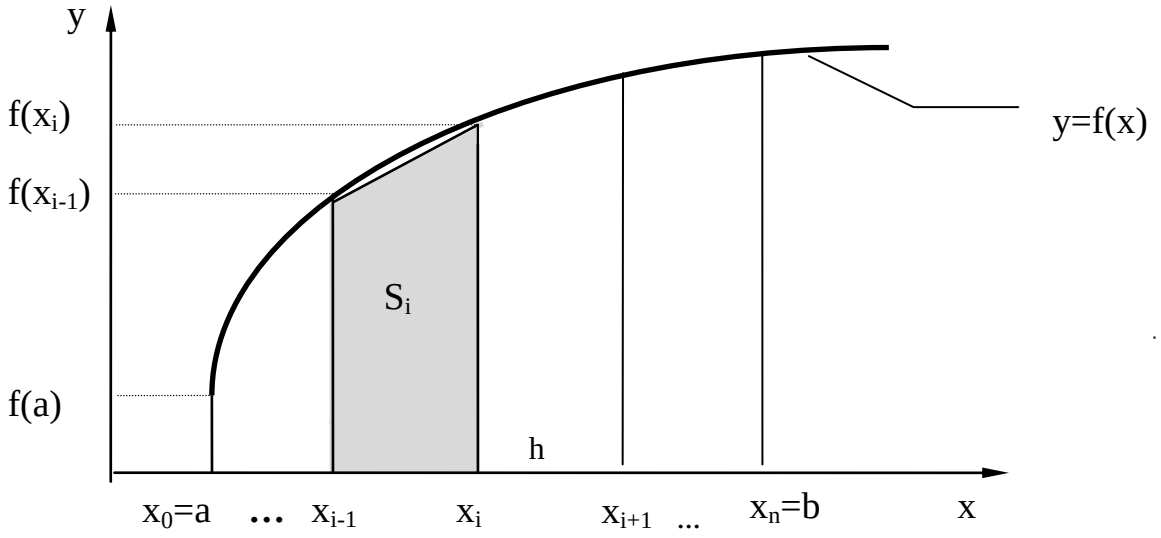
Пример 2. Вычисление площади криволинейной трапеции

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

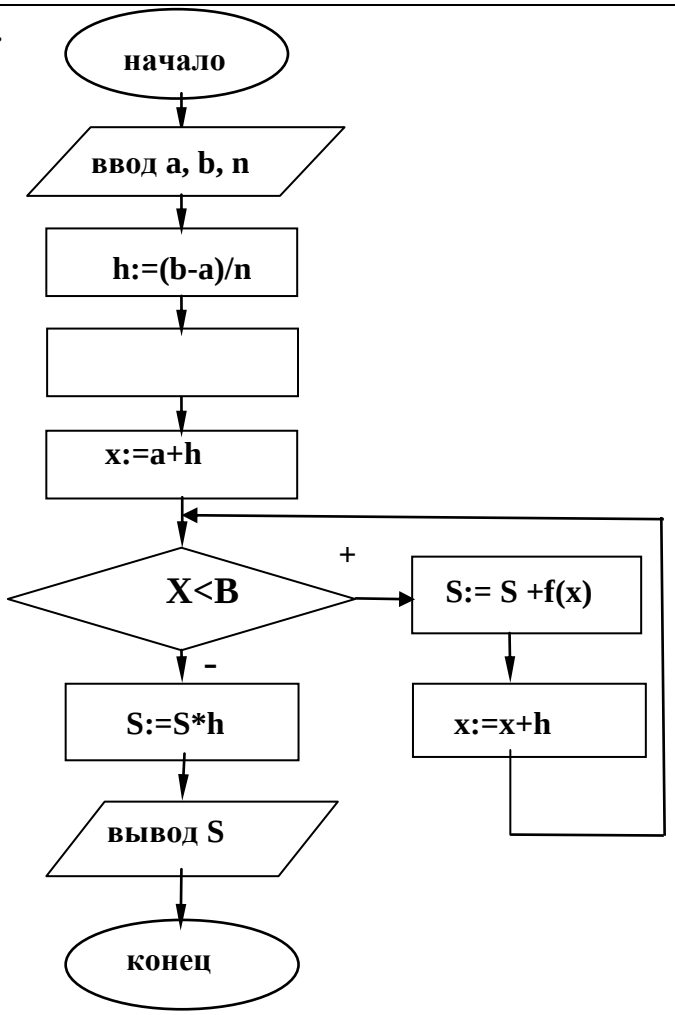
$S_{ABCD}\approx\sum_{i=1}^{n} S_i$, где $S_i=\frac{f(x_{i-1})+f(x_i)}{2}h$. Чем больше **n**, тем точнее результат вычисления.



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

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

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	0	1	0.33333	0.33341
		1000	0	1	0.33333	0.33333
2	$y= e^x$,	100	0	1	1.71828	1.71830

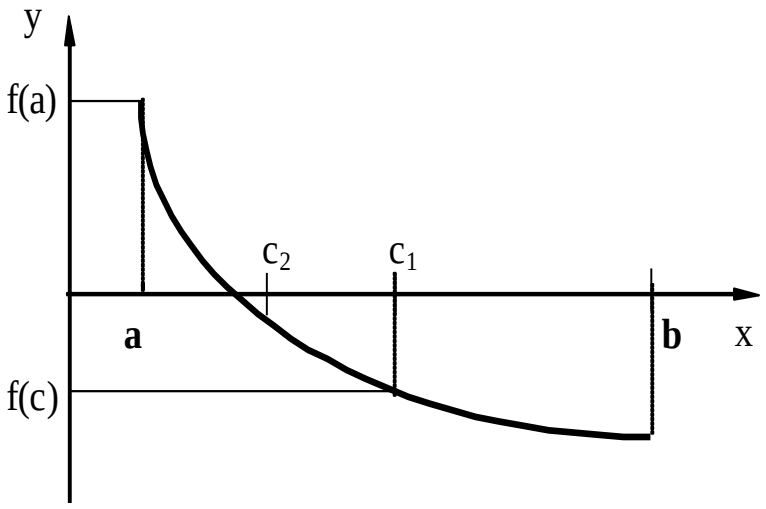
		1000	0	1	1.71828	1.71828

<u>Пример 3.</u>

Решение уравнений методом половинного деления.

Найти приближенное значение корня уравнения $x^3 + 0{,}5\,x^2 - 2=0$ с точностью 0.0001.

1) математическая модель

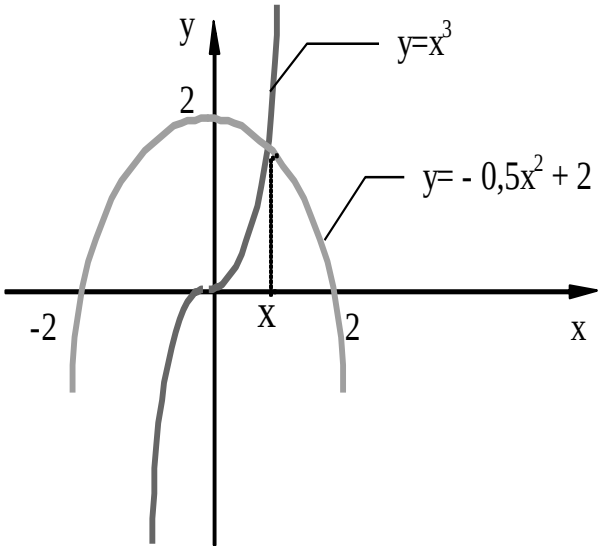


Если уравнение f(x)=0 имеет единственный корень на отрезке [a,b] и функция y=f(x) определена и непрерывна на данном отрезке, то для уточнения значения корня можно воспользоваться методом половинного деления, который заключается в делении отрезка [a,b] пополам и выборе той его половины, которая содержит корень. Выбранная половина снова делится пополам и так далее, пока длина отрезка, содержащего корень, не станет меньше заданной точности E. Тогда любое число, содержащееся между ***a*** и ***b*** можно принять за приближенное значение корня ***x*** с точностью до ***E.***

$c_i = (a+b)/2$ - середина выбранного отрезка

$f(a)*f(c)<0$ -условие принадлежности корня отрезку [a,c]

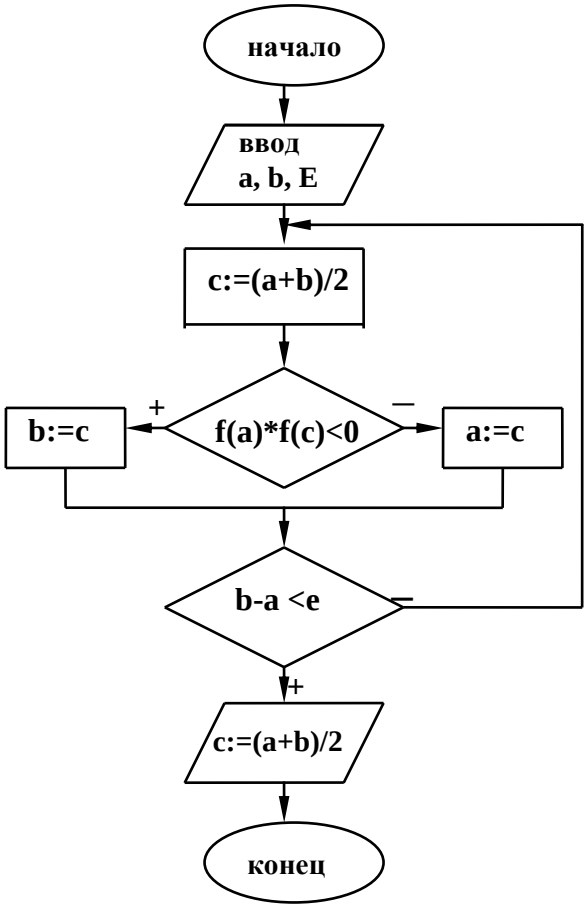
$|b-a| <e$ - условие завершения процесса деления отрезка



Для решения задачи необходимо определить отрезок, содержащий искомый корень. Для этого перепишем уравнение $x^3 + 0{,}5\,x^2 - 2=0$ в виде $x^3 = -0{,}5\,x^2 + 2$.

Построив графики функций $y= x^3$ и $y= -0{,}5\,x^2 + 2$, можно определить, что исходное уравнение имеет единственный корень на отрезке [0,2].

3) блок-схема



3) программа

```
Program  Pr1;
    Const  E=0.00001;
    Var a,b,c: real;
 Function f(x: real): real;
        Begin
            f:=x*x*x+0.5*x*x-2;
 End;

 BEGIN
     a:=0;    b:=2;
     repeat
        c:=(a+b)/2;
        if  f(a)*f(c)<0 then b:=c else a:=c;
     Until b-a<E;
     Writeln('x=', (a+b)/2:8:5);
 END.
```

4) получение и проверка результатов

Результат выполнения программы: c=1.4505

Ручная проверка.

Исходные данные: $f(x)= x^3 + 0,5\ x^2 - 2$, a=0, b=2, E=0.1

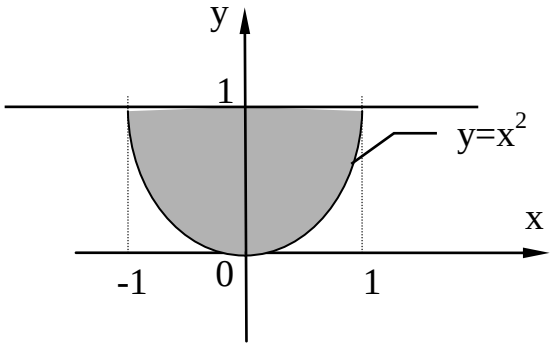
c=(a+b)/2	f(a)*f(c)	a	b	b-a	b-a<e
(0+2)/2=1	f(0)*f(1)>0	1	2	1	false
(1+2)/2=1.5	f(1)*f(1.5)<0	1	1.5	0.5	false
(1+1.5)/2=1.25	f(1)*f(1.25)>0	1.25	1.5	0.25	false
(1.25+1.5)/2=1.38	f(1.25)*f(1.38)>0	1.38	1.5	0.12	false
(1.38+1.5)/2=1.44	f(1.38)*f(1.44)>0	1.44	1.5	0.06	true

x=(1.44+1.5)/2≈1.47

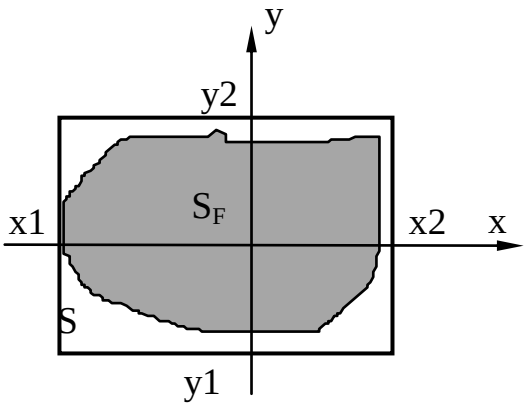
Пример 4. Вычисление площади произвольной фигуры методом Монте- Карло.

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.
3.



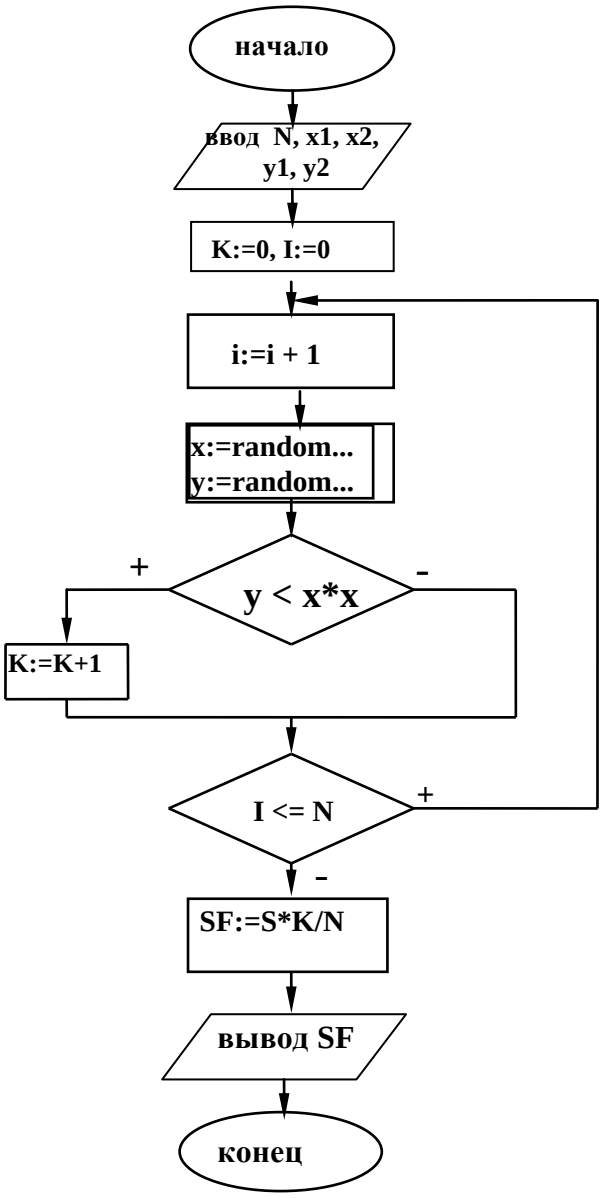
Для этого фигура помещается внутрь прямоугольника, образованного линиями x=x1, x= x2, y= y1, y=y2 (x1, x2, y1, y2 подбираются таким образом, чтобы фигура полностью лежала внутри полученного прямоугольника). Прямоугольник случайным образом заполняется N точками. Если внутрь фигуры F попали K точек, то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

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

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
  begin
    {выбор случайной точки}
          x:=random*(x2-x1)+x1;
         y:=random*(y2-y1)+y1;
    {подсчет количества точек, по-
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 программы 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		прогнозируемый	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

Контрольные вопросы

1. Основные этапы решения задач на компьютере.
2. Назначение каждого из этапов.
3. Принцип подбора контрольных примеров для тестирования

Порядок выполнения работы

1. Изучить теоретические сведения по теме “Основные этапы решения задач ”.
2. Ответить на контрольные вопросы.
3. Получить индивидуальное задание у преподавателя и разработать программу в соответствии с поставленной задачей.
4. Показать работающую программу преподавателю.
5. Подготовить письменный отчет.

