Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»
Кафедра информационных систем и технологий

И. Н. Коренская, А. А. Навроцкий, Е.В. Николаенко

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ В СРЕДЕ VISUAL C++

Задания к контрольной работе по курсу
«Основы алгоритмизации и программирования»
для студентов 1 курса заочной формы обучения
всех специальностей БГУИР ИИТ

Минск БГУИР 2010

УДК
ББК

Р е ц е н з е н т ы:

Василенко Ж. В., доцент кафедры «Технологии программирования» факультета прикладной математики и информатики Учреждения образования Белорусский государственный университет, кандидат технических наук, доцент

Шакирин А.И., доцент кафедры «Вычислительная техника» Учреждения образования Белорусский государственный аграрный университет, кандидат технических наук, доцент

Коренская, И. Н.
Основы алгоритмизации и программирования в среде Visual C++: задания к контрольной работе по курсу «Основы алгоритмизации и программирования» для студ. 1–го курсов всех спец. БГУИР / Коренская И. Н. , А. А. Навроцкий, Е.В. Николаенко – Минск : БГУИР, 2010. – 54 с.: ил. 12

ISBN - - -

Приведено 6 заданий по контрольной работе на языке C++ в среде Microsoft Visual Studio с примерами выполнения; представлены индивидуальные задания; дана справочная информация.

УДК
ББК

ISBN -- -

СОДЕРЖАНИЕ

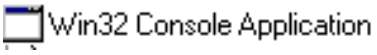
ЗАДАНИЕ №1
СРЕДА ПРОГРАММИРОВАНИЯ VISUAL C++. ПРОГРАММИРОВАНИЕ ЛИНЕЙНЫХ АЛГОРИТМОВ

1.1. Консольный режим работы среды Visual C++ 6.0

Программа, создаваемая в среде Visual C++, всегда оформляется в виде отдельного проекта. Проект (*project*) – это набор взаимосвязанных исходных файлов, предназначенных для решения определенной задачи, компиляция и компоновка которых позволяет получить выполняемую программу. В проект входят как файлы, непосредственно создаваемые программистом, так и файлы, которые автоматически создает и редактирует среда программирования.

Для **создания нового проекта** необходимо:

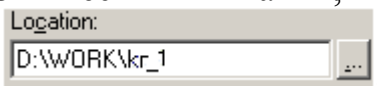
• выбрать в главном меню **File – New**;

• в открывшемся окне (закладка **Projects)** выбрать тип создаваемого проекта 

;

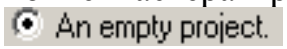
• в поле **Project Name** ввести имя проекта 

;

• в поле **Location** ввести имя папки, в которой будет размещен проект, и полный путь к ней 

. Папку также можно выбрать, используя диалоговое окно **Choose Directory**, для чего надо щелкнуть мышью по кнопке  ;

• щелкнуть мышью по кнопке **ОК;**

• в открывшемся окне мастера приложений **Win32 Console Application – Step 1 of 1** выбрать 

(пустой проект) и щелкнуть по кнопке 

;

• в открывшемся окне **New Project Information** (информация о новом проекте) щелкнуть мышью по кнопке **ОК.**

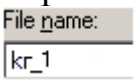
Для работы с консольным приложением необходимо создать **новый** или добавить **существующий** файл с кодом программы.

Для **создания нового файла** необходимо:

• выбрать **File – New**;

• в открывшемся окне (закладка **Files)** выбрать тип файла 

;

• в поле **File name** ввести имя файла: 

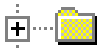
(желательно, чтобы вводимое имя совпадало с именем проекта);

• щелкнуть мышью по кнопке **ОК.**

Для **добавления в проект существующего файла** с кодом программы необходимо:

• скопировать имеющийся файл (расширение **cpp**) в рабочую папку проекта;

• в окне **Workspace** (закладка 

) щелкнуть правой кнопкой мыши по папке 

;

• в раскрывшемся меню выбрать пункт 

, в диалоговом окне 

указать добавляемый файл и щелкнуть мышью по кнопке **ОК.**

В папке проекта, как правило, размещено несколько файлов и одна вложенная папка. **Файлы имеют** следующее **назначение**:

- файл с расширением *.**dsw** (например, *my_lab1.dsw*) – файл проекта, который объединяет все входящие в проект файлы;
- файл с расширением *.**cpp** (например, *my_lab1.cpp*) – файл кода программы.

Для компиляции, компоновки и запуска программы на выполнение используются следующие пункты подменю **Build**:

Compile (Ctrl+F7)  – *компиляция* выбранного файла. Результаты компиляции выводятся в окно **Output.**

Build (F7)  – *компоновка* проекта. Компилируются все файлы, в которых произошли изменения с момента последней компоновки.

Rebuild All – *перекомпоновка* проекта. Компилируются все файлы проекта независимо от того, были ли в них произведены изменения или нет.

Execute (Ctrl+F5)  – *выполнение* исполняемого файла, созданного в результате компоновки проекта. Для файлов, в которые были внесены изменения, выполняется перекомпилирование и перекомпоновка.

Если в процессе компиляции были обнаружены синтаксические ошибки, то выводится соответствующее сообщение. В этом случае необходимо последовательно исправить все ошибки и компилировать проект снова. Если синтаксических ошибок нет, но результат выполнения программы неверный, то необходимо искать логические ошибки. Для этого следует использовать встроенный в систему отладчик (см. лабораторную работу №3 пункт **3.8 Отладка программы**).

Для открытия сохраненного ранее проекта необходимо выбрать в меню **File – Open Workspace**... В открывшемся диалоговом окне выбрать папку проекта и открыть в ней файл с расширением **dsw.**

1.2. Функции библиотеки math.lib

Функции для расчета математических выражений находятся в библиотеке **math**.lib (подключение библиотеки: **#include <math.h>**). Все аргументы в тригонометрических функциях задаются в *радианах*. Для преобразования величины угла из градусов в радианы используется формула $\frac{a \cdot \pi}{180}$, где a — величина угла в градусах, а $\pi \approx 3{,}1415$.

Параметры и аргументы всех остальных функций имеют тип **double** (кроме abs(x)).

Математическая функция	**Функция библиотеки math.lib**	**Описание**
$\|x\|$	**abs(x)**	Вычисление абсолютного значения (**только для целых чисел!**)
$\|x\|$	**fabs(x)**	Вычисление абсолютного значения *x*
$\sqrt{x}$	**sqrt(x)**	Вычисление квадратного корня *x*
x^y	**pow(x, y)**	Возведение *x* в степень *y*
sin(*x*)	**sin(x)**	Вычисление синуса *x*
sh(*x*)= $(e^x–e^{-x})/2$	**sinh(x)**	Вычисление синуса гиперболического *x*
cos(*x*)	**cos(x)**	Вычисление косинуса *x*
ch(x)= $(e^x+e^{-x})/2$	**cosh(x)**	Вычисление косинуса гиперболического *x*
tg(*x*)	**tan(x)**	Вычисление тангенса *x*
tgh(*x*)	**tanh(x)**	Вычисление тангенса гиперболического *x*
arccos(*x*)	**acos(x)**	Вычисление значения арккосинуса *x*
arctg(*x*)	**atan(x)**	Вычисление значения арктангенса *x*
arctg(*x*/*y*)	**atan2(x,y)**	Вычисление значения арктангенса двух аргументов *x* и *y*
e^x	**exp(x)**	Вычисление экспоненты числа *x*
ln(*x*)	**log(x)**	Вычисление натурального логарифма *x*
$lg_{10}(x)$	**log10(x)**	Вычисление десятичного логарифма *x*
Округление к большему	**ceil(x)**	Функция возвращает действительное значение, соответствующее наименьшему целому числу, которое больше или равно *x*
Округление к меньшему	**floor(x)**	Функция возвращает действительное значение, соответствующее наибольшему целому числу, которое меньше или равно *x*
Остаток от деления *x* на *y*	**fmod(x,y)**	Функция возвращает действительное значение, соответствующее остатку от целочисленного деления *x* на *y*

Например:

```
double x, y,
double z = pow (sin (fabs(x)),2);                    // z = sin²|x|
double z1 = exp(1)+exp(x*x)+exp(2*pow(x,3));         // z1 = e +e^x² +e^2x³
double z2 = pow (x, pow (y, 4+pow (x, 1/4.)));       // z2 = x^y^(4+4√x)
```

```
cout<<"fmod( 5, 2)="<<fmod( 5, 2)<<endl;
cout<<"ceil (5.6)="<<ceil (5.6)<<endl;
cout<<"ceil (5.1)="<<ceil (5.1)<<endl;
cout<<"ceil (-5.1)="<<ceil (-5.1)<<endl;
cout<<"ceil (-5.8)="<<ceil (-5.8)<<endl;
cout<<"floor (5.1)="<<floor (5.1)<<endl;
cout<<"floor (5.6)="<<floor (5.6)<<endl;
cout<<"floor (-5.6)="<<floor (-5.6)<<endl;
cout<<"floor (-5.1)="<<floor (-5.1)<<endl;
```

```
fmod( 5, 2)=1
ceil (5.6)=6
ceil (5.1)=6
ceil (-5.1)=-5
ceil (-5.8)=-5
floor (5.1)=5
floor (5.6)=5
floor (-5.6)=-6
floor (-5.1)=-6
Press any key to continue
```

Замечания:

1. Для ввода значений переменных **x, y** и **z** необходимо набрать с клавиатуры:

2.45 ***(x = 2,45)***

-0.423e-2 ***(y = –0,423·10^{-2})***

1.232e3 ***(z = 1,232·10^{3})***

2. В языке С++ при вычислении арифметических выражений происходит *автоматическое приведение типов*, следовательно, при делении целого значения на целое, результат будет *целым числом*. Например, при вычислениии "**1/3**" результат будет равен нулю, так как целая часть вычисленного выражения равна нулю. Для получения результата, имеющего дробную часть, необходимо, чтобы один из операндов имел действительный тип. Для этого можно использовать функцию *явного приведения типа*, а для констант достаточно *поставить точку после числа*, например: "**1/3.**", или "**1./3.**", или "**1./3**".

Например:

```
int s, n;
double sr = static_cast<double> (s) / n;    // явное приведение типа
double y = pow (x, 3/4.);                   // y=∜(x^3) = x^(3/4)
```

3. Язык С чувствителен к регистру букв, т.е. прописные и строчные буквы воспринимаются как *разные* символы.

Например: **count**, **Count**, **COUNT** – разные идентификаторы.

4. При выводе информации для перехода на новую строку применяется манипулятор (функция управления выводом) **endl** или '**\n**', для выравнивания выводимой информации – '**\t**' (вставляет символ табуляции).

5. Главная функция **int main ()** автоматически вызывается при запуске программы и возвращает операционной системе по окончании значение **0** (**return 0;**).

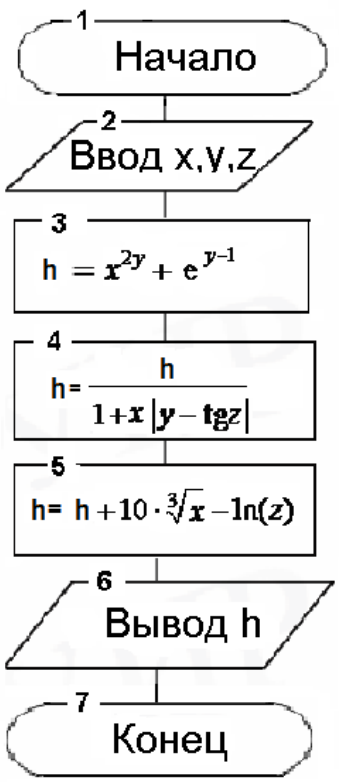
1.3. Пример выполнения задания

Составить программу вычисления линейного арифметического выражения

$$h = \frac{x^{2y} + e^{y-1}}{1 + x\left|y - \mathrm{tg}z\right|} + 10 \cdot \sqrt[3]{x} - \ln(z).$$

При x = 2,45; y = – 0,423·10^{-2}; z = 1,232·10^{3} **ответ:** h = 6,9465.

Блок-схема алгоритма



Код программы

```
#include <iostream.h>
#include <math.h>

int main ()
    {                                          // Начало главной функции
     double x, y, z, h;                        // Объявление переменных

     cout << "Vvedite x:  ";                   // Ввод значений x, y и z
     cin >> x;
     cout << "Vvedite y:  ";
     cin >> y;
     cout << "Vvedite z:  ";
     cin >> z;

     h = pow(x, 2*y) + exp(y-1);               // Вычисление выражения
     h / = 1+x * fabs(y - tan(z));
     h + = 10 * pow(x, 1/3.) - log(z);
     cout << "Result h= " << h << endl;        // Вывод результата
     return 0;                                 // Завершение выполнения программы
    }                                          // Конец главной функции
```

Ход выполнения работы

1. В окне редактирования (рис. 1) наберите код программы, приведенный выше.

