Системы реального времени

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ КОНТРОЛЬНОЙ РАБОТЫ

Изучение дисциплины «Системы реального времени» предусмотрено Государственным образовательным стандартом высшего профессионального образования, регламентирующим процесс подготовки инженеров по специальности 010503 «Математическое обеспечение и администрирование информационных систем». В соответствии с этими же стандартами данная дисциплина должна быть обеспечена практикумом.

Представляется целесообразным в рамках контрольной работы по дисциплине «Системы реального времени» изучить основы проектирования систем реального времени. Так как системы реального времени в основном реализованы на языке программирования C/C++, то задания контрольной работы необходимо реализовать на языке высокого уровня C++ в соответствующей среде разработки. Задания контрольной работы будут способствовать закреплению знаний по соответствующим разделам теоретической части курса, более глубокому пониманию студентами основных вопросов компиляции и трансляции программ, а также генерации машинного исполняемого кода.

Контрольная работа выполняется в соответствии с требованиями оформления контрольных работ, установленных кафедрой Высшей математики и информационных технологий.

Контрольная работа выполняется студентом самостоятельно. Вариант заданий определяется согласно списка студентов группы.

Отчетный документ необходимо оформить в печатном виде на листах формата А4.

Требование к оформлению:

- теоретическая часть;
- условие задачи;
- блок-схема алгоритма решения задачи;
- текст программы;
- результат выполнения программы фиксируется снимком экрана.

Задание 1. Арифметические операции и математические функции языка C++

1. Цель работы

Целью лабораторной работы является получение практических навыков в программировании выражений и использовании математических функций библиотеки языка С.

2. Темы для предварительной проработки

арифметические операции
порядок выполнения операций
стандартные математические функции

3. Задание для выполнения

Составьте программу, которая подсчитывает и выводит значение t1 и t2 по формулам, которые приведены в Вашем варианте индивидуального задания. Определите области допустимых значений параметров формул и задайте произвольные значения из этих областей. Параметры, которые имеют имена: n и m - целые, остальные параметры - с плавающей точкой. Значения параметров с именами x и y должны вводиться с клавиатуры, значения остальных - задаваться как начальные значения при объявлении соответствующих переменных. Допускается (и даже желательно) упростить / разложить формулы для того, чтобы обеспечить минимизацию объема вычислений.

4. Разработка алгоритма решения.

4.1.Основной алгоритм

Алгоритм решения задачи - линейный и состоит из:
ввода значений x и y;

- вычисления значения ***t1***;
- вычисления значения ***t2***;
- вывода значений ***t1*** и ***t2***.

4.2.Оптимизация алгоритма

Перед непосредственным программированием алгоритма проанализируем, как в нем можно изменить объем вычислений.

Выражение ***ax*** встречается один раз в первой формуле и дважды - во второй. Следовательно, можно один раз произвести умножение ***a*x***, а потом использовать этот результат.

Во второй формуле дважды встречается умножение квадратного корня на тангенс - это вычисление можно так же сделать один раз.

Выражение c^2-b^2 можно разложить на ***(c+b)(c-b)***. До разложения в выражении было две операции умножения (возведение в степень 2) и одна - сложения. После разложения - два сложения и одно умножение, что выгоднее для вычислений.

4.3. Ограничения на значения параметров

Аргумент функции, которую вычисляет логарифм, не может быть 0 или меньше. Отсюда вытекают требования к значениям:

$a * x + b > 0; y * x + d > 0$

Аргумент функции извлечения квадратного корня не может быть меньше 0, отсюда:

$c2 - b2 >= 0$

В знаменателе выражения не может быть 0, отсюда:

$a \neq 0; b \neq 0; c \neq 0; y \neq 0$

кроме того:

$\sqrt{c^2 - b^2}\ tg\ ax \neq 2$

4.4. Определение переменных программы и создание проекта

Для решения задачи нам понадобятся переменные для представления каждого параметра формул - ***a, b, c, d, x, y*** и результатов - ***t1, t2***. Кроме того, придется ввести

дополнительную переменную ***ax*** для хранения промежуточного результата , необходимого для оптимизации. Тип всех переменных - ***double.***

Создание проекта программного приложения начинается с диалогового окна. Для создания шаблона окна выполним команду File-New. В появившемся диалоговом окне, переходим на вкладку Projects (см. рис. 1) и выбираем Win32 Console Application (exe).