Форма представления отчета:

- Тема лабораторной работы
- Условие задачи
- Текст программы с указанием исходных данных и результата.
- Результаты тестирования программы.

Задания для самостоятельной работы

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. 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)=|2cos x +1|,	a=0,	b=π.	n=100, 500, 1000.
e)	$f(x)=-(x+3)^2+3$,	a=-1,	b=3,	n=100, 200, 1000.
f)	$f(x)=(x+4)^2$,	a=-3,	b=2,	n=100, 500, 1000.
g)	f(x)=ln(2x+1),	a=1,	b=5,	n=100, 200, 1000.
h)	f(x)=|2sin x|,	a=0,	b=π/2.	n=100, 500, 1000.

3. Найти корень уравнения методом половинного деления с точностью E=0.0001:

a) cos x - ln x = 0;

b) cos x - $\sqrt{x}$ = 0;

c) tg x - 2^x = 0;

d) cos x - x^3 = 0;

a) 2cos x - ln x = 0;

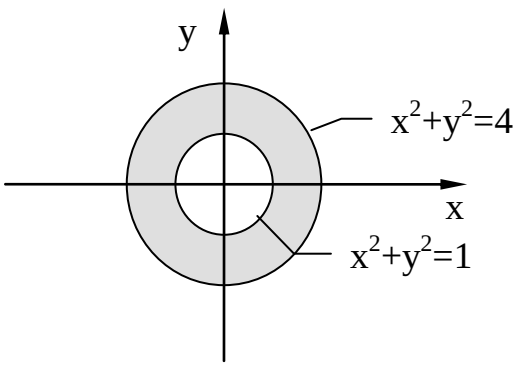
b) sin x - $\sqrt{x}$ = 0;

c) tg 2x – cos x = 0;

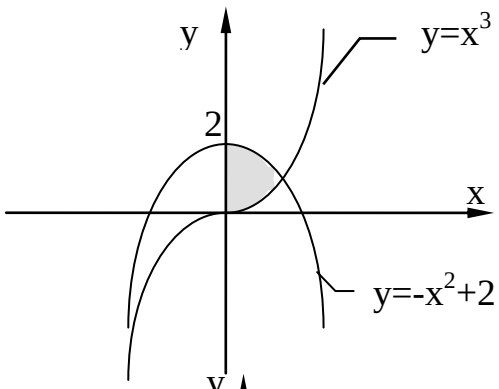
d) cos x - 3x = 0;

4. Найти площадь фигуры методом Монте-Карло.

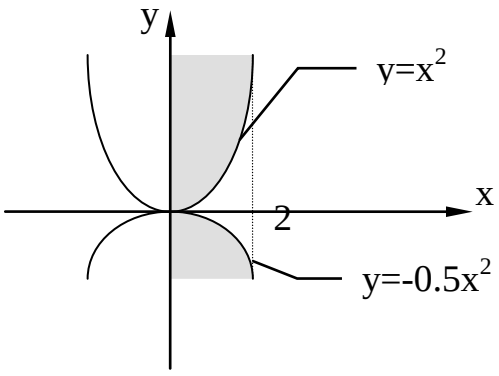
a)



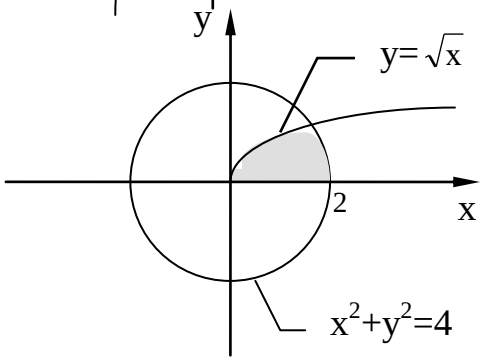
b)



c)



d)



2. СТРУКТУРЫ ДАННЫХ

2.1. СТЕК

Краткие теоретические сведения.

Организация данных для обработки является важным этапом разработки программ. Для реализации многих приложений выбор структуры данных – единственное важное решение: когда выбор сделан, разработка алгоритмов не вызывает затруднений. Для одних и тех же данных различные структуры будут занимать неодинаковое дисковое пространство. Одни и те же операции с различными структурами данных создают алгоритмы неодинаковой эффективности.

Наиболее важными из простейших структур данных являются *стек* и *очередь*.

Стек – это динамическая линейная структура данных, для которой определены всего две операции изменения набора элементов: добавление элемента в конец и удаление последнего элемента. Стек реализует принцип **LIFO** (Last in, First Out) – последним пришел и первым ушел.

Стек применяется довольно часто, причем в самых разных ситуациях. Объединяет их следующая цель: нужно сохранить некоторую работу, которая еще не выполнена до конца, при необходимости переключения на другую задачу. Стек используется для временного сохранения состояния не выполненного до конца задания. После сохранения состояния компьютер переключается на другую задачу. По окончании ее выполнения состояние отложенного задания восстанавливается из стека, и компьютер продолжает прерванную работу.

Например, в ходе выполнения программного кода, вычислительная машина при необходимости вызвать процедуру или функцию сначала заносит указатель на место ее вызова в стек, чтобы при завершении выполнения ее кода корректно вернуться к следующей после точки вызова инструкции. Такая структура данных называется стеком вызовов подпрограмм. Стек позволяет организовать рекурсию, т.е. обращение подпрограммы к самой себе либо непосредственно, либо через цепочку других вызовов

Реализовать стек можно любым удобным для программиста способом: например, массивом. Тогда началом стека (его «верхним» элементом) будет последний компонент массива, а освобождение стека будет происходить в направлении от конца массива к его началу. При такой реализации нет необходимости в постоянном перемещении компонент массива.

<u>Пример1. Моделирование стека</u>

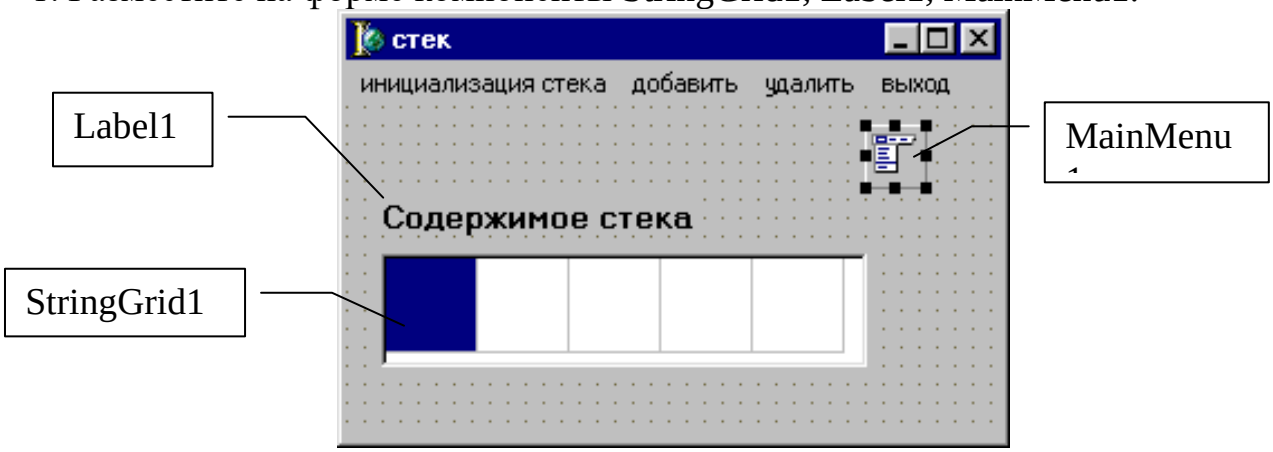
Создать в среде DELPHI проект, позволяющий продемонстрировать механизм работы стека, добавляя и удаляя элементы из него.

В проекте использованы следующие переменные:

St - передаваемый в стек элемент, **i** - номер последнего элемента в стеке, **n** – емкость стека.

Разработка проекта:

1. Разместите на форме компоненты StringGrid1, Label1, MainMenu1.



Для компонента StringGrid1 установите значение свойств:
FixedCols – 0, FixedRows – 0, RowCount – 1, DefaultColWidth – 40,
DefaultRowHeight - 40

2. Создайте главное меню формы (двойной щелчок по MainMenu1) следующей структуры

инициализация стека	добавить	удалить	выход

3. В раздел Var добавьте описание следующих переменных.

```
var
  Form1: TForm1;
  i, n : integer;
  St, Sn : string;
```

4. Создайте процедуру для инициализации стека (щелчок мышкой по пункту меню **инициализация стека**)

```
procedure TForm1.N1Click(Sender: TObject);
begin
 i:=0;
 n:=StrToInt(InputBox('емкость стека', 'введите количество элементов стека(1<n<20)', Sn));   {в окне ввода InputBox вводим количество элементов стека }
 StringGrid1.ColCount:=n;      {определяем заданное количество ячеек таблицы}
 StringGrid1.Width:=n*40+10;                    {задаем нужную ширину таблицы}
end;
```

5. Создайте процедуру, позволяющую добавить элемент в стек (щелчок мышкой по пункту меню **добавить**)

```
procedure TForm1.N2Click(Sender: TObject);
begin
  st:=InputBox('добавление элемента', 'введите элемент',St);   {вводим элемент}
  If   i<n then                                   {если стек заполнен не полностью}
            begin              {помещаем введенный элемент в следующую ячейку}
                 i:=i+1;       StringGrid1.Cells[I-1,0]:=St
            end
        else   ShowMessage('ПЕРЕПОЛНЕНИЕ СТЕКА!');
end;
```

6. Создайте процедуру, позволяющую извлечь элемент из стека (щелчок мышкой по пункту меню **удалить**)

```
procedure TForm1.N3Click(Sender: TObject);
begin
   If i<>0 then                                                  {если стек не пуст}
            begin                                          {удаляем элемент из него}
                 i:=i-1;        StringGrid1.Cells[I,0]:=";
            end
        else  ShowMessage('СТЕК ПУСТ!');
end;
```

7. Создайте процедуру, позволяющую завершить работу проекта (щелчок мышкой по пункту меню **выход**)

```
procedure TForm1.N4Click(Sender: TObject);
begin
  close
end;
```

8. Запустите проект на исполнение и проверьте его работу.

9. Доработайте проект таким образом, чтобы он позволял найти сумму произвольных элементов стека (предусмотреть возможность суммирования извлекаемых элементов стека)

<u>Пример 2.</u>

Составить в среде PascalABC программу, которая позволит сформировать стек из N натуральных чисел, просмотреть его и уменьшить нечетные его элементы на 1.

Работа стека моделируется с помощью одномерного массива **S[1..N].** Для стека определены операции: инициализация стека, добавление элемента в стек, извлечение элемента из стека, просмотр стека. Операции оформлены в программе как процедуры. **St** - передаваемый или извлекаемый из стека элемент, **i** - номер последнего элемента в стеке, **n** – емкость стека.

```
Program Stek;
    Var   s: array[1..100] of integer;
          i, j, n, st: integer;
Procedure IniSt;                                        {инициализация стека}
   Begin
        Write('Емкость стека'); Readln(n);   i:=0;
   End;
Procedure DobSt (st: integer);                          {добавление элемента в стек}
    Begin
        If i<n then  begin   i:=i+1; s[i]:=st;   end
                 else  begin   write('Стек переполнен'); Readln;  end;
    End;
Procedure IsvSt(var st:integer);                        {извлечение элемента из стека}
    Begin
        If i<>0 then begin   st:=s[i];   i:=i-1;   end
                 else begin  write('Стек пуст'); Readln;   end;
    End;
Procedure ProsmSt;                                      {просмотр стека}
    Var   k: integer;
    Begin
          Writeln('ПРОСМОТР   СТЕКА');
          If i<>0   then  For   k:=i  downto 1 do Writeln(k,' -й элемент: ', s[k])
                    else  Writeln('-----СТЕК ПУСТ---');
          Readln ;
    End;
                                  {основная программа}
BEGIN
    IniST;                                              {инициализация стека}
    For   j:=1  to  n  do  begin          {заполнение стека введенными элементами}
                  write('Ввод', j,'-го элемента'); readln(st);   DobSt(st);
```

```
    End;
    ProsmSt;                                        {просмотр стека}
    Repeat                          {извлечение и проверка элементов стека}
        IsvSt(st);
        If  st mod 2 =1 then st:=st-1;
        Writeln(st);
    Until i=0;
END.
```

Контрольные вопросы

1. Понятие стека.
2. Принцип работы стека.
3. Основные операции над стеком.
4. Примеры использования стека.
5. Способы реализации стека.

Порядок выполнения работы

1. Изучить теоретические сведения по теме “Структуры данных. Стек ”.
2. Ответить на контрольные вопросы.
3. Реализовать в среде Delphi пример 1 и продемонстрировать работу проекта преподавателю.
4. Получить индивидуальное задание у преподавателя и разработать программу в соответствии с поставленной задачей.
5. Показать работающую программу преподавателю.
6. Подготовить письменный отчет.

Форма представления отчета:

- Тема лабораторной работы
- Условие задачи
- Текст программы с указанием исходных данных и результата.
- Результаты тестирования программы.

Задания для самостоятельной работы

1. Заполнить стек 10 случайными числами из интервала [-10;20]. Просмотреть содержимое стека. Найти сумму положительных чисел, хранящихся в стеке.
2. Сформировать стек из 6 чисел. Извлечь элементы из стека, увеличить каждое из них на 1 и снова поместить в стек в том же порядке.
3. Сформировать стек из 10 чисел. Извлечь элементы из стека, положительные заменить на 1, а отрицательные на -1 и снова поместить в стек в том же порядке.
4. Сформировать стек из 6 чисел. Извлечь элементы из стека, найти их сумму и произведение. Результат поместить в стек.

5. Заполнить стек 10 случайными числами из интервала [-10;80]. Найти количество чисел, которые при делении на 5 дают в остатке 3.

6. Сформировать стек из 10 чисел. Найти сумму первых 5 чисел и результат поместить в стек.

7. Сформировать стек из 8 чисел. Найти произведение второго и третьего чисел. Результат поместить в стек.

8. Сформировать стек из 7 чисел. Найти сумму 3-го, 4-го и 5-го чисел из стека. Результат поместить в стек.

9. Сформировать стек из 5 чисел. Найти произведение 3-го и 4-го чисел из стека. Результат поместить в стек.

10. Заполнить стек 10 случайными числами из интервала [-10;20]. Найти максимальное число.

11. Сформировать стек из 8 чисел. Извлечь элементы из стека, найти их среднее арифметическое. Результат поместить в стек.

12. Заполнить стек 10 случайными числами из интервала [10; 100]. Найти количество чисел, которые при делении на 7 дают в остатке 2.

13. Сформировать стек из 10 чисел. Найти сумму последних 5 чисел и результат поместить в стек.

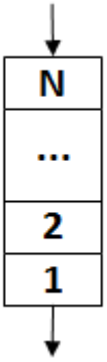
14. Сформировать стек из 8 чисел. Найти среднее арифметическое второго и третьего чисел. Результат поместить в стек.

15. Заполнить стек 12 случайными числами из интервала [-50; 70]. Просмотреть содержимое стека. Найти сумму отрицательных чисел, хранящихся в стеке. Результат поместить в стек.

2.2. ОЧЕРЕДЬ

Краткие теоретические сведения.

