Computer Science. Module 2.
Laboratory works
Table of contents

2.1 Loop statements

Loops are used to repeat a ***statement*** a certain number of times or while a condition is true.

2.1.1 FOR loop

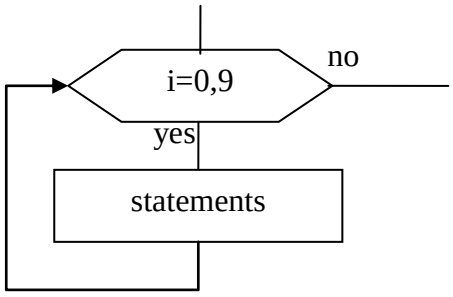
For is a C++ constructions that allow you to repeat a statement, or set of statements, for a known number of times or until some condition is true. The syntax for the **for loop** is as follows:

for (initialization; condition of continuation; modification of parameters)
{statementBlock;}

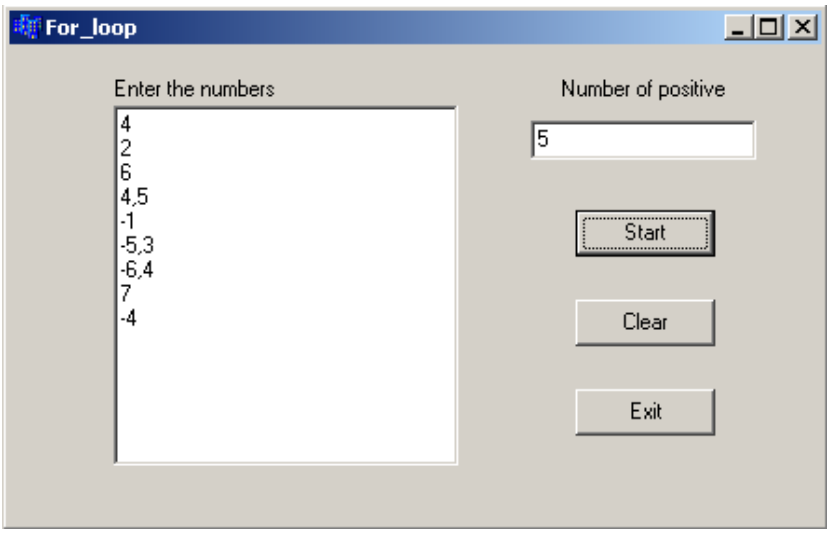
where

- **initialization** declares and initializes the loop control variable, for example int i = 0
- **condition** of continuation is a test that will stop the loop as soon as it is false. Or, in other words, the repetition will continue as long as the condition is true.
- **Modification of parameters** is a statement that modifies the loop control variable appropriately, for example: i = i+1
- **statementBlock** is either a single statement or a group of statements inside the braces { ... }

The scheme of for loop:

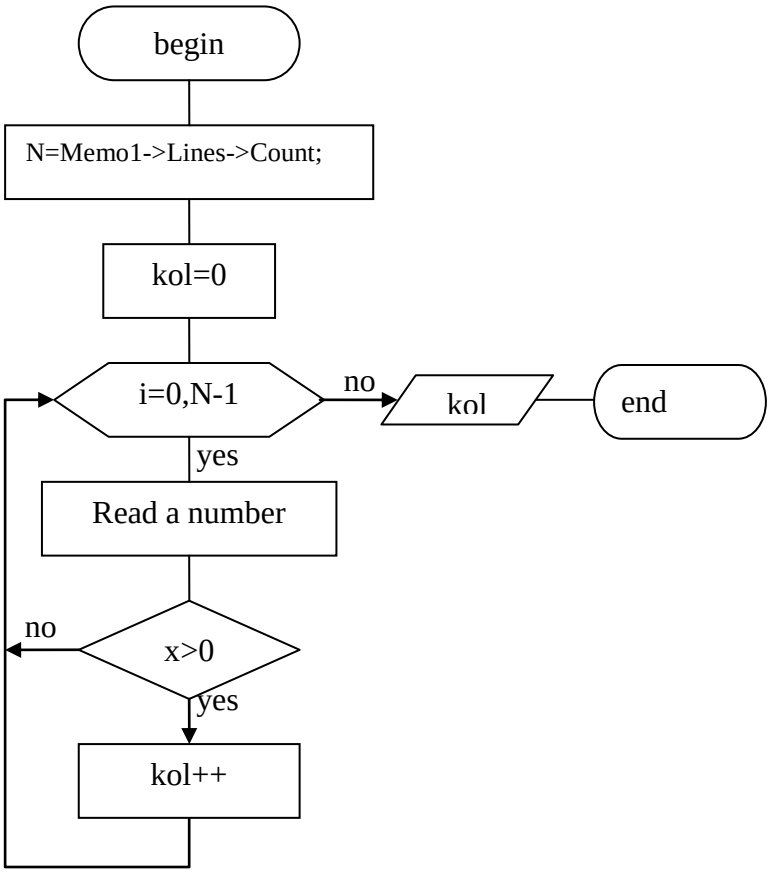


Example 1.1 Write a program that counts the number of positive doubles in Memo.




```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
int N=Memo1->Lines->Count;        //number of lines in Memo
double x;
int kol=0;                        //at first kol must be 0
for (int i=0; i<N; i++)
  { x=StrToFloat(Memo1->Lines->Strings[i]);       //read a number from Line with number i
    if (x>0) kol++;               //if the number is positive, kol increases by 1
  }
Edit1->Text=IntToStr(kol);
}
```

The scheme is:



Each repetition of the loop is called *iteration*.

2.1.2 The WHILE loop (loop with precondition)

While loop is used when we do not know, how many times we need to repeat statements, but we do know the **condition** of repeating.

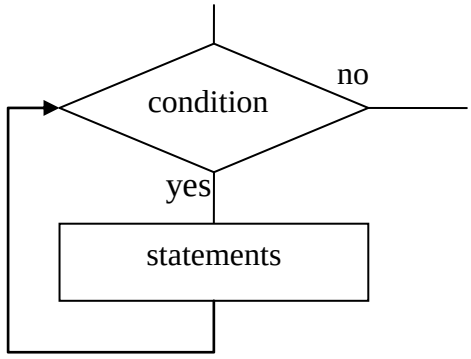
Its format is:

while (Condition)

{statements;}

and its function is simply to repeat ***statements*** while ***Condition*** is true.

The compiler first examines the Condition. If the Condition is true, then it executes the Statements. After executing the Statements, the Condition is checked again. AS LONG AS the Condition is true, it will keep executing the Statements. When or once the Condition becomes false, it exits the loop.
The scheme of while is:



2.1.3 The DO-WHILE loop (loop with postcondition)

The do-while loop is used when it is conveniently to check up the condition **after** statements.

Format:

do{

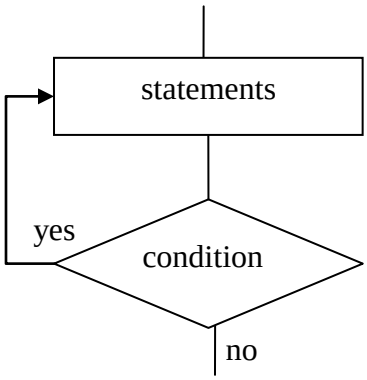
statement;

} ***while*** (condition);

The **do-while** statement executes a *Statement* first. After the first execution of the *Statement*, it examines the *Condition*. If the *Condition* is true, then it executes the *Statement* again. It will keep executing the *Statement* AS LONG AS the *Condition* is true. Once the *Condition* becomes false, the looping (the execution of the *Statement*) would stop.

The **do…while** loop statement can be used to insist on getting a specific value from the user.

The scheme of do-while is:



Example 1.2 Write a program that will read numbers from Memo, until number zero is given. Find the smallest positive number.

The difficulty is to find the first positive number and assign it to min. We use two loops: first to except negative numbers on the beginning of Memo and second – to search minimum

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int min, x, i=0;
  int n=Memo1->Lines->Count;
  do{
    x=StrToInt(Memo1->Lines->Strings[i]);
    i++;
  }while(x <= 0 && i<n);
  min = x;                  // we assume first given number is the smallest
  while (x != 0 && i<n) {
     x=StrToInt(Memo1->Lines->Strings[i]);
     if(x > 0 && x < min )
        min = x;
     i++;
  }

  Edit1->Text =IntToStr(min);
}
```

Alternative approach

We read numbers from Memo. If the number is negative, we ignore it. If the number is positive and min is 0, it means that this positive number is the first and we must assign it to min without any comparison. If the number is positive and min is not 0, it means that this positive number is not the first and min has some value (temporary minimum). In this case we compare the number with min and assign the number to min if it is smaller than min.

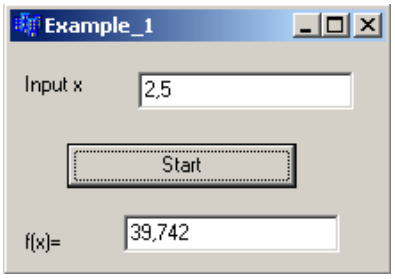
```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  int min=0, x,i=0;
  int n=Memo1->Lines->Count;
  do{
    x=StrToInt(Memo1->Lines->Strings[i]);
    if(x > 0)
       if(min==0 || x < min )
```

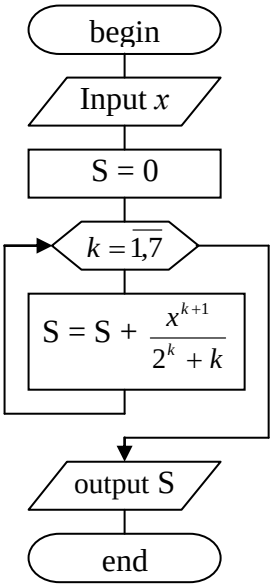
```
        min = x;
    i++;
  }while(x != 0 && i<n);

  Edit1->Text =IntToStr(min);

}
```

Laboratory work 2.1
The organization of cyclic calculations by means of the FOR loop operator

The purpose: To study possible variants of the cyclic calculations organization in the program by means of the loop with parameter operator. To get a skill in calculation of sums with finite number of addends and function tabulation.

Example 1.3 Calculate the value of $f(x)=\sum_{k=1}^{7}\frac{x^{k+1}}{2^k+k}$, enter from the screen the value of *x*.





The number of addends in this example is 7 and we have to repeat the loop 7 times. The addends are $\frac{x^{k+1}}{2^k+k}$, where k is a order number of addend.
We may use k as a loop counter (usually it is variable ***i***).
Let x=2. Instruction

```
for (int k=1; k<=7; k++)
        s+=pow(x,k+1)/(pow(2,k)+k);
```

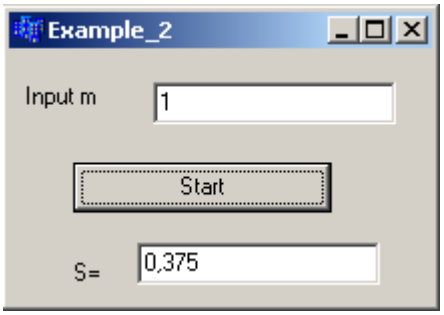
means that:

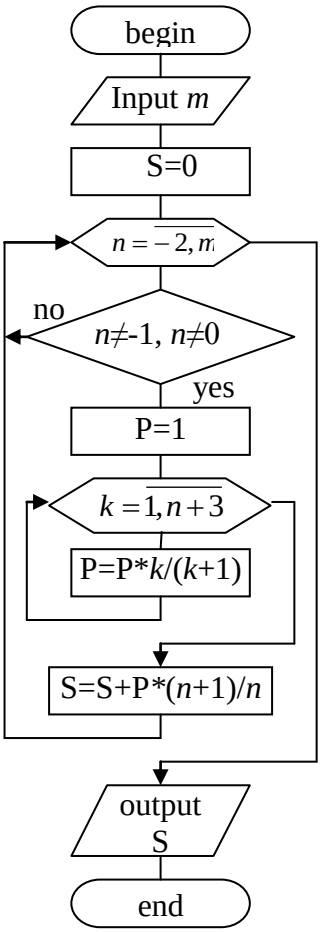
1. At first k=1. We put this value in the addend's formula ($\frac{x^{1+1}}{2^1+1}=\frac{2^2}{3}=\frac{4}{3}=1.33$) and add it to sum (s becomes 1.33). After this k increases by 1 and becomes 2.
2. We check up the condition k<=7. It is true, therefore we repeat again. Put the value of k=2 in the addend's formula $\frac{x^{2+1}}{2^2+2}=\frac{2^3}{6}=\frac{8}{6}=1.33$ and add it to sum: s=1.33+1.6=2.93. After this k increases by 1 and becomes 3.
3. Again check up the condition k<=7. It is true. We put the value of k=3 in the addend's formula $\frac{x^{3+1}}{2^3+3}=\frac{2^4}{11}=\frac{16}{11}=1.45$ and add it to sum: s=2.66+1.45=4.11. After this k increases by 1 and becomes 4.
4. Repeat the same for values of k: 4, 5, 6, 7. After this k becomes 8. The condition k<=7 is false and the loop stops.
5. Now we have only to output the result.

The program is:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{ float x=StrToFloat(Edit1->Text);
  float s=0;
  for (int k=1; k<=7; k++)
        s+=pow(x,k+1)/(pow(2,k)+k);  //s=s+pow(x,k+1)/(pow(2,k)+k);
  Edit2->Text = FormatFloat("0.000",s);
}
```

Example 1.4 Calculate the value of $S=\sum_{n=-2}^{m}\frac{n+1}{n}\prod_{k=1}^{n+3}\frac{k}{k+1}$, enter from the screen the value of *m*.





This example is similar to the previous one, but little more complex.

The number of addends is m+3. The addend is $\frac{n+1}{n}\prod_{k=1}^{n+3}\frac{k}{k+1}$.

Therefore, on each step we have to add $\frac{n+1}{n}\prod_{k=1}^{n+3}\frac{k}{k+1}$ to the sum. To do this, we have to calculate the product $\prod_{k=1}^{n+3}\frac{k}{k+1}$. It consists of n+3 multipliers: $\frac{k}{k+1}$.

To calculate each addend, we write the for loop:

```
for (k=1; k<=n+3; k++)
    if  (k!=-1 && k!=0) p*=(float) k/(k+1);     //p=p*k/(k+1.);
```

When k=-1, there is zero in denominator. When k=0, the multiplier equals 0 (and all product also equals 0). To exclude these values we check up the condition:

```
if  (k!=-1 && k!=0).
```

To calculate the sum, we must write another for loop:

```
for (n=-2; n<=m; n++)        //loop for calculation sum
  if (n!=-1 && n!=0)
    { p=1;                   // first value of product is 1
      for (k=1; k<=n+3; k++)        //loop for calculation the product
        if  (k!=-1 && k!=0) p*=(float) k/(k+1);
      S+=(float)(n+1)*p/n;  //adding the addend to sum
    }
```

The condition if (n!=-1 && n!=0) is used to exclude division by 0 and zero addend.

Pay your attention that loop with parameter ***k*** is inside the loop with parameter ***n*** (parameters are different!). When one loop is inside another, their parameters must be different.