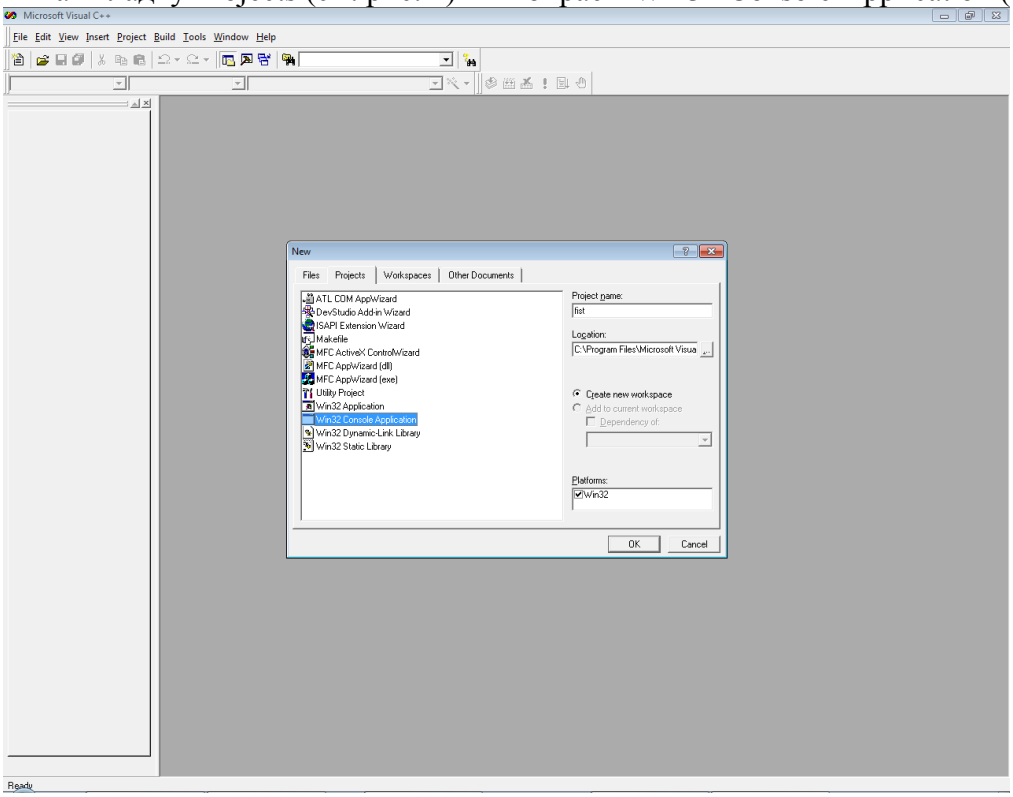


Рис.1.

В строку Project Name (Имя проекта) вводим имя собственного проекта. Далее в следующем диалоговом окне (рис. 2.) всё оставляем без изменений.