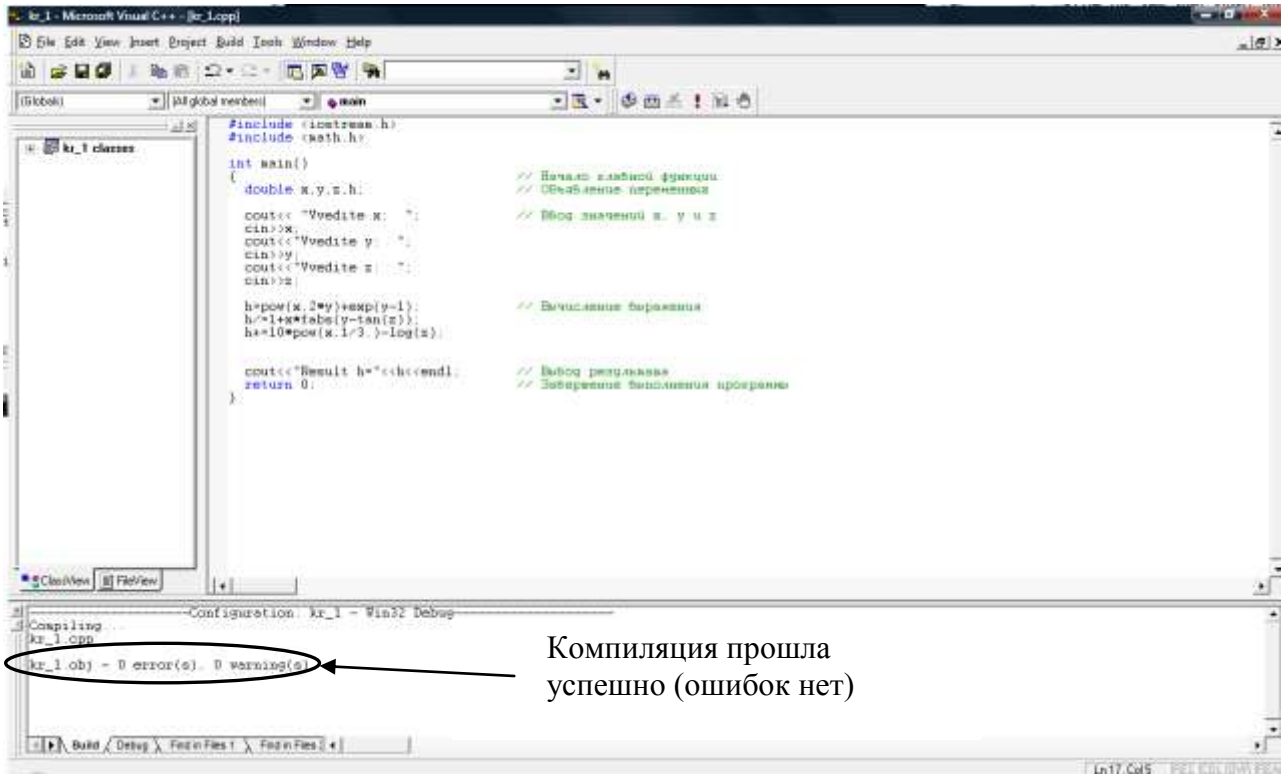


Рисунок 1 – Компиляция программы

2. Выполните *компиляцию* набранного файла.

Для этого нажмите **Ctrl+F7** или пиктограмму **.** Результат компиляции выведется в окно **Output** (рис. 1). При наличии в программе ошибок исправьте их и заново повторите компиляцию.

3. Выполните исполняемый файл.

Для этого нажмите **Ctrl+F5** или пиктограмму **.**

4. В появившемся окне введите с клавиатуры значения переменных x, y и z (рис. 2):

2.45	*(значение x)*
– 0.423e-2	*(значение* y*)*
1.232e3	*(значение* z*)*

После ввода каждого значения нажимайте клавишу Enter.

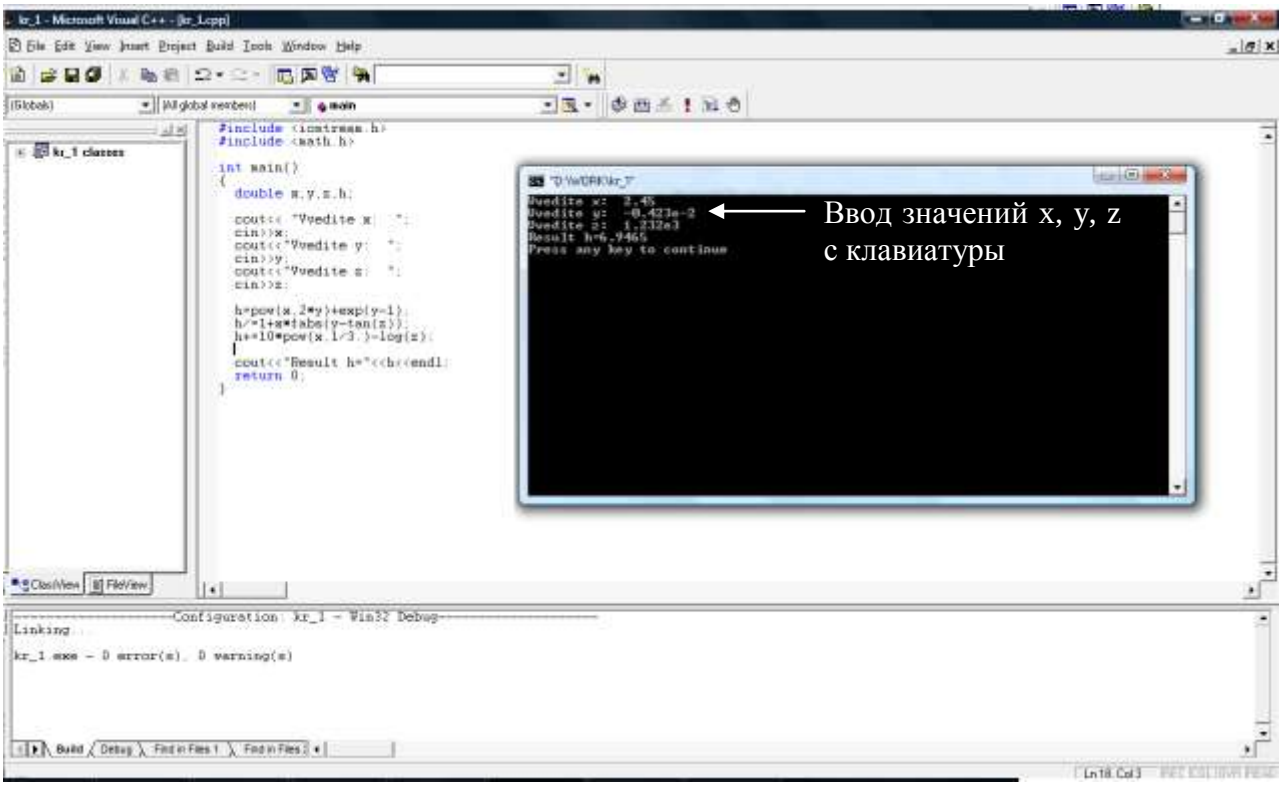


Рисунок 2 – Результат выполнения программы

1.4. Индивидуальные задания

Составить согласно индивидуальному варианту блок-схему алгоритма и программу вычисления значения выражения при заданных исходных данных. Сравнить полученное значение с указанным правильным результатом.

1. $s = \dfrac{2\cos\left(x - \dfrac{2}{3}\right)}{\dfrac{1}{2} + \sin^2 y}\left(1 + \dfrac{z^2}{3 - z^2/5}\right)$

при **x** = 14,26; **y** = –1,22; **z** = $3{,}5 \cdot 10^{-2}$. **Ответ s** = 0,749155.

2. $s = \dfrac{\sqrt[3]{9 + (x-y)^2}}{x^2 + y^2 + 2} - e^{|x-y|}\mathrm{tg}^3 z$

при **x** = –4,5; **y** = $0{,}75 \cdot 10^{-4}$; **z** = $-0{,}845 \cdot 10^{2}$. **Ответ s** = –3,23765.

3. $s = \dfrac{1 + \sin^2(x+y)}{\left|x - \dfrac{2y}{1 + x^2 y^2}\right|} x^{|y|} + \cos^2\left(\mathrm{arctg}\dfrac{1}{z}\right)$

при **x** = $3{,}74 \cdot 10^{-2}$; **y** = –0,825; **z** = $0{,}16 \cdot 10^{2}$. **Ответ s** = 1,05534.

4. $s=\left|\cos x-\cos y\right|^{\left(1+2\sin^2 y\right)}\left(1+z+\frac{z^2}{2}+\frac{z^3}{3}+\frac{z^4}{4}\right)$

при **x** = $0{,}4\cdot10^4$; **y** = –0,875; **z** = $-0{,}475\cdot10^{-3}$. **Ответ s** = 1,98727.

5. $s=\ln\left(y^{-\sqrt{|x|}}\right)\left(x-\frac{y}{2}\right)+\sin^2(\operatorname{arctg}(z))$

при **x** = –15,246; **y** = $4{,}642\cdot10^{-2}$; **z** = 21. **Ответ s** =–182,038.

6. $s=\sqrt{10\left(\sqrt[3]{x}+x^{y+2}\right)}\left(\arcsin^2 z-\left|x-y\right|\right)$

при **x** = $16{,}55\cdot10^{-3}$;**y** = –2,75; **z** = 0,15. **Ответ s** = –40,6307.

7. $s=5\operatorname{arctg}(x)-\frac{1}{4}\arccos(x)\frac{x+3\left|x-y\right|+x^2}{\left|x-y\right|z+x^2}$

при **x** = 0,1722; **y** = 6,33; **z** = $3{,}25\cdot10^{-4}$. **Ответ s** = –205,306.

8. $s=\frac{e^{|x-y|}\left|x-y\right|^{x+y}}{\operatorname{arctg}(x)+\operatorname{arctg}(z)}+\sqrt[3]{x^6+\ln^2 y}$

при **x** = $-2{,}235\cdot10^{-2}$; **y** = 2,23; **z** = 15,221. **Ответ s** = 39,3741.

9. $s=\left|x^{\frac{y}{x}}-\sqrt[3]{\frac{y}{x}}\right|+(y-x)\frac{\cos y-\frac{z}{(y-x)}}{1+(y-x)^2}$

при **x** = $1{,}825\cdot10^2$; **y** = 18,225; **z** = $-3{,}298\cdot10^{-2}$. **Ответ s** = 1,21308.

10. $s=2^{-x}\sqrt{x+\sqrt[4]{|y|}}\sqrt[3]{e^{x-1/\sin z}}$

при **x** = $3{,}981\cdot10^{-2}$; **y** = $-1{,}625\cdot10^3$; **z** = 0,512. **Ответ s** = 1,26185.

11. $s=y^{\sqrt[3]{|x|}}+\frac{\cos^3(y)}{e^{|x-y|}+\frac{x}{2}}\cdot\left|x-y\right|\left(1+\frac{\sin^2 z}{\sqrt{x+y}}\right)$

при **x** = 6,251; **y** = 0,827; **z** = 25,001. **Ответ** s = 0,712122.

12. $s=2^{\left(y^x\right)}+\left(3^x\right)^y-\frac{y\left(\operatorname{arctg}z-\frac{1}{3}\right)}{|x|+\frac{1}{y^2+1}}$

при **x** = 3,251; **y** = 0,325; **z** = $0{,}466\cdot10^{-4}$. **Ответ s** = 4,23655.

13. $s=\frac{\sqrt[4]{y+\sqrt[3]{x-1}}}{\left|x-y\right|\left(\sin^2 z+\operatorname{tg}z\right)}$

при **x** = 17,421; **y** = $10{,}365\cdot10^{-3}$; **z** = $0{,}828\cdot10^5$. **Ответ s** = 0,330564.

14. $s = \dfrac{y^{x+1}}{\sqrt[3]{|y-2|}+3} + \dfrac{x+\dfrac{y}{2}}{2|x+y|}(x+1)^{-1/\sin z}$

при **x** = $12{,}3\cdot10^{-1}$; **y** = 15,4; **z** = $0{,}252\cdot10^{3}$. **Ответ s** = 82,8256.

15. $s = \dfrac{x^{y+1}+e^{y-1}}{1+x|y-\mathrm{tg}z|}(1+|y-x|) + \dfrac{|y-x|^2}{2} - \dfrac{|y-x|^3}{3}$

при **x** = 2,444; **y** = $0{,}869\cdot10^{-2}$; **z** = $-0{,}13\cdot10^{3}$. **Ответ s** = –0,498707.

ЗАДАНИЕ №2
ПРОГРАММИРОВАНИЕ РАЗВЕТВЛЯЮЩИХСЯ АЛГОРИТМОВ

2.1. Логические операции и операции сравнения

Операции сравнения применяются при работе с *двумя* операндами и возвращают **true** (**1**), если результат сравнения – истина, и **false** (**0**), если результат сравнения – ложь. В языке С определены следующие *операции сравнения*: **<** (меньше**), <=** (меньше или равно), **>** (больше), **>=** (больше или равно), **!=** (не равно), **==** (равно).

Логические операции работают с операндами скалярных типов и возвращают результат *булева* типа. Существует три *логические операции*:

! – отрицание или логическое **НЕ**;

&& – логическое **И**;

|| – логическое **ИЛИ**.

Например:

$0 \le x \le 10$ => `0<=x && x<=10`

$x \ne y$ или $z = x$ => `x!=y || z==x`

2.2. Оператор условной передачи управления if

Форматы оператора **if**:

1. Полная форма: **if (***логическое_выражение***)** *оператор_1*;
else *оператор_2*;

Если *логическое выражение* истинно, то выполняется *оператор_1*, иначе – *оператор_2*.

2. Сокращенная форма: **if (***логическое_выражение***)** *оператор_1*;

Если *логическое_выражение* истинно, то выполняется *оператор_1*.

3. Вложенная форма:

if (*логическое_выражение_1***)** *оператор_1*;
else if (*логическое_выражение_2***)** *оператор_2*;
else *оператор_3*;

Если *логическое_выражение_1* истинно, то выполняется *оператор_1*, иначе, если *логическое_выражение_2* истинно, то выполняется *оператор_2*, иначе выполняется *оператор_3*.