The code of the program is:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{ int n, k, m=StrToInt(Edit1->Text);
  float S=0, p;
  for (n=-2; n<=m; n++)
     if (n!=-1 && n!=0)
       { p=1;
         for (k=1; k<=n+3; k++)
            if  (k!=-1 && k!=0) p*=(float) k/(k+1);
         S+=(float)(n+1)*p/n;
       }
 Edit2->Text=FloatToStrF(S,ffGeneral,4,3);  }
```

Function **FloatToStrF**(S,ffGeneral,4,3) converts the float number S to the string. Parameter ffGeneral is the General number format. The value is converted to the shortest possible decimal string using fixed or scientific format. Trailing zeros are removed from the resulting string, and a decimal point appears only if necessary. The resulting string uses fixed point format if the number of digits to the left of the decimal point in the value is less than or equal to the specified precision (4 in our program), and if the value is greater than or equal to 0.00001. Otherwise the resulting string uses scientific format, and the last parameter specifies the minimum number of digits in the exponent (between 0 and 4).

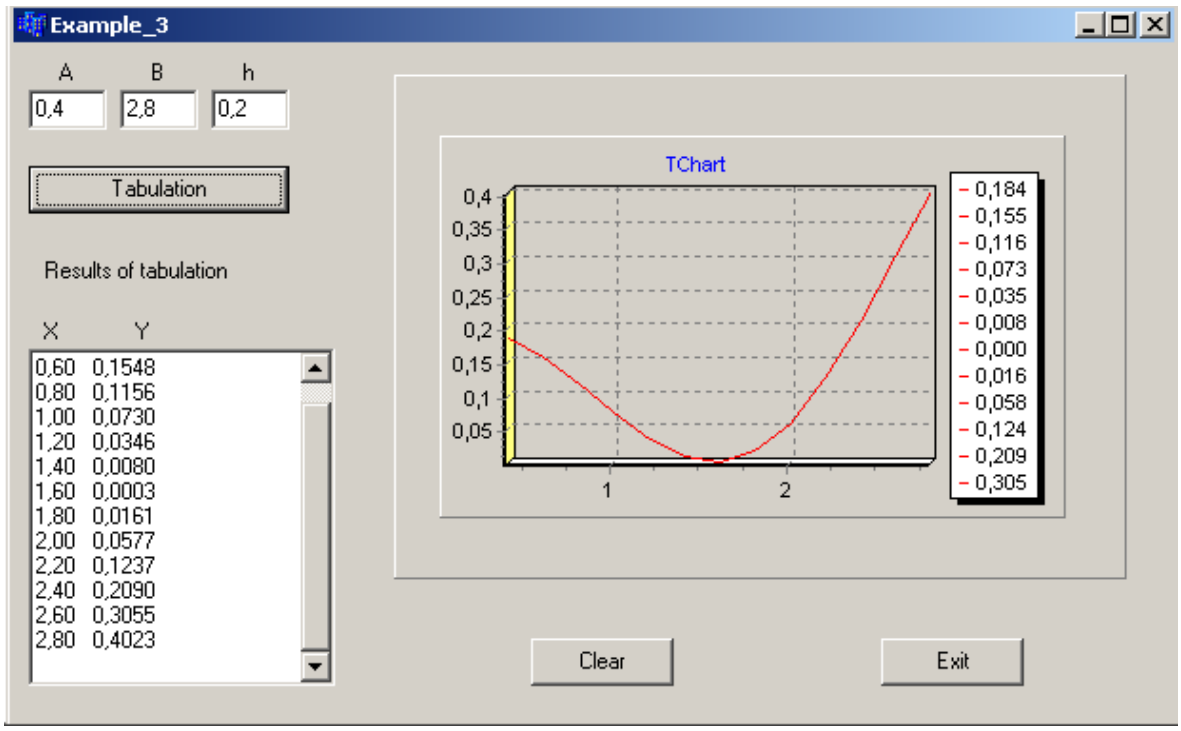
Example 1.5 Make a scheme of the algorithm and a program of function tabulation. Create a plot of the function.

$$f(x)= \sum_{k=1}^{10} \frac{x^{k-1} \cdot \cos^2 x}{5^k}$$

The value of x changes from A= 0,4 to B=2,8 with step h=0,2.

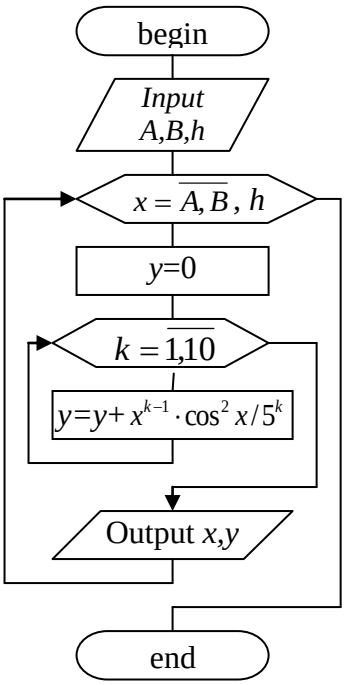
There are 13 values of x between 0.4 and 2.8 if step equals 0.2 ((2.8-0.4)/0.2+1). They are: 0.4, 0.6, 0.8, 1, 1.2, 1.4 and etc.

To tabulate the function means to calculate the value of function for each of these values of x. In this example we will use the for loop too. The loop variable is x. Its first value is 0.4. The condition is x<=2.8, changing of parameter is: x+=0.2. On each iteration of the loop we will calculate the value of function.



To place a graph:
Place a Panel on the form.
Place a Chart from the Additional Component Palette.
Do right-click on the Chart and select Edit Chart and click Add and choose the type of chart.

```
#include <math.h>

void __fastcall TForm1::Button1Click(TObject *Sender)
{ float A,B,h,x,y;
  A=StrToFloat(Edit1->Text);  B=StrToFloat(Edit2->Text);
  h=StrToFloat(Edit3->Text);
  for(x=A; x<=B +0.1*h; x+=h)
   {   //calculate y – the value of function
       y=0;
       for (int k=1; k<=10; k++) y+=pow(cos(x),2)*pow(x,k-
1)/pow(5,k);
       //add values of x and y to Memo1
       Memo1->Lines->Add(FormatFloat("0.00",x)+"   "+
                                          FormatFloat("0.0000",y));
       //add values of x and y to the chart
       Series1->AddXY(x,y,"",clRed);    }
}
void __fastcall TForm1::Button2Click(TObject *Sender)
{ Edit1->Clear();  Edit2->Clear();  Edit3->Clear();
  Memo1->Clear(); Series1->Clear();
}
void __fastcall TForm1::Button3Click(TObject *Sender)
{   Close();  }
```



Individual task

1. Make a scheme of algorithm and a program of sum (product) calculation. Choose variants from the tables 2.1-2.2 accordingly to your log number.
2. Make a scheme of the algorithm and a program of function tabulation. Variable *x* changes from 0.4 to 2.8 with step 0.2. Choose the function *f*(*x*) from the table 2.1 accordingly to your log number (ignore the value of variable x in the table).

Table 2.1 – Individual tasks of the low level of complexity

1	$\sum_{k=1}^{7}\frac{2^k\sin(x+k)}{(x+1)^k};x=0{,}52$	2	$\sum_{k=1}^{12}\frac{Cos(kx)}{k};x=\frac{3\pi}{5}$	3	$\sum_{k=1}^{8}\sin(x)(x+\cos(x+2));x=1{,}1$
4	$\sum_{k=1}^{9}\frac{x^{k+1}}{(k+1)^x};x=1{,}79$	5	$\sum_{k=1}^{8}\frac{\sin x^k}{4k};x=1{,}75$	6	$\sum_{k=2}^{10}\frac{\operatorname{arctg}^3(2kx)}{1.2\ln(k+x)};x=1{,}39$
7	$\sum_{k=1}^{12}\frac{Sin(kx)+k}{\sqrt[k]{x+0.1}+6k};x=0{,}94$	8	$\sum_{k=1}^{7}\frac{\ln^k(3x)}{(2+x)^k};x=1{,}35$	9	$\sum_{k=1}^{11}\frac{\sin(x)^k+0.3}{(2^k)};x=0{,}81$
10	$\sum_{k=1}^{9}\frac{Ln(x+1)}{(x+k)^k};x=3{,}84$	11	$\sum_{k=1}^{8}\sqrt[k]{\ln(x+1)};x=1{,}21$	12	$\sum_{k=1}^{6}\frac{k^2\sin^2(x/k)-kx^2}{e^{kx}};x=0{,}8$
13	$\sum_{k=1}^{9}\frac{Sin(2kx)+0.2}{2k+5};x=2{,}73$	14	$\sum_{k=6}^{1}\frac{x^k}{k^3+x^{k+2}};x=0{,}92$	15	$\sum_{k=1}^{12}\frac{Cos(x^k)}{(x+5)^k+k};x=1{,}51$
16	$\sum_{k=1}^{7}\frac{kx\cos(x+k)}{\ln(2+x)+2k};x=1{,}85$	17	$\sum_{k=1}^{11}\frac{\sin(x^k-1)}{4k^2+1};x=5{,}2$	18	$\sum_{k=2}^{9}\frac{\sin(x+1)+1.5}{\lg(5kx)+2.1};x=0{,}37$
19	$\sum_{k=2}^{6}\frac{\sin(0.17x^k)}{2k+x};x=1{,}34$	20	$\sum_{k=1}^{8}\frac{\ln x^{2k-1}}{2^k(2k-1)};x=0{,}4$	21	$\sum_{k=1}^{7}\frac{2(x+1)^{3-k}}{(k+1)^x+k^3};x=1{,}22$
22	$\sum_{k=1}^{8}\frac{5\ln(2kx)}{arctg(2x)+k^2};x=0{,}25$	23	$\sum_{k=2}^{9}\frac{\operatorname{tg}(e^x)}{3k^2+1};x=5{,}25$	24	$\sum_{k=1}^{10}\cos\left(k^3-\frac{kx}{5}\right);x=2{,}73$
25	$\sum_{k=2}^{9}\frac{tg(x)-x^2/k}{(k^2-1)};x=1{,}92$	26	$\sum_{k=3}^{10}\frac{x^{k-1}\cos x}{(12^k-1)};x=1{,}84$	27	$\sum_{k=1}^{7}\frac{x\sin(x-k)}{e^{2+x}+k};x=2{,}12$
28	$\sum_{k=1}^{7}\frac{\sin(x^k-\pi)}{\ln k^2+0.3};x=0{,}77$	29	$\sum_{k=3}^{11}\frac{\cos^{2+k}x}{2k-1};x=2{,}342$	30	$\sum_{k=2}^{6}x\cdot\operatorname{arctg}\frac{x-4.4k}{x+\sin(x+k/5)};x=2{,}2$

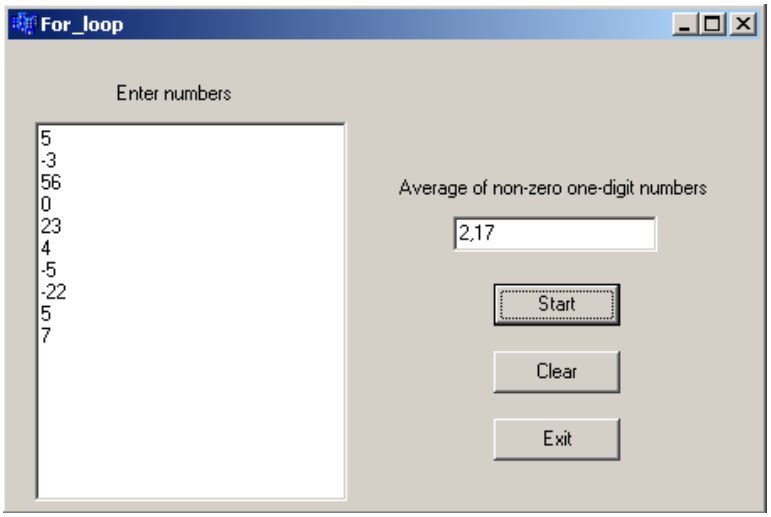
Table 2.2 – Individual task of the medium level of complexity

1	$S=\sum_{k=1}^{n}\frac{(k+1)}{(k-5)}\prod_{m=1}^{k+7}\frac{m^2-9}{m-2}$	2	$P=\prod_{j=1}^{k}\frac{(j-6)j}{j-3}\sum_{i=j}^{12}\frac{\sqrt[3]{i+5}}{i-11}$	3	$R=\sum_{i=1}^{k}\frac{(1-i)^i}{(i+3)}\prod_{n=i}^{2k}\frac{n-i}{n+2}$
4	$Z=\prod_{j=-4}^{k}\frac{(j+2)j}{j-3}\sum_{i=j}^{k+5}\left(\frac{\sqrt[5]{i+5}}{i-11}+5i\right)$	5	$A=\prod_{j=1}^{k}\frac{(j-4)j}{j-3}\sum_{i=j}^{12}\frac{\sqrt[3]{i+5}}{i-1}$	6	$Q=\sum_{k=1}^{n}\frac{(-1)^{k+1}(k+7)}{k!}$
7	$S=\sum_{k=3}^{n}\frac{(-2)^{k-1}}{(k-5)}\prod_{i=1}^{k+7}\frac{i^3-27}{i-7}$	8	$P=\prod_{j=2}^{k}\frac{(j-6)j}{(j-3)(j-1)!}$	9	$W=\sum_{i=1}^{k}\frac{(-1)^i(i-3)^2}{i!}$
10	$Z=\prod_{j=-3}^{k}\frac{(j+2)j}{j-3}\sum_{i=j}^{k+5}\left(\frac{i+5}{i-11}-3{,}5i\right)$	11	$Q=\sum_{k=1}^{n}\frac{(-1)^k(k+3)^2}{k!}$	12	$A=\prod_{j=1}^{k}\frac{(j^2-4)j}{j-k+1}\sum_{i=j}^{9}\frac{i-3}{i-7}$
13	$W=\sum_{i=-2}^{k}\frac{(-1)^i(i+3)!}{i^2-4}$	14	$U=\prod_{t=2}^{k}\frac{\cos(t)}{t-3}\sum_{i=1}^{t}\left(\frac{i-2}{i-7}\right)$	15	$P=\prod_{j=1}^{k}\frac{(j-6)j}{j-3}\sum_{i=j}^{12}\frac{\sqrt[3]{i+5}}{i-11}$
16	$Y=\sum_{n=1}^{k}\frac{(-1)^{2-n}(n^2-9)^2}{(n+1)!}$	17	$S=\sum_{k=1}^{n}\frac{(-3)^{3k+1}}{(k-2)^{3k+1}}\prod_{m=1}^{k+n}\frac{m^3-8}{m-3}$	18	$Z=\prod_{n=-2}^{k}\frac{(n+1)\lvert n-9\rvert}{(n+3)!}$

19	$W=\sum_{i=1}^{k}\frac{(-1)^i}{(i-3)^2}\prod_{n=i}^{2k}\frac{n^3-8}{n+2}$	20	$Z=\prod_{j=-4}^{k}\frac{(j+2)j}{j-3}\sum_{i=j}^{k+5}\left(\frac{\sqrt[5]{i+5}}{i-11}+5i\right)$	21	$p=\prod_{i=1}^{n}\frac{(3i-1)(i-3)}{(2i-1)!}$
22	$L=\prod_{j=1}^{k}\frac{(j-5)}{j-3}\sum_{i=k}^{12}\frac{\sqrt{\lvert i+5\rvert}}{i-1}$	23	$Y=\sum_{i=1}^{k}\frac{(i-1)^i}{(3-i)^2}\prod_{n=i}^{2+k}\frac{n+0.8}{n+i}$	24	$W=\sum_{i=1}^{k}\frac{(-1)^i}{(i-3)^2}\prod_{n=i}^{2k}\frac{n^3-8}{n+2}$
25	$Q=\sum_{k=1}^{n}\frac{(n-1)^{k+1}(k+3)}{(k+1)!}$	26	$G=\prod_{j=-3}^{k}\frac{(2j-1)}{4j-3}\sum_{i=j}^{k+5}\left(\frac{i+5}{1-k+j}\right)$	27	$Y=\sum_{i=-1}^{k}\frac{(k-i)^i(i+2)!}{i^2-4}$
28	$Z=\prod_{t=0}^{k}\frac{t+k}{\cos(t)-3}\sum_{i=1}^{t}\left(\frac{3i-2}{i+7}\right)$	29	$D=\sum_{i=-2}^{k}\frac{\left(-2^i\right)\sin^2(i+3)}{(i+3)!}$	30	$F=\sum_{n=0}^{k}\frac{\left(n+2^k\right)\lvert n-4\rvert}{(n)!}$

Laboratory work 2.2
The organization of cyclic calculations by means of the FOR loop operator.

Example 1.6 Enter float numbers in Memo. Calculate the average of non-zero one-digit numbers.

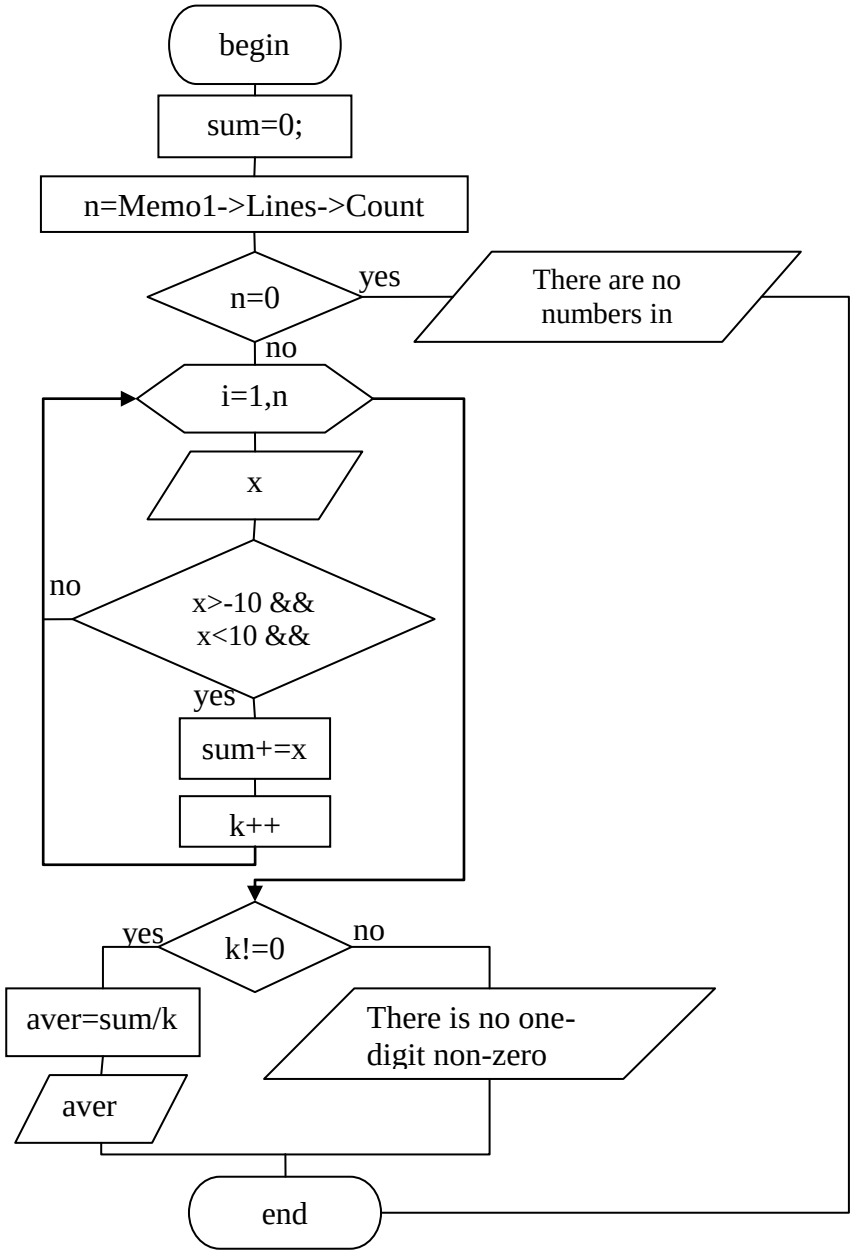


To calculate the average of non-zero one-digit numbers we have to calculate the sum and the quantity of these numbers and then divide.

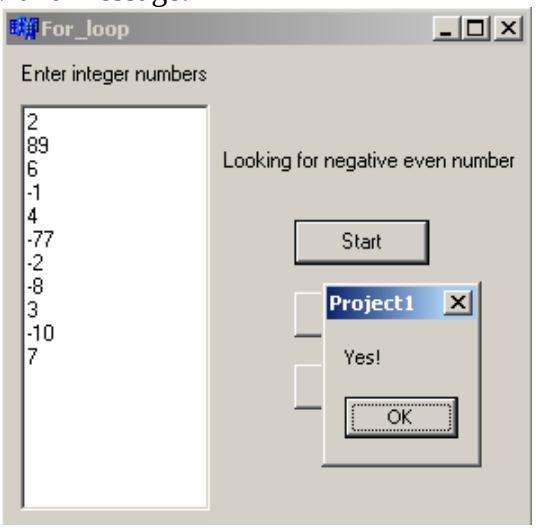
Variable x is for numbers from Memo1, sum is for sum, k is for quantity, aver is for average.

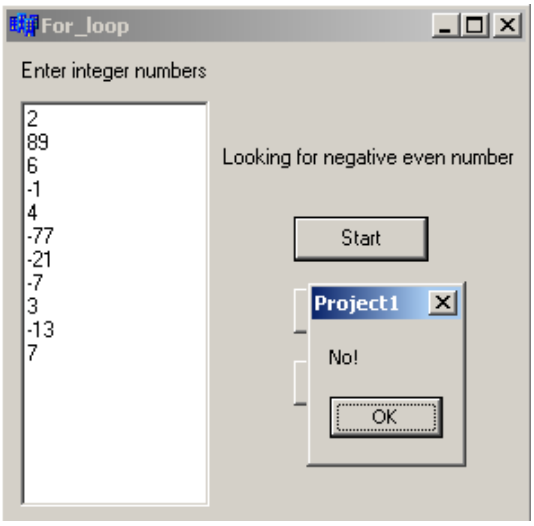
The text of Button1Click function is:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
float x, sum=0, k=0, aver;
int i, n;
n=Memo1->Lines->Count;    // n is the quantity of lines (numbers) in Memo1
if (n==0)                 //if there are no numbers in Memo1
      {ShowMessage ("There are no numbers in Memo1"); return;}
for (i=0; i<n; i++)       //repeat n times (sequentially for each number)
 {x=StrToFloat(Memo1->Lines->Strings[i]);      //read the number from Memo1 (line i) to
variable x
  if (x>-10 && x<10 && x!=0)          //if x is a non-zero one-digit number
   {sum+=x;                           //add it to sum
   k++;}                              //increase the quantity by 1
}
if (k!=0)                 //if the quantity is not 0 (there are such numbers) – to avoid division
by 0
    {aver=sum/k;          //calculate the average
     Edit1->Text=FormatFloat("0.00",aver); }
else                      //if the quantity is 0, we have nothing to calculate
      ShowMessage("There is no one-digit non-zero numbers");
}
```