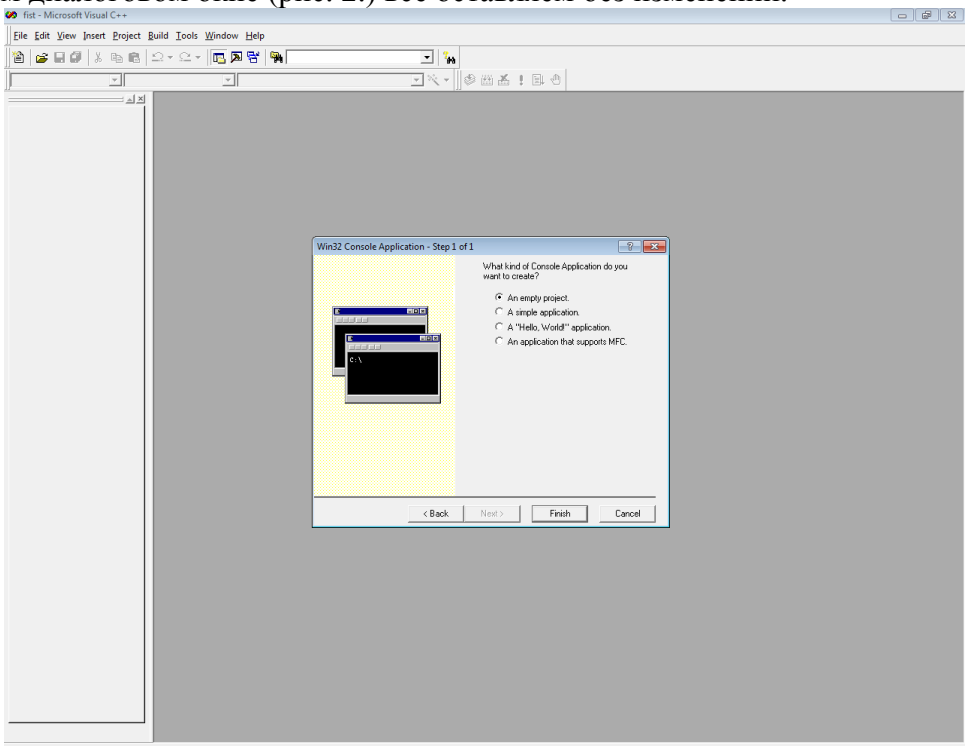


Рис.2.

Далее выполним команду File-New. В появившемся диалоговом окне, переходим на вкладку Files (см. рис. 3) и выбираем C++ Source File (exe). В строку File Name (Имя файла) вводим имя своей программы.

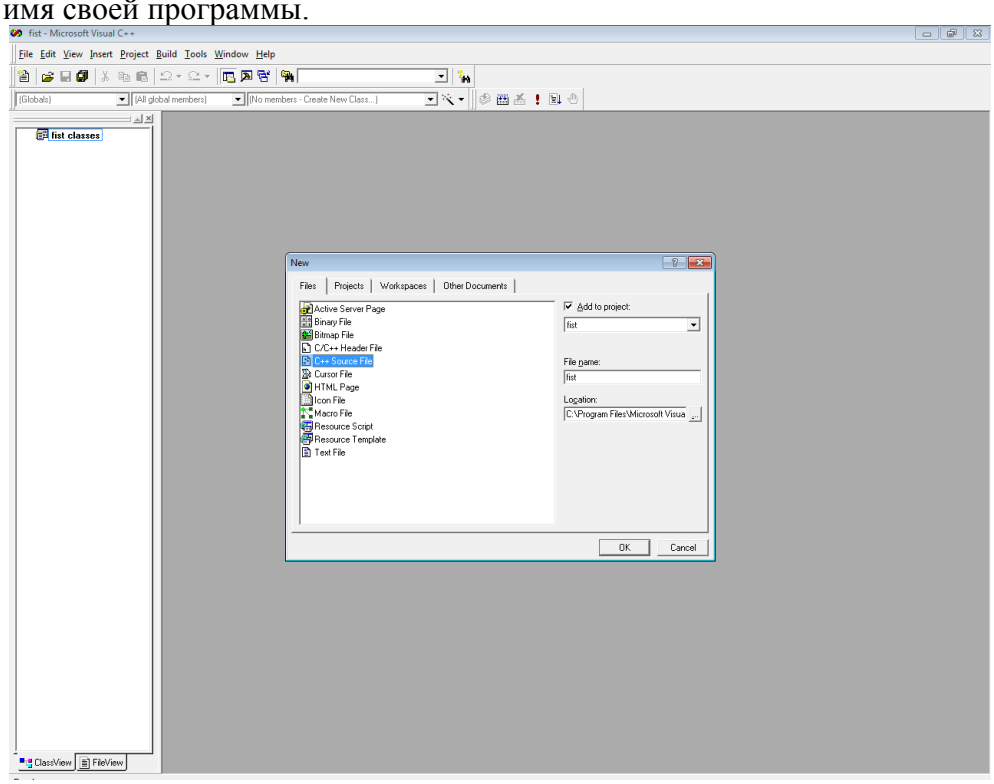


Рис. 3.

В появившемся окне (рис.4) вводим текст программы.