Примеры:

1. найти максимальное значение из x, y, z (max(x,y,z)):

```
if (x>y) max=x;
    else max=y;
if (z>max) max=z;
```

2. *Вычислить выражение:*

$$s=\begin{cases}\ln(x)+\sqrt[3]{|y|}, & x/y>0\\ \ln|x/y|\cdot(x+y)^3, & x/y<0\\ (x^2+y)^3, & \text{иначе}\end{cases}$$

Так как на ввод значения переменной y по условию примера не предусмотрено никаких ограничений, то при вводе значения **y=0** может возникнуть ситуация **деление на 0**, поэтому следует записывать оператор **if**, начиная с ветви **иначе**:

```
if (!x || !y)    s =pow(pow(x,2)+y,3);        // x=0 или y=0 - иначе
      else if (x/y >0)    s=log(x)+ pow(fabs(y),1./3);    // x/y>0
              else s= log(fabs(x/y))* pow(x+y,3);       // x/y<0
```

2.3. Оператор множественного выбора switch

Общая форма оператора:

```
switch (переменная_выбора)
 {        case const_1: операторы_1;   break;
                                …
          case const_N: операторы_N;   break;
          default:      операторы_N+1;
 }
```

переменная_выбора, **const_1,…, const_N** – константа, переменная или выражение *целого*, *символьного* или *логического* типа.

При использовании оператора **switch** сначала анализируется *переменная_выбора* и проверяется, совпадает ли её значение со значением одной из констант **const_1, …, const_N**. При совпадении выполняются операторы этого **case**. Конструкция **default** (может отсутствовать) выполняется, если результат выражения не совпал ни с одной из констант.

Например: выбор функции f(x): x^2, |x|, $\sqrt{x}$

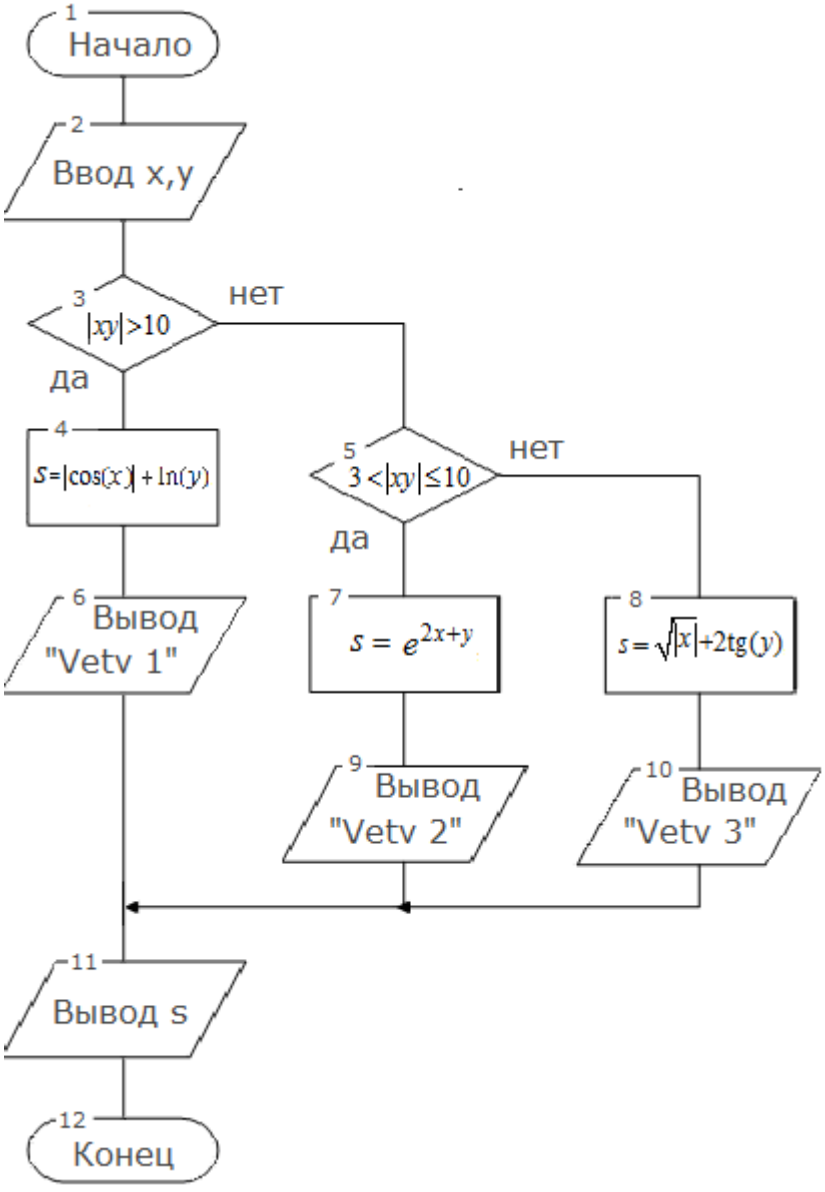
```
switch (k)
{
  case 1: f=pow(x,2); break;
  case 2: f=fabs(x);  break;
  case 3: f=sqrt(x);  break;
  default:  cout<<"\nf(x) ne zadana!\n";
            return 1;
}
```

2.4. Пример выполнения задания

Написать программу вычисления выражения $s=\begin{cases}|\cos(x)|+\ln(y), & |xy|>10\\ e^{2x+y}, & 3<|xy|\le 10\\ \sqrt{|x|}+2\mathrm{tg}(y), & \text{иначе}\end{cases}$

Предусмотреть вывод информации о выбранной ветви вычислений.

Блок-схема алгоритма



Код программы

```
#include <iostream.h>
#include <math.h>

int main()
{
```

```
  double x, y, s, f_xy;


  cout << "Vvedite x:  ";
  cin >> x;
  cout << "Vvedite y:  ";
  cin >> y;


  f_xy=fabs(x*y);
  if (f_xy>10) {                                          //|xy|>10
                    s=fabs(cos(x))+log(y);
                    cout<<"\nVetv 1\n";
                  }
      else if (f_xy>3 && f_xy<=10)                        //3<|xy|≤10
                                {
                                  s=exp(2*x+y);
                                  cout<<"\nVetv 2\n";
                                 }
                          else {                          // иначе
                                  s=sqrt(fabs(x))+2*tan(y);
                                  cout<<"\nVetv 3\n";
                                }
  cout<<"\nResult="<<s<<endl;
  return 0;
}
```

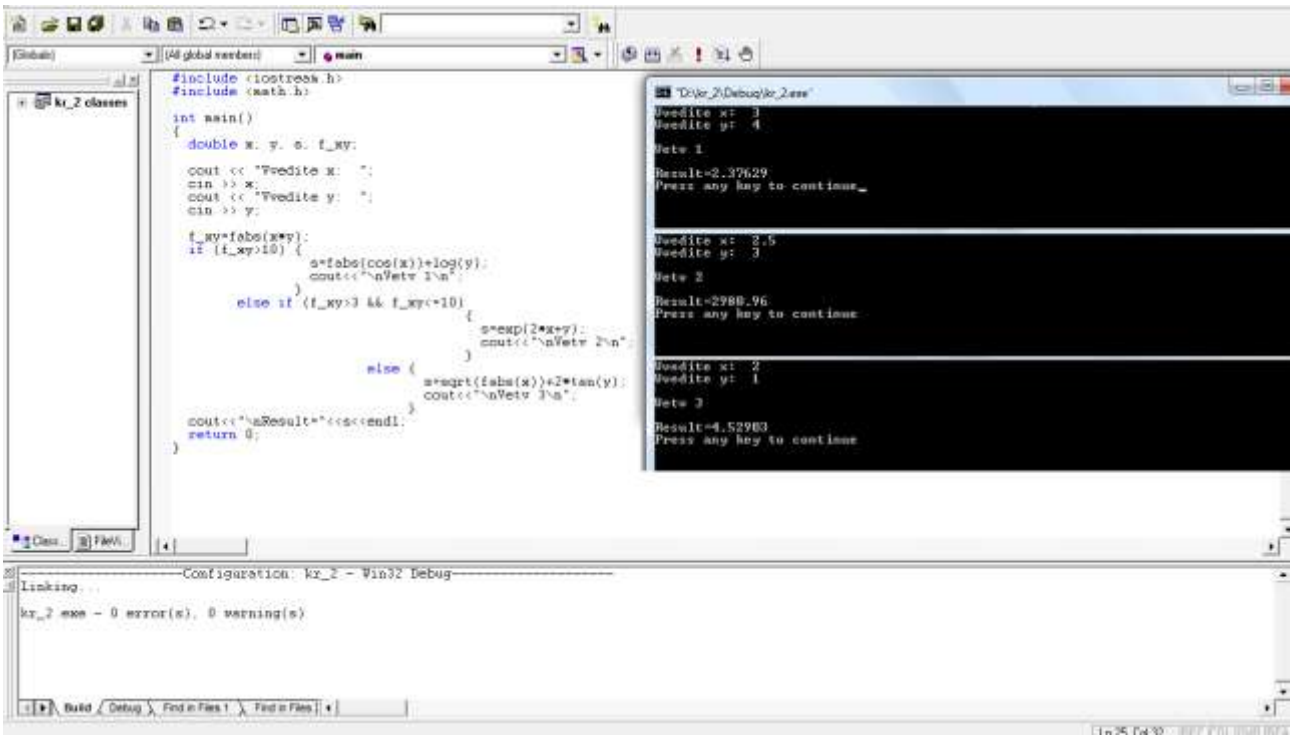


Рисунок 3 – Результат выполнения программы

2.5. Индивидуальные задания

Составить согласно индивидуальному варианту блок-схему алгоритма и программу вычисления выражения. Предусмотреть вывод информации о выбранной ветви вычислений.

1. $s=\begin{cases}(x+y)^2-\sqrt[3]{|x|}, & xy>0\\ (x+y)^2+\sin(x), & xy<0\\ (x+y)^2+y^3, & \text{иначе}\end{cases}$

2. $s=\begin{cases}\ln(x)+\sqrt[3]{|y|}, & x/y>0\\ \ln|x/y|\cdot(x+y)^3, & x/y<0\\ (x^2+y)^3, & \text{иначе}\end{cases}$

3. $s=\begin{cases}x^2+\sqrt[3]{y}+\sin(y), & x-y=0\\ (x-y)^2+\ln(|x|), & x-y>0\\ (y-x)^2+\operatorname{tg}(y), & \text{иначе}\end{cases}$

4. $s=\begin{cases}\sqrt[3]{|x-y|}+\operatorname{tg}(x), & x>y\\ (y-x)^3+\cos(x), & x<y\\ (y+x)^2+x^3, & \text{иначе}\end{cases}$

5. $s=\begin{cases}y\sqrt{|x|}+3\sin(x), & x>y\\ x\sqrt{|x|}, & x<y\\ \sqrt[3]{|x|}+x^3/y, & \text{иначе}\end{cases}$

6. $s=\begin{cases}e^{x-|y|}, & 0{,}5<xy<10\\ \sqrt[3]{|x+y|}, & 0{,}1<xy<0{,}5\\ 2x^2, & \text{иначе}\end{cases}$

7. $s=\begin{cases}e^{-x}, & 1<xb<10\\ \sqrt[3]{|x+4y|}, & 12<xb<40\\ y\cdot x^2, & \text{иначе}\end{cases}$

8. $s=\begin{cases}(x^2+y)^3, & x/y<0\\ \ln|x/y|+x/y, & x/y>0\\ \sqrt[3]{|\sin(y)|}, & \text{иначе}\end{cases}$

9. $s=\begin{cases}2x^3+3y^2, & x>|y|\\ |x-y|, & 3<x<|y|\\ \sqrt[3]{|x-y|}, & \text{иначе}\end{cases}$

10. $s=\begin{cases}\ln(|x|+|y|), & |xy|>10\\ e^{x+y}, & |xy|<10\\ \sqrt[3]{|x|}+y, & \text{иначе}\end{cases}$

11. $s=\begin{cases}\operatorname{tg}(x)+\dfrac{x}{\sqrt[3]{y}}, & xy>0\\ \ln|x^2\cdot y|, & xy<0\\ x^3+\sin^2(y), & \text{иначе}\end{cases}$

12. $s=\begin{cases}\operatorname{tg}(x)+x^2, & y>2x\\ |x+y|^3, & y<2x\\ \sqrt[3]{x}\cdot\sin(x), & \text{иначе}\end{cases}$

13. $s=\begin{cases}(x+\ln(|y|))^3, & x/y>0\\ 2/3+\ln(|\sin(y)|), & x/y<0\\ \sqrt[3]{x^2}+y, & \text{иначе}\end{cases}$

14. $s=\begin{cases}\ln(x)^3, & x^3>0\\ \operatorname{tg}(x^3)+y\cdot x, & x^3<0\\ \sqrt[3]{|y^3-x^2|}, & \text{иначе}\end{cases}$

15. $s=\begin{cases}(x^2+y^3)/y, & x>0\\ \ln|x^3|+\cos(y), & x<0\\ \sqrt[3]{\sin^2(y)}, & \text{иначе}\end{cases}$

ЗАДАНИЕ №3
ПРОГРАММИРОВАНИЕ ЦИКЛИЧЕСКИХ АЛГОРИТМОВ. ФУНКЦИИ ПОЛЬЗОВАТЕЛЯ

3.1. Оператор цикла с параметром for

Общий вид оператора:

```
for (инициализирующее_выражение; условие;
                                  инкрементирующее_выражение)
    {
      тело цикла;
     }
```

Инициализирующее_выражение выполняется только один раз в начале выполнения цикла и, как правило, инициализирует счетчик цикла.