Example 1.7 Enter integer numbers in Memo. Define, is there negative even number. Output the result with a message.

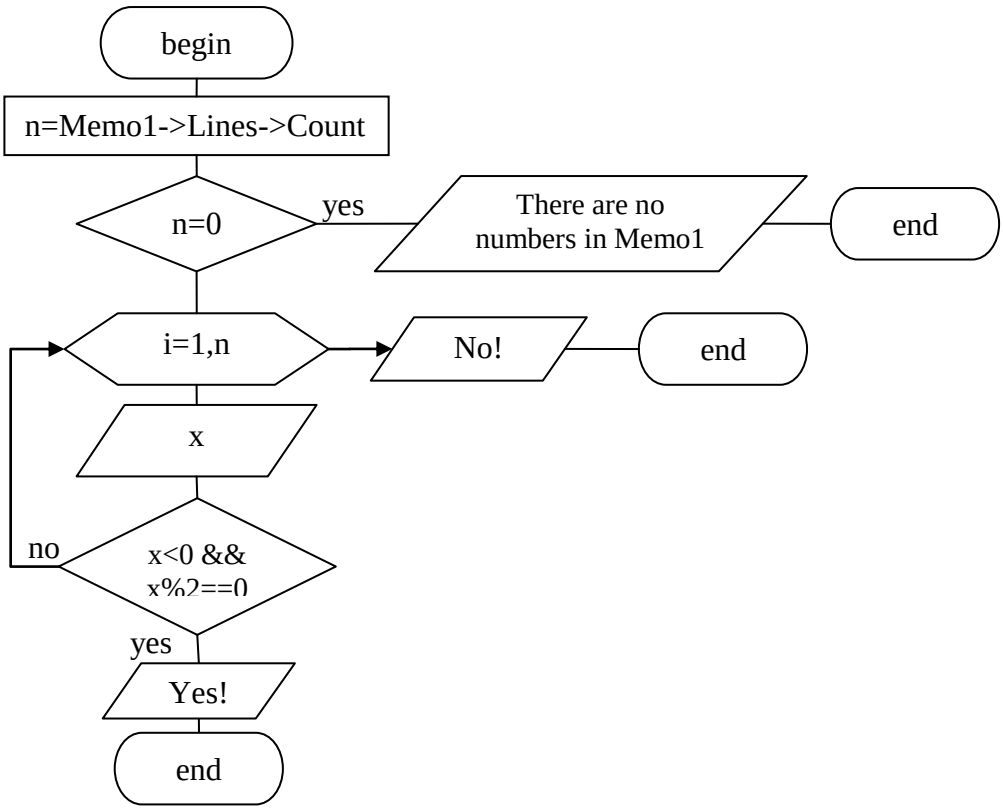




To decide, is there a negative even number, we may calculate the quantity of negative even numbers (as it was done in the previous example) and then if this quantity is greater than 0, we may say that there is at least one negative even number; otherwise we may say that there are no negative even numbers.

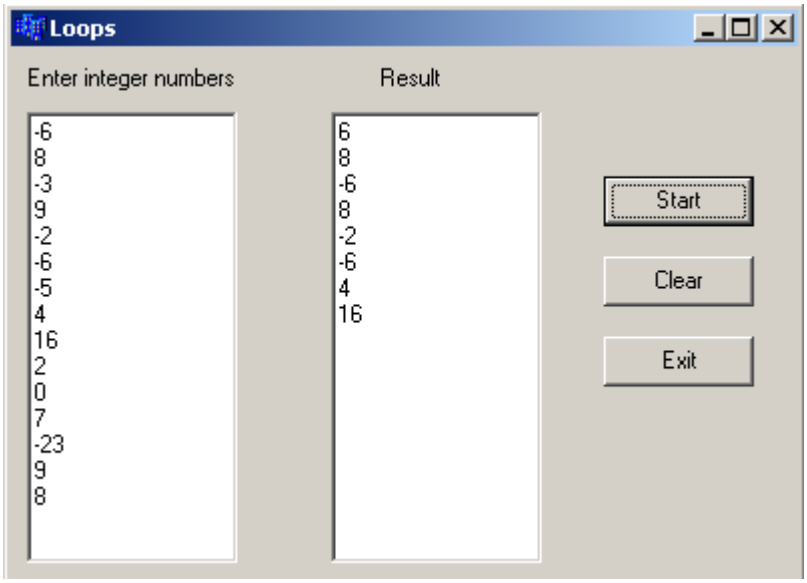
But if we find one negative even number it is not necessary to read and check up other numbers from Memo1. We may output the result immediately and stop. For this purpose we may use return command to stop the execution of Button1Click function. If the loop fulfilled all its iterations and stopped and Button1Click function is still working, it means that we found no negative even number. The text of Button1Click function:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
int x, i, n;
n=Memo1->Lines->Count;
if (n==0) {ShowMessage ("There are no numbers in Memo1"); return;}
for (i=0; i<n; i++)
 {x=StrToInt(Memo1->Lines->Strings[i]);
  if (x<0 && x%2==0)          //if x is a negative even number
   {ShowMessage("Yes!");     //output the result
    return;}                  //stop the function
}
ShowMessage("No!");
}
```



Example 1.8 Enter integer numbers in Memo1. Copy even numbers from Memo1 to Memo2 until their sum will become greater than 10 (use three loops).

We have to stop copying numbers when all numbers in Memo1 are read (there are no more numbers to read) or the sum of copied numbers is greater than 10.



With for loop:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
int x, i, n, S=0;
n=Memo1->Lines->Count;
if (n==0) {ShowMessage ("There are no numbers in Memo1"); return;}
for (i=0; i<n && S<=10; i++)          //the first part of condition means that still there are numbers in Memo1
                    //the second part of condition means that the sum of copied numbers still is not greater than 10
 {x=StrToInt(Memo1->Lines->Strings[i]);
  if (x%2==0)
    {Memo2->Lines->Add(IntToStr(x));
     S+=x;}
}
}
```

With while loop:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
int x, i=0, n, S=0;
n=Memo1->Lines->Count;
if (n==0) {ShowMessage ("There are no numbers in Memo1"); return;}
while (i<n && S<=10)
 {x=StrToInt(Memo1->Lines->Strings[i]);
  if (x%2==0)
    {Memo2->Lines->Add(IntToStr(x));
     S+=x;}
  i++;
}
}
```

With do-while loop:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
int x, i=0, n, S=0;
```

```
n=Memo1->Lines->Count;
if (n==0) {ShowMessage ("There are no numbers in Memo1"); return;}
do{
  x=StrToInt(Memo1->Lines->Strings[i]);
  if (x%2==0)
     {Memo2->Lines->Add(IntToStr(x));
      S+=x;}
  i++;
}while (i<n && S<=10);
}
```

Individual tasks

Task 1 (low level of complexity)

1. Enter float numbers in Memo1. Calculate the product of positive numbers.
2. Enter double numbers in Memo1. Calculate the sum of numbers from interval [-4, 6.5].
3. Enter float numbers in Memo1. Calculate the sum of negative numbers.
4. Enter double numbers in Memo1. Calculate the average of numbers from interval [-1, 3].
5. Enter integer numbers in Memo1. Calculate the product of odd numbers.
6. Enter float numbers in Memo1. Calculate the quantity of numbers, which absolute value is less than 7.
7. Enter integer numbers in Memo1. Calculate the quantity of one-digit numbers.
8. Enter integer numbers in Memo1. Calculate the average of two-digit numbers.
9. Enter integer numbers in Memo1. Calculate the product of numbers, which are multiple to 3 and greater than 4.
10. Enter integer numbers in Memo1. Calculate the sum of numbers, which are not multiple to 3 and less than 10.
11. Enter double numbers in Memo1. Calculate the quantity of numbers from interval (-2, 6].
12. Enter integer numbers in Memo1. Calculate the sum of two-digit even numbers.
13. Enter integer numbers in Memo1. Calculate the product of one-digit negative numbers.
14. Enter integer numbers in Memo1. Calculate the average of positive odd numbers.
15. Enter float numbers in Memo1. Calculate the average of numbers from interval [-5, 4).
16. Enter double numbers in Memo1. Calculate the sum of squares of positive numbers, which are less than 20.
17. Enter integer numbers in Memo1. Calculate the sum of absolute values of numbers, which are multiple to 3 and 4.
18. Enter float numbers in Memo1. Define, is there a number 2.5. Output the result with a message.
19. Enter float numbers in Memo1. Calculate the quantity of numbers greater than average.
20. Enter double numbers in Memo1. Calculate the sum of numbers, which are greater than first number.
21. Enter integer numbers in Memo1. Calculate the product of numbers, which are multiple to 3 and not multiple to 2.
22. Enter float numbers in Memo1. Define, is there a number from interval (-2, 3]. Output the result with a message.
23. Enter integer numbers in Memo1. Calculate the quantity of numbers with absolute value from interval [4, 11].
24. Enter float numbers in Memo1. Calculate the difference between the sum of positive and the product of negative numbers.