Очередь – очень похожая на стек, динамическая структура данных, с той лишь разницей, что она реализует принцип **FIFO** (First in, First out) – первым пришел и первым ушел. Очередь содержит элементы, как бы выстроенные друг за другом в цепочку. У очереди есть начало и конец. Добавлять новые элементы можно только в конец очереди, забирать элементы можно только из начала. В отличие от обычной очереди, которую всегда можно при желании покинуть, из середины программистской очереди удалять элементы нельзя.



Очередь.

Использование очереди в программировании почти соответствует ее роли в обычной жизни. Очередь практически всегда связана с обслуживанием запросов, в тех случаях, когда они не могут быть выполнены мгновенно. С помощью очередей, например, обрабатывают события пользовательского интерфейса, обращения клиентов к веб - сервисам и прочие информационные запросы.

<u>Пример1. Моделирование очереди</u>

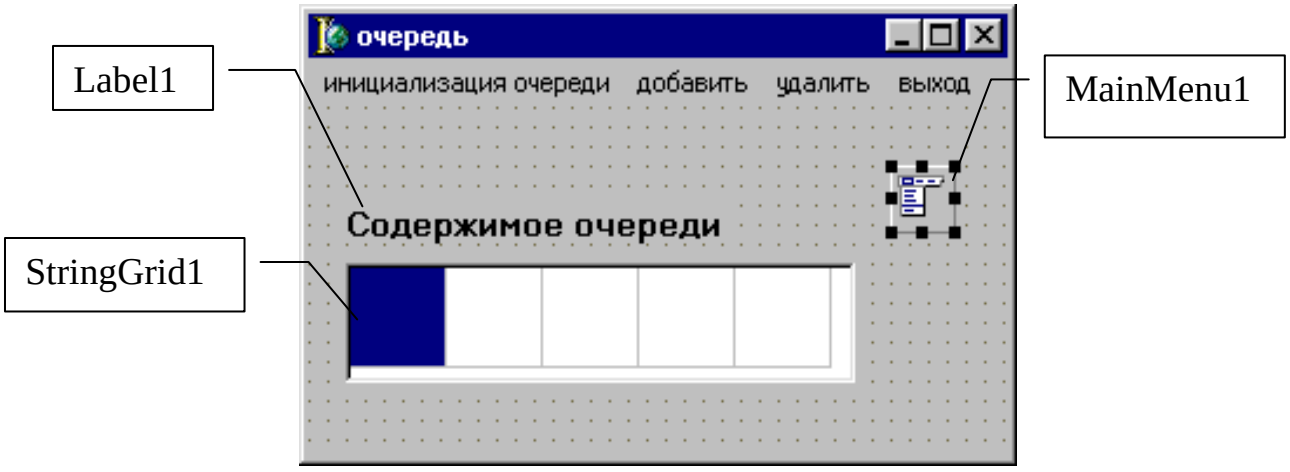
Создать в среде DELPHI проект, позволяющий продемонстрировать механизм работы очереди, добавляя и удаляя элементы из нее.

В проекте использованы следующие переменные:

t - передаваемый в очередь элемент, **Ob** - номер первого элемента в очереди, **Oe** - номер последнего элемента в очереди, **n**- емкость очереди.

Разработка проекта:

1. Разместите на форме компоненты StringGrid1, Label1, MainMenu1.



Для компонента StringGrid1 установите значение свойств:
FixedCols – 0, FixedRows – 0, RowCount – 1, DefaultColWidth – 40, DefaultRowHeight - 40

2. Создайте главное меню формы (двойной щелчок по MainMenu1) следующей структуры

инициализация очереди	добавить	удалить	выход

3. В раздел Var добавьте описание следующих переменных.

```
var
  Form1: TForm1;
  Ob, Oe,  n : integer;
  t, Sn : string;
```

4. Создайте процедуру для инициализации стека (щелчок мышкой по пункту меню **инициализация очереди**)

```
procedure TForm1.N1Click(Sender: TObject);
begin
    Ob:=0; Oe:=0;
    n:=StrToInt(InputBox('емкость очереди', 'введите количество элементов очереди(1<n<20)', Sn));    {в окне ввода InputBox вводим количество элементов очереди}
    StringGrid1.ColCount:=n;     {определяем заданное количество ячеек таблицы}
    StringGrid1.Width:=n*40+10;                    {задаем нужную ширину таблицы}
 end;
```

5. Создайте процедуру, позволяющую добавить элемент в очередь (щелчок мышкой по пункту меню **добавить**)

```
procedure TForm1.N2Click(Sender: TObject);
begin
  st:=InputBox('добавление элемента', 'введите элемент',t);     {вводим элемент}
  if Oe<>N then                                  {если очередь заполнена не полностью}
        begin                                     {помещаем в конец очереди элемент}
              Oe:=Oe+1;
              StringGrid1.Cells[Oe-1,0]:=t
        end
       else  ShowMessage('ПЕРЕПОЛНЕНИЕ ОЧЕРЕДИ!');
end;
```

6. Создайте процедуру, позволяющую извлечь элемент из очереди (щелчок мышкой по пункту меню **удалить**)

```
procedure TForm1.N3Click(Sender: TObject);
begin
    if Ob<>Oe then                                        {если очередь не пуста}
                  begin                        {извлекаем элемент из головы очереди}
                      Ob:=Ob+1;  StringGrid1.Cells[Ob-1,0]:='';
                  end
```

```
                    else  ShowMessage('ОЧЕРЕДЬ ПУСТА!');
end;
```

7. Создайте процедуру, позволяющую завершить работу проекта (щелчок мышкой по пункту меню **выход**)

```
procedure TForm1.N4Click(Sender: TObject);
begin
  close
end;
```

8. Запустите проект на исполнение и проверьте его работу.

9. Доработайте проект таким образом, чтобы он позволял найти произведение произвольных элементов очереди (предусмотреть возможность умножения извлекаемых элементов очереди)

<u>Пример 2.</u>

Составить в среде PascalABC программу, которая позволит сформировать очередь из N натуральных чисел, просмотреть ее и найти сумму второго и четвертого элементов очереди. Результат поместить в очередь.

Работа очереди моделируется с помощью одномерного массива **O[1..N].** Для очереди определены операции: инициализация очереди, добавление элемента в очередь, извлечение элемента из очереди, просмотр очереди, которые оформлены в программе как процедуры. **t** - передаваемый или извлекаемый из стека элемент, **Ob** - номер первого элемента в очереди, **Oe** - номер последнего элемента в очереди, **n**- емкость очереди.

```
Program Ochered;
    Var    O: array[1..100]  of  integer;
             t, S, Ob, Oe, n, j:  integer;
Procedure IniO;                                        {инициализация очереди}
     begin
         Write('Емкость очереди');   Readln(n);
          Ob:=0; Oe:=0;
    end;
 Procedure DobO(t: integer);                           {добавление элемента в очередь}
    begin
         if Oe<>N then  begin  Oe:=Oe+1; o[Oe]:=t;   end
                    else  begin  write('Переполнение'); Readln;  end;
    end;
 Procedure IsvO(Var t:integer);                        {извлечение элемента из очереди}
    begin
          if Ob<>Oe then begin   Ob:=Ob+1;t:=o[Ob];   end
                     else begin   write('Очередь пуста'); Readln;   end;
```

```
    end;
 Procedure ProsmO;                                          {просмотр очереди}
     var   k:  integer;
     begin
          writeln('ПРОСМОТР ОЧЕРЕДИ');
          for   k:=Ob+1   to   Oe   do  writeln(k,' -й элемент ',o[k]);
          writeln('----------------------------------------------------');
          readln;
    end;
BEGIN                          {ОСНОВНАЯ ПРОГРАММА}
    IniO;
    for   j:=1   to  n  do  begin                          { заполнение очереди}
          write('Ввод', j,'-го элемента '); readln(t);
          DobO(t);
    end;
    ProsmO;
    S:=0;    IsvO(t);                    {извлечение первого элемента  очереди}
    IsvO(t);  S:=S+t;              {извлечение и суммирование второго элемента}
    IsvO(t);                          {извлечение третьего элемента  очереди}
    IsvO(t);  S:=S+t;             {извлечение и суммирование четвертого элемента}
    t:=S; DobO(t);                                  {добавление суммы  в очередь}
    ProsmO;   readln
END.
```

Контрольные вопросы

1. Понятие очереди.
2. Принцип работы очереди.
3. Основные операции над очередью.
4. Примеры использования очереди.
5. Способы реализации очереди.

Порядок выполнения работы

1. Изучить теоретические сведения по теме “Структуры данных. Очередь ”.
2. Ответить на контрольные вопросы.
3. Реализовать в среде Delphi пример 1 и продемонстрировать работу проекта преподавателю.
4. Получить индивидуальное задание у преподавателя и разработать программу в соответствии с поставленной задачей.
5. Показать работающую программу преподавателю.
6. Подготовить письменный отчет.

Форма представления отчета:

- Тема лабораторной работы

- Условие задачи
- Текст программы с указанием исходных данных и результата.
- Результаты тестирования программы.

Задания для самостоятельной работы

1. Сформировать очередь из 8 чисел. Утроить модуль разности между 2-м и 3-м числом очереди. Результат поместить в очередь.
2. Сформировать очередь из 7 чисел. Найти модуль разности между 2-м и 3-м числом очереди.
3. Сформировать очередь из 10 чисел. Найти сумму первых 5 чисел из очереди. Результат поместить в очередь.
4. Сформировать очередь из 8 чисел. Найти произведение первых 4 чисел из очереди.
5. Сформировать очередь из 8 чисел. Найти сумму 3-го, 4-го и 5-го чисел из очереди.
6. Сформировать очередь из 8 чисел. Найти произведение 3-го и 4-го чисел из очереди. Результат поместить в очередь.
7. Заполнить очередь 8 случайными числами из интервала [-20;50]. Найти среднее арифметическое 2-го и 3-го чисел очереди. Результат поместить в очередь.
8. Заполнить очередь 10 случайными числами из интервала [10;80]. Найти количество чисел, которые при делении на 7 дают в остатке 4.
9. Заполнить очередь 8 случайными числами из интервала [0;50]. Найти среднее арифметическое четных чисел. Результат поместить в очередь.
10. Сформировать очередь из 8 чисел. Найти сумму 2-го и 4-го чисел из очереди. Результат поместить в очередь.
11. Сформировать очередь из 10 чисел. Найти произведение 3-го, 4-го и 5-го чисел из очереди.
12. Заполнить очередь 8 случайными числами из интервала [20; 70]. Найти среднее арифметическое 2-го и 4-го чисел очереди.
13. Заполнить очередь 10 случайными числами из интервала [-40; 50]. Найти количество положительных чисел. Результат поместить в очередь.
14. Заполнить очередь 15 случайными числами из интервала [0;100]. Найти среднее арифметическое нечетных чисел. Результат поместить в очередь.
15. Сформировать очередь из 8 чисел. Найти среднее арифметическое первых 4 чисел из очереди. Результат поместить в очередь.

3. РЕКУРСИВНЫЕ АЛГОРИТМЫ

Краткие теоретические сведения.

Рекурсия – это определение объекта через обращение к самому себе.

Рекурсивный алгоритм – это алгоритм, в описании которого прямо или косвенно содержится обращение к самому себе. В технике процедурного программирования данное понятие распространяется на функцию, которая реализует решение отдельного блока задачи посредством вызова из своего тела других функций, в том числе и себя самой. Если при этом на очередном этапе работы функция организует обращение к самой себе, то такая функция является *рекурсивной*.

Главное требование к рекурсивным процедурам заключается в том, что вызов рекурсивной процедуры должен выполняться по условию, которое на каком-то уровне рекурсии станет ложным. Это позволит завершить рекурсию за конечное число шагов.

Прямое обращение функции к самой себе предполагает, что в теле функции содержится вызов этой же функции, но с другим набором фактических параметров. Такой способ организации работы называется *прямой* рекурсией. Например, чтобы найти сумму первых n натуральных чисел, надо сумму первых (n-1) чисел сложить с числом n, то есть имеет место зависимость: Sn=Sn-1+n.

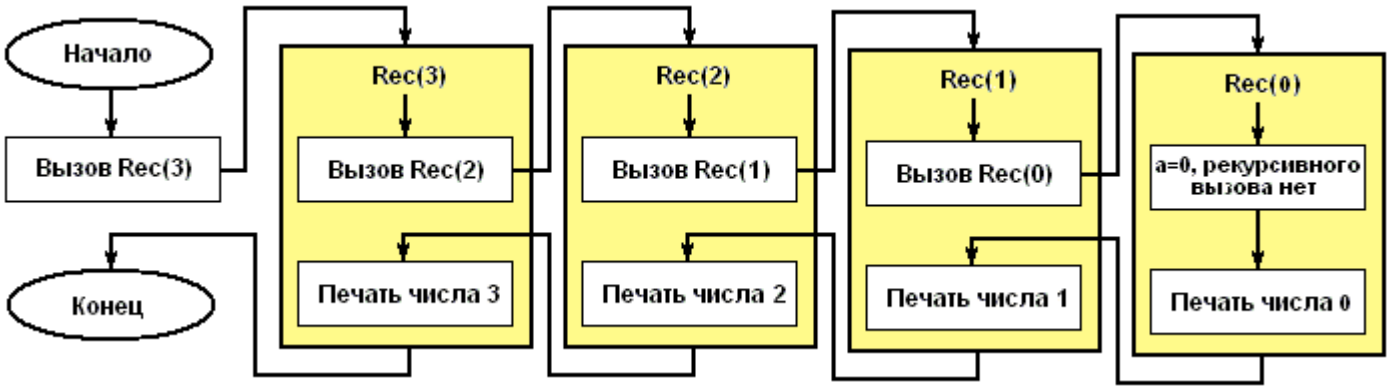
Косвенной или *взаимной* рекурсией называется организация вызовов нескольких процедур и функций по кругу (первая процедура вызывает вторую, вторая - третью, ..., n-я процедура вызывает первую).

Реализация рекурсивных вызовов функций в практически применяемых языках и средах программирования, как правило, опирается на механизм стека вызовов — адрес возврата и локальные переменные функции записываются в стек, благодаря чему каждый следующий рекурсивный вызов этой функции пользуется своим набором локальных переменных и за счёт этого работает корректно.

Пример рекурсивной процедуры, печатающей последовательные целые числа от 0 до заданного значения a:

```
procedure Rec(a: integer);
begin
    if  a>0  then  Rec(a-1);
   writeln(a);
end;
```

Рассмотрим, что произойдет, если в основной программе поставить вызов, например, вида Rec(3). Ниже представлена блок-схема, показывающая последовательность выполнения операторов.



Процедура Rec вызывается с параметром a = 3. В ней содержится вызов процедуры Rec с параметром a = 2. Предыдущий вызов еще не завершился, поэтому можете представить себе, что создается еще одна процедура и до окончания ее работы первая свою работу не заканчивает. Процесс вызова заканчивается, когда параметр a = 0. В этот момент одновременно выполняются 4 экземпляра процедуры. Количество одновременно выполняемых процедур называют *глубиной рекурсии*.

Четвертая вызванная процедура (Rec(0)) напечатает число 0 и закончит свою работу. После этого управление возвращается к процедуре, которая ее вызвала (Rec(1)) и печатается число 1. И так далее пока не завершатся все процедуры. Результатом исходного вызова будет печать четырех чисел: 0, 1, 2, 3.

Классический пример функции, определение которой может задаваться в рекурсивной форме - факториал F(n)=n!

N!=1*2*3*…*N

Вычисление факториала рекурсивно можно определить следующим образом:

$$N! = \begin{cases} если N = 1, то N! = 1 \\ если N > 1, то N! = N * (N-1)! \end{cases}$$

Например, 5!= 5*4!=5*4*3!=5*4*3*2!=5*4*3*2*1!= 5*4*3*2*1

<u>Пример 1.</u>

Вычисление факториала натурального числа **N**.

```
Program Pr1;
Var
    N: integer;
Function Fact(N:integer):real;
begin
  if N=1 then Fact:=1 else Fact:=N*Fact(N-1);
end;
BEGIN                    {основная  программа}
  Write('N='); Readln(N);
  Writeln(N, '!=', Fact(N):4:0);
  Readln
```

END.