Условие содержит операцию отношения, которая выполняется в начале каждого цикла. Если условие равно **true** (**1**), то цикл повторяется, иначе выполняется следующий за телом цикла оператор.

Инкрементирующее_выражение, как правило, предназначено для изменения значения счетчика цикла. Модификация счетчика происходит после каждого выполнения тела цикла.

Например: вычислить значение факториала $n = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n = n!$ (0!=1)

```
f=1;
for (i=1;i<=n;i++)
      f*=i;
```

3.2. Оператор цикла с предусловием while

Общий вид оператора:

```
while (условие)
    {
      тело цикла;
    }
```

Операторы *тела цикла* повторяются до тех пор, пока *условие* истинно.

Например: найти НОД(a,b)

```
while (a!=b)
    if (a>b) a-=b;
        else b-=a;
```

3.3. Оператор цикла с постусловием do

Общий вид оператора:

```
do {
      тело цикла;
   } while (условие);
```

Операторы *тела цикла* повторяются до тех пор, пока *условие* истинно.

Например: ввести такие длины сторон, чтобы из них можно было составить треугольник

```
do {
    cout<<"\nVvedite dlini storon (a,b,c): ";
    cin>>a>>b>>c;
} while (a+b>c || c+b>a || a+c>b);
```

3.4. Операторы перехода

Оператор **break** прекращает выполнение ближайшего к нему цикла или оператора **switch**.

```
while (условие)
    {
     оператор;
     break;
     оператор;
    }
оператор;
```

```
do {
    оператор;
    break;
    оператор;
  } while (условие);
оператор;
```

Оператор **continue** передает управление на проверку условия циклов **while** и **do while**, либо на *инкрементирующее_выражение* цикла **for**.

```
while (условие)
    {
     оператор;
     continue;
     оператор;
    }
```

```
do {
    оператор;
    continue;
    оператор;
  } while (условие);
```

3.5. Объявление функции

Функция – это последовательность операторов, оформленная таким образом, что ее можно вызвать по имени из любого места программы. **Функция описывается** следующим образом:

```
тип_возвращаемого_значения имя_функции (список_параметров)
{
  тело функции
}
```

Первая строка описания называется *заголовком функции*.

Тип_возвращаемого_значения может быть любым, кроме *массива* или *функции*. Допустимо не возвращать никакого значения (тип **void**).

Список параметров представляет собой список конструкций следующей формы:

тип параметра имя параметра

Например:

```
int Sum (int a, double b, char c);
void Prints (char c, int f);
```

Если функция не получает никаких данных, то скобки остаются пустыми:

```
int Mem ();
```

Как правило, помимо описания функции в программу вставляется *прототип* функции (ее предварительное объявление). Прототип аналогичен заголовку функции, только на конце его ставится точка с запятой, а имена формальных параметров не указываются (остаются только типы):

```
int Sum (int, double, char);
```

Правила оформления тела функции такие же, как и любого другого участка программы. Все объявления носят локальный характер, т.е. объявленные переменные доступны только внутри функции.

В C++ не допускается вложение функций друг в друга.

Выход из функции осуществляется следующими способами:

1. Если нет необходимости возвращать вычисленное значение, то выход осуществляется по достижении закрывающей скобки или при выполнении оператора **return**.

2. Если необходимо вернуть полученное значение, то выход осуществляется оператором

return *выражение*;

3.6. Передача параметров

При работе важно соблюдать следующее *правило*: при объявлении и вызове функции параметры должны соответствовать по *количеству*, *порядку следования* и *типам*. Функция может не иметь параметров, в этом случае после имени функции обязательно ставятся круглые скобки. Существует три основных способа передачи параметров: передача по значению, по ссылке или по указателю.

3.6.1. Передача параметров по значению

В момент обращения к функции в памяти создаются временные переменные с именами, указанными в списке параметров, в которые копируются значения фактических (передаваемых в функцию) параметров. После завершения работы функции временные переменные удаляются из памяти.

Пример. *Вычислить сумму двух переменных* **x** *и* **y.**

```
double Sum(double a, double b)
 {
   return a+b;          // Вычисление и передача результата
 }
  …
 s = Sum (x, y);          // Вызов функции Sum
```

Функция Sum не может изменить значения используемых при вызове функции переменных **x** и **y**, так как работает только с их *локальными копиями*.

3.6.2. Передача параметров по ссылке

При передаче параметров по ссылке передается **адрес** соответствующей переменной, а не ее значение. Для получения адреса используется операция разадресации («**&**»).

Пример. *Поменять местами значения двух переменных* **x** *и* **y**.

```
void Swap_Ref (double &a, double &b)
 {
   int tmp = a;
   a = b;
   b = tmp;
  }
   ...
Swap_Ref (x, y);          // Вызов функции Swap_Ref
```

При таком способе передачи параметры **a** и **b** будут инициализированы в качестве псевдонимов переменных-аргументов **x** и **y**. Поэтому любые изменения параметров **a** и **b** будут приводить к соответствующему изменению переменных **x** и **y**.

3.6.3. Передача параметров по указателю

Так же как и при передаче параметров по ссылке, в данном способе используется не значение соответствующей переменной, а ее **адрес**. Отличие от предыдущего способа состоит в том, что используется **операция косвенной адресации** (*).

Пример. *Поменять местами значения двух переменных* **x** *и* **y**

```
void Swap_Ptr (double *a, double *b)
{
   double tmp= *a;
   *a = *b;
   *b = tmp;
}
   ...
Swap_Ptr (&x, &y);          // Вызов функции Swap_Ptr
```

Функция **Swap_Ptr** требует явного указания адресов при своем вызове (&x, &y) и явного их разыменования в функции (*a и *b).

3.7. Перегрузка функций

В C++ допустимо использование нескольких функций с одинаковым именем, но различным числом или типами параметров. Такое свойство называется *перегрузкой функций*. Перегруженные функции различаются компилятором *по типам* и *числу параметров*.

Пример. *Написать функцию, суммирующую* ***два*** *или* ***три*** *целых числа.*

```
#include <iostream.h>
#include <conio.h>
```

```
// Прототипы функций
int Sum(int, int);
int Sum(int, int, int);

int main()
{
  cout << Sum(5, 3)     << endl;
  cout << Sum(5, 3, 11) << endl;
  return 0;
}

int Sum(int a, int b)          // Функция суммирования двух чисел
{
  return a+b;
}

int Sum(int a, int b, int c) // Функция суммирования трех чисел
{
  return a+b+c;
}
```

3.8. Отладка программы

Для поиска логических ошибок используется встроенный отладчик.

Для пошагового выполнения программы необходимо нажимать клавишу **F10**. При каждом нажатии выполняется текущая строка. Если необходимо пошагово проверить код вызываемой функции, то следует нажать **F11**. Для досрочного выхода из функции нажать **Shift+F11**. Если необходимо начать отладку с определенного места программы, то надо установить курсор в соответствующую строку программы и нажать **Ctrl+F10**.

Другим способом отладки является установка *точек прерывания* программы. Для этого надо поместить курсор в нужную строку и нажать **F9**. Точка прерывания обозначается красным кружком на специальном поле, расположенном слева от окна кода программы. Для удаления точки прерывания следует в необходимой строке повторно нажать **F9.** Количество точек прерывания в программе может быть любым.

Для выполнения программы до точки прерывания необходимо нажать **F5**. Для продолжения отладки применяется клавиша **F5** (выполнение программы до следующей точки прерывания) или используются клавиши для пошаговой отладки.

Желтая стрелка на поле слева от окна кода программы указывает на строку, которая будет выполнена на следующем шаге отладки.

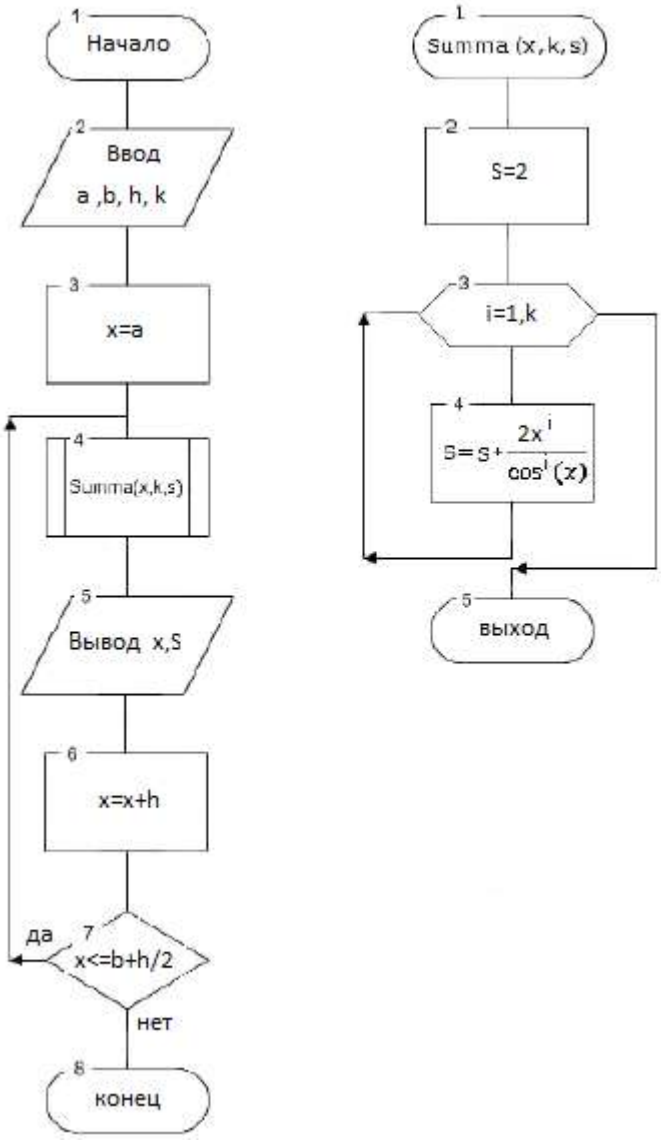
Для контроля за значениями переменных удобно использовать следующий способ: подвести указатель мыши к интересующей переменной и задержать его на несколько секунд. На экране рядом с именем переменной появится окно, содержащее текущее значение этой переменной. Кроме этого, значения переменных будут отображаться в окнах, расположенных снизу. В левом нижнем окне отображаются значения последних использованных программой переменных. В пра-

вом нижнем окне (**Watch**) можно задать имена переменных, значения которых необходимо контролировать.

3.9. Пример выполнения задания

Написать программу вывода на экран таблицы значений функции $\sum_{k=0}^{20} \frac{2x^k}{\cos^k(x)}$ *для x, изменяющегося от a = 0,1 до b = 1 с шагом h = 0,1. Вычисление суммы оформить в виде функции пользователя. Предусмотреть передачу параметров в функцию разными способами.*

Блок-схема алгоритма



Код программы

```
#include <iostream.h>
#include <iomanip.h>
#include <math.h>

// Прототипы функций
```

```
double Summa (double, int);            // Передача параметров по значению
void Summa (double*, int*,double*);// Передача параметров по указателю
void Summa (double&, int&,double&); // Передача параметров по ссылке

int main()
{
  double  s, x, a, b, h;
  int     k;
  cout<<"Vvedite a, b, h, k:\n";
  cin>>a>>b>>h>>k;              // Ввод значений:: 0.1  1  0.1  20
// Вывод строки заголовка таблицы
  cout<<"\n     Value"<<setw(20)<<"Pointer"<<setw(24)<<"Reference\n";
  x=a;
  do                                  // Начало цикла по  x
  {    // Вывод таблицы
                                      // Передача параметров по значению
    cout<<setw(5)<<x<<setw(10)<<Summa(x, k);

    Summa (&x, &k,&s);            // Передача параметров по указателю
    cout<<setw(10)<<x<<setw(10)<<s;

    Summa (x, k, s);              // Передача параметров по ссылке
    cout<<setw(10)<<x<<setw(10)<<s<<endl;
     x+=h;                        // Изменение значения x на величину шага h
  } while (x<=b+h/2);             // Проверка условия продолжения цикла по x
  cout<<endl;                     // Переход на новую строку
  return 0;
}

double Summa (double x, int k)     // Передача параметров по значению
{
   double s;
   int i;
   s=2;                           // Начальное значение при k=0

   for (i=1; i<=k; i++)           // Вычисление суммы  \sum_{k=1}^{20} \frac{2x^k}{\cos^k(x)}

        s+=2*pow(x,i)/pow(cos(x),i);
   return s;                      // Передача результата s в главную функцию
}
                                  // Передача параметров по указателю
void Summa (double *x, int *k, double *s)
{
   int i;
   *s=2;                          // Начальное значение при k=0
```

```
   for (i=1; i<=*k; i++)        // Вычисление суммы  $\sum_{k=1}^{20} \frac{2x^k}{\cos^k(x)}$
       *s+=2*pow(*x, i)/pow(cos(*x),i);
}
                                // Передача параметров по ссылке
void Summa (double &x, int &k, double &s)
{
   int i;
   s=2;                         // Начальное значение при k=0
   for (i=1; i<=k; i++)         // Вычисление суммы  $\sum_{k=1}^{20} \frac{2x^k}{\cos^n(x)}$
       s+=2*pow(x,i)/pow(cos(x),i);
}
```