Task 2 (medium level of complexity)

1. Enter float numbers in Memo1. Find the bigger number.
2. Enter double numbers in Memo1. Find the smaller number.
3. Enter float numbers in Memo1. Find the position (line number) of the bigger number.

4. Enter double numbers in Memo1. Find the position (line number) of the smaller number.
5. Enter integer numbers in Memo1. Calculate the difference between the bigger and the smaller numbers.
6. Enter integer numbers in Memo1. Define whether the numbers are allocated in ascending order.
7. Enter integer numbers in Memo1. Define whether the numbers are allocated in descending order.
8. Enter integer numbers in Memo1. Find the biggest of the positive numbers.
9. Enter integer numbers in Memo1. Calculate the quantity of numbers, which are allocated between the first and the last even numbers.
10. Enter double numbers in Memo1. Define, whether there are equal numbers.
11. Enter integer numbers in Memo1. Define whether all numbers are different.
12. Enter integer numbers in Memo1. Define whether the smallest number is odd.
13. Enter integer numbers in Memo1. Find the smallest of the positive numbers.
14. Enter float numbers in Memo1 in ascending order. Find the position (line number), where we can insert number 5 without violation of the order.
15. Enter double numbers in Memo1. Define, what is bigger – the smallest absolute value of positive numbers or the biggest absolute value of negative numbers.
16. Enter integer numbers in Memo1. Find the number with smallest absolute value.
17. Enter float numbers in Memo1. Calculate the quantity of different numbers.
18. Enter float numbers in Memo1. Find the biggest of the negative numbers.
19. Enter double numbers in Memo1. Calculate the product of numbers, which meet more than once.
20. Enter integer numbers in Memo1. Define, what is bigger – the smallest even number or the biggest odd number.
21. Enter float numbers in Memo1. Find the number with biggest absolute value.
22. Enter integer numbers in Memo1. Calculate the sum of numbers, which are allocated after the biggest number.
23. Enter float numbers in Memo1 in descending order. Find the position (line number), where we can insert number 0 without violation of the order.
24. Enter double numbers in Memo1. Define whether the biggest number is even.

Task 3 (high level of complexity)

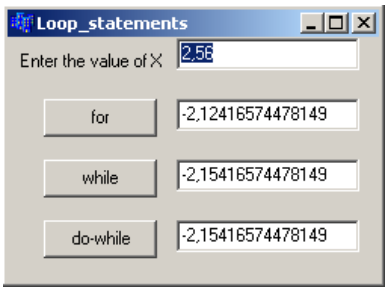
1. Enter float numbers in Memo1. Copy numbers from Memo1 to Memo2 until their product will become negative (using three loops). What loop is the best for solving this problem? Why?
2. Enter double numbers in Memo1. Copy 10 positive numbers from Memo1 to Memo2 (using three loops). What loop is the best for solving this problem? Why?
3. Enter float numbers in Memo1. Copy negative numbers from Memo1 to Memo2 until their product will become positive greater than 50 (using three loops). What loop is the best for solving this problem? Why?
4. Enter double numbers in Memo1. Copy positive numbers from Memo1 to Memo2 until the first negative number will be found (using three loops). What loop is the best for solving this problem? Why?
5. Enter integer numbers in Memo1. Copy even numbers from Memo1 to Memo2 until their product will become multiple to 16 (using three loops). What loop is the best for solving this problem? Why?
6. Enter float numbers in Memo1. Copy 7 negative numbers from Memo1 to Memo2 (using three loops). What loop is the best for solving this problem? Why?
7. Enter integer numbers in Memo1. Calculate the quantity of the positive numbers going successively in beginning of Memo1 before the first zero or negative number (using three loops). What loop is the best for solving this problem? Why?

8. Enter integer numbers in Memo1. Copy numbers from Memo1 to Memo2 until the number from the interval [-2, 8) will be found (using three loops). What loop is the best for solving this problem? Why?
9. Enter integer numbers in Memo1. Calculate the quantity of the even numbers going successively in beginning of Memo1 before the first odd number (using three loops). What loop is the best for solving this problem? Why?
10. Enter integer numbers in Memo1. Copy even numbers from Memo1 to Memo2, while their product belongs to interval (-10, 25] (using three loops). What loop is the best for solving this problem? Why?
11. Enter double numbers in Memo1. Calculate the quantity of non-zero numbers going successively in beginning of Memo1 before the first zero number (using three loops). What loop is the best for solving this problem? Why?
12. Enter integer numbers in Memo1. Copy 6 numbers multiple to 3 from Memo1 to Memo2 (using three loops). What loop is the best for solving this problem? Why?
13. Enter integer numbers in Memo1. Copy odd numbers from Memo1 to Memo2, until their sum will become negative (using three loops). What loop is the best for solving this problem? Why?
14. Enter integer numbers in Memo1. Calculate the quantity of one-digit even numbers, going successively in beginning of Memo1 (using three loops). What loop is the best for solving this problem? Why?
15. Enter float numbers in Memo1. Copy numbers from Memo1 to Memo2, until the number with absolute value 15 will be found (using three loops). What loop is the best for solving this problem? Why?
16. Enter double numbers in Memo1. Copy numbers from Memo1 to Memo2, until their product will become negative (using three loops). What loop is the best for solving this problem? Why?
17. Enter integer numbers in Memo1. Copy odd numbers from Memo1 to Memo2, until their sum will become negative (using three loops). What loop is the best for solving this problem? Why?
18. Enter float numbers in Memo1. Copy numbers greater than 4 from Memo1 to Memo2 until their sum will become greater than 50 (using three loops). What loop is the best for solving this problem? Why?
19. Enter float numbers in Memo1. Calculate the quantity of numbers with absolute value greater than 3, going successively in beginning of Memo1 (using three loops). What loop is the best for solving this problem? Why?
20. Enter double numbers in Memo1. Calculate the sum of numbers, going before the number 100 (using three loops). What loop is the best for solving this problem? Why?
21. Enter integer numbers in Memo1. Calculate the product of odd numbers, going before the number 25 (using three loops). What loop is the best for solving this problem? Why?
22. Enter float numbers in Memo1. Copy positive numbers from Memo1 to Memo2, until the zero number will be found (using three loops). What loop is the best for solving this problem? Why?
23. Enter integer numbers in Memo1. Calculate the average of numbers, going before the zero number (using three loops). What loop is the best for solving this problem? Why?
24. Enter float numbers in Memo1. Calculate the sum of positive numbers, going before the first negative number or number 100 (using three loops). What loop is the best for solving this problem? Why?

Laboratory work 2.3
Calculation with use of conditional loop statements

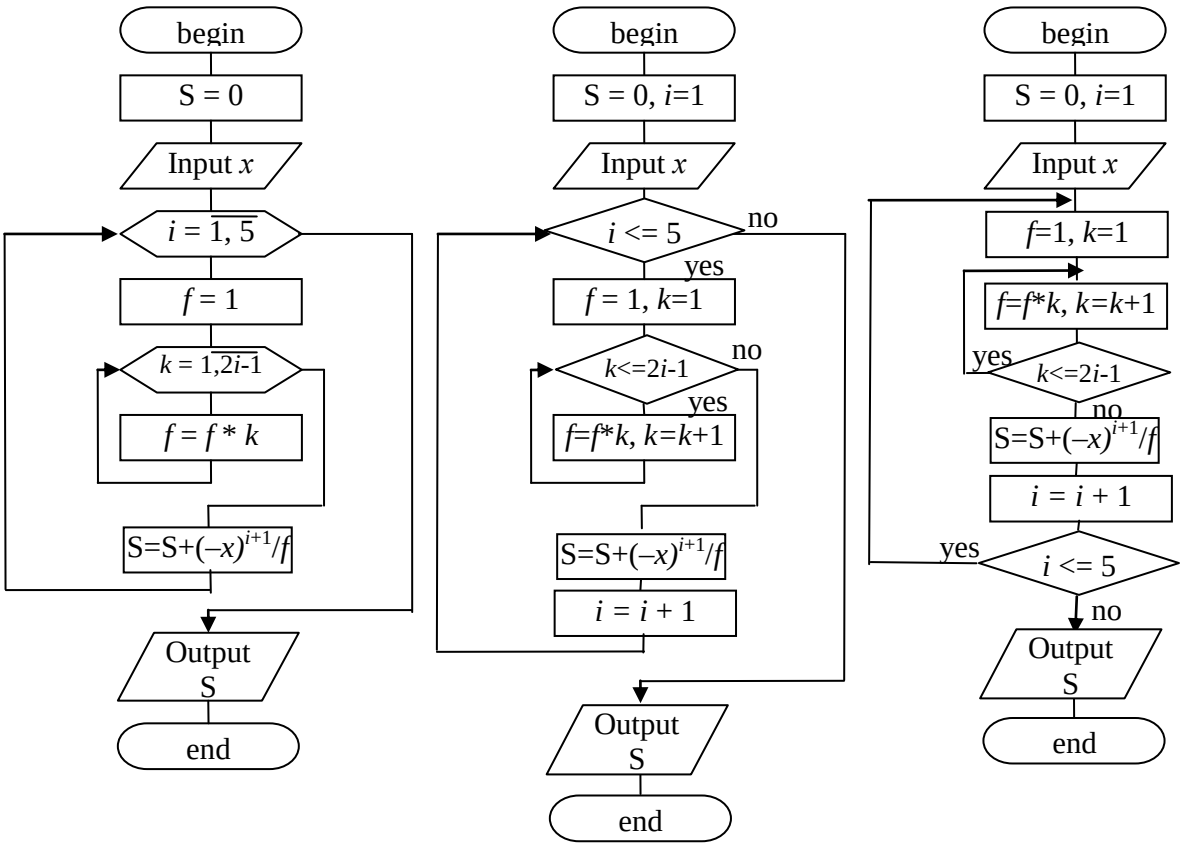
The purpose: To study conditional loop statements while and do-while. To get skills of their usage in cyclic programs.

Example 1.9 Calculate the sum of the alternating series $S = \sum_{i=1}^{5} \frac{(-1)^{i+1} x^{i+1}}{(2i-1)!}$ (using **for**, **while** and **do-while** loops).



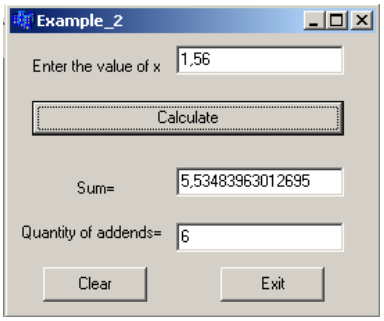

```
void __fastcall TForm1::Button1Click(TObject *Sender)   //for loop statement
{  int i,f,k;
   float s=0, x=StrToFloat(Edit1->Text);
   for( i=1;i<=5;i++)
    {  for( k=1,f=1;k<=2*i-1;k++)
              f*=k;
      s+=pow(-x, i+1)/f;
    }
   Edit2->Text=FloatToStr(s);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button2Click(TObject *Sender) //while loop statement
{ int i=1,f,k;
  float s=0, x=StrToFloat(Edit1->Text);
  while( i<=5)
   {  f=1; k=1;
      while(k<=2*i-1)
          {f*=k; k++; }
      s+=pow(-x, i+1)/f;
      i++;
   }
  Edit3->Text=FloatToStr(s);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button3Click(TObject *Sender) //do-while loop statement
{ int i=1,f,k;
  float s=0, x=StrToFloat(Edit1->Text);
   do
    { f=1; k=1;
      do {f*=k; k++; }while(k<=2*i-1) ;
      s+=pow(-x, i+1)/f;
      i++;
   } while( i<=5);
  Edit4->Text=FloatToStr(s);
}
```

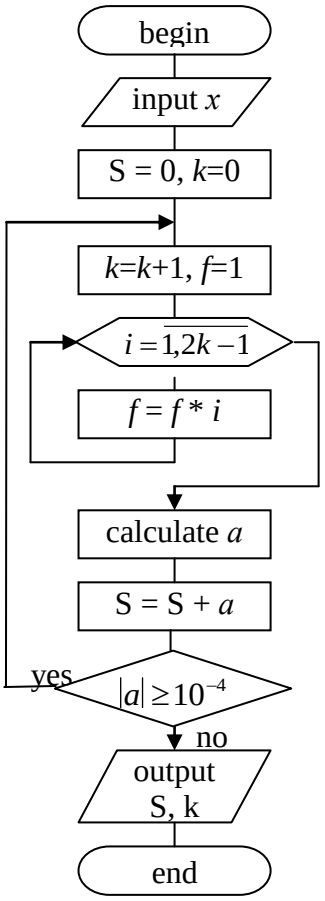
Schemes of the program with different loop statements:



for ***while*** ***do-while***

Example 1.10 Calculate the sum of series $S = \sum_{k=1}^{\infty} \frac{x^{2k+1}}{(2k-1)!}$, sum only those terms of the series, which absolute values are greater than given exactitude $\varepsilon = 10^{-4}$. Define the number of addends. Enter the value of x (-2<*x*<2) from the screen.





We can use the for loop in this program. But it is better to use while or do-while, as we do not know the number of repetitions.

The addend is:

$$\frac{x^{2k+1}}{(2k-1)!}$$

The loop continues while the condition $\left|\frac{x^{2k+1}}{(2k-1)!}\right| < \varepsilon$ is true.

Before the first iteration we do not know the value of first addend, therefore we can not check up the condition. That is why we need to check up the condition after the iteration. It is possible when we use do-while loop.

On each iteration we increase the number of addend, calculate the factorial (using for-loop), then calculate the addend and add it to sum.