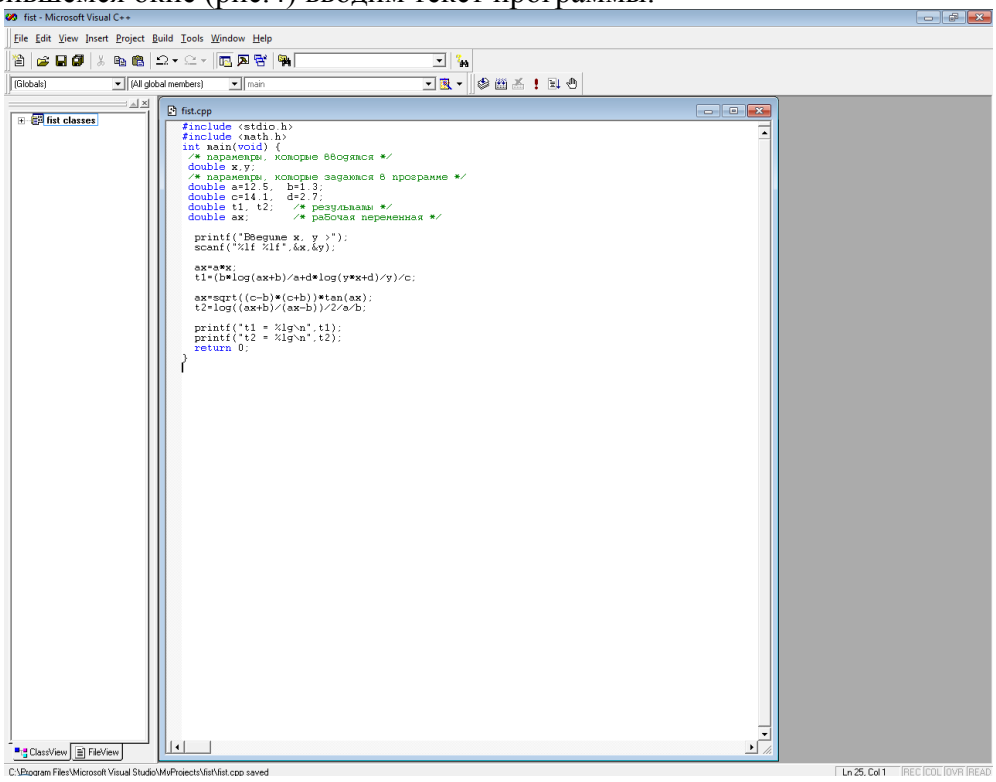


Рис.4.

4.5. Разработка текста программы

Программа начинается с включения файлов:

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

в которых находятся описания функций ввода- вывода и математических функций соответственно.

Далее открываем главную функцию:

```
int main(void)
```

Включаем описания переменных (см. п.5.2):

```
double x,y;
    double a=12.5,  b=1.3;
    double c=14.1,  d=2.7;
    double t1, t2;
    double ax;
```

Вводятся значения для переменных ***x*** и ***y***:

```
    printf("Введите x, y >");
    scanf("%lf %lf",&x,&y);
```

Далее вычисляется первое промежуточное значение:

```
    ax=a*x;
```

и первый результат:

```
    t1=(b*log(ax+b)/a+d*log(y*x+d)/y)/c;
```

Вычисляется второй промежуточный результат:

```
    ax=sqrt((c-b)*(c+b))*tan(ax);
```

и вычисляется второй окончательный результат:

```
    t2=log((ax+b)/(ax-b))/2/a/b;
```

Полученные результаты выводятся на экран:

```
    printf("t1 = %lg\n",t1); printf("t2 = %lg\n",t2);
```

Полный текст программы приводится ниже.

```
#include <stdio.h>
#include <math.h>
int main(void) {
 /* параметры, которые вводятся */
 double x,y;
 /* параметры, которые задаются в программе */
 double a=12.5,  b=1.3;
 double c=14.1,  d=2.7;
 double t1, t2;   /* результаты */
 double ax;       /* рабочая переменная */

  printf("Введите x, y >");
  scanf("%lf %lf",&x,&y);

  ax=a*x;
  t1=(b*log(ax+b)/a+d*log(y*x+d)/y)/c;

  ax=sqrt((c-b)*(c+b))*tan(ax);
  t2=log((ax+b)/(ax-b))/2/a/b;

  printf("t1 = %lg\n",t1);
  printf("t2 = %lg\n",t2);
  return 0;
}
```

4.6. Отладка и компиляция программы

Компиляция проекта выполняется из меню Build (рис.5.) или комбинацией клавиш Crtl+F7.