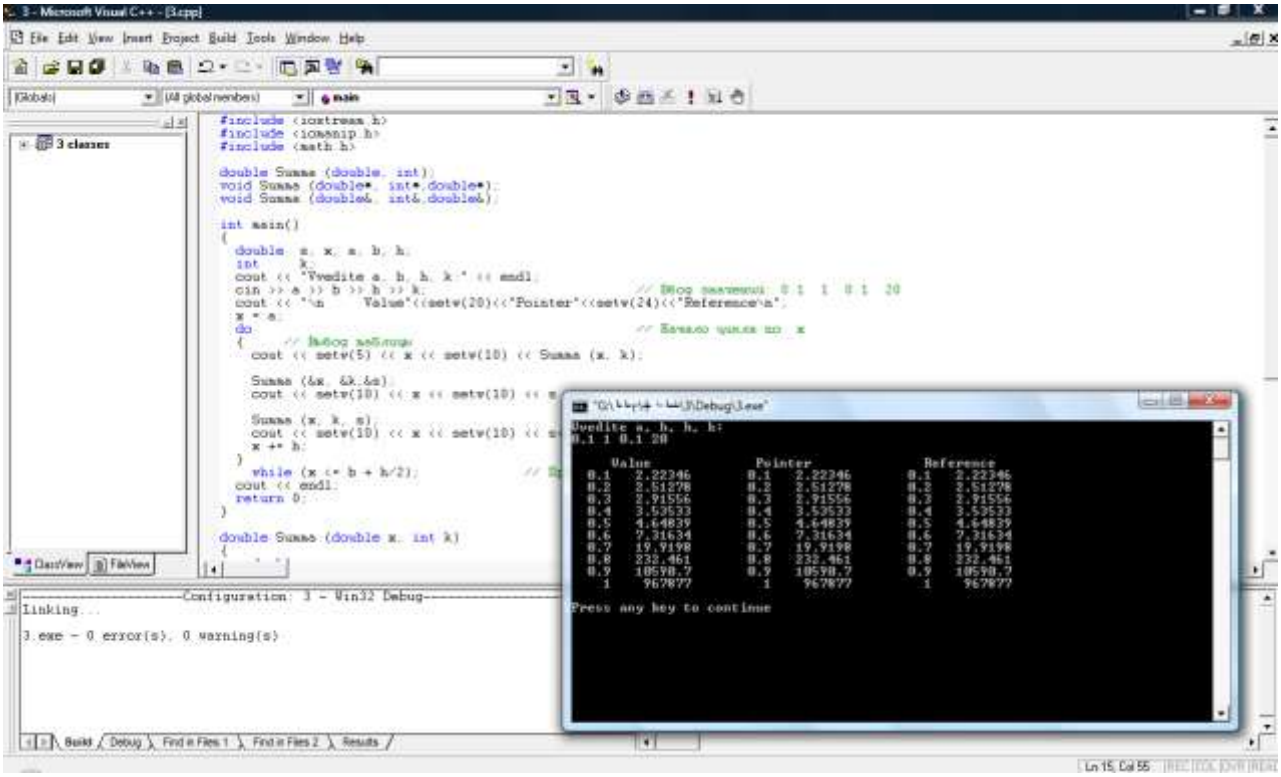


Рисунок 4 – Результат выполнения программы

3.10. Индивидуальные задания

Составить согласно индивидуальному варианту блок-схему алгоритма и программу вывода на экран таблицы значений функции y(x) для x, изменяющегося от a=0,1 до b=1,2 с шагом h=0,1. Вычисление y(x) оформить в виде функции. Предусмотреть передачу параметров в функцию разными способами.

1. $y=\sum_{n=1}^{20}\frac{x^{n-1}}{2n+1}$.

2. $y=\sum_{n=0}^{20}\frac{(2x)^n}{n+1}$.

3. $y=\sum_{n=1}^{20}\frac{x^{n-1}}{\sin(nx)}$.

4. $y=\sum_{n=1}^{20}\frac{n^2+1}{n}\left(\frac{x}{2}\right)^n$.

5. $y=\sum_{n=1}^{20}\frac{\cos\left(n\cdot\frac{\pi}{4}\right)}{n+1}x^n$.

6. $y=\sum_{n=1}^{20}\frac{x^{2n-2}}{2n+1}$.

7. $y=\sum_{n=0}^{20}\frac{x^{2n}}{\cos(nx)}$.

8. $y=\sum_{n=1}^{20}\frac{2n^2+1}{2n}x^{2n-2}$.

9. $y=\sum_{n=1}^{20}\frac{2n+1}{\sin(nx)}x^{n-1}$.

10. $y=\sum_{n=0}^{20}\frac{\cos^n(x)}{2n+1}$.

11. $y=\sum_{n=1}^{20}\frac{n\cdot x^{n-1}}{\sin(2n+x)}$.

12. $y=\sum_{n=1}^{20}\frac{(1+x)^{n-1}}{\left|\sin^n(x)\right|}$.

13. $y=\sum_{n=1}^{20}\frac{x^{2n-2}}{4\cos(nx^2)}$.

14. $y=\sum_{n=1}^{20}\frac{n^2}{(2n+1)}x^{n-1}$.

15. $y=\sum_{n=1}^{20}\frac{\cos\left(n\frac{\pi}{4}\right)}{n^2}x^n$.

ЗАДАНИЕ №4
ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ ОДНОМЕРНЫХ МАССИВОВ

4.1. Одномерные статические массивы

Массив – это набор данных **одинакового типа**, расположенных в непрерывной области памяти таким образом, чтобы по индексу элемента можно было вычислить адрес его значения: ***адрес(a[i]) = адрес(a[0]) + i*k***,
где ***k*** – количество байт, отводимых под элемент массива;

i – индекс элемента массива.

Для доступа к элементу массива необходимо указать его имя и индекс (порядковый номер элемента в масиве):

имя_массива [индекс]

В программе одномерный массив объявляется следующим образом:

***тип имя_массива* [*размер*];**

Пример *декларации массива*:

```
int mas[4];
```

Индексы в массиве начинаются с **0,** т. е. массив, приведенный в примере, будет содержать следующие элементы: *mas[0]*, *mas[1]*, *mas[2]* и *mas[3]*. *Выход индекса за пределы массива не проверяется.*

Пример 1. Найти произведение нечетных элементов массива, расположенных до первого нулевого элемента.

```
for (pr=1,k=i=0; i<n, a[i]; i++) //пока ai≠0
  if (a[i]%2)    { // ai - нечетное
                     pr*= a[i];
                     k++;
                   }
if (!k) pr=0;  // если таких элементов нет, то произведение =0
```

Пример 2. *Упорядочить элементы массива по возрастанию их значений, т.е. для всех элементов массива должно выполняться условие:* $a_i < a_{i+1}$.

```
   …
for (k=1; k<n; k++)              // k – номер просмотра массива
   for (i=0; i<n-k; i++)         // Просмотр элементов массива
    if (a[i] > a[i+1])           // Сравнение элементов массива
        {
          temp=a[i];          // Перестановка элементов ai и ai+1,
          a[i]=a[i+1];           // если они стоят неправильно
          a[i+1]=temp;
        }
```

Пример 3*. Удалить из одномерного массива все отрицательные элементы*

Для решения данной задачи необходимо выполнить следующие действия:

```
        ...
    for (i=0; i<n; i++)
      if (a[i]<0)            // Если найден отрицательный элемент, то
        {
          for (j=i+1; j<n; j++)   // сдвинуть все элементы, стоящие
                    a[j-1]=a[j];  // после удаляемого на одну позицию
         n--;                     // Уменьшение размера массива
         i--;                     // Возврат к предыдущему индексу
        }
```

Пример 4. *Даны одномерные упорядоченные по возрастанию массивы:* ***X*** *размером* ***n*** *элементов и* ***Y*** *размером* ***m*** *элементов. Объединить элементы этих массивов в массив* ***Z*** *так, чтобы и он оказался упорядоченным по возрастанию.*

```
    ...
k=i=j=0;
while(i<n && j<m)
{
  if (a[i]<b[j]) { c[k]=a[i];
                   i++;
                 }
          else { c[k]=b[j];
                 j++;
                }
  k++;
}
while(i<n)
  {
    c[k]=a[i];
    i++;
    k++;
  }
while(j<m)
  {
    c[k]=b[j];
    j++;
    k++;
  }
```

4.2. Пример выполнения задания

Составить программу поиска минимального и максимального элементов одномерного массива и их индексов.

```
#include <iostream.h>
#include <iomanip.h>
#include <math.h>

void MinMax (int a[],int,int*,int*,int*,int*);  // Прототип функции
```

```
int main()
{
  int a[10], i, n, min, imin, max, imax;

  cout<<"Vvedite razmer massiva: ";        // Ввод размерности массива
  cin>>n;
  cout<<"\nVvedite  massiv:\n";
  for (i=0; i<n; i++)                      // Ввод одномерного массива
     {
       cout<<"Vvedite a["<<i<<"]=";
       cin>>a[i];
     }

  cout<<"\nMassiv a:\n";                  // Вывод одномерного массива
  for (i=0; i<n; i++)
       cout<<setw(7)<<a[i];
  cout<<endl;

  MinMax (a, n, &min, &max, &imin, &imax);          // Вызов функции

  cout<<"\nMax="<<max<<setw(10)<<"i="<<imax;
  cout<<"\nMin="<<min<<setw(10)<<"i="<<imin<<endl;
  return 0;
}
// Функция поиска минимального и максимального элементов
// одномерного массива и их индексов
void MinMax (int a[],int n,int *min,int *max,int *imin,int *imax)
{
  int i;
  *min=*max=a[0];              // Инициализация значений
  *imin=*imax=0;
  for (i=1; i<n; i++)
     if (a[i]<*min) {          // Поиск минимального элемента и его индекса
                     *min=a[i];
                     *imin = i;
                    }
        else
          if (a[i]>*max) { // Поиск максимального элемента и его индекса
                          *max=a[i];
                          *imax = i;
                         }
}
```

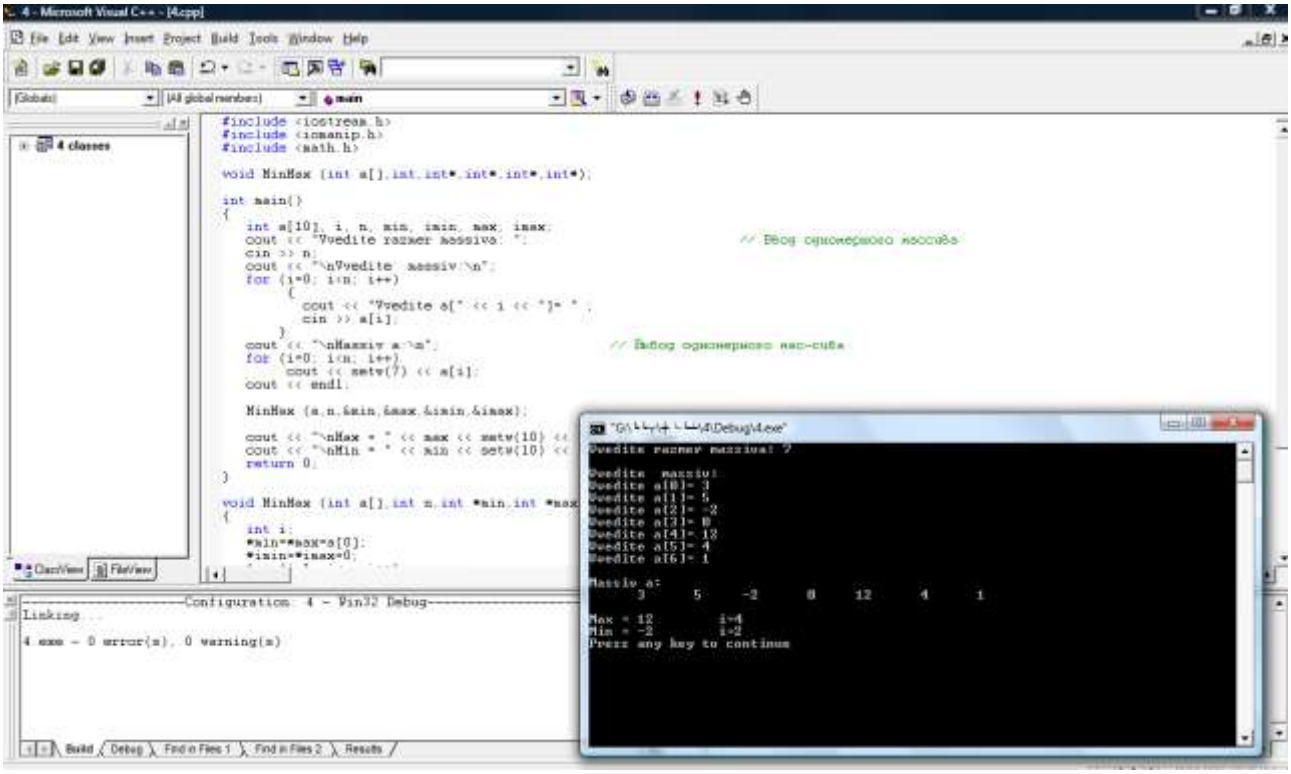


Рисунок 5 – Результат выполнения программы

4.3. Индивидуальные задания

Ввести одномерный статический массив из k чисел. Выполнить в соответствии с номером варианта индивидуальное задание и вывести на экран исходные данные и полученный результат. Предусмотреть использование функции пользователя.

1. Найти произведение элементов массива, расположенных между максимальным и минимальным элементами.

2. Найти сумму элементов массива, расположенных после первого нулевого элемента.

3. Найти среднее арифметическое элементов массива, расположенных до максимального элемента.

4. Найти количество элементов массива, расположенных после минимального элемента.

5. Найти произведение и количество элементов массива, расположенных до первого отрицательного элемента.

6. Найти сумму и количество элементов массива, расположенных после первого положительного элемента.