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{ float x, a, s=0;
  int f, i, k=0;
```

```
x=StrToFloat(Edit1->Text);
do                                  // Loop with postcondition
{  k++;                             // increase the variable k by 1, k is a number of addend
   for(i=1,f=1; i<=2*k-1; i++) f*=i; // calculation of factorial (2k−1)!
   a=pow(x, 2*k+1)/f;               // calculation of the k-th addend
   s+=a;    }                       // add the calculated addend to sum
while(fabs(a)>=1e-4);  //loop continues while the condition fabs(a)>=1e-4 is true and
                       //it stops when the absolute value of addend becomes less than 0.0001
Edit2->Text=FloatToStr(s) ;         // Output the sum
Edit3->Text=IntToStr(k);            // Output the number of addends
}
```

Example 1.11 Calculate the sum of series $y=\sum_{k=1}^{\infty}\frac{(-1)^k x^{2k}}{2k(2k-1)!}$, sum only those terms of the series, which absolute values are greater than given exactitude $\varepsilon=10^{-4}$. Define the number of addends. Enter the value of x ($-2<x<2$) from the screen.

We may write this program similarly to the previous one. But in situation, when an addend contains factorial and power it is better to use **recurrent factor**.

Let's write the sum of series $y=\sum_{k=1}^{\infty}\frac{(-1)^k x^{2k}}{2k(2k-1)!}$ as $y=\sum_{k=1}^{\infty}u_k$, where $u_k=\frac{(-1)^k x^{2k}}{2k(2k-1)!}$.

Recurrent factor is a relation of two terms of the series with order numbers **k-1** and **k**:

$$u_2=R\cdot u_1,\quad u_3=R\cdot u_2,\quad \ldots,\quad u_k=R\cdot u_{k-1}.$$

$$\boxed{R=\frac{u_k}{u_{k-1}}}. \qquad (1)$$

To calculate the recurrent factor, substitute $u_k=\frac{(-1)^k x^{2k}}{2k(2k-1)!}$ and u_{k-1} in the formula (1). To calculate u_{k-1} substitute (k-1) instead of k in formula of u_k:

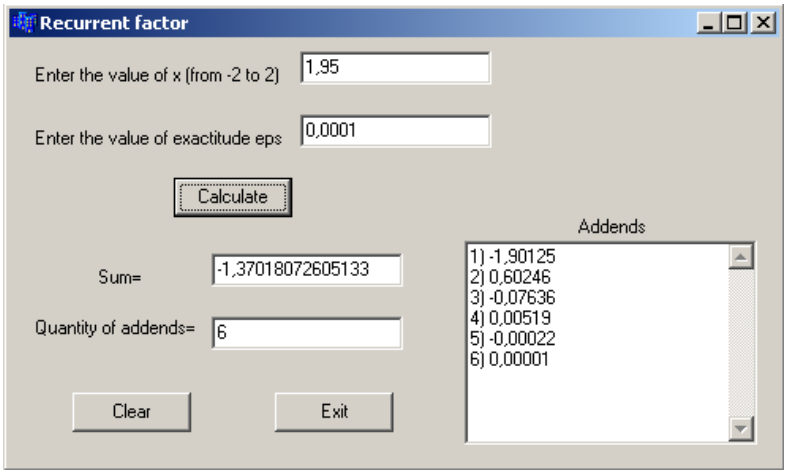
$$u_{k-1}=\frac{(-1)^{k-1}x^{2(k-1)}}{2(k-1)(2(k-1)-1)!}=\frac{(-1)^{k-1}x^{2(k-1)}}{2(k-1)(2k-3)!}.$$

$$R=\frac{u_k}{u_{k-1}}=\frac{(-1)^k x^{2k}}{2k(2k-1)!}\cdot\frac{2(k-1)(2k-3)!}{(-1)^{k-1}x^{2(k-1)}}=\frac{(-1)^k\cdot x^{2k}\cdot(2k-2)\cdot(2k-3)!}{(-1)^{k-1}\cdot x^{2k-2}\cdot 2k\cdot(2k-1)!}=$$

$$=\frac{(-1)^{k-k+1}\cdot x^{2k-(2k-2)}\cancel{(2k-2)}}{2k}\cdot\frac{\cancel{1\cdot 2\cdot\ldots\cdot(2k-3)}}{\cancel{1\cdot 2\cdot\ldots\cdot(2k-3)}\cdot\cancel{(2k-2)}\cdot(2k-1)}=-\frac{x^2}{2k(2k-1)}.$$

In program to calculate u_2 we have to know the value of u_1 (addend with k=1) before it:

$$u_1=\frac{(-1)^1 x^2}{2(2-1)!}=-\frac{x^2}{2}.$$




```
void __fastcall TForm1::Button1Click(TObject *Sender)
      {  float x,y,u,r,eps;    Memo1->Clear();
        x=StrToFloat(Edit1->Text);
        eps=StrToFloat(Edit2->Text);
        int i,f,k=1;
        u=-x*x/2; y=u;                  //calculate the first addend and output it in Memo1
        Memo1->Lines->Add(IntToStr(k)+") "+ FormatFloat("0.00000",u));
        do
         { k++;                         //increase k to calculate the next addend
           r = -x*x/(2*k*(2*k-1));      //calculate the recurrent factor
           u *= r ;                     //calculate the addend (multiply the previous addend by recurrent factor)
           Memo1->Lines->Add(IntToStr(k)+") "+ FormatFloat("0.00000",u));
           y += u;    }                 //add the calculated addend to sum
        while(fabs(u)>=eps)  ;          //the loop continues while fabs(u) is greater than eps
        Edit3->Text=FloatToStr(y);
        Edit4->Text=IntToStr(k);
      }
```

The task for laboratory work

Individual tasks with low level of complexity

Make the schemes of algorithm and programs of calculating the sum of series $\sum_{k=1}^{\infty} u_k$, sum only those terms of the series, which absolute values are greater than given exactitude $\varepsilon = 10^{-4}$. Use two conditional loop statements. Enter the value of x (-2<x<2) from the screen. Run the program and calculate for x=0,5. Function $f(x)$ is from table 2.3.

Individual tasks with medium level of complexity

Make the schemes of algorithm and programs of calculating the sum of series $\sum_{k=1}^{\infty} u_k$, sum only those terms of the series, which absolute values are greater than given exactitude $\varepsilon = 10^{-4}$. Use two conditional loop statements and the recurrent factor. Define the number of addends. Enter the value of x (-2<x<2) from the screen. Run the program and calculate for x=0,5. Function $f(x)$ is from table 2.3.

Table 2.3 – Individual tasks

1	$\sum_{k=1}^{\infty}\frac{(-1)^k x^{2k+1}}{(2k+1)!}$	2	$\sum_{k=1}^{\infty}\frac{(-1)^{k+1} x^{2k}}{(2k-1)!}$	3	$\sum_{k=1}^{\infty}\frac{(-1)^{k-1} x^{k}}{k!}$	4	$\sum_{k=1}^{\infty}\frac{(-1)^{k+1} x^{2k}}{k!\cdot 2^k}$
5	$\sum_{k=1}^{\infty}\frac{(-1)^k x^{2k-1}}{2k\cdot(2k+1)!}$	6	$\sum_{k=1}^{\infty}\frac{(-1)^{k+1} x^{k-1}}{(2k-1)\cdot(k+1)!}$	7	$\sum_{k=1}^{\infty}\frac{(-1)^k x^{2(k+1)}}{(k+2)k!}$	8	$\sum_{k=1}^{\infty}\frac{(-1)^{k-1} x^{3k-1}}{(2k)!}$
9	$\sum_{k=1}^{\infty}\frac{(-1)^{k+1} x^{k+2}}{k(2k+1)!}$	10	$\sum_{k=1}^{\infty}\frac{(-1)^k x^{2k+1}}{(2k-1)!}$	11	$\sum_{k=1}^{\infty}\frac{(-1)^{k-1} x^{3k+1}}{3k\cdot(k+1)!}$	12	$\sum_{k=1}^{\infty}\frac{(-1)^{k+1} x^{k+3}}{k^2(k+2)!}$
13	$\sum_{k=1}^{\infty}\frac{(-1)^k x^{3k-1}}{2k(k+3)!}$	14	$\sum_{k=1}^{\infty}\frac{(-1)^{k+1} x^{3(k-2)}}{(k+3)(3k)!}$	15	$\sum_{k=1}^{\infty}\frac{(-1)^{k-1} x^{3k-2}}{2^{k+1}\cdot k!}$	16	$\sum_{k=1}^{\infty}\frac{(-1)^k x^{2k+1}}{(2k+1)!}$
17	$\sum_{k=1}^{\infty}\frac{(-1)^{k+1} x^{2k}}{(2k+1)!}$	18	$\sum_{k=1}^{\infty}\frac{(-1)^{k+1} x^{2k}}{k!2^{k-1}}$	19	$\sum_{k=1}^{\infty}\frac{(-1)^{k+1} x^{k}}{(k+4)!}$	20	$\sum_{k=1}^{\infty}\frac{(-1)^{k+1} x^{k-1}}{2k(2k+1)!}$
21	$\sum_{k=1}^{\infty}\frac{(-1)^k x^{2k-1}}{(2k-1)(k+1)!}$	22	$\sum_{k=1}^{\infty}\frac{(-1)^{k-1} x^{3k-1}}{(k+2)k!}$	23	$\sum_{k=1}^{\infty}\frac{(-1)^k x^{2(k+1)}}{(2k)!}$	24	$\sum_{k=1}^{\infty}\frac{(-1)^k x^{2k+1}}{k\cdot(2k+1)!}$
25	$\sum_{k=1}^{\infty}\frac{(-1)^{k-1} x^{3k+1}}{3k(2k-1)!}$	26	$\sum_{k=1}^{\infty}\frac{(-1)^k x^{2k+1}}{k^2(k+1)!}$	27	$\sum_{k=1}^{\infty}\frac{(-1)^k x^{3k-1}}{(k+3)(3k)!}$	28	$\sum_{k=1}^{\infty}\frac{(-1)^k x^{3k-1}}{(k+1)!\cdot k^2}$
29	$\sum_{k=1}^{\infty}\frac{(-1)^{k-1} x^{3k-2}}{2k(k+3)!}$	30	$\sum_{k=1}^{\infty}\frac{(-1)^{k+1} x^{3(k-2)}}{(k+3)(3k)!}$	31	$\sum_{k=1}^{\infty}\frac{(-1)^k x^{2k-1}}{k(k+3)!}$	32	$\sum_{k=1}^{\infty}\frac{(-1)^k x^k (2k-1)}{(3k-2)!}$

2.2 Functions

Laboratory work 2.4
Functions

The purpose: To study functions in C++.

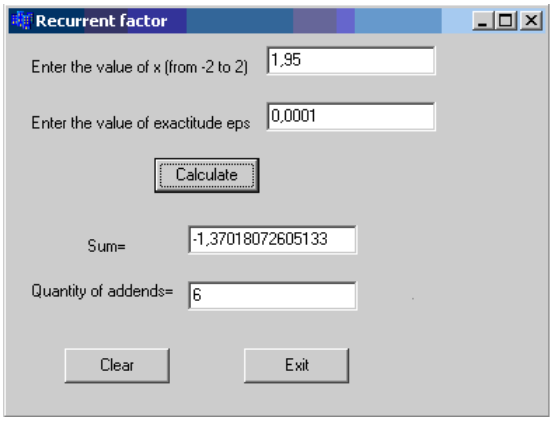
Example 2.1 Calculate the sum of series $y=\sum_{k=1}^{\infty}\frac{(-1)^k x^{2k}}{2k(2k-1)!}$, sum only those terms of the series, which absolute values are greater than given exactitude $\varepsilon=10^{-4}$. Define the number of addends. Enter the value of x (-2<*x*<2) from the screen. Use function to calculate the sum.

Recurrent:

$$u_{k-1}=\frac{(-1)^{k-1}x^{2(k-1)}}{2(k-1)(2(k-1)-1)!}=-\frac{x^2}{2k(2k-1)}.$$

In program to calculate u_2 we have to know the value of u_1 (addend with *k*=1) before it:

$$u_1=\frac{(-1)^1 x^2}{2(2-1)!}=-\frac{x^2}{2}.$$



The program code

```
//function calculates the sum
      double sum (double x, double eps, int &k)
      {double u, r, y;
       u=-x*x/2; y=u;                  //calculate the first addend and output it in Memo1
        do
         { k++;                        //increase k to calculate the next addend
           r = -x*x/(2*k*(2*k-1));     //calculate the recurrent factor
           u *= r ;                    //calculate the addend (multiply the previous addend by recurrent factor)
           y += u;    }                //add the calculated addend to sum
        while(fabs(u)>=eps)  ;         //the loop continues while fabs(u) is greater than eps
    return y;                          //return calculated value in Button1
        }
//Button1 inputs x and eps, calls the function sum and outputs the result
void __fastcall TForm1::Button1Click(TObject *Sender)
      { int k=0;
        float x,Y,eps;    Memo1->Clear();
```

```
  x=StrToFloat(Edit1->Text);
  eps=StrToFloat(Edit2->Text);
  Y=sum(x, eps, k);      //function call
  Edit3->Text=FloatToStr(Y);
    Edit4->Text=IntToStr(k);
}
```

Individual task

Write the same programs as in the previous lab work (Table 3.1), **use function** to calculate the sum.

2.3 One-dimensional arrays

An array is a group of values of the same data type. Because the items are considered in a group, they are declared as one variable but the declaration must indicate that the variable represents various items. The items that are part of the group are also referred to as members or elements of the array. To declare an array, you must give it a name.

*DataType ArrayName***[***Dimension***]**;

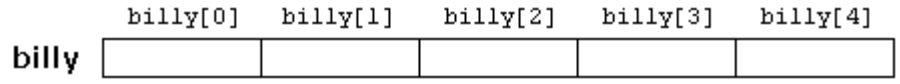
The declaration of array starts by specifying a data type, the DataType in our syntax. This indicates the kind of values shared by the elements of the array. It also specifies the amount of memory space that each member of the array will need to store its value. Like any other variable, an array must have a name, which is the ArrayName in our syntax. The name of the array must be followed by an opening and closing square brackets "[]". Inside of these brackets, you must type the number of items that the array is made of; that is the Dimension in our syntax.

Examples of declaration:

int student[5]; *//array of 5 integer elements*

float billy [10]; *//array of 10 float elements*

When declaring an array, before using it, we saw that you must specify the number of members of the array. This declaration allocates an amount of memory space to the variable. The first member of the array takes a portion of this space. The second member of the array occupies memory next to it:



Each member of the array can be accessed using its position. The position is also referred to as an **index**. The elements of an array are arranged starting at index 0, followed by index 1, then index 2, etc. This system of counting is referred to as "zero-based" because the counting starts at 0. To locate a member, type the name of the variable followed by an opening and closing square brackets. Inside the bracket, type the zero-based index of the desired member. After locating the desired member of the array, you can assign it a value, exactly as you would any regular variable.
For example, to store the value 7.5 in the third element of billy, we could write the following statement:
billy[2] = 7.5;
and, for example, to pass the value of the third element of billy to a variable called a, we could write:
a = billy[2];
Therefore, the expression billy[2] is for all purposes like a variable of type int.

Notice that the third element of billy is specified billy[2], since the first one is billy[0], the second one is billy[1], and therefore, the third one is billy[2]. By this same reason, its last element is

billy[4]. Therefore, if we write billy[5], we would be accessing the sixth element of billy and therefore exceeding the size of the array.
Some other valid operations with arrays:
billy[0] = a;
billy[a] = 75;
b = billy [a+2];
billy[billy[a]] = billy[2] + 5;

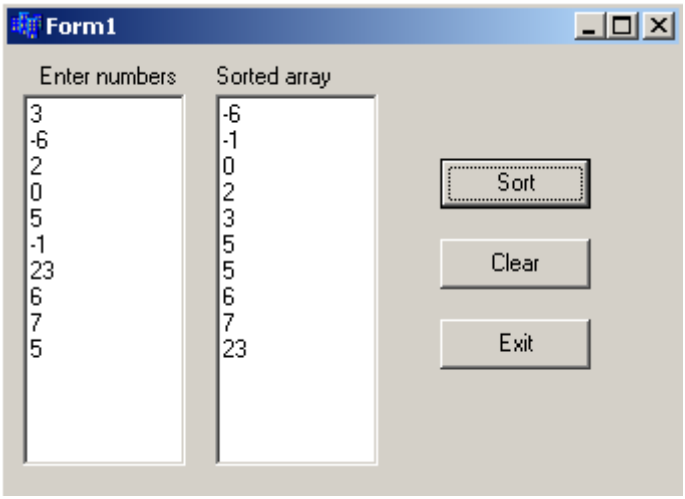
SORTING
Bubble SORT

The technique: Make several passes through the array. On each pass, successive pairs of elements are compared. If a pair is in increasing order (or values are identical), do nothing. If a pair is in decreasing order, swap the values.

Assume you have an array a[10]. First the program compares a[0] to a[1], then a[1] to a[2], then a[2] to a[3], and so on, until it completes the pass by comparing a[8] to a[9]. Although there are 10 elements, we need only 9 comparisons. On the first pass, the largest value is guaranteed to move to a[9]. On the second pass, the second largest value is guaranteed to move to a[8]. On the 9-th pass, the 9-th largest value will move to a[1], which will leave the smallest value in a[0].

Bubble sort algorithm:

- Advantage: easy to program
- Disadvantage: runs slowly, not appropriate for large arrays