При отладке программы можно проверять правильность выполнения каждой операции. Для этого сложные операторы-выражения, разбиваются на последовательность операторов-выражений, в каждом из которых выполняется только одна операция.

4.7. Результаты работы программы

Запуск проекта выполняется также из меню Build (рис.6.).

При работе программы на экран было выдано следующее:

```
Введите x, y >3.3 1.1
t1 = 0.348897
```

t2 = 0.0133405

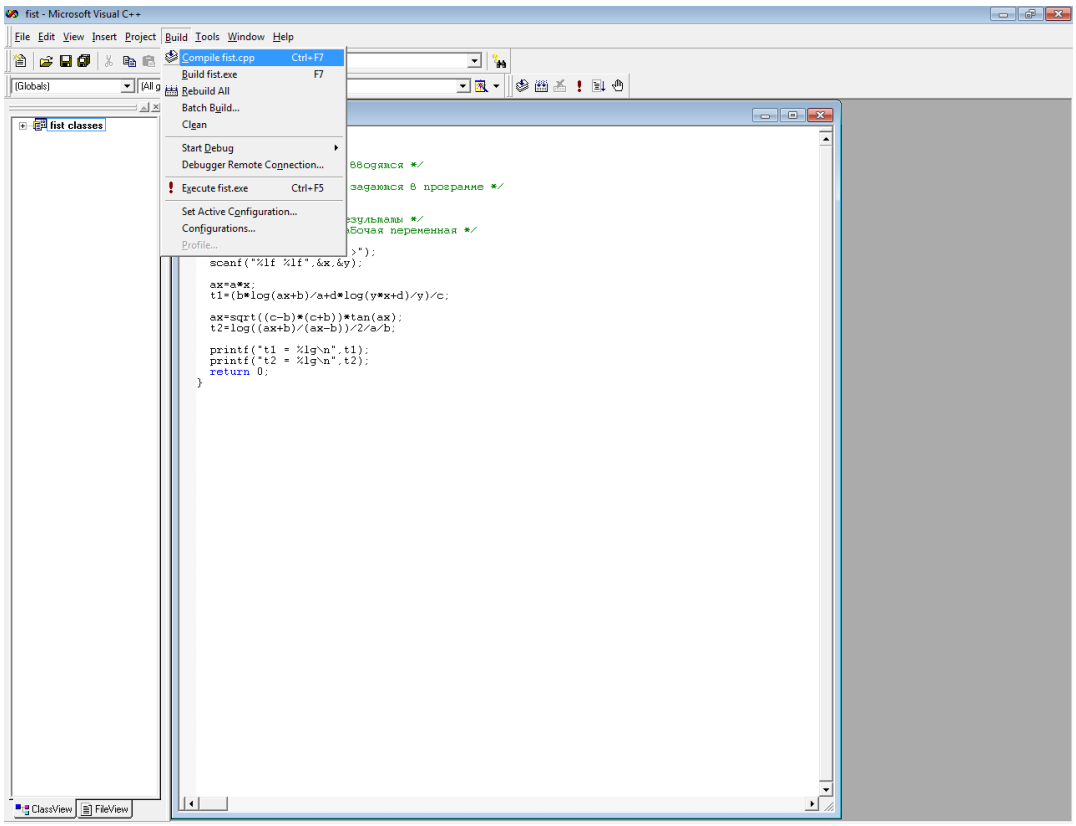


Рис.5.