7. Найти среднее арифметическое положительных кратных трем элементов массива, расположенных до минимального элемента.

8. Найти произведение четных отрицательных элементов массива, расположенных после минимального элемента.

9. Найти сумму и количество нечетных элементов массива, расположенных до последнего положительного элемента.

10. Найти среднее арифметическое модулей кратных пяти элементов массива, расположенных после максимального элемента.

11. Найти произведение модулей элементов массива, расположенных после минимального элемента.

12. Найти сумму модулей элементов массива, расположенных после последнего нулевого элемента.

13. Найти среднее арифметическое модулей четных элементов массива, расположенных между первым отрицательным и последним положительным элементами.

14. Найти максимальное значение между суммами четных и нечетных элементов массива, расположенных до минимального элемента.

15. Расположить элементы массива в обратном порядке.

ЗАДАНИЕ №5
УКАЗАТЕЛИ. ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ ДИНАМИЧЕСКИХ ДВУМЕРНЫХ МАССИВОВ

5.1. Объявление указателя

Для всех переменных выделяются участки памяти размером, соответствующим типу переменной. Программист имеет возможность работать непосредственно с адресами, для чего определен соответствующий тип данных – *указатель*. Указатель имеет следующий формат:

*тип *имя_указателя;*

Например:

```
int *a;
double *b, *d;
char *c;
```

Знак «**звездочка**» относится к имени указателя. Значение указателя соответствует *первому* байту участка памяти, на который он ссылается. На один и тот же участок памяти может ссылаться *любое* число указателей.

В языке С существует *три вида указателей*:

1. Указатель на объект *известного типа*. Содержит адрес объекта определенного типа.

Например: **int** *ptr;

2. Указатель типа **void**. Применяется, если тип объекта заранее не определен.

Например: **void** *vptr;

3. Указатель на *функцию*. Адрес, по которому передается управление при вызове функции.

Например: void (***func**)(int);

***func** указатель на функцию, принимающей аргумент **int** и не возвращающей никаких значений.

5.2. Операции над указателями

К указателям можно применять две *унитарные операции*:

1. & (взятие адреса).

Указатель получает адрес переменной. Данная операция применима к переменным, под которые выделен соответствующий участок памяти.

```
Например:  int *ptr, var=1;        // ptr – указатель, var – переменная
           ptr = &var;             // В ptr заносится адрес var
```

2. * (операция разадресации).

Предназначена для доступа к значению, расположенному по данному адресу.

```
*ptr = 9;      // В ячейку памяти, с адресом ptr записывается значение 9
var = *ptr;    // Переменной var присваивается значение,
```

// расположенное по адресу **ptr**

Над указателями можно выполнять арифметические операции *сложения*, *инкремента* (увеличения на 1), *вычитания*, *декремента* (уменьшения на 1) и операции *сравнения* (**>**, **>=**, **<**, **<=**, **==**, **!=**). При выполнении арифметических операций с указателями автоматически учитывается размер данных, на которые он указывает.

Например:

```
ptr++;      // Сдвиг указателя ptr на один элемент вперед
(*ptr)++;  // (или ++*ptr;) Увеличение на 1 значения переменной,
           // на которую указывает указатель ptr
*ptr = NULL;  // Очистка указателя ptr1
```

Указатели, как правило, используются при работе с динамической памятью (*heap* или «куча»). Для работы с динамической памятью в языке С определены следующие функции: **malloc**, **calloc**, **realloc** и **free**.

В языке С++ для выделения и освобождения памяти определены операции **new** и **delete** соответственно. Используют две *формы операций*:

1. *Тип *указатель* = **new** *тип* (*значение*); – выделение участка памяти в соответствии с указанным типом и занесение туда заданного значения.

delete *указатель;* – освобождение выделенной памяти.

2. *Тип *указатель* = **new** *тип*[**n**]; – выделение участка памяти размером **n** блоков указанного типа.

delete [ ] *указатель;* – освобождение выделенной памяти.

Пример *работы с одномерным динамическим массивом*:

```
int *a;              // Объявление указателя a
a = new int[n];   // Выделение n блоков памяти целого типа
      …           // Работа с массивом a
delete [] a;      // Освобождение выделенной памяти
```

5.3. Создание двумерного динамического массива

Имя любого массива рассматривается компилятором как *указатель на нулевой элемент* массива. Так как имя двумерного динамического массива является *указателем на указатель*, то сначала выделятся память *под указатели*, а затем под соответствующие этим указателям *строки*. Освобождение выделенной памяти происходит *в обратном порядке*.

5.4. Пример выполнения задания

Написать программу перестановки минимального и максимального элементов двумерного массива размером NxM. Память для массива выделить динамически. результат. Предусмотреть использование функции пользователя.

```
#include <iostream.h>
#include <iomanip.h>
#include <math.h>

// Прототип функции поиска минимального и максимального элементов
```

```
// двумерного массива
void Poisk (double**, int, int, int*, int*, int*, int*);

int main()
{
   double  **a, tmp;
   int i, j, n, m, imin, jmin, imax, jmax;

   cout << "Vvedite razmer massiva A:\n";
   cout << "row n=";
   cin >> n ;
   cout << "column m=";
   cin >>  m;

   a = new double*[n];            // Выделение памяти под массив указателей
   for(i=0; i<n; i++)             // Выделение памяти под соответствующие
      a[i] = new double[m];       // этим указателям строки матрицы

   cout << "\nVvedite massiv A:\n";    // Ввод элементов двумерного массива
   for (i=0; i<n; i++)
     for (j=0; j<m; j++)
       {
          cout << "Vvedite a[" << i << "][" << j << "]: ";
          cin >> a[i][j];
       }

   cout << "\nMassiv A:\n";         // Вывод элементов двумерного массива
   for (i=0; i<n; i++)
      {
         for (j=0; j<m; j++)
            cout << setw (9) << a[i][j];
         cout << endl;
      }

   Poisk (a, n, m, &imin, &imax, &jmin, &jmax);         // Вызов функции

   tmp = a[imin][jmin];                                 // Перестановка элементов
   a[imin][jmin] = a[imax][jmax];
   a[imax][jmax] = tmp;

   cout << "\nResult massiv:\n" ;                       // Вывод результата
   for (i=0; i<n; i++)
      {
```

```
            for (j=0; j<m; j++)
               cout << setw (9) << a[i][j];
            cout << endl;
         }

   for(i=0; i<n; i++)              // Освобождение выделенной памяти
      delete [] a[i];
   delete []a;
   a = NULL;

   return 0;
}

// Функция поиска индексов минимального и максимального элементов массива
void Poisk (double **a,int n,int m,int *imin,int *imax,int *jmin,int *jmax)
{
   int i, j;

   *imin=*jmin=*imax=*jmax=0;  // Инициализация значений индексов
   for (i=0; i<n; i++)
      for (j=0; j<m; j++)
         {
            if (a[i][j]<a[*imin][*jmin]) { // Поиск индексов минимального элемента
                                           *imin=i;
                                           *jmin=j;
                                         }
            if (a[i][j]>a[*imax][*jmax]) { // Поиск индексов максимального элемента
                                           *imax=i;
                                           *jmax=j;
                                         }
         }
}
```

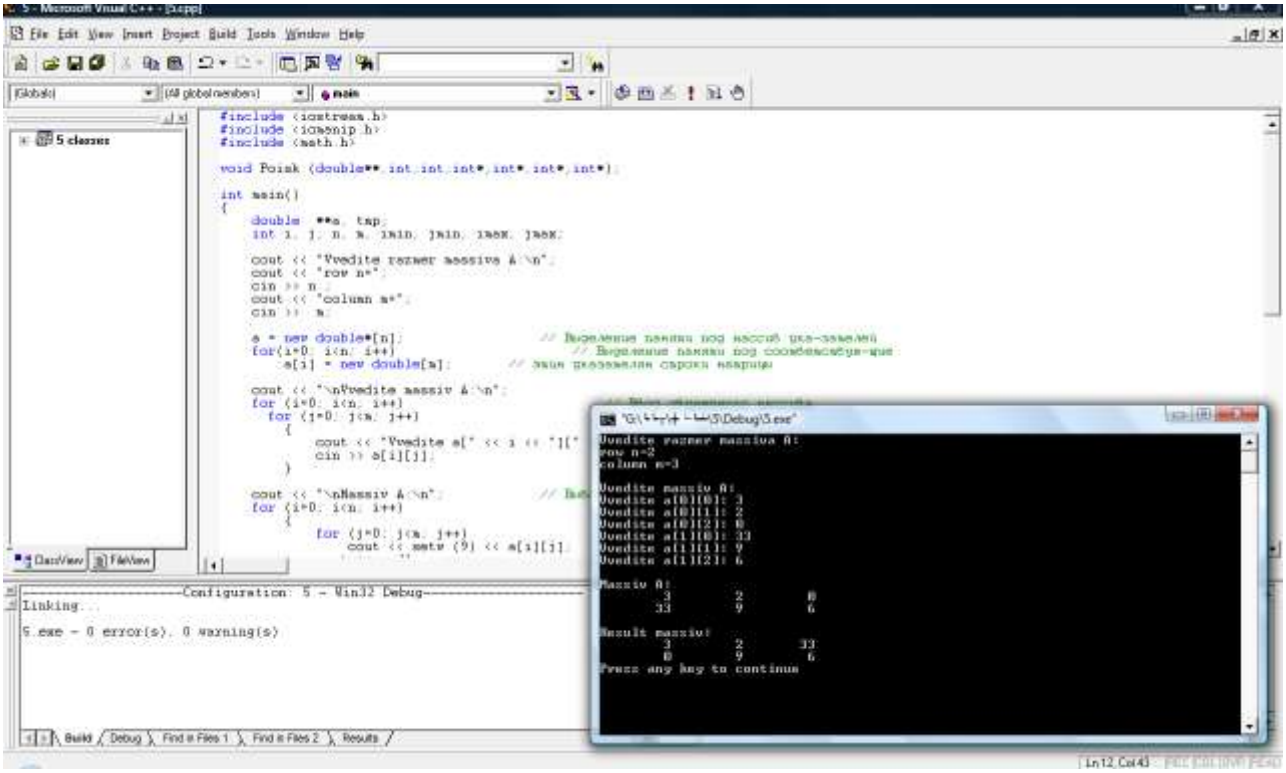


Рисунок 6 – Результат выполнения программы

5.5. Индивидуальные задания

Ввести матрицу размером NxM. Память для массива выделить динамически. Выполнить в соответствии с номером варианта индивидуальное задание и вывести на экран исходные данные и полученный результат. Предусмотреть в программеиспользование функции пользователя.

1. Определить количество положительных элементов, расположенных ниже побочной диагонали матрицы.

2. Определить количество отрицательных элементов, расположенных выше главной диагонали матрицы.

3. Определить сумму отрицательных элементов, расположенных выше побочной диагонали матрицы.

4. Определить произведение положительных элементов, расположенных ниже главной диагонали матрицы.

5. Определить сумму элементов, расположенных на главной диагонали матрицы, и произведение элементов, расположенных на побочной диагонали матрицы.

6. Определить количество четных элементов, расположенных на главной и побочной диагоналях.

7. Найти максимальный среди элементов, лежащих ниже побочной диагонали.

8. Найти минимальный среди элементов, лежащих выше главной диагонали.

9. Найти максимальный среди элементов, лежащих выше побочной диагонали.

10. Найти минимальный среди элементов, лежащих ниже главной диагонали.

11. Определить произведение кратных трем элементов матрицы, расположенных выше ее главной диагонали, включая саму диагональ.

12. Определить сумму отрицательных нечетных элементов матрицы, расположенных ниже ее побочной диагонали, включая саму диагональ.

13. Найти сумму элементов, расположенных в четных (по номеру) строках матрицы.

14. Найти произведение элементов, расположенных в нечетных (по номеру) столбцах матрицы.

15. Подсчитать сумму четных элементов и произведение нечетных элементов матрицы.

ЗАДАНИЕ №6
ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ ФАЙЛОВ И СТРУКТУР

6.1. Объявление структур

Структура – это составной тип данных, в котором под одним именем объединены данные *различных* типов. Отдельные данные структуры называются *полями*. Объявление структуры осуществляется с помощью ключевого слова **struct**, за которым указывается ее имя и список элементов, заключенных в фигурные скобки:

```
struct имя
 {
    тип_элемента_1  имя_элемента_1;
    тип_элемента_2  имя_элемента_2;
                    ...
    тип_элемента_n  имя_элемента_n;
 };
```

Правила работы с полями структуры идентичны работе с переменными соответствующих типов. К полям структуры можно обращаться через *составное имя*. **Формат обращения**:

имя_структуры.*имя_поля*

или

указатель_на_структуру–>*имя_поля*

6.2. Организация работы с файлами

Различают два вида файлов: *текстовые* и *двоичные* (*бинарные*).

Текстовые файлы хранят информацию в виде последовательности символов. В текстовом режиме каждый разделительный символ строки автоматически преобразуется в пару (возврат каретки – переход на новую строку).

Бинарные (или двоичные) файлы предназначены для хранения только числовых значений данных. Структура такого файла определяется программно.

Функции для работы с файлами размещены в библиотеках **stdio**.lib (**#include <stdio.h>**) и **io**.lib (**#include <io.h>**). Каждый файл должен быть связан с указателем, который имеет тип **FILE** и используется во всех операциях с файлами.

Формат *объявления указателя на файл* следующий:

FILE *указатель на файл;

Макрос **NULL** определяет пустой указатель.

Макрос **EOF**, часто определяемый как **–1**, является значением, возвращаемым тогда, когда функция ввода пытается выполнить чтение после конца файла.