Рекурсивный алгоритм обращается к самому себе, пока не выполнится определенное условие, поэтому в любой рекурсивной подпрограмме должна быть нерекурсивная ветвь (в примере 1 это: if N=1 then Fact:=1).

<u>Пример 2.</u>

Расчет суммы первых N натуральных чисел

Вычисление суммы 1+2+3+…+N можно определить следующим образом:

$$\text{Sum(N)} = \begin{cases} \text{если} N = 0, \text{то} Sum = 0 \\ \text{если} N > 0, \text{то} Sum = N + Sum(N-1) \end{cases}$$

```
Program Pr2;
Var
    N: integer;
Function Sum(N: integer): integer;
begin
  if N=0 then Sum:=0 else Sum:= N+Sum(N-1);
end;

BEGIN                {основная программа}
  Write('N='); Readln(N);
  Writeln('S=', Sum(N));
  Readln
END.
```

<u>Пример 3.</u>

Вычисление **n**-го члена последовательности Фибоначчи.

Последовательность Фибоначчи определяется следующим образом: первые два числа равны 1, а каждое следующее равно сумме двух предыдущих (1, 1, 2, 3, 5,…).

Вычисление **n**-го члена последовательности Фибоначчи можно определить следующим образом:

$$\text{Fib(n)} = \begin{cases} \text{если} N = 1 \text{или} N = 2, \text{то} \text{Fib} := 1 \\ \text{если} N > 2, \text{то} \text{Fib} := \text{Fib(n-1)} + \text{Fib(n-2)} \end{cases}$$

```
Program Pr3;
Var
     n:integer;
Function Fib(n:integer):integer;
begin
```

```
  if n<=2 then Fib:=1 else Fib:=Fib(n-1)+Fib(n-2);
end;
BEGIN            {основная  программа}
  Write('n='); Readln(n);
  Write(Fib(n):5);
  Readln
END.
```

<u>Пример 4.</u>

Расчет n-члена арифметической прогрессии, заданной значением первого члена a1 и разностью d.

Вычисление n-члена арифметической прогрессии можно определить следующим образом:

$$a_n = \begin{cases} если N = 1, то a_n = a_1 \\ если N > 1, то a_n = a_{n-1} + d \end{cases}$$

```
Program Pr4;
Var
     n:integer;  d,a1:real;
Function Arifm(a1,d:real; n:integer):real;
begin
  if n=1 then Arifm:=a1 else Arifm:=Arifm(a1,d,n-1) + d;
end;
BEGIN       {основная  программа}
  Write('a1='); Readln(a1);
  Write('d=');  Readln(d);
  Write('n=');  Readln(n);
  Write(Arifm(a1,d, n):5:2);
   Readln
END.
```

<u>Пример 5.</u>

Нахождения суммы **n** членов арифметической прогрессии, заданной значениями первого члена прогрессии **a** и разности **d.**

```
Program Pr5;
Var
     s,a,n,d: integer;
Function sa(n,a,d:integer):integer;
begin
  if n>0 then sa:=a+ sa(n-1,a+d,d) else sa:=0;
end;
```

```
BEGIN        {основная  программа}
  Write('n='); Readln(n);
  Write('a='); Readln(a);
  Write('d='); Readln(d);
  Write(sa(n,a,d));
  Readln
END.
```

<u>Пример 6.</u>

Возведение числа **a** в целую степень **n**.

```
Program Pr6;
Var  a:real;
    n:integer;
Function Stepen(a: real;n:integer):real;
begin
  if n=0 then Stepen:=1
        else if n>0 then Stepen:=a*Stepen(a,n-1)
                  else Stepen:=(1/a)*Stepen(a,n+1);
end;
BEGIN    {основная  программа}
  Write('a='); Readln(a);
  Write('n='); Readln(n);
  Write(Stepen(a,n):6:4);
  Readln
END.
```

<u>Пример 7.</u>

Представление натурального числа в двоичной системе счисления.

```
Program Pr7;
Var
      n:integer;
Procedure Bin_n(n:integer);
begin
  if n>1 then Bin_n(n div 2);
  Write( n mod 2)
end;
BEGIN              {основная  программа}
  Write('n=');  Readln(n);
  Bin_n(n);
  Readln
END.
```

<u>Пример 8.</u>

Нахождения наибольшего общего делителя двух натуральных чисел

```
Program Pr8;
Var
      a,b:integer;
Function Nod(a,b:integer):integer;
begin
  if a=b  then  Nod:=a
           else   if a>b  then Nod:=Nod(a-b, b)
                          else Nod:=Nod(a, b-a)  ;
end;
BEGIN       {основная  программа}
  Write('a='); Readln(a);
  Write('b='); Readln(b);
  Writeln('NOD(',  a,  ',',  b,  ')=',  Nod(a,b));
  Readln
END.
```

Контрольные вопросы

1. Понятие рекурсии и рекурсивного алгоритма.
2. Что означает прямая рекурсия?
3. Что означает косвенная рекурсия?
4. Главное требование к рекурсивным процедурам.
5. Что называют глубиной рекурсии?
6. Каким образом реализуется рекурсивный вызов?

Порядок выполнения работы

1. Изучить теоретические сведения по теме “Рекурсивные алгоритмы ”.
2. Ответить на контрольные вопросы.
3. Получить индивидуальное задание у преподавателя и разработать программу в соответствии с поставленной задачей.
4. Показать работающую программу преподавателю.
5. Подготовить письменный отчет.

Форма представления отчета:

- Тема лабораторной работы
- Условие задачи
- Текст программы с указанием исходных данных и результата.
- Результаты тестирования программы.

Задания для самостоятельной работы

1. Вычислить (a! + b!)/a!, используя рекурсивную функцию вычисления факториала
2. Вычислить m!/(m + n)!, используя рекурсивную функцию вычисления факториала
3. Вычислить (1+2+3+4+5)/(1+2+3+4+5+6+7+8), используя рекурсивную функцию вычисления суммы первых n натуральных чисел.
4. Составить рекурсивную функцию для вычисления S = 2 + 4 + 6 +…+24
5. Составить рекурсивную функцию для вычисления S = 5 + 10 + 15 +…+55
6. Составить рекурсивную функцию для вычисления P = 2* 4* 6*…*12
7. Составить рекурсивную функцию вычисления **n**-го члена арифметической прогрессии 3, 7, … и вывести первые 10 членов прогрессии .
8. Составить рекурсивную функцию вычисления **n**-го члена геометрической 1, 2, … и вывести первые 8 членов прогрессии .
9. Составить рекурсивную функцию вычисления **n**-го члена последовательности: $a_1= 1$, $a_i = a_{i-1}*i$. Найти сумму 2-го и 5-го членов последовательности.
10. Составить рекурсивную функцию вычисления **n**-го члена последовательности: $a_1= 0$, $a_i = 2*a_{i-1}+i$. Найти произведение 3-го и 7-го членов последовательности.
11. Составить рекурсивную функцию нахождения суммы **n** членов арифметической прогрессии 1, 3, …Найти сумму с 5-го по 10-й членов прогрессии
12. Составить рекурсивную процедуру, которая печатает введенное натуральное число в восьмеричном представлении
13. Найти наибольший общий делитель для чисел A, B, C, используя рекурсивную функцию нахождения наибольшего общего делителя двух натуральных чисел
14. Сократить дробь a/b (a, b – введенные натуральные числа), используя рекурсивную функцию нахождения наибольшего общего делителя двух натуральных чисел
15. Вычислить $(4^2 + 2^3)/2^{-2}$, используя рекурсивную функцию возведения в степень

4. ПОИСК ДАННЫХ

Краткие теоретические сведения

Во всех компьютерных информационных системах поиск данных является основным видом обработки информации. При выполнении любого поиска данных имеются три составляющие, которые мы назовем атрибутами поиска:

1) набор данных. Это вся совокупность данных, среди которых осуществляется поиск.

2) ключ поиска. Это величина, по значению которой происходит поиск.

3) критерий поиска, или условие поиска. Это то условие, которому должно удовлетворять значение ключа поиска.

Существуют различные алгоритмы поиска. Рассмотрим поиск на примере одномерного массива.

Если исходный массив не упорядочен, то к нему можно применить метод *последовательного (линейного)* поиска.

Линейный поиск заключается в последовательном переборе элемента массива и сравнения каждого из них с ключом поиска k. Перебор ведется до тех пор, пока не будет найден элемент массива, равный k или пока не будет просмотрен весь массив. В этом случае поиск ведется до первого вхождения ключа поиска в массив.

Если исходные данные упорядочены (по возрастанию или по убыванию), в этом случае можно применить более эффективный **бинарный** поиск.

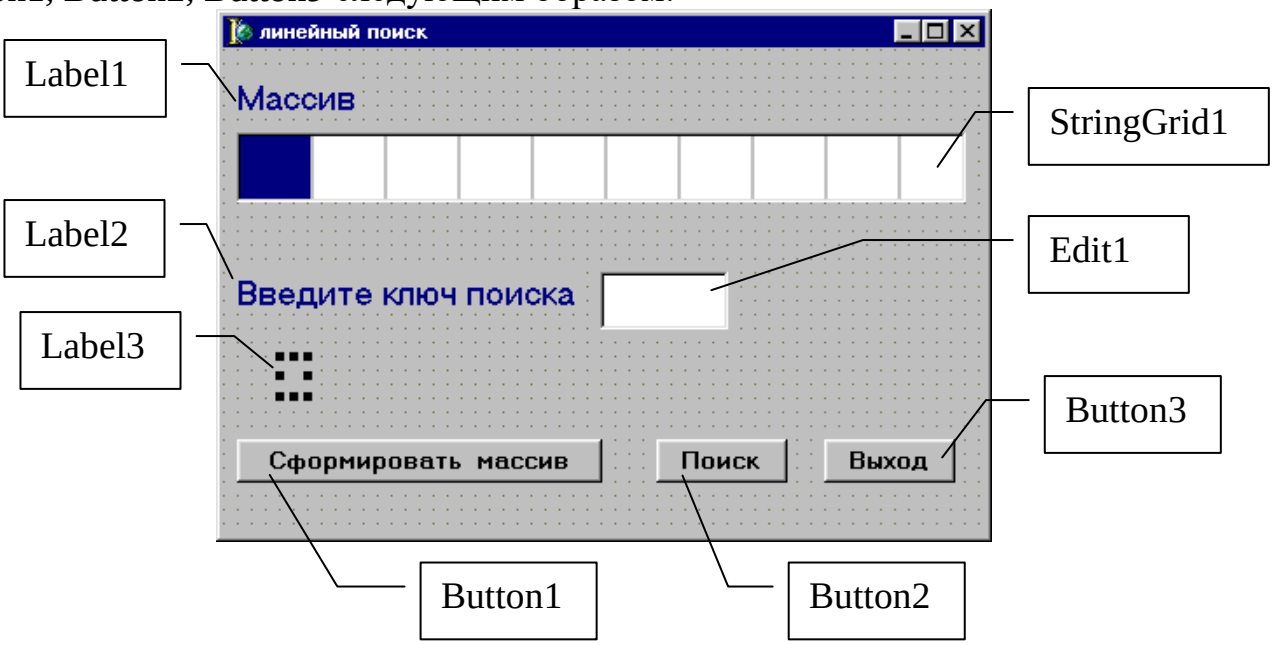
В этом случае массив делится на две части путем выбора серединного элемента, который сравнивается с ключом поиска. Если они равны, то поиск завершен. Если серединный элемент не совпал с ключом поиска, то выбирается та часть массива, где может находиться искомый элемент. Она снова делится на две части и т.д. до тех пор, пока элемент не будет найден или пока не будут рассмотрены все возможные варианты.

<u>Пример 1.</u> Линейный поиск

Создать проект, позволяющий заполнить массив a[0..9] целыми случайными числами и определить, содержит ли он заданное число.

Разработка проекта:

1. Разместите на форме компоненты StringGrid1, Label1, Label2, Label3, Edit1, Button1, Button2, Button3 следующим образом:



Для компонента StringGrid1 установите значение свойств:

FixedCols – 0, FixedRows – 0, ColCount – 10, RowCount – 1, DefaultColWidth – 30, DefaultRowHeight - 30

2. В раздел Var добавьте описание следующих переменных.

```
Var
   Form1: TForm1;
    a: array[0..9] of integer;
    i, k, kod:integer;
```

3. Создайте процедуру для формирование массива (двойной щелчок мышкой по Button1)

```
procedure TForm1.Button1Click(Sender: TObject);
begin
 randomize;                    {выбираем новую последовательность случайных чисел}
 for i:=0 to  9 do begin                                          {формирование массива}
     a[i]:= random(100);                    {выбираем целое случайное число от 0 до 99}
     StringGrid1.Cells[i,0]:=IntToStr(a[i]);      {помещаем число в i-ю ячейку таблицы}
     end;
end;
```

4. Создайте процедуру для поиска заданного элемента в массиве (двойной щелчок мышкой по Button2)

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  val(edit1.Text, k, kod); {переводим введенное строковое значение ключа в числовое}
  if kod=0 then begin                          {если преобразование выполнено успешно,}
   i:=0;    While (a[i]<>k) and ( i<=9) do  i:=i+1;                  { осуществляем поиск}
   if i<=9                                                {проверяем условие выхода из цикла}
            then   label3.Caption:='Элемент найден. Его номер - '+ IntToStr(i)
            else   label3.Caption:='Элемент не найден';
  end;
end;
```

5. Создайте процедуру для завершения работы приложения (двойной щелчок мышкой по Button3)

```
procedure TForm1.Button3Click(Sender: TObject);
begin
    close
end;
```

6. Запустите проект на исполнение и проверьте его работу, задавая различные значения ключа поиска.

7. Доработайте данный проект следующим образом:

а) разместите на форме кнопку **Удалить;**

б) создайте процедуру, которая при щелчке по этой кнопке позволит удалить найденный элемент массива в случае успешного завершения поиска.

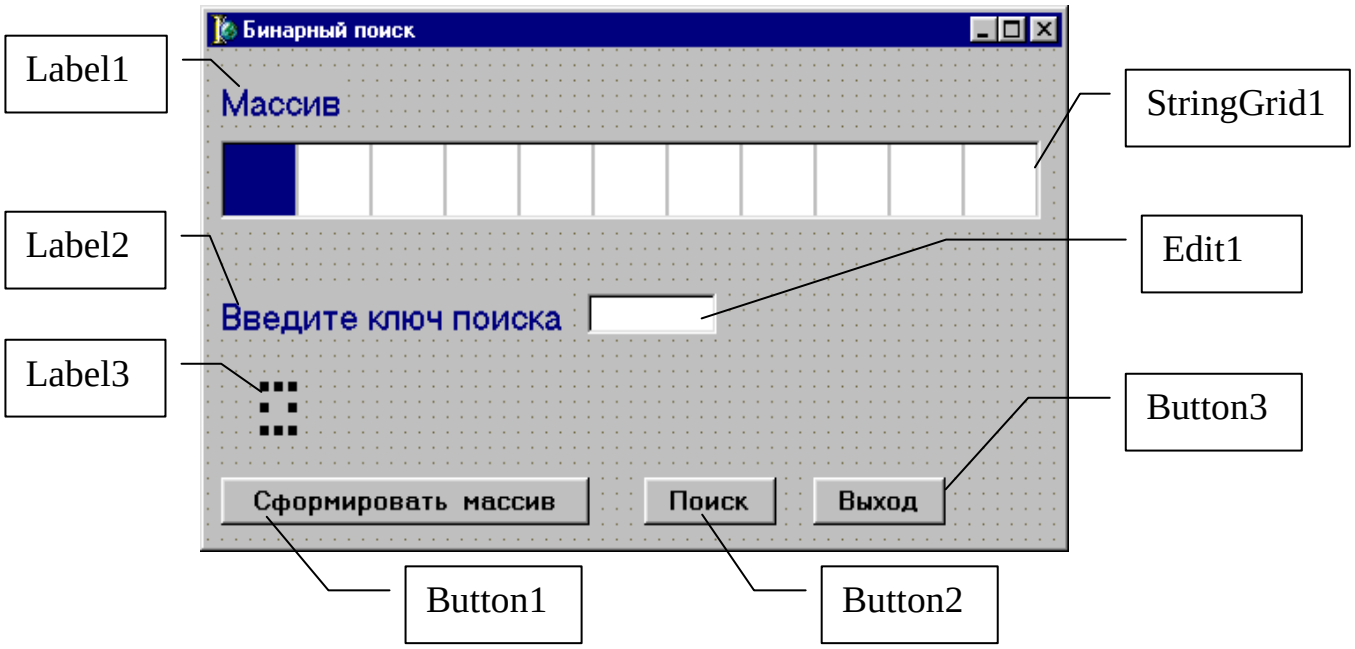
<u>Пример 2.</u> Бинарный поиск

Создать в среде DELPHI проект, позволяющий сформировать упорядоченный по возрастанию массив a[0..9] и определить, содержит ли он введенное число.

В программе использованы следующие обозначения: l - левая граница, r - правая граница, m - номер серединного элемента.

Разработка проекта:

1. Разместите на форме компоненты StringGrid1, Label1, Label2, Label3, Edit1, Button1, Button2, Button3 следующим образом:



Для компонента StringGrid1 установите значение свойств:
FixedCols – 0, FixedRows – 0, ColCount – 11, RowCount – 1, DefaultColWidth – 30, DefaultRowHeight - 30

2. В раздел Var добавьте описание следующих переменных.