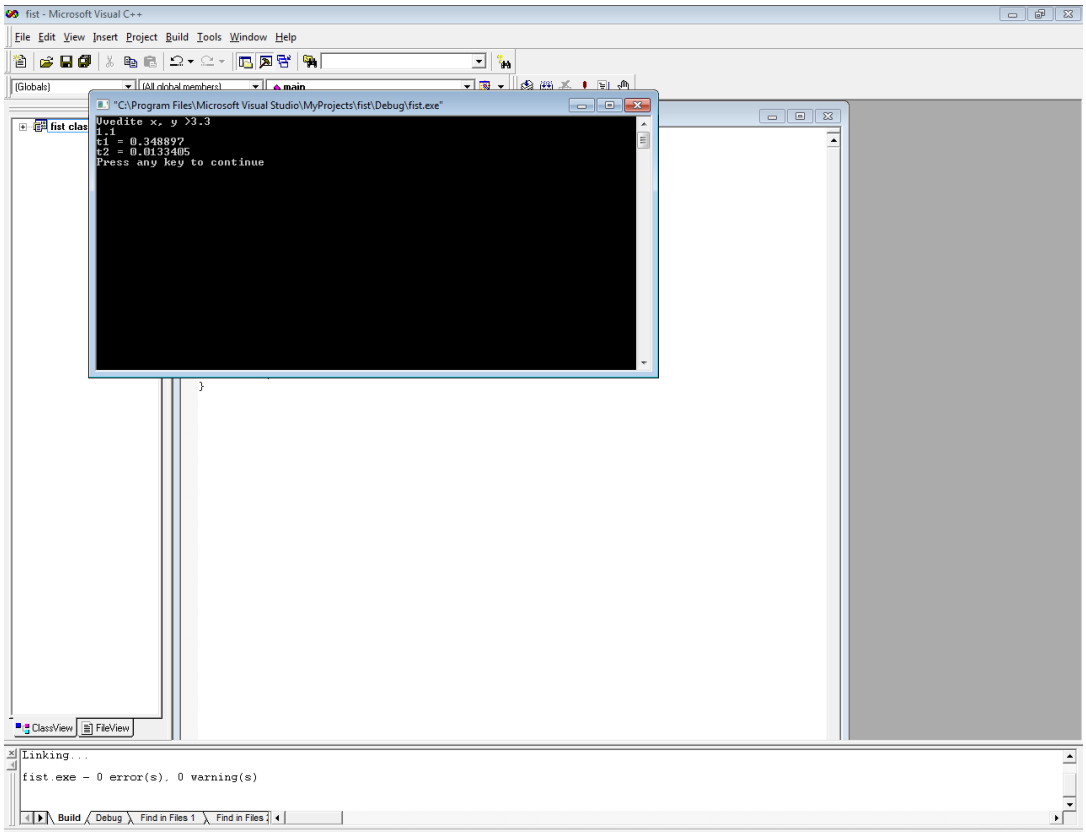


Рис. 6.

5. Варианты индивидуальных заданий

Вариант 1

$$t1 = \frac{ax}{y} + \frac{b}{y^2} lg(yx+c)$$

$$t2=\frac{1}{2ab}\ln\frac{\sqrt{c^2-b^2}\,tg\,\alpha x+2}{\sqrt{c^2-b^2}\,tg\,\alpha x-2}$$

Вариант 2

$$t1=\frac{1}{c}\left[\frac{b}{a}\ln(ax+b)+\frac{d}{y}\ln(yx+d)\right]$$

$$t2=\frac{1}{a(n-1)}\,\frac{\sin\alpha x}{\cos^{n-1}\alpha x}$$

Вариант 3

$$t1=\frac{1}{c}\left(\frac{1}{ax+b}+\frac{y}{c}\ln\frac{yx+a}{ax+b}\right)$$

$$t2=\frac{\sin\alpha x}{2a\cos^2 x}+\frac{1}{2a}\ln tg\frac{\alpha x}{2}$$

Вариант 4

$$t1=\frac{b}{(a-b)(b+x)}-\frac{a}{(a-b)^2}\ln\frac{a+x}{b+x}$$

$$t2=\frac{1}{a}\left(\ln tg\,\frac{\alpha x}{2}-\frac{1}{\sin\alpha x}\right)$$

Вариант 5

$$t1=\frac{-1}{(a-b)^2}\left(\frac{1}{a+x}+\frac{1}{1+x}\right)+\frac{2}{(a-b)^3}\ln\frac{a+x}{b+x}$$

$$t2=-\frac{1}{2a}\left(\frac{\cos\alpha x}{\sin^2\alpha x}-\ln tg\frac{\alpha x}{2}\right)$$

Вариант 6

$$t1=\frac{1}{a}\left(\frac{-1}{(n-2)x^{n-2}}+\frac{b}{(n-1)x^{n-1}}\right)$$

$$t2=\frac{2x}{a^2}\sin\alpha x-\left(\frac{x^2}{a}-\frac{2}{a^3}\right)\cos\alpha x$$