Макрос **FOPEN_MAX** определяет целое значение, равное максимальному числу одновременно открытых файлов.

6.3. Функции для работы с файлами

Перед началом работы с файлом его необходимо открыть функцией

FILE ***fopen** (**const char** **имя_файла*,
const char **режим_открытия*);

которая связывает файл с потоком и возвращает указатель на открытый файл. *Имя_файла* и *режим_открытия* – указатели на строки символов, содержащие соответственно путь к файлу, его имя и режим открытия файла. Допустимые *режимы*:

r – открытие текстового файла для **чтения**;

w – создание текстового файла для **записи**;

a – **добавление** информации в конец текстового файла.

При работе с текстовыми файлами к символу, указывающему режим открытия, добавляется символ «**t**» (по умолчанию), а при работе с бинарными – «**b**». Если необходимо читать и записывать в файл, то добавляется символ «**+**». При возникновении ошибки во время открытия файла функция **fopen** возвращает значение **NULL**.

После завершения работы с файлом его необходимо закрыть функцией

int fclose (FILE * *указатель_на _файл*);

которая закрывает поток, открытый с помощью вызова **fopen ()**, и записывает в файл данные, оставшиеся в дисковом буфере. Результатом работы функции может быть значение нуля (успешная операция закрытия) или EOF (ошибка). Доступ к файлу после выполнения функции будет запрещен.

Функция

int fcloseall (void);

закрывает все открытые файлы и возвращает количество закрытых файлов или **EOF**, если возникает ошибка.

Функция

int putc (int *символ*, **FILE *** *указатель_на _файл*);

записывает один символ в текущую позицию указанного открытого файла.

Функция

int getc (FILE * *указатель_на _файл*);

читает один символ из текущей позиции указанного открытого файла.

Функция

int feof (FILE * *указатель_на _файл*);

возвращает отличное от нуля значение (**true**), если конец файла не достигнут, и ноль (**false**), если достигнут конец файла.

Функция

int fputs (const char * *строка*, **FILE *** *указатель_на _файл*);

записывает строку символов в текущую позицию указанного открытого файла.

Функция

char *fgets (char * *строка*, **int** *длина*,
FILE * *указатель_на _файл*);

читает строку символов из текущей позиции указанного открытого файла до тех пор, пока не будет прочитан символ перехода на новую строку или количество прочитанных символов не станет равным **длина – 1**.

Функция

int *fprintf (FILE * *указатель_на _файл*,
const char * *управляющая_строка*);

записывает форматированные данные в файл. *Управляющая_строка* определяет строку форматирования аргументов, заданных своими адресами. Обычно эта строка состоит из последовательности символов «**%**», после которых следует *символ типа данных*:

I или **i** – десятичное, восьмеричное или шестнадцатеричное целое;

D или **d** – десятичное целое;

U или **u** – десятичное целое без знака;

E или **e** – действительное с плавающей точкой;

s – строка символов;

c – символ.

Функция

int *fscanf (FILE * *указатель_на _файл*,
const char * *управляющая_строка*);

читает форматированные данные из файла. Строка форматирования строится аналогично функции **fprintf**.

Функция

void rewind (FILE * *указатель_на _файл*);

устанавливает указатель текущей позиции выделенного файла в начало файла.

Функция

int ferror (FILE * *указатель_на _файл*);

определяет, произошла ли ошибка во время работы с файлом.

Функция

size_t fwrite (const void * *записываемое_данное*,
size_t *размер_элемента*, **size_t** *число_элементов*,
FILE * *указатель_на _файл*);

записывает в файл заданное число данных определенного размера. Размер данных задается в байтах. Тип size_t определяется как целое значение без знака.

Функция

size_t fread (void * *считываемое_данное*,
size_t *размер_элемента*, **size_t** *число_элементов*,
FILE * *указатель_на _файл*);

считывает из файла указанное число данных заданного размера. Размер задается в байтах. Функция возвращает число прочитанных элементов. Если число прочитанных элементов не равно заданному, то при чтении возникла ошибка или встретился конец файла.

Функция

int fileno (FILE * *указатель_на _файл*);

возвращает значение *дескриптора* (логический номер файла для заданного потока) указанного файла.

Функция

long filelength (int *дескриптор*);

возвращает длину файла с соответствующим дескриптором в байтах.

Функция

int fseek (FILE * *указатель_на _файл*, **long int** *число_байт*,
int *точка_отсчета*);

устанавливает указатель в заданную позицию. Заданное количество байт отсчитывается от позиции, которая задаётся следующими макросами: **SEEK_SET** или **0** – начало файла, **SEEK_CUR** или **1** – текущая позиция, **SEEK_END** или **2** – конец файла.

6.4. Пример выполнения задания

Написать программу формирования файла, содержащего экзаменационную ведомость студентов: фамилию и оценки по математике и программированию. Предусмотреть возможность чтения из файла. Вывести список студентов, сдавших экзамен по программированию с оценкой 9, и записать эту информацию в текстовой файл.

```
#include <iostream.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <string.h>

FILE *fl;
typedef struct
{
   char fio[30];
   unsigned char matem;
   unsigned char oaip;
} TStudent;

TStudent stud[30];          // Массив структур
char name[20];              // Имя файла
int nst = 0;                // Число введенных структур
int Menu();                 // Создание меню
void Nnf();                 // Ввод имени файла
void Newf();                // Создание нового файла
void Spisok();              // Формирование файла
void Opf();                 // Открытие файла
void Resc();                // Вывод результата на экран
void Resf();                // Вывод результата в файл
```

```
int main()
{
   while (true)
   {
      switch (Menu())
      {
         case 1: Nnf();       break;
         case 2: Newf();      break;
         case 3: Spisok();    break;
         case 4: Opf();       break;
         case 5: Resc();      break;
         case 6: Resf();      break;
         case 7: return 0;
        default: puts("Viberite pravilno!");
      }
      puts ("Press any key to continue");
      getch ();              // Ожидание нажатия любой клавиши
      system ("cls");        // Очистка экрана
   }
}

int Menu()                   // Меню
{
   cout << "VIBERITE:" << endl;
   cout << "1. Vvod file name" << endl;
   cout << "2. New file" << endl;
   cout << "3. Vvesti spisok" << endl;
   cout << "4. Open file" << endl;
   cout << "5. Vivesti result" << endl;
   cout << "6. Vivesti v fail" << endl;
   cout << "7. Exit" << endl;
   int i;
   cin >> i;                          // Ввод выбранного пункта меню
   return i;
}

void Nnf()                           // Ввод имени файла
{
   cout << "Vvedite file name" << endl;
   cin >> name;
}

void Newf()                          // Создание нового файла
{
```

```
   if ((fl = fopen(name,"wb")) == NULL)
      {
         cout << "Oshibka pri sozdanii" << endl;
         exit(1);
      }
      cout << "OK" << endl;
      fclose(fl);
}

void Spisok()                    // Ввод данных в файла
{
   if ((fl = fopen(name,"rb+")) == NULL)
      {
         cout << "Oshibka pri sozdanii" << endl;
         exit(1);
      }
   cout << "Vvedite chislo studentov" << endl;
   cin >> nst;
   for (int i=0; i<nst; i++)
      {
         cout << "Vvedite imya: ";
         cin >> stud[i].fio;
         cout << "Vvedite otcenku po matematike: ";
         cin >> stud[i].matem;
         cout << "Vvedite otcenku po OAiP: ";
         cin >> stud[i].oaip;
         fwrite (&stud[i], sizeof(TStudent), 1, fl);
      }
      fclose (fl);
}

void Opf()                       // Открытие бинарного файла
{
   if ((fl = fopen (name,"rb")) == NULL)
   {
      cout << "Oshibka pri otkritii" << endl;
      exit(1);
   }
   nst = 0;
   TStudent std;
   while (true)
      {
```

```
            int  nwrt = fread (&std, sizeof(TStudent), 1, fl);
            if (nwrt != 1) break;
            stud[nst] = std;
            cout << stud[nst].fio << "  " << stud[nst].matem << "  "
                 << stud[nst].oaip << endl;
            nst++;
         }
      fclose(fl);
}

void Resc()                    // Вывод результата на экран
{
   for (int i=0; i<nst; i++)
      if (stud[i].oaip == '9')
                cout << stud[i].fio << endl;
}

void Resf()                    // Вывод результата в текстовый файл
{
   char namet[30];
   FILE *ft;
   cout << "Vvedite imya faila" << endl;
   cin >> namet;
   if ((ft = fopen (namet,"w")) == NULL)
      {
         cout << "Oshibka pri sozdanii" << endl;
         exit(1);
      }
   char s[80];
   for (int i=0; i<nst; i++)
      if (stud[i].oaip == '9')
      {
         strcpy (s, stud[i].fio);
         strcat (s, "\n");              // Добавление разделителя строк
         fputs (s, ft);
      }
   fclose(ft);
}
```

6.5. Индивидуальные задания

Составить программу формирования файла, содержащего данные согласно варианту индивидуального задания. В программе предусмотреть сохра-

нение вводимых данных в файле и возможность чтения из ранее сохраненного файла. Вывести результаты на экран и в текстовой файл.

1. Список товаров, имеющихся на складе, включает в себя наименование товара, количество единиц товара, цену единицы товара. Вывести список товаров, стоимость которых превышает *1 000 000* рублей.

2. Для получения места в общежитии формируется список студентов, который включает ФИО студента, группу, средний балл, доход на члена семьи. Вывести информацию о студентах, у которых доход на члена семьи менее двух минимальных зарплат.

3. В справочной автовокзала имеется расписание движения автобусов. Для каждого рейса указаны его номер, пункт назначения, время отправления и прибытия. Вывести информацию о рейсах, которыми можно воспользоваться для прибытия в пункт назначения раньше заданного времени.

4. Информация о сотрудниках фирмы включает ФИО, количество проработанных часов за месяц, почасовой тариф. Вывести размер заработной платы каждого сотрудника фирмы.

5. Информация об участниках спортивных соревнований содержит название команды, ФИО игрока, возраст. Вывести информацию о спортсменах, возраст которых не достиг *18* лет.

6. Для книг, хранящихся в библиотеке, указаны автор, название, год издания, количество страниц. Вывести список книг, изданных после заданного года.

7. На заводе производится несколько наименований деталей. Сведения о деталях, выпущенных за последний месяц, включают номер цеха, наименование детали, количество выпущенных деталей. Вывести информацию о продукции, выпущенной заданным цехом.

8. Информация о сотрудниках предприятия содержит ФИО, номер отдела, должность, дату начала работы. Вывести список сотрудников, проработавших на предприятии более *20* лет.

9. Ведомость абитуриентов содержит ФИО, город проживания, суммарный балл. Вывести информацию об абитуриентах, проживающих в городе *Минске* и имеющих балл *больше 220*.

10. В справочной аэропорта имеется расписание вылета самолетов на следующие сутки. Для каждого рейса указаны номер рейса, пункт назначения, время вылета. Вывести для заданного пункта назначения номера рейсов и время вылета самолетов.

11. У администратора железнодорожных касс хранится информация о свободных местах в поездах. Информация представлена в следующем виде: номер поезда, пункт назначения, время отправления, число свободных мест. Вывести информацию о поездах, в которых имеются свободные места до заданного пункта назначения.

12. Ведомость студентов, сдававших сессию, содержит ФИО и оценки по четырем предметам. Вывести список студентов, сдавших сессию со средним баллом *больше 7*.

13. В радиоателье хранятся квитанции о сданных в ремонт телевизорах. Каждая квитанция содержит марку телевизора, дату приемки в ремонт, состояние готовности заказа (выполнен, не выполнен). Вывести информацию о невыполненных на текущий момент заказах.

14. На АТС информация о разговорах содержит номер телефона абонента, время разговора и тариф. Вывести для заданного абонента сумму оплаты за разговоры.

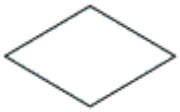
15. В магазине составлен список людей, которым выдана карта постоянного покупателя. Каждая запись этого списка содержит номер карточки, ФИО, предоставляемую скидку. Вывести информацию о покупателях, имеющих *10 %*-ную скидку в магазине.

ПРИЛОЖЕНИЕ 1
БЛОК-СХЕМА АЛГОРИТМА

Благодаря своей наглядности данный способ записи алгоритмов получил наибольшее распространение. При построении блок-схемы алгоритм изображается геометрическими фигурами (блоками), связанными линиями (направление потока информации) со стрелками. Внутри блоков записывается последовательность действий.

В таблице 1 приведены виды и назначение основных блоков, применяемых при составлении блок-схемы алгоритма.

Таблица 1. Виды и назначение основных блоков

Наименование блока	**0бозначение блока**	**Функции блока**
процесс		выполнение действий, изменяющих значение, форму представления или расположение данных
ввод-вывод		ввод или вывод данных
условие		выбор направления выполнения алгоритма в зависимости от заданного в блоке условия
предопределенный процесс		вызов подпрограммы
пуск-останов		начало или конец описания алгоритма
цикл с параметром		цикл с параметром; внутри блока указывается изменение параметра (переменной цикла), *например*, i=1,10
внутристраничный соединитель		переход между блоками в пределах данной страницы
межстраничный соединитель		переход между блоками, расположенными на разных листах

Наименование блока	0бозначение блока	Функции блока
комментарий		пояснение действий, указанных в блоке

Правила оформления блок-схем:

- в пределах одной схемы блоки изображают *одинаковых размеров*;
- все *блоки нумеруются* (номер ставится в верхнем левом углу блока с разрывом линии);
- *линии*, соединяющие блоки и указывающие последовательность связей между ними, проводятся *параллельно* линиям рамки;
- стрелка в конце линии *не ставится*, если линия направлена *слева направо* или *сверху вниз*;
- из блока «*условие*» могут выходить *две линии*, из других блоков — только *одна линия*;
- если схема занимает *более одного листа*, то в случае разрыва линии используется *межстраничный соединитель*;
- внутри каждого соединителя указывается *номер блока* (откуда или куда направлена соединительная линия). Внутри *межстраничного* соединителя в первой строке указывается *номер листа*, во второй —*номер* блока куда или откуда передается управление.