```
Var
  Form1: TForm1;
   a : array[0..10] of integer;
   i, l, r, m, k, kod : integer;
```

3. Создайте процедуру для формирования упорядоченного по возрастанию массива (двойной щелчок мышкой по Button1)

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

```
begin
 for i:=0 to  9 do begin
     a[i]:= 3*i+2;  Вычисляем i-й элемент массива
     StringGrid1.Cells[i,0]:=IntToStr(a[i]);       {помещаем его в i-ю ячейку таблицы}
     end;
end;
```

4. Создайте процедуру для поиска заданного элемента в массиве (двойной щелчок мышкой по Button2)

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  val(edit1.Text, k, kod); {переводим введенное строковое значение ключа в числовое}
  if kod=0 then begin                          {если преобразование выполнено успешно,}
      l:=0;    r:=9;                  {устанавливаем левую (l) и правой (r) границы поиска}
      m:=(l+r) div 2;                                        {выбираем серединный элемент}
      while   (l<=r) and (k<>a[m])    do   begin                {поиск заданного элемента}
               if    k<a[m] then   r:=m-1   else    l:=m+1;
               m:=(l+r) div 2;
      end;
      if   k=a[m]      {если значение ключа поиска совпало с элементом массива, то..}
             then  Label3.Caption:='Элемент найден. Его номер - ' + IntToStr(m)
             else   Label3.Caption:='Элемент не найден';
  end;
end;
```

5. Создайте процедуру для завершения работы приложения (двойной щелчок мышкой по Button3)

```
procedure TForm1.Button3Click(Sender: TObject);
begin
    close
end;
```

6. Запустите проект на исполнение и проверьте его работу, задавая различные значения ключа поиска.

7.Доработайте данный проект следующим образом:

а) разместите на форме кнопку **Вставить**;

б) создайте процедуру, которая при щелчке по этой кнопке позволит вставить в конец массива число, не найденное в нем.

Поиск данных часто сопровождается необходимостью удаления элемента из массива или вставкой нового элемента в массив. Данные операции можно и реализовать с помощью процедур.

Процедура Del(m: integer) позволяет удалить из массива a[1..n] элемент с номером m.

При выполнении данной процедуры, если удаляется не последний элемент (m<>n), то все элементы, начиная с (m+1)-го смещаются на 1 позицию влево (for j:=m to n-1 do a[j]:=a[j+1];). После чего общее количество элементов массива нужно уменьшить на 1(n:=n-1;). Если удаляется последний элемент, достаточно уменьшить общее количество элементов.

Процедура Ins(k, m: integer) процедура, вставляющая новый элемент k на m-ое место

При добавлении нового элемента общее количество элементов массива нужно увеличить на 1(n:=n+1;). Если элемент добавляется не в конец массива (m<>n), то все элементы, начиная с последнего до m-го смещаются на 1 позицию вправо (for j:=n downto m+1 do a[j]:=a[j-1];). Затем на m-ю позицию вставляется элемент k (a[m]:=k;). В конец массива новый элемент добавляется командой a[m]:=k; без перемещения других элементов.

<u>Пример 3.</u>

Составить в среде PascalABC программу, которая позволит сформировать массив a[1..n] случайным образом и удалить из него элемент с индексом **m**.

```
Program Pr3;
      Var   a:  array[1..100]  of  integer;
            m, i, n:  integer;
                  {процедура,  удаляющая элемент с заданным индексом}
Procedure Del(m:  integer);
     Var   j: integer;
     begin
        if   m<>n   then   for   j:=m  to  n-1 do   a[j]:=a[j+1];
        n:=n-1;
      end;
 BEGIN                        {ОСНОВНАЯ ПРОГРАММА}
      n:=15; randomize;
      for i:=1 to n do begin                {формирование  и вывод элементов массива}
          a[i]:=random(50);   write(a[i]:4);
      end;
      writeln;
      write('Введите номер удаляемого элемента ');   readln(m);
      Del (m);                                      {удаление элемента с индексом m }
      for i:=1 to n do write(a[i]:4);                    { просмотр элементов  массива}
END.
```

<u>Пример 4.</u>

Составить в среде Turbo Pascal программу, которая позволит сформировать массив a[1..n] случайным образом и вставить на **m**-ое место элемент **k** (m и k вводятся с клавиатуры).

```
Program Pr4;
      Var   a:  array[1..100]  of  integer;    k, m, n, i:  integer;
Procedure Ins(k, m:integer);                {процедура, вставляющая  элемент k
                                              на m-ое  место}
Var j:  integer;
   begin
       n:=n+1; if m<>n then   for   j:=n   downto   m+1  do   a[j]:=a[j-1];
             a[m]:=k;
   end;
BEGIN                    {ОСНОВНАЯ ПРОГРАММА}
      write('Введите количество элементов массива'); readln(n);
      randomize;    for   i:=1   to   n  do  begin          {формирование и вывод массива}
                                  a[i]:=random(20)-10;    write(a[i]:4);
                     end;
      writeln;  write('Введите число и его номер');  readln(k, m);
      Ins (k,m);                              {вставка элememета  k  на   m-ое место}
       for   i:=1 to n do  write(a[i]:4);                                {просмотр массива}
END.
```

<u>Пример 5.</u>

Составить в среде Turbo Pascal программу, которая позволит сформировать массив a[1..n], элементы которого выбираются случайным образом из интервала [0,50]. Удалить из него те элементы, которые больше 40.

```
Program Pr5;
      Var   a:  array[1..100]   of   integer;
            m, i, n: integer;
 Procedure Udal(m:  integer); {процедура,  удаляющая элемент с заданным индексом}
      Var j:  integer;
            begin
                If    m<n   then    For   j:=m   to   n-1 do   a[j]:=a[j+1];
                n:=n-1;
            end;
BEGIN                    {ОСНОВНАЯ ПРОГРАММА}
      Write('n=');   Readln(n);
      Randomize;
      For i:=1 to n do   begin                          {формирование и вывод массива}
              a[i]:=random(51);   write(a[i]:4);
      end;
      Writeln;
      i:=1;   While  i<=n  do       {перебор и удаление элементов массива, больших 40}
                     if  a[i]>40  then  Udal(i)  else  i:=i+1;
```

```
    For i:=1 to n do Write(a[i]:4);                    {просмотр массива}
    Readln
END.
```

Контрольные вопросы

1. Опишите алгоритм линейного поиска.
2. Для каких массивов можно применить линейный поиск?
3. Опишите алгоритм бинарного поиска.
4. Для каких массивов можно применить бинарный поиск?

Порядок выполнения работы

1. Изучить теоретические сведения по теме “Поиск данных ”.
2. Ответить на контрольные вопросы.
3. Реализовать в среде Delphi примеры 1, 2 и продемонстрировать работу проекта преподавателю.
4. Получить индивидуальное задание у преподавателя и разработать программу в соответствии с поставленной задачей.
5. Показать работающую программу преподавателю.
6. Подготовить письменный отчет.

Форма представления отчета:

- Тема лабораторной работы
- Условие задачи
- Текст программы с указанием исходных данных и результата.
- Результаты тестирования программы.

Задания для самостоятельной работы

1. Сформировать массив a[1..n], элементы которого выбираются случайным образом из интервала [100, 200]. Определить, содержит ли он заданное число. Если элемент найден, то удалить его из массива.

2. Сформировать массив a[1..n], элементы которого выбираются случайным образом из интервала [100, 200]. Определить, содержит ли он заданное число. Если элемент не найден, то вставить его в конец массива.

3. Сформировать массив a[1..n], элементы которого выбираются случайным образом из интервала [10,50]. Удалить из него максимальный элемент.

4. Сформировать массив a[1..n], элементы которого выбираются случайным образом из интервала [50,90]. Удалить из него минимальный элемент.

5. Сформировать массив a[1..n], упорядоченный по возрастанию. Определить, содержит ли он заданное число. Если элемент не найден, то вставить его в массив на первое место.

6. Сформировать массив a[1..n], упорядоченный по возрастанию. Определить, содержит ли он заданное число. Если элемент найден, то вставить перед ним два новых элемента.

7. Сформировать массив b[1..n], упорядоченный по убыванию. Определить, содержит ли он заданное число. Если элемент не найден, то вставить его в массив с сохранением упорядоченности.

8. Сформировать массив X[1..n], упорядоченный по убыванию. Определить, содержит ли он заданное число. Если элемент найден, то удалить его из массива.

9. Сформировать массив X[1..n], упорядоченный по возрастанию. Определить, содержит ли он заданное число. Если элемент не найден, то вставить его в массив на второе место.

10. Сформировать массив X[1..n], упорядоченный по возрастанию. Определить, содержит ли он заданное число. Если элемент найден, то вставить после него новый элемент.

11. Сформировать массив a[1..n], элементы которого выбираются случайным образом из интервала [10, 90]. Определить, содержит ли он заданное число. Если элемент найден, то вставить перед ним элемент, вдвое больший найденного.

12. Сформировать массив a[1..n], элементы которого выбираются случайным образом из интервала [100, 500]. Определить, содержит ли он заданное число. Если элемент не найден, то вставить его на 3-е место.

13. Сформировать массив X[1..n], упорядоченный по возрастанию. Определить, содержит ли он заданное число. Если элемент найден, то удалить стоящий перед ним элемент.

14. Сформировать массив X[1..n], упорядоченный по возрастанию. Определить, содержит ли он заданное число. Если элемент найден, то вставить после него новый элемент.

15. Сформировать массив X[1..n], упорядоченный по возрастанию. Определить, содержит ли он заданное число. Если элемент не найден, то вставить его в массив на 4-е место.

5. СОРТИРОВКА ДАННЫХ

Краткие теоретические сведения

Существует множество алгоритмов сортировки. Один из самых простых - это сортировка выбором. Алгоритм сортировки выбором находит в исходном массиве максимальный или минимальный элементы, в зависимости от того как необходимо сортировать массив, по возрастанию или по убыванию. Если массив должен быть отсортирован по возрастанию, то из исходного массива необходимо выбирать минимальные элементы. Если же массив необходимо отсортировать по убыванию, то выбирать следует максимальные элементы.

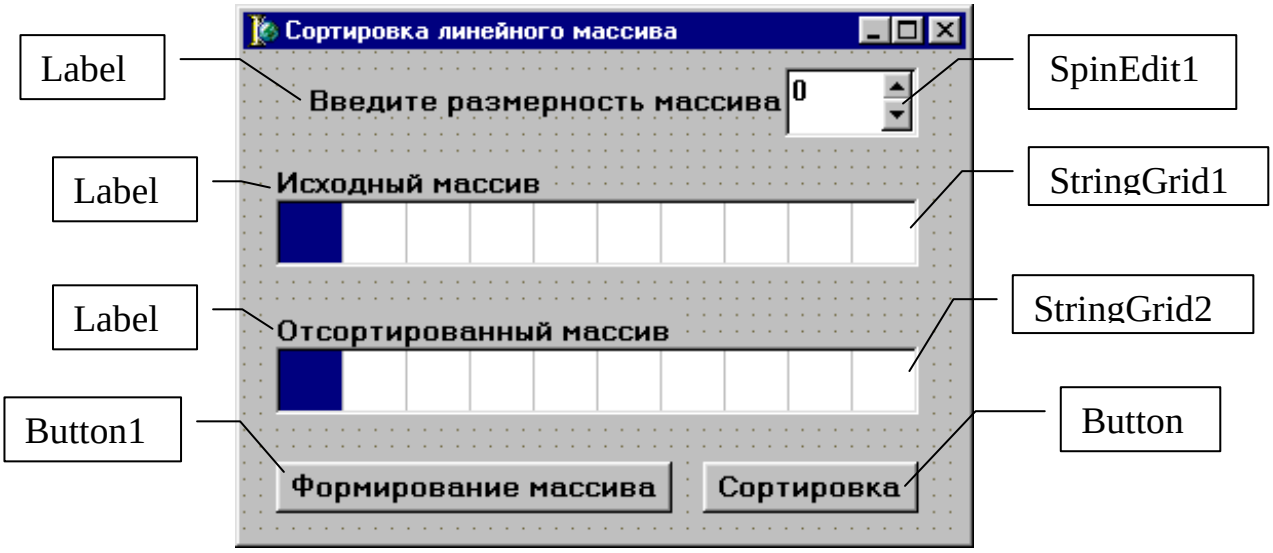
Допустим необходимо отсортировать массив по возрастанию. В исходном массиве находим минимальный элемент, меняем его местами с первым элементом массива. В результате из всех элементов массива один элемент стоит уже на своём месте. Теперь будем рассматривать неотсортированную часть массива, то есть все элементы массива, кроме первого. В неотсортированной части массива опять ищем минимальный элемент. Найденный минимальный элемент меняем местами со вторым элементом массива и т. д. Таким образом, суть алгоритма сортировки выбором сводится к многократному поиску минимального (максимального) элементов в неотсортированной части массива.

<u>Пример 1. Сортировка линейного массива методом простого выбора</u>

Создать в среде DELPHI проект, позволяющий сформировать массив заданной размерности (от 2 до 20 элементов) случайным образом и отсортировать его по возрастанию.