Вариант 7

$$t1=\frac{1}{a^3}\left(\ln x+\frac{2b}{x}-\frac{b^2}{2x^2}\right)$$

$$t2=\frac{\cos\alpha x}{2a\sin^2\alpha x}+\frac{1}{2a}\ln tg\frac{\alpha x}{2}$$

Вариант 8

$$t1=\frac{1}{a^4}\left(\frac{x^3}{3}-3bx+3b^2\ln x+\frac{b^3}{x}\right)$$

$$t2=\frac{1}{1-\sin\alpha x}+\frac{1}{a}tg\frac{\alpha x}{2}$$

Вариант 9

$$t1=\frac{1}{b^2}\left(\ln\frac{y}{x}+\frac{\alpha x}{y}\right)$$

$$t2=-\frac{x}{a}tg\frac{ax}{2}+\frac{2}{a^2}ln\ sin\frac{ax}{2}$$

Вариант 10

$$t1=\frac{1}{b^3}\left(ln\frac{y}{x}-\frac{a^2x^2}{2y^2}\right)$$

$$t2=\frac{1}{a}tg\frac{ax}{2}+\frac{1}{a}ln\ tg\frac{ax}{2}$$

Вариант 11

$$t1=-a\left(\frac{1}{b^2y}+\frac{1}{ab^2x}-\frac{2}{b^3}ln\frac{y}{x}\right)$$

$$t2=\frac{1}{2a}ctg\frac{ax}{2}+\frac{1}{6a}ctg^3\frac{ax}{2}$$

Вариант 12

$$t1=\frac{1}{b^3}\left(a^2ln\frac{y}{x}+\frac{2ax}{y}+\frac{y^2}{2x^2}\right)$$

$$t2=\frac{1}{2\sqrt{2}a}+\frac{3sin^2ax-1}{sin^2ax-1}$$

Вариант 13

$$t1=\frac{1}{b^4}\left(3a^3ln\frac{y}{x}+\frac{a^2x}{y}-\frac{3ay}{x}\right)$$

$$t2=\frac{2b\ tg\frac{ax}{2}}{a\sqrt{b^2-c^2}}$$

Вариант 14

$$t1=\frac{1}{2(n-1)x^{n-1}}+\frac{a}{2nx^n}$$

$$t2=\frac{1}{2a}tg^2ax+\frac{1}{a}ln\ cos\ ax$$

Задание 2. Условный оператор в языке С++

1. Цель работы

Целью лабораторной работы является получение практических навыков в работе с условным оператором и разветвленными алгоритмами в языке С.

2. Темы для предварительной проработки

- логические операции
- условный оператор

3. Пример решения задачи

Дать ответ, когда точка попадает в заштрихованную область на рис.7.

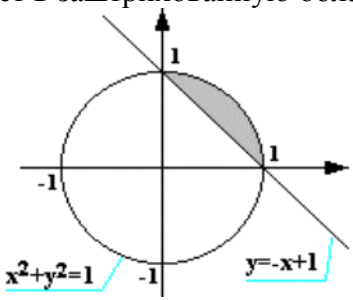


Рис.7.

3.1. Разработка алгоритма решения.

Первым шагом алгоритма должен быть ввод координат точки: x и y. Для большего удобства при анализе результатов можно вывести введенные значения на экран

Уравнение этой прямой:

y=-x+1

Уравнение окружности:

x2+y2=1

Для проверки попадания точки в заданную область нужно проверить условия того, что:

1). Точка лежит выше прямой или на ней, т.е.:

y>=-x+1

2). Точка лежит внутри окружности или на ней, т.е.:

x2+y2<=1

Точка лежит в области, если выполняются оба условия, если же не выполняется хотя бы одно из них, точка лежит вне области. Следовательно, эти условия должны быть объединены логической операцией "И".

Схема алгоритма приведена на рисунке ниже.

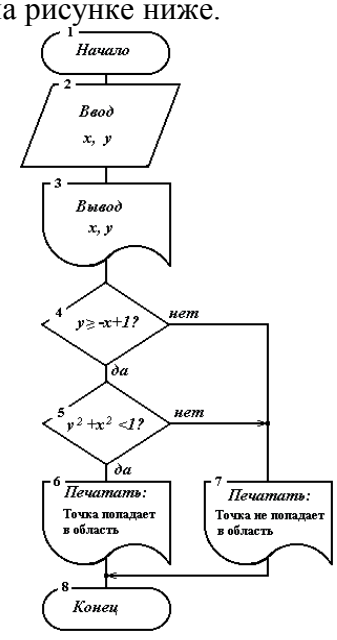


Рис.8.