```
void __fastcall
TForm1::Button1Click(TObject *Sender)
{
  const int SIZE = 10;
  int a[SIZE];
  int i, temp;                    // temp will be
used in swap

//Input the array to sort
 for ( i = 0; i < SIZE; i++ )
      a[i]=StrToInt(Memo1->Lines->Strings[i]);
//Sorting
  for ( int j = 0; j < SIZE - 1; j++ )    // repeat SIZE-1 times
  {
    for ( i = 0; i < SIZE - 1; i++ ){    // for each element (except the last) of array
      if ( a[i] > a[i + 1] ) {           // compare it with the next element and if they  are in wrong
order
        temp = a[i];                     // swap them
        a[i] = a[i + 1];
        a[i + 1] = temp;
      }
    }
  }
 //Output the sorted array
  for ( i = 0; i < SIZE; i++ )
    Memo2->Lines->Add(IntToStr(a[i]));
}
```

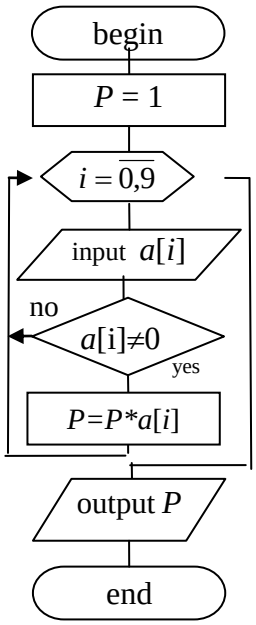
This algorithm can be optimized: repeat passes until the array becomes sorted (there can be less than SIZE-1 passes). We may write do-while loop instead of first for. And condition is: while array is not sorted. We need in special variable to indicate, is array sorted or not. We may count how many times we replaced elements during this pass, and non-zero value shows that array is not sorted.

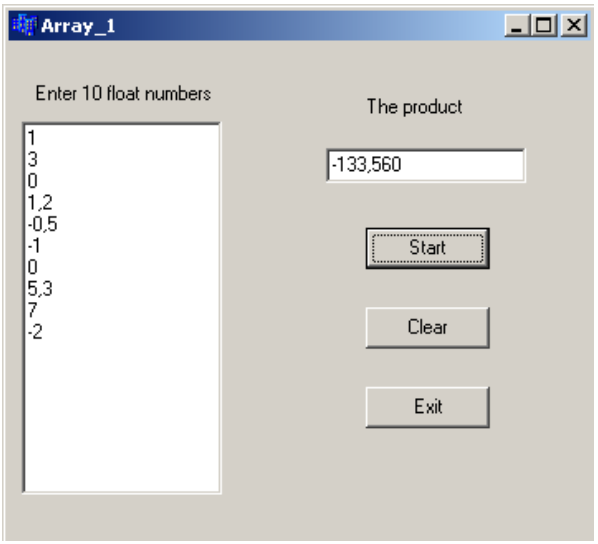
```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  const int SIZE = 10;
  int a[SIZE];
  int i, temp;
  int counter;                //number of replacements on each pass
  for ( i = 0; i < SIZE; i++ )
      a[i]=StrToInt(Memo1->Lines->Strings[i]);
  do{                         // passes
    counter=0;                // before each pass counter is 0
    for ( i = 0; i < SIZE - 1; i++ ){
  if ( a[i] > a[i + 1] ) {     // compare it with the next element and if they  are in wrong order
        temp = a[i];           // swap them
        a[i] = a[i + 1];
        a[i + 1] = temp;
        counter++;            //and increase the counter
      }
    }
  }while(counter>0);          //if array is sorted counter will be 0 after the pass

  for ( i = 0; i < SIZE; i++ )
    Memo2->Lines->Add(IntToStr(a[i]));

}
```

Laboratory work 2.5
Arrays

The purpose: to get acquaintance with the easiest structured type – one-dimensional array; to learn the means of working with one dimensional arrays in C++.

Example 3.1 Enter an array of 10 float elements and calculate the product of non-zero elements of this array.

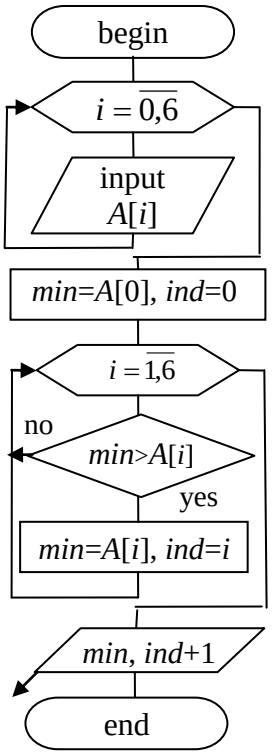


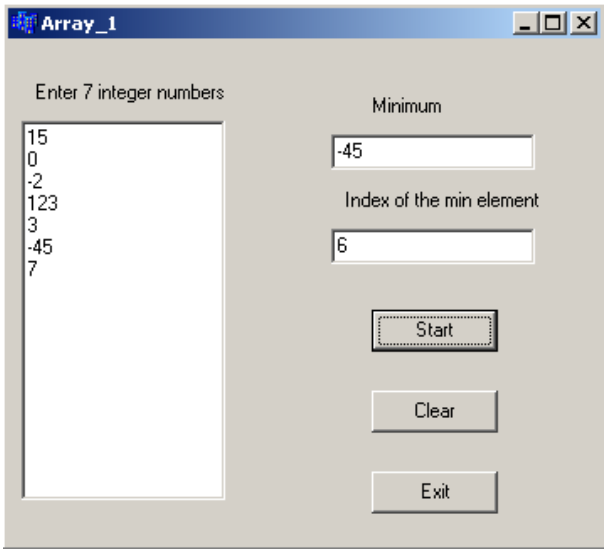


The program code:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{ float a[10], P = 1;
  for(int i = 0; i < 10; i++)
   {  a[i] = StrToFloat(Memo1->Lines->Strings[i]);
      if(a[i] != 0)  P *= a[i];
   }
  Edit1->Text = FormatFloat("0.000", P);
}
```

Example 3.2 Enter an array of 7 integer elements. Find the minimum of the elements and its index.



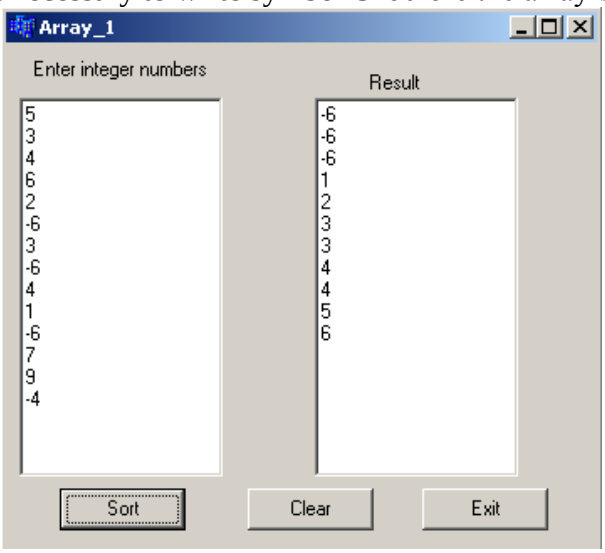


The program code:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{ int A[15] ;  int i;
  for(i=0; i<7; i++)
     A[i]= StrToInt(Memo1->Lines->Strings[i]);
  int min=A[0], ind=0;
  for(i=1; i<7; i++)
    if(min>A[i])
      { min=A[i];  ind=i;  }
  Edit1->Text = IntToStr(min);
  Edit2->Text = IntToStr(ind+1);  }
```

Example 3.3 Enter the array of 11 (or less) integer numbers in Memo. Sort these numbers in ascending order.

We'll write the function to sort our array. The elements of the array will change after sorting and we want to save these changing. Arrays are transmitted in subroutines by reference, therefore it is **not** necessary to write symbol **&** before the array's name.



The program code:

```
sort(int a[ ], int n)
```

```
{  int i, j, tmp;
   for (i = 0; i < n - 1; i++)
     for (j = i + 1; j < n; j++)
       if (a[i] > a[j])      // compare two elements
          {  tmp = a[i];   // replace elements a[i] and a[j]
             a[i] = a[j];     // variable tmp is necessary for
             a[j] = tmp;    // temporary storage of a[i]
           }
}
void __fastcall TForm1::Button1Click(TObject *Sender)
{ int  n, i, a[11];
  n = Memo1->Lines->Count;
  if (n>11) n=11;
  for (i = 0; i < n; i++)
      a[i] = StrToInt(Memo1->Lines->Strings[i]);
  sort (a,n);
  Memo2->Clear();
  for (i = 0; i < n; i++)
     Memo2->Lines->Add(IntToStr(a[i]));
}
```

The laboratory task

Individual tasks with low level of complexity.

1. Enter array of 10 float numbers in Memo. Calculate the product of positive elements.
2. Enter array of 9 double numbers in Memo. Calculate the sum of elements with values from interval [-4, 6.5].
3. Enter array of 11 float numbers in Memo. Calculate the sum of negative elements.
4. Enter array of 12 double numbers in Memo. Calculate the average of elements with values from interval [-1, 3].
5. Enter array of 10 integer numbers in Memo. Calculate the product of odd elements.
6. Enter array of 9 float numbers in Memo. Calculate the quantity of elements, which absolute value is less than 7.
7. Enter array of 8 integer numbers in Memo. Calculate the quantity of one-digit elements.
8. Enter array of 7 array of integer numbers in Memo. Calculate the average of two-digit elements.
9. Enter array of 9 integer numbers in Memo. Calculate the product of elements, which are multiple to 3 and greater than 4.
10. Enter array of 13 integer numbers in Memo. Calculate the sum of elements, which are not multiple to 3 and less than 10.
11. Enter array of 11 double numbers in Memo. Calculate the quantity of elements with values from interval (-2, 6].
12. Enter array of 12 integer numbers in Memo. Calculate the sum of two-digit even elements.
13. Enter array of 9 integer numbers in Memo. Calculate the product of one-digit negative elements.
14. Enter array of 10 integer numbers in Memo. Calculate the average of positive odd elements.
15. Enter array of 7 float numbers in Memo. Calculate the average of elements with values from interval [-5, 4).
16. Enter array of 9 double numbers in Memo. Calculate the sum of squares of positive elements, which are less than 20.
17. Enter array of 8 integer numbers in Memo. Calculate the sum of absolute values of elements, which are multiple to 3 and 4.

18. Enter array of 12 float numbers in Memo. Define, is there an element with the value = 2.5. Output the result with a message.
19. Enter array of 11 float numbers in Memo. Calculate the quantity of elements, which are greater than average.
20. Enter array of 10 double numbers in Memo. Calculate the sum of elements, which are greater than first number.
21. Enter array of 8 integer numbers in Memo. Calculate the product of elements, which are multiple to 3 and not multiple to 2.
22. Enter array of 11 float numbers in Memo. Define, is there an element with value from interval (-2, 3]. Output the result with a message.
23. Enter array of 9 integer numbers in Memo. Calculate the quantity of elements with absolute value from interval [4, 11].
24. Enter array of 10 float numbers in Memo. Calculate the difference between the sum of positive and the product of negative elements.

Individual tasks with high level of complexity

1. Enter the array of 12 float numbers in StringGrid. Find the bigger element.
2. Enter the array of 11 double numbers in StringGrid. Find the smaller element.
3. Enter the array of 9 float numbers in StringGrid. Find the position (line number) of the bigger element.
4. Enter the array of 10 double numbers in StringGrid. Find the position (line number) of the smaller element.
5. Enter the array of 8 integer numbers in StringGrid. Calculate the difference between the bigger and the smaller elements.
6. Enter the array of 13 integer numbers in StringGrid. Define whether the elements are allocated in ascending order.
7. Enter the array of 11 integer numbers in StringGrid. Define whether the elements are allocated in descending order.
8. Enter the array of 10 integer numbers in StringGrid. Find the biggest of the positive elements.
9. Enter the array of 8 integer numbers in StringGrid. Calculate the quantity of elements, which are allocated between the first and the last even elements.
10. Enter the array of 11 double numbers in StringGrid. Define, whether there are equal elements.
11. Enter the array of 10 integer numbers in StringGrid. Define whether all elements are different.
12. Enter the array of 12 integer numbers in StringGrid. Define whether the smallest element is odd.
13. Enter the array of 11 integer numbers in StringGrid. Find the smallest of the positive elements.
14. Enter the array of 9 float numbers in StringGrid in ascending order. Find the position (line number), where we can insert number 5 without violation of the order.
15. Enter the array of 7 double numbers in StringGrid. Define, what is bigger – the smallest absolute value of positive elements or the biggest absolute value of negative elements.
16. Enter the array of 10 integer numbers in StringGrid. Find the number with smallest absolute value of the elements.
17. Enter the array of 8 float numbers in StringGrid. Calculate the quantity of different numbers.
18. Enter the array of 9 float numbers in StringGrid. Find the biggest of the negative numbers.
19. Enter the array of 11 double numbers in StringGrid. Calculate the product of numbers, which meet more than once.
20. Enter the array of 12 integer numbers in StringGrid. Define, what is bigger – the smallest even number or the biggest odd number.

21. Enter the array of 8 float numbers in StringGrid. Find the number with biggest absolute value.
22. Enter the array of 10 integer numbers in StringGrid. Calculate the sum of numbers, which are allocated after the biggest number.
23. Enter the array of 9 float numbers in StringGrid in descending order. Find the position (line number), where we can insert number 0 without violation of the order.
24. Enter the array of 11 double numbers in StringGrid. Define whether the biggest number is even.

Laboratory work 2.6
Arrays

Example 3.4 *Enter the array of 10 integer numbers. Create the new array of the first array elements divided by the sum of its elements with odd indexes.*

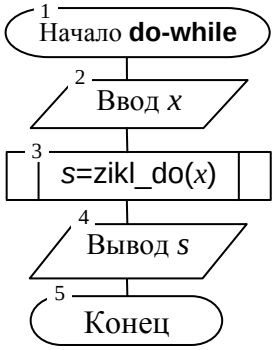
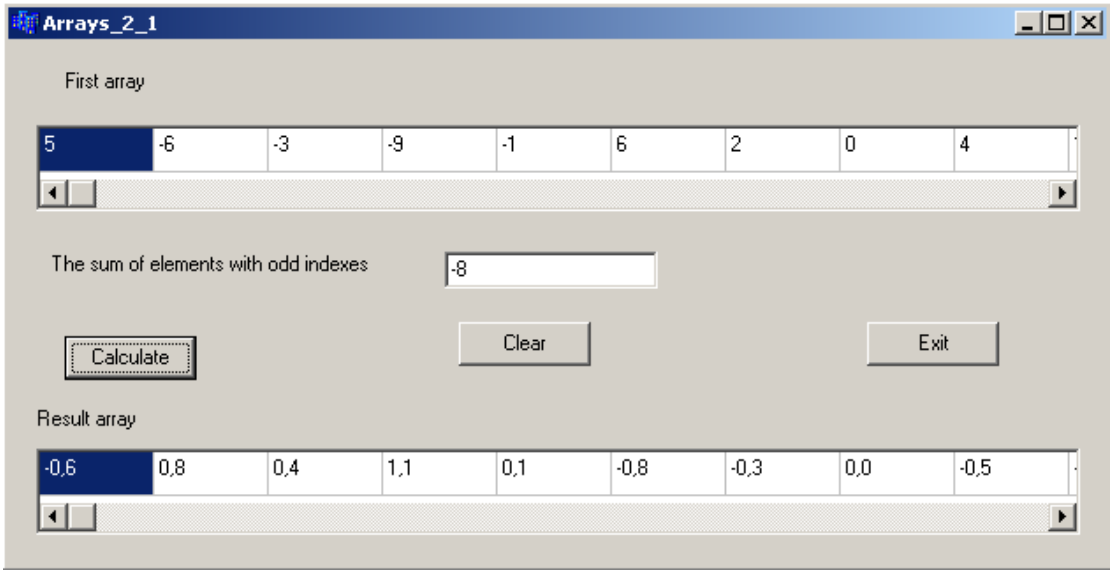


Рисунок 2.11 – Блок-схема для кнопки «**do-while**»

To get a new array, we must calculate the sum of elements with odd indexes. After this we can sequentially divide all elements by sum and assign the result to elements of the new array. New array has the float type (after division).

Let's enter the first array and output the new array in StringGrid (from Additional pannel). This component is used to represent tables. It consists of Cells. Each Cell has Column and Row indexes. Our StringGrid will consist of 1 row and 10 columns.

- Place StringGrid on the form. Its name is StringGrid1. Let's rename it. Enter the new name: SG1.
- Change the properties ColCount (количество столбцов) =10 and RowCount (количество строк) =1.
- Change the properties FixedCols (количество фиксированных столбцов) =0 and FixedRows (количество фиксированных строк) =0.
- Resize our StringGrid.
- We want to enter numbers in Cells of StringGrid. Change the property Options – goEditing on true value and goTabs – also on true.
- Copy this StringGrid and Paste. Rename new StringGrid on SG2**.** This StringGrid is for new array.



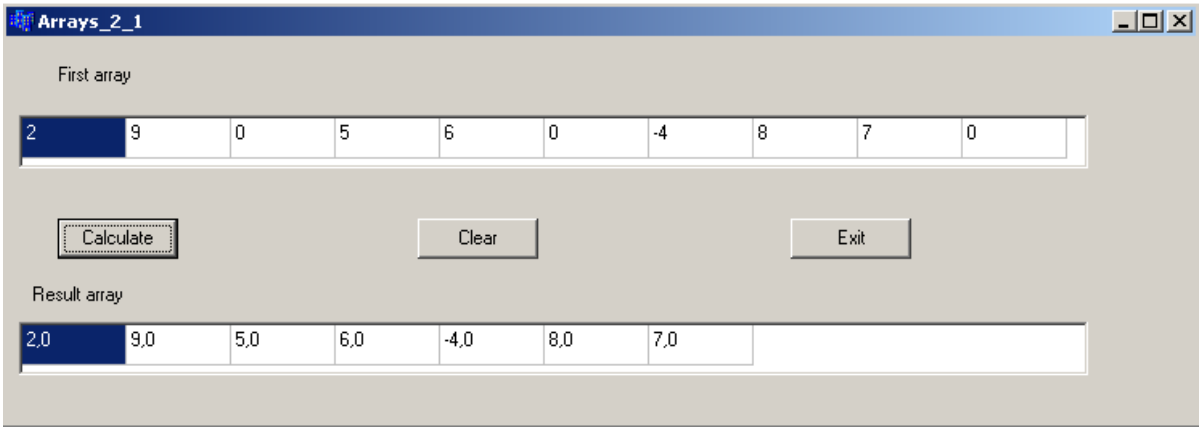
The text of program:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
const int N=10;          //constant N is for quantity elements in array
int a[N], i, sum=0;      //a is the first array
float b[N];              //b is the new array
for (i=0; i<N; i++)
  a[i]=StrToInt(SG1->Cells[i][0]);     //read the value of elements from Cells of StringGrid with row
0 and
```

```
                                        //column i
for (i=0; i<N; i++)                     //calculate the sum
  if (i%2!=0)sum+=a[i];
Edit1->Text=IntToStr(sum);
for (i=0; i<N; i++)
  b[i]=1.*a[i]/sum;              //divide elements a[i] by sum and assign the result to b[i]
for (i=0; i<N; i++)
  SG2->Cells[i][0]=FormatFloat("0.0",b[i]);
}
```

Example 3.5 *Enter the array of ≤10 floating point numbers. Create the new array of the non-zero elements.*

We do not know the number of non-zero elements. Therefore we declare the new array with 10 elements. In example 1 indexes of the elements in both arrays were equal. In this example, indexes are not equal, because zero-numbers will be missed.



The text of program:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
const int N=10;
float a[N],b[N];
int  i,j=0, sum=0;
for (i=0; i<N; i++)
  {a[i]=StrToInt(SG1->Cells[i][0]);
   b[i]=0;                  //elements of the new array are at first 0
  }
for (i=0; i<N; i++)
  if (a[i]!=0) {
     b[j]=a[i];             //if element a[i] is non-zero, we assign it to b[j] and increase j
     j++;}
SG2->ColCount=j;    //now we know the real number of cells in SG2
for (i=0; i<j; i++)
  SG2->Cells[i][0]=FormatFloat("0.0",b[i]);
}
```

Individual tasks

Table 2.4

№	Array size	Element type	Individual task

№	Array size	Element type	Individual task
1	15	integer	Calculate the sum S of positive odd elements of array and replace the odd elements with S.
2	10	floating point	Replace minimal element end penultimate (предпоследний) element
3	12	integer	Calculate the factorial of the first element of array, which value is less than 8.
4	8	floating point	Define, is array sorted in ascending order.
5	14	integer	Calculate the new array as difference of the elements of first array and its average.
6	18	floating point	Calculate universal element (maximum) and replace it with the last element.
7	11	integer	Sort array in ascending order and define index of element, which is equal F. Enter number F from the form.
8	14	floating point	Calculate percentage of negative, positive and zero elements in array.
9	7	integer	Calculate the product of odd elements and create the new array, which elements are equal to difference of first array's elements and calculated product
10	19	floating point	Calculate the number of elements, which are greater than average.
11	17	integer	Create a new array of the first, place all positive and zero an elements on the beginning of it, after them – negative elements, saving their order
12	9	floating point	Replace all negative elements with universal element (maximum).
13	15	integer	Calculate the minimum element and replace it with the first element.
14	10	floating point	Calculate maximum and minimum and replace them.
15	8	integer	Calculate the factorial of the maximum's index.
16	12	floating point	Calculate the sum of all elements from the beginning up to the first negative. If there is no negative number, output the message and calculate the sum of all elements.
17	20	integer	Calculate the sum of elements with even indexes and create a new array, which elements are differences of first array's elements and calculated sum.
18	18	floating point	Calculate the sum of elements, which value is greater than average.
19	11	integer	Calculate the minimum of the odd positive elements.
20	9	floating point	Calculate the maximum of the negative elements.
21	16	integer	Create the new array, which elements are remainders of division of the first array elements of first array by 3.

Table 2.5

№	Array size	Element type	Individual task
1	≤ 9	integer	Enter two arrays. Calculate the number of equal elements in them.
2	≤ 10	floating point	Enter two arrays. Create third array, which elements are from both arrays sorted in ascending order.
3	≤ 12	integer	Enter the array, which elements are 0, 1 or 2. Replace zero elements on the

№	Array size	Element type	Individual task
			beginning, then all elements with value=1 and then elements with value= 2.
4	≤ 8	floating point	Enter array and number C. Place elements, which are less than or equal C, in the beginning of array, then elements, which are greater than C, saving their order.
5	≤ 14	integer	Define the first number, which presents in each of three arrays. These arrays are sorted in ascending order.
6	≤ 7	floating point	Enter two arrays. Create the third array of common elements of two arrays.
7	≤ 11	integer	Enter the array, in which there only 2 equal elements. Find their indexes.
8	≤ 14	floating point	Enter the array and number L. Create the new array of elements which are less than L, and sort new array in descending order.
9	≤ 19	integer	Find the pair of neighbor elements, which value are nearest to each other (the value of $\|x_{i+1} - x_i\|$ is minimal)
10	≤ 12	floating point	Calculate the product P of negative numbers with even indexes, which are not greater than given number L, and divide all positive numbers by P
11	≤ 17	integer	Enter array. Create two new arrays: first of them consists of elements with odd indexes, second consists of elements which are multiple to 5
12	≤ 9	floating point	Enter the array and number P. Calculate the array element, which value is nearest to P (the value of $\|x_i - P\|$ is minimal)
13	≤ 15	integer	Sort the positive elements on increase and place them on the beginning of array. Sort the negative elements on decrease and place them in the end
14	≤ 10	floating point	Enter two arrays and calculate the quantity of unique elements.
5	≤ 8	integer	Enter two arrays and replace elements of the first array, which do not present in the second one, with 0
16	≤ 12	floating point	Calculate the sum S of negative elements, which are not greater than given number L, and divide the last positive element by S
17	≤ 20	integer	Enter the array and number K. Create two new arrays: the first consists of the elements, which are less than or equal K, and the second consists of elements, which are greater than 1, the second consists of elements, which are greater than K, saving their order
18	≤ 18	floating point	Enter array (which has a lot of zero numbers). Replace all the zero groups with the quantity of zeros in these groups.
19	≤ 11	integer	Calculate the sum S of the positive odd elements and replace all the even elements with S
20	≤ 9	floating point	Enter two arrays.
21	≤ 16	integer	Enter the array. Create the new array of the elements with negative and zero elements with even indexes
22	≤ 19	floating point	Enter two arrays. Calculate the minimum of the first array the maximum of the second array and replace them.

2.4 Multi-dimensional arrays.

A **two-dimensional array** is a collection of components, all of the same type, structured in two dimensions, (referred to as **rows** and **columns**). Individual components are accessed by a pair of indexes representing the component's position in each dimension.

The common convention is to treat the first index as denoting the row and the second as denoting the column.

Syntax:

DataType ArrayName [ConstIntExpr] [ConstIntExpr] ;

Examples:

```
int table1[5][3];       // 5 rows, 3 columns
                        // (row variable changes from 0 to 4, column variable changes from 0
                to 2)

float table2[3][4];     // 3 rows, 4 columns
                        // (row variable changes from 0 to 2, column variable changes from 0
                        to 3)

float hiTemp[52] [7]; // 52 rows, 7 columns
                        // (row variable changes from 0 to 51, column variable changes from
                0 to 6)
```

Laboratory work 2.7
Multi-dimensional arrays

The purpose: to get skills in operation with multi-dimensional arrays.

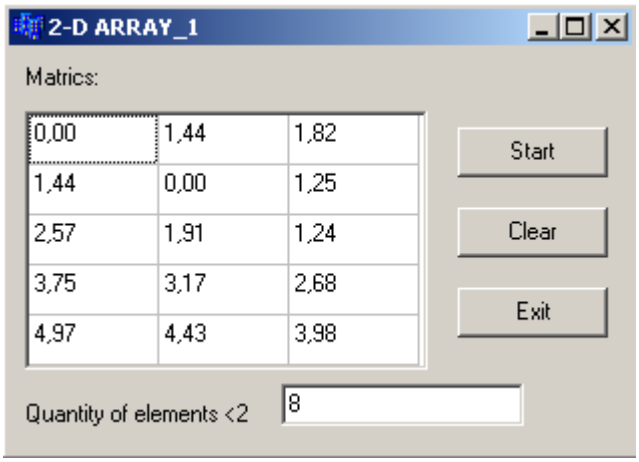
Examples:

Example 4.1 Fill the matrix 5x3 with numbers by formula:

$$a_{ij} = \frac{\sqrt{|i^3 - j^2|}}{\ln(i + j + 1)}.$$

Calculate the quantity of elements, which are less than 2.

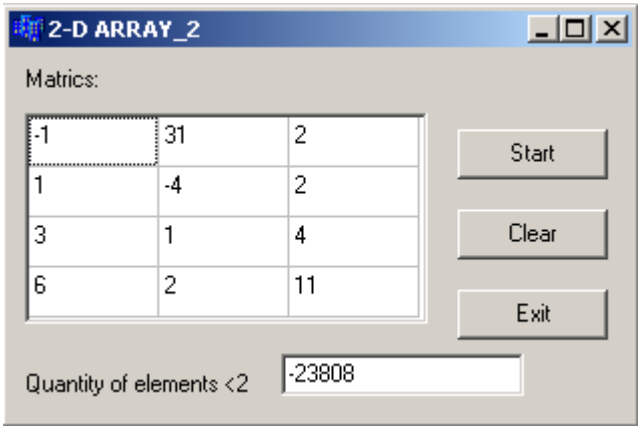
This formula can be calculated for all values of i and j except i=0 and j=0 simultaneously (let in this case a_{ij}=0).