Разработка проекта:

Разместите на форме компоненты StringGrid1, StringGrid2, SpinEdit1, Label1, Label2, Label3, Button1, Button2 следующим образом:



Для компонентов StringGrid1 и StringGrid2 установите значение свойств: FixedCols – 0, FixedRows – 0, ColCount – 10, RowCount – 1, DefaultColWidth – 30, DefaultRowHeight – 30

Для компонента SpinEdit1 установите значение свойств: MinValue – 2, MaxValue - 20

2. В раздел Var добавьте описание следующих переменных.

```
Var
Form1: TForm1;
 a : array[0..19] of integer;
 i, j, t, n : integer;
```

3. Создайте процедуру для формирования массива (двойной щелчок мышкой по Button1)

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   Randomize;
   n:= SpinEdit1.Value;                          {количество элементов массива}
   StringGrid1.ColCount:=n;              {определяем нужное число ячеек в таблице}
   StringGrid2.ColCount:=n;
   For i:= 0 to n-1 do {формируем массив и размещаем его в таблице StringGrid1}
     begin
       a[i]:=Random(100);
       StringGrid1.Cells[i,0]:=IntToStr(a[i]);
   end;
end;
```

4. Создайте процедуру для сортировки массива (двойной щелчок мышкой по Button2)

```
procedure TForm1.Button2Click(Sender: TObject);
begin
                                    {сортировка массива}
  for i:=0 to n-2 do
    for j:=i+1 to n-1 do
       if a[i]>a[j] then begin
                              t:=a[i]; a[i]:=a[j]; a[j]:=t; { обмен элементов}

                     end;
                 {вывод отсортированного массива в таблице StringGrid2}
for i:= 0 to n-1 do
     StringGrid2.Cells[i,0]:=IntToStr(a[i]);
end;
```

5. Запустите проект на исполнение и проверьте его работу, задавая различную размерность массива.

6. Доработайте данный проект следующим образом:

а) разместите на форме компоненты CheckBox1 и CheckBox2. При установке флажка в CheckBox1 должны сортироваться элементы массива с четными индексами. При установке флажка в CheckBox2 должны сортироваться элементы массива с нечетными индексами. При установке обоих флажков должны быть отсортированы все элементы массива.

б) разместите на форме компонент RadioGroup, позволяющий выбрать одно из направлений сортировки: по возрастанию или по убыванию.

Еще один простой алгоритм сортировки – обменом или «пузырьковая».

Принцип действий: обходим массив от начала до конца, попутно меняя местами неотсортированные соседние элементы: 1-й со 2-м, 2-й с 3-м и т.д. В результате первого прохода на последнее место «всплывёт» максимальный элемент. Теперь снова обходим неотсортированную часть массива (от первого элемента до предпоследнего) и меняем по пути неотсортированных соседей. Второй по величине элемент окажется на предпоследнем месте. Продолжая в том же духе, будем обходить всё уменьшающуюся неотсортированную часть массива, перемещая найденные максимумы в конец.

<u>Пример 2. Метод «пузырьковой» сортировки</u>

Составить в среде Turbo Pascal программу, которая позволит сформировать массив a[1..N], элементы которого выбираются случайным образом. Отсортировать его по возрастанию.

```
Program Pr2;
Const N=15;
Var a: array[1..N] of integer;
      i, j, t: integer;        {i – номер просмотра, j – номер первого элемента
                                рассматриваемой пары, t –дополнительная переменная для
                                перестановки элементов массива}
BEGIN
   Randomize;
   Writeln('Исходный массив');
   For i:=1 to N do begin
        a[i]:=random(100);
        write(a[i]:4);
    end;
  Writeln;
  Writeln('Упорядоченный массив');
  For i:=1 to N-1 do                          {цикл по номеру просмотра}
     For j:=1 to N-i do
            If a[j]>a[j+1]  then
               begin
                    t:=a[j];   a[j]:=a[j+1];   a[j+1]:=t; {перестановка элементов}
               end;
  For i:=1 to N do write(a[i]:3);
  Readln
END.
```

<u>Пример 3.</u>

Составить в среде Turbo Pascal программу, которая позволит сформировать массив a[1..20], элементы которого выбираются случайным образом. Отсортировать по неубыванию элементы, занимающие нечетные позиции.

```
Program Pr3;
        Var   a[1..20] of integer;
              i, j, t:    integer;
BEGIN
       for   i:=1 to 20 do begin                    {формирование исходного массива}
             a[i]:=random(100); write(a[i]:4);
      end;
      writeln;
      for   i:=1   to   19  do                                                  {сортировка}
            for   j:=i+1   to   20  do
                  if   (a[i]>a[j]) and (i mod 2=1)  and (j mod 2=1)    then
                                    begin    t:=a[i];   a[i]:=a[j];   a[j]:=t;   end;
           for   i:=1   to  20   do                    {вывод отсортированного массива}
                 write(a[i]:4);
END.
```

<u>Пример 4.</u>

В матрице NxN, элементы которой выбираются случайным образом, отсортировать по неубыванию элементы каждой строки.

```
Program Pr4;
      Const N=5;
      Type mas=array[1..N,1..N] of integer;
      Var   a:  mas;
            i, j, k, m, t: i  nteger;
 BEGIN
      for   i:=1   to  N  do  begin            {формирование и вывод на экран матрицы}
            for   j:=1  to  N  do  begin
                  a[i,j]:=random(50);   write(a[i,j]:3);
            end;
            writeln;
      end;
      writeln;
      for  i:=1  to  N  do                          {сортировка матрицы по строкам}
          for   k:=1  to  N-1  do
             for  m:=k+1  to  N  do
                 if   a[i,k]>a[i,m]   then
                          begin  t:=a[i,k];  a[i,k]:=a[i,m];  a[i,m]:=t;  end;
      for   i:=1  to  n  do  begin        {вывод на экран отсортированной матрицы}
            for   j:=1   to  N  do   write(a[i,j]:3);
            writeln;
      end;
END.
```

Пример 5.

Определить, какое максимальное число можно получить из цифр введенного натурального числа.

Для решения задачи достаточно отсортировать по невозрастанию цифры введенного числа (число вводится как строка цифр)

```
Program Pr5;
      Var   St:  string;
             i, j, n: integer;   c:char;
BEGIN
      write('Введите число  ');   readln(St);
      n:= length(st);                                          {количество цифр}
      for   i:=1   to   n-1  do                                        {сортировка}
            for   j:=i+1   to  n  do
                  if    st[i]<st[j]   then    begin   c:=st[i];st[i]:=st[j]; st[j]:=c;    end;
      writeln(st);                                   {вывод отсортированной строки}
END.
```

Пример 6.

Составить в среде Turbo Pascal программу, которая позволит сформировать массив a[1..10], элементы которого выбираются случайным образом. Отсортировать его методом бинарных вставок.

В программе использованы следующие переменные:
n – размерность массива, **l** – левая граница упорядоченной последовательности, **r** – правая граница, **m** – номер серединного элемента, **x** – элемент, перемещаемый в упорядоченную последовательность.

```
Program Bin_Insert;
Const n=10;
Var    a:array[1..n] of integer;     i, j, l, r, m, x:integer;
BEGIN
   Randomize;                          {инициируем датчик случайных чисел}
   Writeln('Исходный массив');
   For i:=1 to n do  begin                    {формируем и выводим на экран  массив}
        a[i]:=random(100);    write(a[i]:4);
   end;
   Writeln;
                                          {сортировка}
For i:=2 to n do begin
    l:=1; r:=i-1;                          {устанавливаем левую и правую границу}
    x:=a[i];                                    {выбираем перемещаемый элемент}
```

```
  While l<=r do{определяем для него место в упорядоченной последовательности}
      begin
          m:=(l+r) div 2;
          If   x<a[m]  then  r:=m-1  else  l:=m+1;
      end;
        {вставка элемента в упорядоченную последовательность}
    For j:=i-1 downto l do a[j+1]:=a[j];
     a[l]:=x;
end;
  Writeln('Отсортированный массив');
  For i:=1 to n do   write(a[i]:4);
  readln
END.
```

Контрольные вопросы

1. Опишите алгоритм сортировки выбором.
2. Опишите алгоритм сортировки обменом.

Порядок выполнения работы

1. Изучить теоретические сведения по теме “Сортировка данных ”.
2. Ответить на контрольные вопросы.
3. Реализовать в среде Delphi примеры 1, 2 и продемонстрировать работу проекта преподавателю.
4. Получить индивидуальное задание у преподавателя и разработать программу в соответствии с поставленной задачей.
5. Показать работающую программу преподавателю.
6. Подготовить письменный отчет.

Форма представления отчета:

- Тема лабораторной работы
- Условие задачи
- Текст программы с указанием исходных данных и результата.
- Результаты тестирования программы.

Задания для самостоятельной работы

1. Задан массив B[1..20]. Отсортировать все элементы, стоящие на нечетных местах по невозрастанию.
2. Задан массив B[1..30]. Отсортировать все элементы с n-го по k-ый по неубыванию.
3. Задан массив a[1..20]. Отсортировать все элементы стоящие на четных местах по невозрастанию.
4. Задан массив B[1..20]. Отсортировать элементы с 1-го по 10-ый по неубыванию, а с 11-го по 20-й - по невозрастанию.

5. Задана матрица NxN. Отсортировать четные строки по невозрастанию.
6. Задана матрица MxN. Отсортировать нечетные строки по неубыванию.
7. Задана матрица NxN. Отсортировать все столбцы по невозрастанию.
8. Задана матрица NxM. Отсортировать четные столбцы по неубыванию.
9. Протабулировать функцию $y=\sin x^2$ на отрезке [1,3] с шагом h=0.1 и упорядочить значения функции по возрастанию.
10. Протабулировать функцию $y= \cos x + \sin x$ на отрезке [0, 1] с шагом h=0.05 и упорядочить значения функции по убыванию.
11. Протабулировать функцию $y= \cos x^2 + \sin 2x$ на отрезке [0, 4] с шагом h=0.1 и упорядочить значения функции по убыванию.
12. Задан массив X[1..20[, элементы которого выбираются случайным образом из отрезка [-30, 80]. Отсортировать все элементы с 5-го по 15-й по невозрастанию.
13. Задан массив B[1..15], элементы которого выбираются случайным образом из отрезка [0, 70[. Отсортировать все элементы с n-го по k-ый по неубыванию.
14. Задан массив a[1..20], элементы которого выбираются случайным образом из отрезка [-50, 50]. Отсортировать все элементы, стоящие на нечетных местах по невозрастанию.
15. Задан массив B[1..20] , элементы которого выбираются случайным образом из отрезка [30, 100]. Отсортировать элементы с 1-го по 10-ый по невозрастанию, а с 11-го по 20-й - по. неубыванию

6. ПОСТРОЕНИЕ ГРАФИЧЕСКИХ ИЗОБРАЖЕНИЙ

6.1. Графика в PascalABC

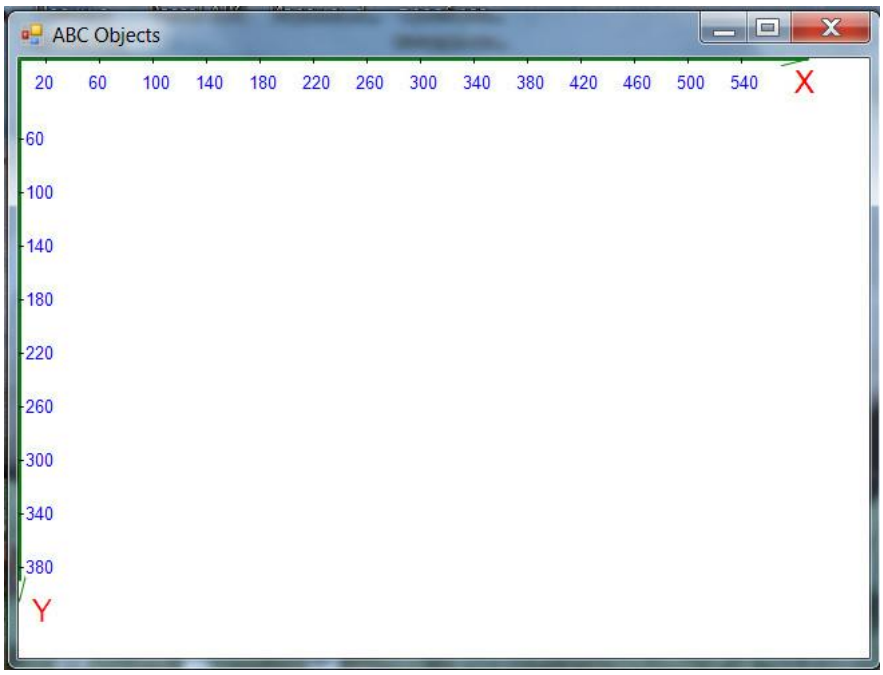
Краткие теоретические сведения

После запуска PascalABC, по умолчанию, запускается текстовый режим. Для работы с графикой служит отдельное графическое окно. Чтобы его открыть, необходимо подключить модуль GraphABC. В этом модуле содержится обширный набор процедур и функций, предназначенных для работы с графическим экраном, а также некоторые встроенные константы и переменные, которые могут быть использованы в программах с графикой. С их помощью можно создавать разнообразные графические изображения и сопровождать их текстовыми надписями. Подключение осуществляется в разделе описаний.

Формат подключения модуля GraphABC:

uses GraphABC;

Начало координат графического окна модуля находится в верхнем левом углу. Ось *x* направлена вправо, а ось *y* - вниз. Координаты исчисляются в пикселях.

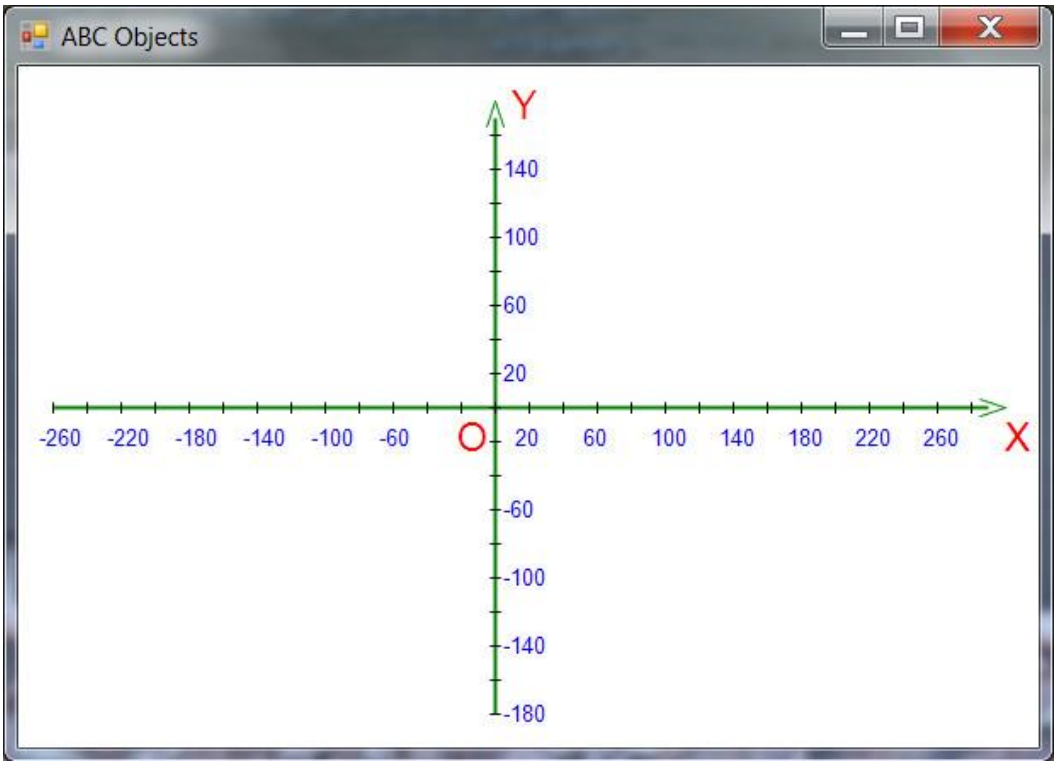


Для перехода к привычной для нас системе координат надо выполнить преобразование системы координат таким образом, чтобы точка (0, 0) находилась посредине окна (в любой нужной точке), а ось OY была направлена вверх.

$$\begin{cases} x_1 = x_0 + x \\ y_1 = y_0 - y \end{cases}$$

Здесь x0 и y0 – координаты точки, в которую мы хотим перенести начало координат новой системы, x и y – координаты точки в старой системе, x1 и y1 – координаты точки в новой системе.

В результате система координат будет иметь следующий вид:



Все команды библиотеки GraphABC являются подпрограммами и описаны в виде процедур и функций. Для того, что бы команда выполнилась необходимо указать команду и задать значения параметрам.

Некоторые команды библиотеки GraphABC

Имя и параметры команды	**Действие команды**
Line(x1,y1,x2,y2)	Рисует отрезок из точки с координатами (x1,y1) в точку с координатами (x2,y2)
LineTo(x, y)	Рисует отрезок из текущей точки в точку с координатами (x, y) (соответствует команде сместиться_в с опущенным пером для исполнителя Чертежник)
Rectangle(x1,y1,x2,y2)	Рисует контур прямоугольника со сторонами параллельными сторонам экрана. Точки с координатами (x1, y1) и (x2, y2) определяют диагональные вершины прямоугольника.
FillRect(x1,y1,x2,y2);	Заливает прямоугольник, заданный координатами противоположных вершин (x1,y1) и (x2,y2), цветом текущей кисти.
Circle(x,y, radius);	Рисует окружность с центром в точке с координатами (x,y) и радиусом radius.
Ellipse(x1,y1,x2,y2);	Рисует эллипс, заданный своим описанным прямоугольником с координатами противоположных вершин `(x1,y1)` и `(x2,y2)`.
TextOut (x, y, text)	Выводит Text на экран. Начало текста в точке с координатами (x, y)
SetPixel(x,y,color)	Закрашивает цветом color точку с координатами (x, y)
SetPenColor(color)	Задает цвет рисования линий. Параметр color – число в промежутке от 0 до 256^3-1.
SetPenWidth(n)	Задает толщину рисования линий. Параметр n – число в промежутке от 1 до 50
SetPenStyle(ps)	Устанавливает стиль пера, задаваемый параметром ps.Стили пера задаются следующими именованными константами: **Значение** **Описание** psSolid Сплошная линия psDash Штриховая линия psDot Пунктирная линия psDashDot Штрихпунктирная линия psDashDotDotЛиния – штрих и два пунктира psClear Отсутствие линии
Arc(x,y,r,a1,a2)	Рисует дугу окружности с центром в точке (x,y) и радиусом r, заключенной между двумя лучами, образующими углы a1 и a2 с осью OX (a1 и a2 – вещественные, задаются в градусах и отсчитываются против часовой стрелки).
FloodFill(x,y,color);	Заливает область одного цвета цветом color, начиная с точки (x,y).
SetBrushColor(color)	Устанавливает цвет кисти, задаваемый параметром `color`
SetBrushStyle(bs);	Устанавливает стиль кисти, задаваемый параметром bs.Стили кисти задаются следующими именованными константами: Константа Константа bsSolid bsClear bsCross dsDiagCross bsHorizontal bsVertical

	bsBDiagonal bsFDiagonal
SetFontColor(color);	Устанавливает цвет шрифта
SetFontSize(sz);	Устанавливает размер шрифта в пунктах.
ClearWindow;	Очищает графическое окно белым цветом
ClearWindow(c);	Очищает графическое окно цветом (c).
SetWindowSize(w,h)	Устанавливает ширину и высоту графического окна.

Пример 1. Программа, рисующая окружность в квадрате

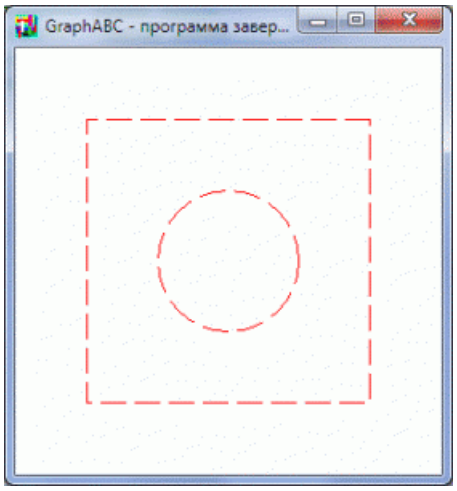
```
program graf_1;