В зависимости от поставленной задачи выделяют три **основных вида алгоритмов**:

- линейный;
- разветвляющийся;
- циклический.

Линейным называется алгоритм, в котором все действия, указанные в блоках, выполняются по порядку их следования.

Пример блок-схемы алгоритма вычисления площадей прямоугольника и квадрата (рис. 7).

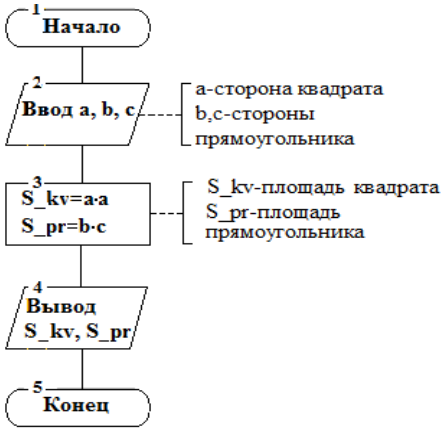


Рис. 7 - Блок-схема линейного алгоритма

Разветвляющимся называют алгоритм, в котором в зависимости от значения условия (выполняется или не выполняется) изменяется последовательность выполнения действий алгоритма.

Пример блок-схемы алгоритма определения принадлежности точки с координатами (x, y) номеру сектора (рис. 8).

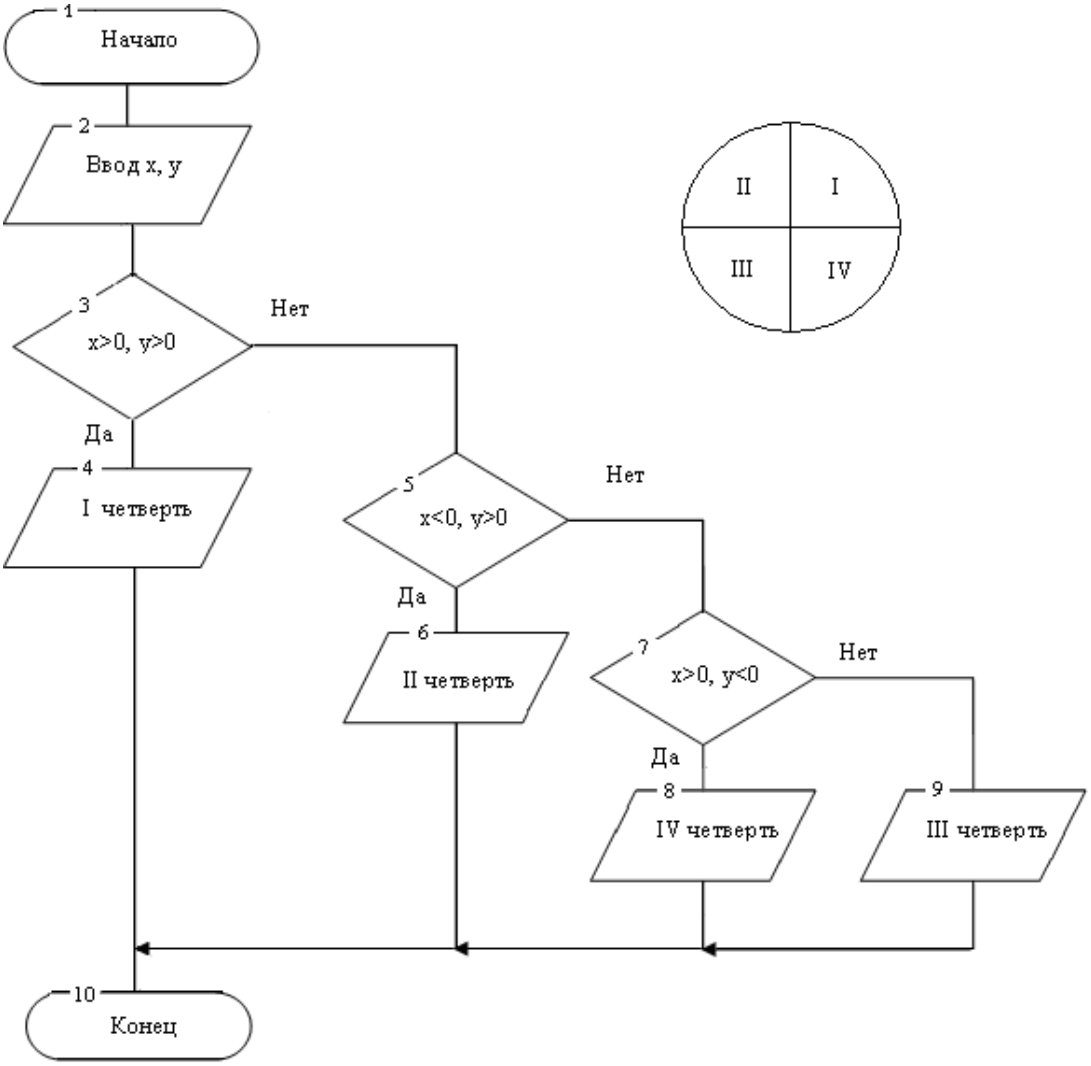


Рис. 8 - Блок-схема разветвляющегося алгоритма

Циклическим называется алгоритм, в котором некоторая последовательность действий повторяется определенное количество раз.

Тело цикла – действия, выполняемые в цикле.

Циклические алгоритмы могут быть:

- *с предусловием* – условие, при котором выполняется тело цикла, задается в начале цикла (рис. 9);
- *с постусловием* – условие, при котором выполняется тело цикла, задается в конце цикла (рис. 10);
- *с параметром* — в начале цикла задается правило изменения его параметра (рис. 11).

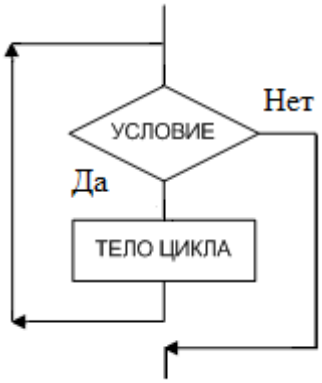


Рис. 9 - Цикл с предусловием

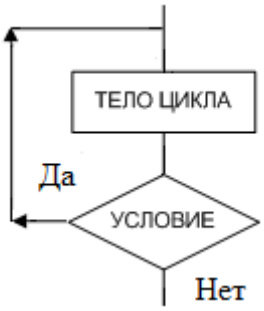


Рис. 10 - Цикл с постусловием

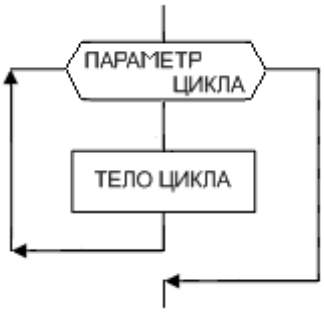


Рис. 11 - Цикл с параметром

Пример блок-схемы циклического алгоритма нахождения НОД (наибольшего общего делителя) двух значений a и b.

Правило нахождения НОД (a, b) формулируется следующим образом:

1. определяется наибольшее из значений a и b — max (a, b);
2. из наибольшего значения вычитается оставшееся значение;
3. п.п. 1 и 2 повторяются до тех пор, пока значения a и b не станут равными.

Полученное значение будет НОД (a, b).

Решение:

Предположим, что a=20 и b=15.

Тогда:

max (a, b) = max (20, 15) = 20 → a=20-15=5 b=15
max (a, b) = max (5, 15) = 15 → a=5 b=15-5=10
max (a, b) = max (5, 10) = 10 → a=5 b=10-5=5

a = b => **НОД (a, b) =5**

Составим блок-схему алгоритма нахождения НОД (a, b) (рис. 12):

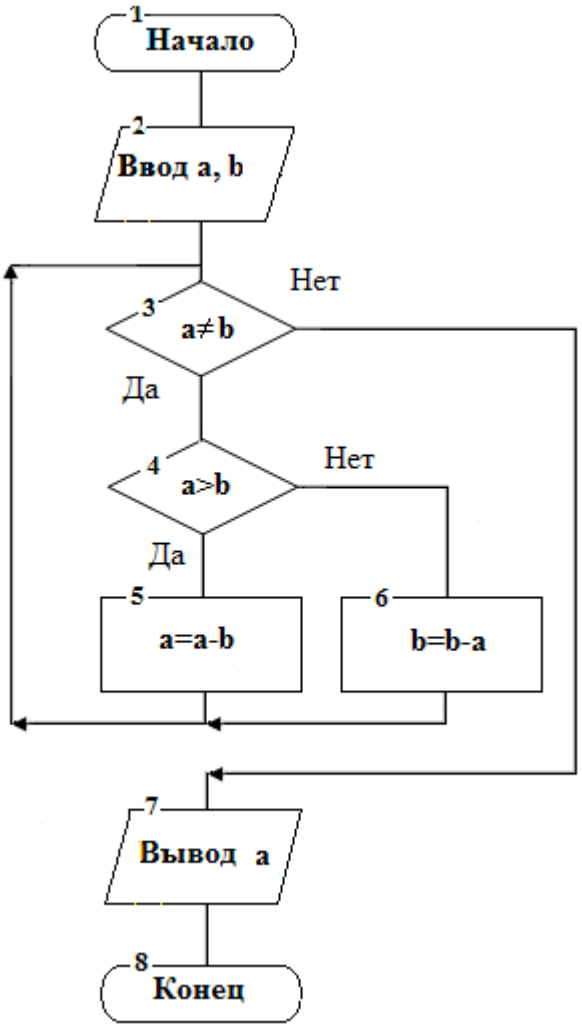


Рис. 12 Блок-схема алгоритма нахождения НОД (a, b).

Для реализации алгоритма на компьютере необходимо описать его на языке программирования.

ПРИЛОЖЕНИЕ 2
ПРИМЕР ОФОРМЛЕНИЯ

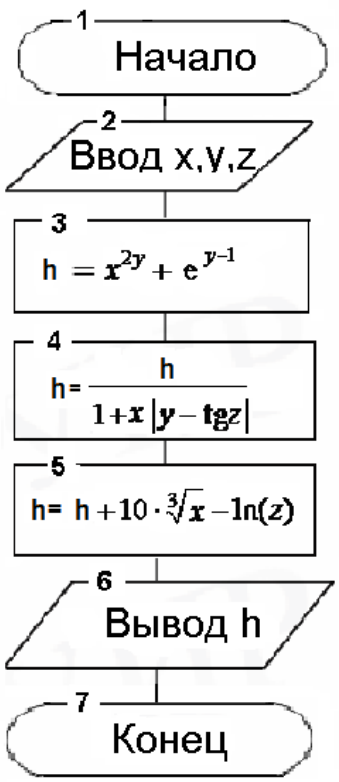
Задание 1

Составить программу вычисления линейного арифметического выражения

$$h = \frac{x^{2y} + e^{y-1}}{1 + x|y - tgz|} + 10 \cdot \sqrt[3]{x} - \ln(z).$$

При x = 2,45; y = – 0,423·10^{-2}; z = 1,232·10^{3} **ответ:** h = 6,9465.

Блок-схема алгоритма



Код программы

```
#include <iostream.h>
#include <math.h>

int main ()
    {
     double x, y, z, h;

     cout << "Vvedite x:  ";
     cin >> x;
```

```
 cout << "Vvedite y:  ";
 cin >> y;
 cout << "Vvedite z:  ";
 cin >> z;
 h = pow(x, 2*y) + exp(y-1);
 h / = 1+x * fabs(y - tan(z));
 h + = 10 * pow(x, 1/3.) - log(z);
 cout << "Result h= " << h << endl;
 return 0;
}
```

Результаты выполнения программы

ЛИТЕРАТУРА

1. Основы алгоритмизации и программирования. Язык Си: учеб. пособие // М. П. Батура [и др.] / – Минск: БГУИР, 2007.

2. Основы алгоритмизации и программирования: конспект лекций для студ. всех спец. и всех форм обуч. БГУИР / В. Л. Бусько [и др.] /. – Минск: БГУИР, 2004.

3. Вирт, Н. Алгоритмы и структуры данных/ Н. Вирт– СПб.: Невский диалект, 2005.

4. Кнут, Д. Искусство программирования: В 3 т. Т. 3/ Д. Кнут Сортировка и поиск / – М.: Вильямс, 2000.

5. Хопкрофт, Дж. Структуры данных и алгоритмы / Дж. Хопкрофт, Дж.Ульман, А. Ахо– М.: Вильямс, 2003.

6. Павловская, Т. А. С/С++. Программирование на языке высокого уровня / Павловская Т. А. – СПб.: Питер, 2004.

7. Павловская Т. А. С++. Объектно-ориентированное программирование: практикум / Т. А.Павловская, Ю. А. Щупак – СПб.: Питер, 2004.

8. Керниган Б. Язык программирования Си / Б. Керниган, Д. Ритчи – М.: Финансы и статистика, 1992.

9. Демидович Е. М. Основы алгоритмизации и программирования. Язык СИ / Е. М. Демидович – Минск: Бестпринт, 2001.

10. Страуструп Б. Язык программирования С++ / Б. Страуструп – СПб.: БИНОМ, 1999.