```
#include <math.h>
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
double a[5][3];
int i, j, k=0;
//Calculation of matrix elements
for (i=0; i<5; i++)
  for (j=0; j<3; j++)
     if (i==0 && j==0) a[i][j]=0;
     else a[i][j]= sqrt(fabs(pow(i,3)-j*j))/log(i+j+1);
//Output the elements
for (i=0; i<5; i++)
  for (j=0; j<3; j++)
     SG1->Cells[j][i]=FormatFloat("0.00",a[i][j]);
//Calculation of quantity of elements <2
for (i=0; i<5; i++)
  for (j=0; j<3; j++)
     if (a[i][j]<2) k++;
Edit1->Text=IntToStr(k);
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button2Click(TObject *Sender)
{
for (int i=0; i<5; i++)
```

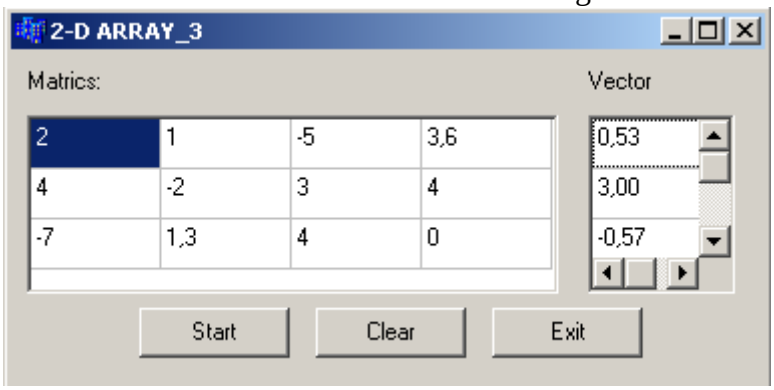
```
  for (int j=0; j<3; j++)
    SG1->Cells[j][i]="";
Edit1->Clear();
}
//---------------------------------------------------------------------------
```

Example 4.2 Enter an integer matrix 4x3. Calculate the product of elements which are multiple at least to one of its indexes.




```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
int a[4][3];
int i, j, k=0;
long p=1;
//Input matrix elements
for (i=0; i<4; i++)
  for (j=0; j<3; j++)
     a[i][j]=StrToInt(SG1->Cells[j][i]);
//Calculate the product
for (i=0; i<4; i++)
  for (j=0; j<3; j++)
     if (i!=0 && a[i][j]%i==0 || j!=0 && a[i][j]%j==0)
        {p*=a[i][j];
         k++;}
if (k>0) Edit1->Text=IntToStr(p);
else ShowMessage ("No numbers");
}
```

Example 4.3 Enter float matrix 3x4. Create vector of the average of row elements.




```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
float a[3][4],s, b[3];
```

```
int i, j, k;
long p=1;
//Input matrix elements
for (i=0; i<3; i++)
  for (j=0; j<4; j++)
     a[i][j]=StrToFloat(SG1->Cells[j][i]);
//Calculate the averages and assign them to elements of new array
for (i=0; i<3; i++)
{ s=0;
  for (j=0; j<4; j++)
    s+=a[i][j];
  b[i]=s/3;
}
//Output new array
for(i=0; i<3; i++)
  SG2->Cells[0][i]=FormatFloat("0.00",b[i]);
}
```

Example 4.4 There is a branch of mathematics that deals with two-dimensional arrays. In maths, however, they are called **matrices**. Matrices are written using large brackets, like this:

$$\begin{pmatrix} 24 & 17 & -6 \\ 5 & -10 & 11 \end{pmatrix} + \begin{pmatrix} -9 & 12 & 6 \\ -14 & 37 & 20 \end{pmatrix} = \begin{pmatrix} 15 & 29 & 0 \\ -9 & 27 & 31 \end{pmatrix}$$

This example shows two matrices being added together to produce a third. The trick is to add the corresponding elements of each array (matrix).

Multiplying matrices by a constant number is equally easy. You just multiply each element of the matrix by the constant number:

$$6 \times \begin{pmatrix} 3 & 4 & 1 \\ 2 & 9 & 7 \end{pmatrix} = \begin{pmatrix} 18 & 24 & 6 \\ 12 & 54 & 42 \end{pmatrix}$$

Multiplying one matrix by another is not just a case of multiplying the corresponding elements. Each element of the result comes from a complicated formula:

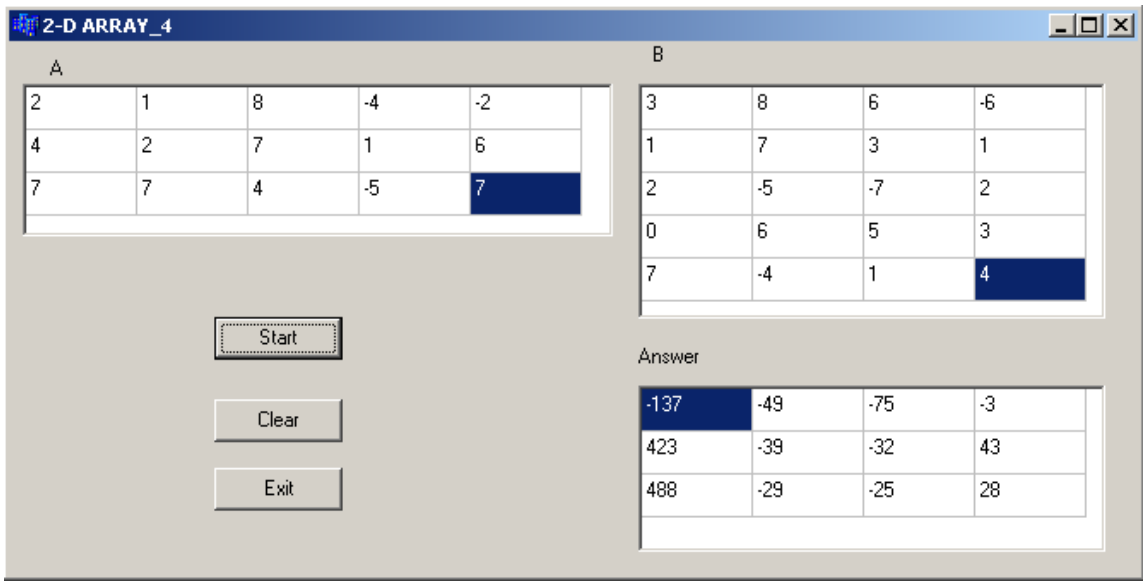
$$\begin{pmatrix} 3 & 4 & 7 & 2 \\ 5 & 1 & 2 & 0 \end{pmatrix} \times \begin{pmatrix} 1 & 2 & 8 \\ 3 & 0 & 1 \\ 5 & 1 & 4 \\ 1 & 2 & 7 \end{pmatrix} = \begin{pmatrix} . & 17 & . \\ . & . & . \end{pmatrix}$$

$3 \times 2 + 4 \times 0 + 7 \times 1 + 2 \times 2$

The diagram above shows how the second element in the top row is calculated. You have to work along the top row of the first matrix (top row of the first matrix to correspond to the top row of the answer matrix) and the whole of the second column of the second matrix (to correspond to the second column of the answer)

Take each of the elements in the row and column in turn and multiply them in pairs. These multiplied pairs are then added to give the single value that goes in that position in the answer, so you multiply the first two, then add the product of the second two etc.

You will see that, unlike previous matrix operations, the answer doesn't need to be the same shape as either of the two matrices that combine to make it. When you multiply a matrix with 2 rows and 4 columns (a 4-by-2 matrix) by one with 4 rows and 3 columns (a 3-by-4 matrix) you get an answer with 2 rows and 3 columns.



The program code

```
{
  int a[3][5];
  int b[5][4];
  int answer[3][4];

  int i,j,k;    // Loop counters in FOR loops
  int temp;     // Used to build answer

  // Read values for first array
for (i = 1; i < 3; i++)
    for (j = 1; j <5; j++)
        a[i][j]=StrToInt(SG1->Cells[j][i]);
  // Read values for second array
for (i = 1; i < 5; i++)
    for (j = 1; j <4; j++)
        b[i][j]=StrToInt(SG2->Cells[j][i]);

 //Calculate answers in answer array
  for (i = 1; i <3; i++)
    for (j = 1; j <4; j++)
      {
        temp = 0;
        for (k = 1; k <5; k++)
          temp += a[i][k] * b[k][j];
        answer[i][j] = temp;
      }
  //Now display the answers
  for (i = 1; i <3; i++)
    for (j = 1; j <4; j++)
      SG3->Cells[j][i]= IntToStr(answer[i][j]);
```

}

Individual tasks:

Table 2.6 Individual tasks with low level of complexity.

№ var	Type of elements	Dimension	Individual task
1	float	5x4	Calculate the quantity of positive, negative and zero elements.
2	int	4×5	Calculate the order number of the row, which contains the maximum element.
3	int	5×5	Calculate minimum element of the main diagonal and the order number of row, which contains this element.
4	float	4×4	Replace elements of the first row with elements of its secondary diagonal
5	int	3×5	Change all negative numbers with 0 and calculate the number of exchanges.
6	float	6×6	Replace maximum and minimum elements.
7	int	4×6	Calculate the biggest negative element with even columns
8	float	5×5	Calculate the sum of secondary diagonal elements
9	int	5×5	Transpose the matrix
10	float	3×6	Calculate the column number with the minimum element
11	int	5×5	Calculate the minimum element of the secondary diagonal and its column number
12	int	7×3	Calculate the vector of sums of row elements
13	float	7×4	Change in odd rows all negative elements with 0, and positive elements – with 1
14	float	5×5	Calculate the difference between the sum of main diagonal elements and secondary diagonal elements
15	float	7×5	Calculate the sum of negative elements of first 4 rows
16	int	4×5	Change all negative numbers with their absolute values
17	int	4×5	Calculate the product of minimum element and average of all elements
18	int	6×4	Calculate the average of positive elements
19	int	5×4	Change positive elements in even rows with 1 and negative elements in odd rows with -1
20	float	6×3	Calculate the product of negative elements in even rows
21	int	3×5	Calculate the quantity of elements which are less than average
22	float	5×3	Change all elements which are greater than 2.5 with -1 and calculate the quantity of exchanges
23	float	4×5	Calculate the vector of absolute values of row elements sums
24	int	7×4	Calculate the minimum positive element and maximum of negative elements and replace them

Table 2.7 Individual tasks with medium level of complexity.

№ var	Individual task
1	Create vector of sums of odd columns elements. Matrix of 3×7 int numbers.
2	Create vector of scalar product of float matrix 4×4 on its last column.
3	Create vector of products of even rows odd elements. Matrix of 6×4 int numbers.
4	of scalar product of first row elements on columns of matrix. Matrix of 4×4 int numbers
5	Find number of column with minimum sum of elements. Matrix of 4×5 float numbers.
6	Create vector of products of columns elements. Matrix of 3×6 int numbers.
7	Create vector of products of odd columns even elements. Matrix of 4×5 int numbers.

№ var	Individual task
8	Define number of row with maximum elements sum. Matrix of 4×5 int numbers.
9	Create vector of even rows odd elements sums. Matrix of 6×6 int numbers.
10	Create vector of scalar products of columns elements on the main diagonal. Matrix of 3×3 float numbers.
11	Create vector of columns minimum elements. Matrix of 4×6 float numbers.
12	Find number of string with minimum sum of elements absolute values. Matrix of 7×3 float numbers.
13	Change main diagonal elements with sums of columns elements. Matrix of 5×5 int numbers.
14	Create vector of averages of even rows positive elements. Matrix of 7×8 int numbers.
15	Create vector of maximum elements of rows. Matrix of 7×6 float numbers.
16	Change secondary diagonal elements with sums of row elements. Matrix of 4×4 int numbers.
17	Create vector of scalar products of rows on the main diagonal. Matrix of 5×5 int numbers.
18	Create vector of the row, which contains the maximum element. Matrix of 5×6 int numbers.
19	Change main diagonal elements with sum of maximum and minimum elements. Matrix of 6×6 int numbers.
20	Change secondary diagonal elements with maximum element. Matrix of 4×4 int numbers.
21	Change zero elements with maximum element. Matrix of 5×5 float numbers.
22	Replace the first and the last negative elements. Matrix of 7×3 float numbers.
23	Create vector of positive elements sums of rows. Sort this vector on ascending. Matrix of 6×5 int numbers.
24	Find the row, for which sum of odd elements is minimal, and create vector of this row. Matrix of 5×4 int numbers.

Table 2.8 Individual tasks with high level of complexity.

№ var	Individual tasks
1	Matrix of 5×7 float numbers. Create new matrix, in which each element is calculated as a half of sum of according row and column averages.
2	Matrix of 7×7 float numbers. Calculate determinant of matrix.
3	Matrix of 5×5 float numbers. Create vector of maximum elements of rows.
4	Matrix of 5×5 int numbers. Calculate the product of matrix together with its transposed matrix.
5	Matrix of 7×3 float numbers. Find number of row, which length is maximum.
6	Matrix of 8×3 float numbers. Create vector of row, which distance to the second row is minimal. Distance is calculated be formula: $d_i = \sum_{j=1}^{n} \left| a_{ij} + a_{2j} \right|$
7	Matrix of 4×4 float numbers. Sort in ascending order (from left to right) row elements, then in ascending order (from top to bottom) – column elements.
8	Matrix of 5×4 float numbers. Change the row with minimal length with row with maximum length row.
9	Matrix of 5×5 float numbers. maximal distance from the first row. Distance from the first row to *i-th* : $d_i = \sum_{j=1}^{n} \left| a_{ij} \right| + \left| a_{1j} \right|, \ i \neq 1$
10	Matrix of 6×4 float numbers. Create vector of the row with minimum sum $\sum_{j=1}^{n} a_{ij}^2$.
11	Matrix of 3×7 float numbers. Create vector of the column with maximum weight, column

	weight is: $W_j = \sum_{i=1}^{n} \lvert a_{ij} \rvert$
12	Matrix of 8×6 float numbers. Create vector of the column with maximum sum: $\sum_{j=1}^{n} \lvert a_{ij} \rvert + a_{ij}$
13	Matrix of 4×8 float numbers. Each negative element change with sum of positive elements those rose, where this element is.
14	Matrix of 6×8 float numbers. Create vector of the column with maximum distance from the first row. Distance is: $d_j = \sum_{i=1}^{n} \lvert a_{ij} \rvert \cdot \lvert a_{i1} \rvert, \quad j \neq 1$
15	Matrix of 5×3 float numbers. Create vector of the row with minimum weight, weight is: $W_i = \sum_{j=1}^{n} \lvert a_{ij} \rvert$
16	Enter two matrices of 4×5 float numbers. If maximum element of the first row of the first matrix is smaller than maximum element of the second row of the second matrix, then replace rows, which contain maximum elements. Otherwise, output the message.
17	Matrix of 4×8 float numbers. Create vector of positive numbers.
18	Find minimum element in the sector above the main diagonal and minimum element under the main diagonal. Change elements of main diagonal with bigger of these two numbers.
19	Matrix of 6×5 int numbers. Matrix consists of numbers from 0 to 9. If number of element's repetitions is equal to its value than change it with 0.
20	Matrix of 7×5 int numbers. Create vector from the row, which distance from the third row is maximum. Distance is $d_j = \sum_{i=1}^{n} \lvert a_{ij} \rvert + \lvert a_{3,j} \rvert$
21	Matrix of 3×6 float numbers. Create vector with minimum value of: $W_j = \sqrt{\sum_{i=1}^{n} a_{ij}^2}$
22	Matrix of 3×5 int numbers. Matrix consists of numbers from 0 to 9. Calculate the percentage of each of these numbers in matrix.
23	Matrix of 6×4 int numbers. Matrix consists of numbers from 0 to 9. Create vector with length 10, which elements are numbers of repetitions these constants in matrix.
24	Matrix of 6×5 float numbers. Create vector from the column, which distance to (n – 1)-th columns maximum. Distance is: $d_j = \sum_{i=1}^{n} \lvert a_{ij} \rvert + \lvert a_{i,n-1} \rvert$.