    uses graphABC;
begin

    SetWindowSize(300,300);     //установка размера окна
    SetPenColor(clred);          // установка цвета линий
    SetPenStyle(psdash);        // установка стиля линий
    SetPenWidth(10);            // установка толщины линий
    Rectangle(50,50,250,250);   // рисование прямоугольника
    Circle(150,150,50);          // рисование окружности

end.
```

Результат выполнения программы



Пример 2. Программа рисует окружность относительно графического окна (x0 = 0, y0 = 0), а через 3 секунды эта окружность появляется относительно нарисованной в центре системы координат.

```
program graf_2;

uses   GraphABC;
const
        r = 90; //радиус окружности
```

```
var
        x0, y0, x, y: integer;
Begin
 { Начало координат: }
  x0 := WindowWidth div 2;              //половина ширины окна
  y0 := WindowHeight div 2;             //половина высоты окна
  x := 100; y := 100;                   //координаты центра окружности
  Line(x0 - 250, y0, x0 + 250, y0);     //ось OX
  Line(x0, y0 - 200, x0, y0 + 200);     //ось OY
  Circle(x, y, r);                      //окружность с центром (x, y) и радиусом r
  Circle(x, y, 2);                      //центр окружности (маленькая окружность)
  TextOut(x + 5, y, '(100, 100)');      //надпись возле центра окружности
  sleep(3000);                          //делаем паузу на 3 секунды
  ClearWindow;                          //очищаем графическое окно белым цветом
  Line(x0 - 250, y0, x0 + 250, y0);     //ось OX
  Line(x0, y0 - 200, x0, y0 + 200);     //ось OY
 { Делаем преобразование координат: }
  x := x0 + x;
  y := y0 - y;
 { Рисуем окружность, центр окружности и надпись: }
  Circle(x, y, r);
  Circle(x, y, 2);
  TextOut(x + 5, y, '(100, 100)');
End.
```

Здесь:

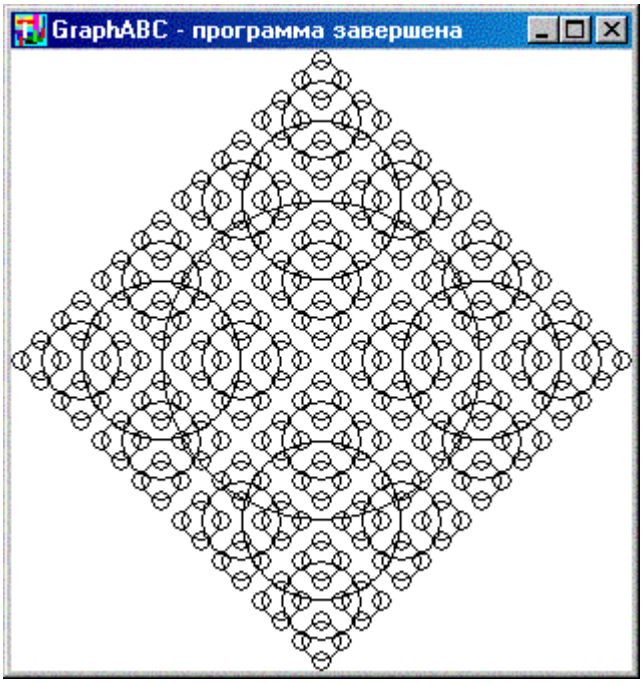
WindowWidth - ширина графического окна;
WindowHeight - высота графического окна (в пикселях);
ClearWindow - эта процедура очищает окно белым цветом;
sleep(ms) - задержка (пауза) на ms миллисекунд.

Пример 3. Рекурсивная прорисовка кругов

```
program graf_3;
uses GraphABC;
var
        r,x,y: integer; // координатные переменные
procedure krug(xx,yy,rr:integer);
begin
      if (rr div 4)>1 then begin
      krug(xx-rr,yy,rr div 2);
      krug(xx+rr,yy,rr div 2);
      krug(xx,yy-rr,rr div 2);
```

```
        krug(xx,yy+rr,rr div 2);
        end;
        Arc(xx,yy,rr,0,360);
end;
        Begin
        SetWindowSize(310,310); // размер окна
        // 1 смайлик
        x:=155;y:=155;
        r:=80;
        krug(x,y,r);
        End.
```

Результат выполнения программы



Пример 4. Пятиконечная звезда

```
program graf_3;

uses GraphABC;
var
i,x,y: integer; // координатные переменные
x0,y0: integer; // альтернативные координаты
x1,y1: integer; // альтернативные координаты
x2,y2: integer; // альтернативные координаты
begin
SetWindowSize(300,300); // размер окна
```

```
// 1 смайлик
x:=150;y:=150;
for i:=1 to 5 do begin
x0:=x+round(57*cos((18+36*(i*2-1))*pi*2/360));
y0:=y-round(57*sin((18+36*(i*2-1))*pi*2/360));
x1:=x+round(150*cos((18+36*i*2)*pi*2/360));
y1:=y-round(150*sin((18+36*i*2)*pi*2/360));
x2:=x+round(57*cos((18+36*(i*2+1))*pi*2/360));
y2:=y-round(57*sin((18+36*(i*2+1))*pi*2/360));
line(x,y,x0,y0);
line(x,y,x1,y1);
line(x,y,x2,y2);
line(x1,y1,x2,y2);
line(x0,y0,x1,y1);
end;
// раскраска
FloodFill(x+3,y+3,clMaroon );
FloodFill(x+3,y-3,clMaroon );
FloodFill(x-3,y+3,clRed);
FloodFill(x-3,y-3,clMaroon );
FloodFill(x+2,y-6,clMaroon );
FloodFill(x-2,y-6,clRed);
FloodFill(x+2,y+6,clRed);
FloodFill(x-2,y+6,clRed);
FloodFill(x+6,y,clRed);
FloodFill(x-6,y,clRed);
end.
```

Результат выполнения программы

Контрольные вопросы

1. Какой модуль необходимо подключить для перехода в графический режим?
2. Где находится центр координат в графическом окне?
3. Как следует преобразовать координаты для перехода к привычной системе координат?
4. Какие процедуры позволяют изобразить основные графические примитивы (отрезок, прямоугольник, окружность, дугу, эллипс)?

Порядок выполнения работы

1. Изучить теоретические сведения по теме “ Графика в PascalABC ”.
2. Ответить на контрольные вопросы.
3. Получить индивидуальное задание у преподавателя и разработать программу в соответствии с поставленной задачей.
4. Показать работающую программу преподавателю.
5. Подготовить письменный отчет.

Форма представления отчета:

- Тема лабораторной работы
- Условие задачи
- Текст программы с указанием исходных данных и результата.
- Результаты тестирования программы.

Задания для самостоятельной работы

1. Построить на экране прямоугольную систему координат, обозначить начало и оси координат. Построить два закрашенных треугольника, симметрич-

ных относительно оси X. Вершины треугольников соединить пунктирными линиями.

2. Построить на экране прямоугольную систему координат, обозначить начало и оси координат. Построить в этой системе координат окружность по заданным координатам центра и радиуса. Окружность закрасить.

3. Изобразить трапецию, выделив пунктиром ее высоту. Обозначить вершины трапеции и вывести на экран формулу для вычисления ее площади.

4. Изобразить параллелепипед, выделив пунктиром его высоту. Обозначить вершины параллелепипеда и вывести на экран все известные формулы для вычисления его площади.

5. Изобразить два вектора с общим началом и вектор, являющийся их суммой. Вывести соответствующие надписи.

6. Изобразить на экране треугольную пирамиду. Провести высоту пирамиды и выделить пунктиром невидимые линии. Вывести на экран формулу для вычисления объема пирамиды.

7. Изобразить на экране пирамиду, в основании которой лежит квадрат. Провести высоту пирамиды и выделить пунктиром невидимые линии. Вывести на экран формулу для вычисления объема пирамиды.

8. Построить прямую треугольную призму. Провести в ней произвольную диагональ и высоту. Обозначить вершины призмы. Вычислить объём пирамиды, если даны стороны треугольника и высота призмы. Вывести результат на экран.

9. Изобразить на экране конус, выделив пунктиром невидимые линии. Провести высоту пирамиды и радиус основания. Вывести на экран формулу для вычисления объема конуса.

10. Нарисовать на экране домик с трубой и круглым окном.

11. Вывести на экран изображение рыбы.

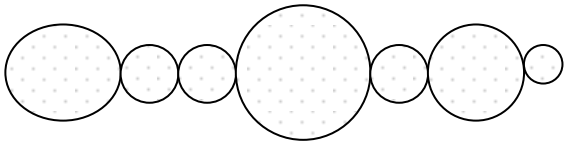
12. Изобразить на экране 5 олимпийских колец.

13. Изобразить на экране стрелковую мишень.

14. Вывести на экран изображение шахматной доски.

15. Изобразить солнце с лучами, длины которых выбираются случайным образом из интервала (20, 80).

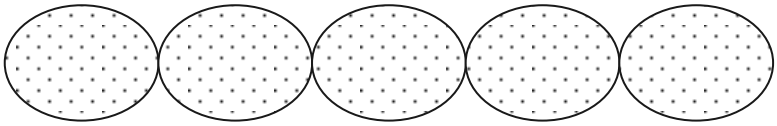
16. Изобразить семейство окружностей. Радиус каждой окружности выбирается случайно из диапазона [10, 40].

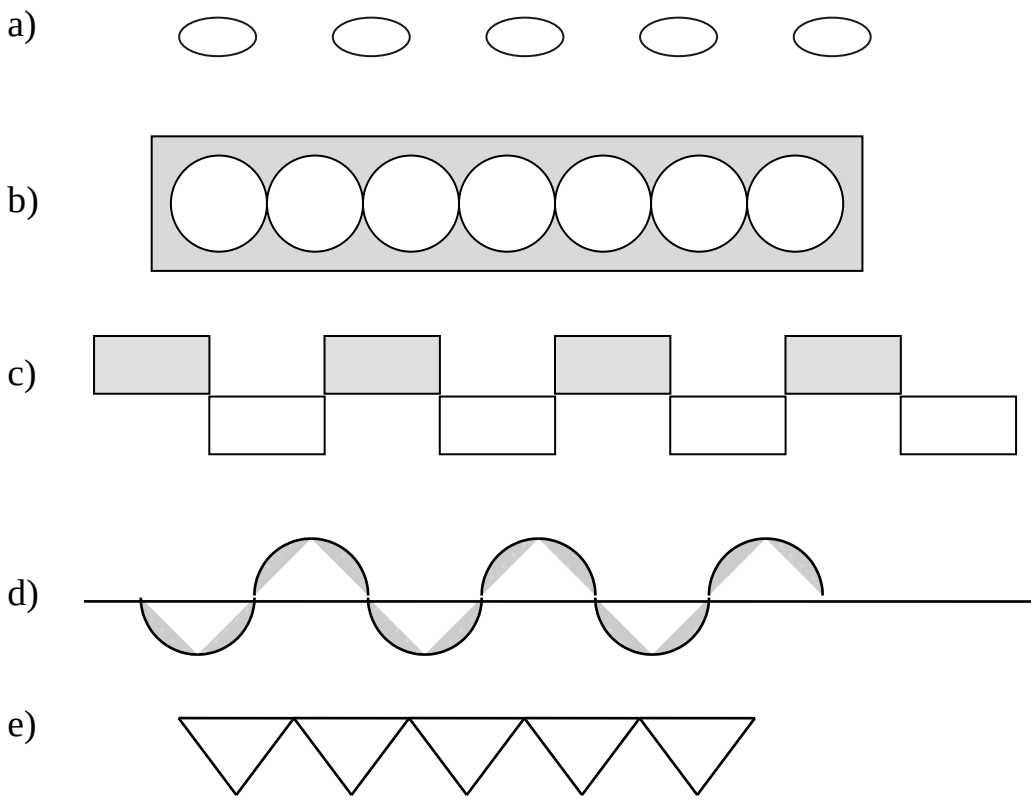


17. Изобразить окружность и вписанный в нее правильный n-угольник. Значение n вводится с клавиатуры.

18. Изобразить циферблат круглых часов с нанесенными делениями и стрелками. Около делений вывести значения от 1 до 12.

19. Изобразить, используя оператор цикла:





6.2. Построение графиков функций в среде LASARUS

Краткие теоретические сведения

Для построения произвольных изображений в LASARUS используется компонент *PaintBox*. PaintBox находится на странице System палитры компонентов. Он представляет из себя простое окно с канвой (холстом), на которой создается изображение. Канва содержится в свойстве *Canvas* компонента.

В этом компоненте имеются следующие свойства для рисования:

Canvas	канва для рисования;
Font	для выбора шрифта на поверхности рисования;
Pen	для установки параметров пера;
Brush	для установки параметров кисти;
Picture	для выбора готового изображения.

А само рисование осуществляется в обработчике события *OnPaint.*

Свойство Canvas доступно при выполнении приложения, поэтому рисунки являются динамическими. Созданный рисунок можно сохранить в графическом файле или вывести на принтер. Рисунки, создаваемые при выполнении приложения, могут быть как неподвижными, так и анимационными, т.е. изменяющими размеры, форму, расположение.

Для рисования на Canvas геометрических фигур используются методы, приведенные в таблице.

Наименование	Рисование	Наименование	Рисование
Arc	дуги	LineTo	Линии
Chord	Хорды эллипса	Polygon	Заполненного прямоугольника
Elipse	эллипса	PolyLine	Незаполненного многоугольника
FillRect	Заполненного прямоугольника	Rectangle	Заполненного прямоугольника
FrameRect	Незаполненного прямоугольника	RoundRect	Заполненного прямоугольника со скругленными углами

<u>Пример 1.</u>

Создать в среде LASARUS проект, позволяющий построить график функции $y = x^2 - 2$, $x \in [-10 ; 10]$.

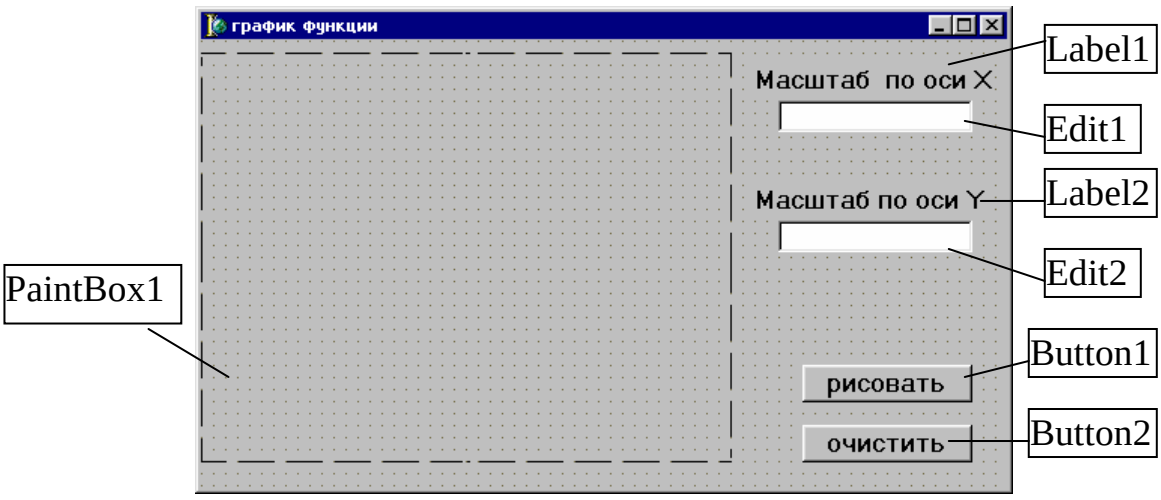
Для перехода от естественных координат к экранным воспользуемся формулами:

$$x1 = Ox + Mx*x.$$
$$y1 = Oy - My*y ,$$

где x , y - естественные координаты, x1 , y1 - экранные координаты, (Ox, Oy) – экранная точка, выбранная в качестве начала координат, Mx , My - масштаб, т.е. длина единичных отрезков в пикселях.

Разработка проекта:

1. Разместить на форме компоненты PaintBox1, Label1, Label2, Edit1, Edit2, Button1, Button2 следующим образом:



2. Создать процедуру для построения графика функции (двойной щелчок мышкой по Button1)

```
procedure TForm1.Button1Click(Sender: TObject);
var
  x1, y1, ox, oy, mx, my, i : integer;
  x, y: real;
begin
   paintbox1.Canvas.Pen.Color:=clblue;                {устанавливаем синий цвет пера}
   ox:=paintbox1.Width div 2;                        {начало координат размещаем в центре
   oy:=paintbox1.Height div 2;                        компонента paintbox1}
   mx:=StrToInt(Edit1.Text);                                          {масштаб по оси x}
   my:=StrToInt(Edit2.Text);                                          {масштаб по оси x}
   paintbox1.Canvas.moveto(0, oy);                                   {строим оси координат}
   paintbox1.Canvas.LineTo(640, oy);
   paintbox1.Canvas.moveto(ox, 0);
  paintbox1.Canvas.LineTo(ox, 480);
   for    i:= -10   to  10 do  begin                                   {разметка оси ox}
         x1:=ox+mx*i;
         paintbox1.Canvas.moveto(trunc(x1), oy-5);
         paintbox1.Canvas.LineTo(trunc(x1), oy+5);
 end;
 for   i:= -10  to  10 do  begin                                     { разметка оси oy }
         y1:=oy-my*i;
         paintbox1.Canvas.moveto(ox-5, trunc(y1));
         paintbox1.Canvas.LineTo(ox+5, trunc(y1));
 end;
  paintbox1.Canvas.Pen.Color:=clred;                {устанавливаем красный цвет пера}
  x:= -10;  y:=sqr(x)-2;              {вычисляем координаты начальной точки графика}
  x1:= trunc(ox+mx*x); y1:= trunc(oy-my*y);          {перевод в экранные координаты}
  paintbox1.Canvas.moveto(x1, y1);                    {перемещаем перо в эту  точку}
   while   x<=10   do                                 { построение графика функции}
```

```
    begin
      y:=sqr(x)-2;  x1:= trunc(ox+mx*x); y1:= trunc(oy-my*y);
      paintbox1.Canvas.LineTo(x1, y1);
      x:=x+0.1;
    end;
end;
```

3. Создать процедуру для очистки изображения графика (двойной щелчок мышкой по Button2)

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  paintbox1.Canvas.Pen.Color:=clblack;
  paintbox1.Canvas.Brush.Color:=clwhite;
  paintbox1.Canvas.Rectangle(0,0,paintbox1.Width,paintbox1.Height);
end;
```

4. Запустить проект на исполнение и проверить его работу, задавая различный масштаб.

5.Доработать данный проект следующим образом:

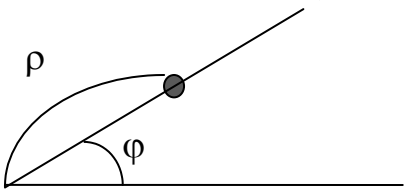
а) разместить на форме компонент RadioGroup , позволяющий выбирать для построения графика одну из функций $y = x^2 - 2$, $y=\sin x$, $y = \sin(e^x)$. $y= e^{\cos x}$;

б) разместить на форме кнопку **Выход,** при щелчке по которой проект завершает работу.

<u>Пример 2.</u>

Создать в среде LASARUS проект, позволяющий построить кривую, заданную уравнением в полярных координатах:

$\rho=\sin(m*\varphi/n)$, m, n -натуральные числа, $0 \le \varphi \le 12\pi$



Каждая точка кривой, заданной уравнением в полярных координатах $\rho=f(\varphi)$, определяется парой чисел φ и ρ, где φ - угол поворота, ρ - радиус

Для перехода от полярных к декартовым координатам воспользуемся формулами:

$x = \rho* \cos \varphi$

$y = \rho* \sin \varphi$, где x, y -естественные координаты

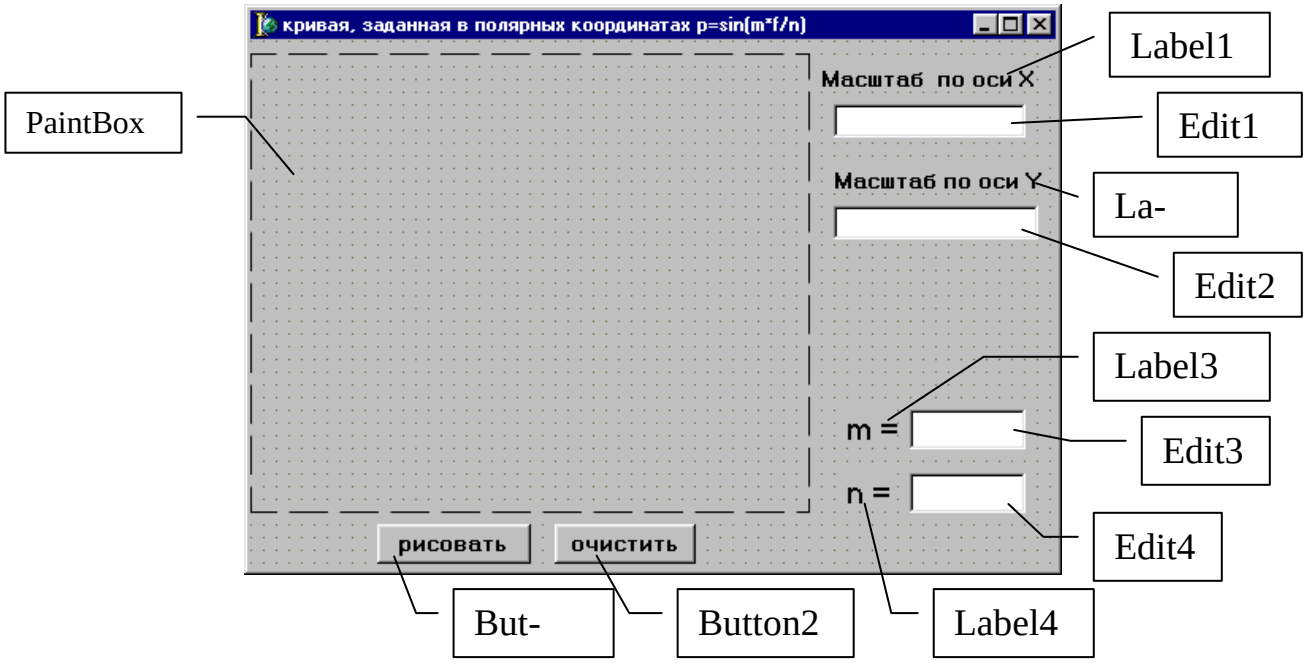
Переходя к экранным координатам, получим соотношения:

$x_э = 0x + Mx*p*\cos \varphi$

$y_э = 0y - My*p*\sin \varphi$

Разработка проекта:

1. Разместить на форме компоненты PaintBox1, Label1, Label2, Label3, Label4, Edit1, Edit2, Edit3, Edit4, Button1, Button2 следующим образом:



2. Создать процедуру для построения графика функции (двойной щелчок мышкой по Button1)

```
procedure TForm1.Button1Click(Sender: TObject);
  var
 n, m, x, y,ox, oy, mx, my, i : integer;
    f, r: real;
begin
   paintbox1.Canvas.Pen.Color:=clblack;        {устанавливаем черный цвет пера}
   ox:=paintbox1.Width div 2;            {начало координат размещаем в центре }
   oy:=paintbox1.Height div 2;                          {  компонента paintbox1}
   mx:=StrToInt(Edit1.Text);                                  {масштаб по оси x}
  my:=StrToInt(Edit2.Text);                                   {масштаб по оси x}
  paintbox1.Canvas.moveto(0, oy);                        {строим оси координат}
  paintbox1.Canvas.LineTo(640, oy);
  paintbox1.Canvas.moveto(ox, 0);
  paintbox1.Canvas.LineTo(ox, 480);
    paintbox1.Canvas.Pen.Color:=clred;        {устанавливаем красный цвет пера}
    m:=StrToInt(Edit3.Text);    {переводим введенные m и n в числовые значения}
    n:=StrToInt(Edit4.Text);
    f:=0;  r:=sin(m*f/n);          { определяем начальные значения  угла и радиуса }
    x:= trunc(ox+mx*r*cos(f));              {переводим в декартовы координаты}
    y:= trunc(oy-my*r*sin(f));
   paintbox1.Canvas.moveto(x, y);                    {перемещаем перо в эту  точку}
```

```
    while f<=12*pi do  begin                    {построение кривой}
            r:=sin(m*f/n);
            x:= trunc(ox+mx*r*cos(f));
            y:= trunc( oy-my*r*sin(f));
            paintbox1.Canvas.LineTo(x, y);
            f:=f+0.01;
        end;
    end;
```

3. Создать процедуру для очистки изображения графика (двойной щелчок мышкой по Button2)

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  paintbox1.Canvas.Pen.Color:=clblack;
  paintbox1.Canvas.Brush.Color:=clwhite;
  paintbox1.Canvas.Rectangle(0,0,paintbox1.Width,paintbox1.Height);
end;
```

4. Запустить проект на исполнение и проверить его работу, задавая различный масштаб и меняя значения m и n.

5. Доработать данный проект следующим образом:

а) разместить на форме компонент RadioGroup, позволяющий выбирать цвет для построения кривой: красный, синий или зеленый;

б) разместить на форме кнопку **Выход,** при щелчке по которой проект завершает работу.

Контрольные вопросы

1. Назначение компонента Paintbox .
2. Основные свойства компонента Paintbox .
3. Формулы перехода от естественных координат к экранным.
4. Формулы перехода от полярных к декартовым координатам.

Порядок выполнения работы

1. Изучить теоретические сведения по теме “ Построение графиков функций в среде LASARUS ”.
2. Ответить на контрольные вопросы.
3. Реализовать в среде LASARUS примеры 1, 2 и продемонстрировать работу проекта преподавателю.

ЛИТЕРАТУРА

1. Бабушкина И.А. Практикум по объектно-ориентированному программированию [Электронный ресурс]/ Бабушкина И.А., Окулов С.М.— Электрон. текстовые данные.— М.: БИНОМ. Лаборатория знаний, 2012.— 366 с.— Режим доступа: http://www.iprbookshop.ru/12254.— ЭБС «IPRbooks»
2. Большаков В.А. Информатика [Электронный ресурс]: лабораторный практикум по программированию на Турбо-Паскале/ Большаков В.А., Воронов Г.И., Савватеева Л.А.— Электрон. текстовые данные.— СПб.: Российский государственный гидрометеорологический университет, 2002.— 190 с.— Режим доступа: http://www.iprbookshop.ru/14906.— ЭБС «IPRbooks»
3. Давыдова Н.А. Программирование [Электронный ресурс]: учебное пособие/ Давыдова Н.А., Боровская Е.В.— Электрон. текстовые данные.— М.: БИНОМ. Лаборатория знаний, 2012.— 238 с.— Режим доступа: http://www.iprbookshop.ru/6485.— ЭБС «IPRbooks»
4. Златопольский Д.М. Программирование. Типовые задачи, алгоритмы, методы [Электронный ресурс]/ Златопольский Д.М.— Электрон. текстовые данные.— М.: БИНОМ. Лаборатория знаний, 2012.— 223 с.— Режим доступа: http://www.iprbookshop.ru/12264.— ЭБС «IPRbooks»
5. Соколова Ю.С. Разработка приложений в среде Delphi. Общие приемы программирования. Часть 1 [Электронный ресурс]: учебное пособие/ Соколова Ю.С., Жулева С— Электрон. текстовые данные.— М.: Горячая линия - Телеком, 2013.— 142 с.— Режим доступа: http://www.iprbookshop.ru/12035.— ЭБС «IPRbooks»