3.2. Определение переменных программы

Для реализации алгоритма нам будут нужны только переменные для хранения значений координат x и y. В условиях задания не приведены требования к точности вычислений, рисунок представлен с довольно невысокой точностью, поэтому для этих переменных было бы достаточно типа float. Но в соответствии с общим стилем программирования на C выберем для них тип double.

3.3. Разработка текста программы

Текст программы начинается с включения фала:

```
#include <stdio.h>
```

т.к. нам обязательно понадобятся функции стандартного ввода-вывода, которые описаны в этом файле.

Далее идет заголовок и открытие главной функции:

```
int main(void) {
```

и объявление переменных, определенных в пункте 5.2.

```
double x, y;
```

Для каждой координаты выводится приглашение на ее ввод и вводится ее значение:

```
printf("Введите координату x >");
scanf("%lf",&x);
printf("Введите координату y >");
scanf("%lf",&y);
```

Введенные значения координат выводятся на экран:

```
printf("x=%6.3lf;  y=%6.3lf\n",x,y);
```

Далее идет проверка условий попадания точки в область. Оба условия проверяются одним выражением. Поскольку точка попадает в область, если выполняются оба условия вместе, условия в выражении соединены операцией "логическое И":

```
if ( (y>=1-x)&& (x*x+y*y<=1) )
```

Если значение логического выражения в условном операторе истинно, то выводится сообщение про попадание:

```
printf("Точка попадает в область\n");
```

В противном случае выводится сообщение про попадание:

```
else printf("Точка не попадает в область\n");
```

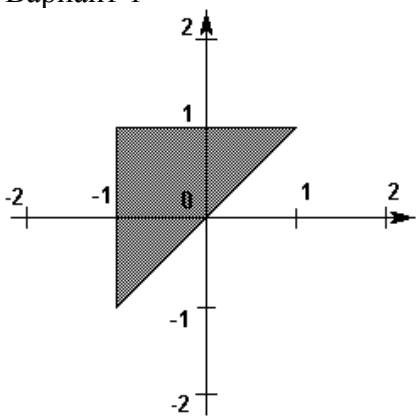
Полный текст программы приведен ниже.

```
#include <stdio.h>
int main(void) {
 double x, y; /* координаты точки */
  /* ввод координат */
  printf("Введите координату x >");
  scanf("%lf",&x);
  printf("Введите координату y >");
  scanf("%lf",&y);
  /* вывод только что введенных значений */
  printf("x=%6.3lf;  y=%6.3lf\n",x,y);
  /* проверка условий */
  if ( (y>=1-x)
       && (x*x+y*y<=1) )
     printf("Точка попадает в область\n");
  else printf("Точка не попадает в область\n");
  return 0;
}
```

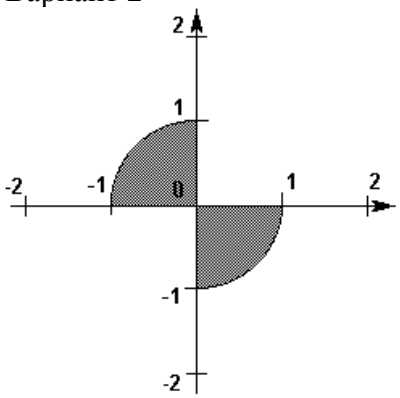
Компиляция и запуск проекта программы выполняется также как в первом задании.

4. Варианты индивидуальных заданий

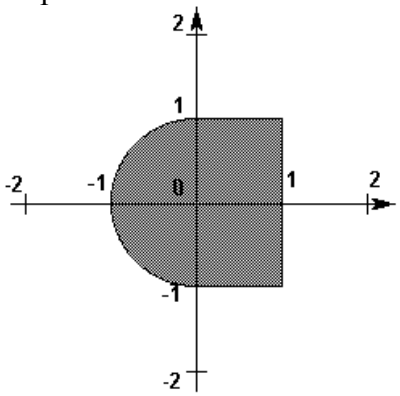
Вариант 1



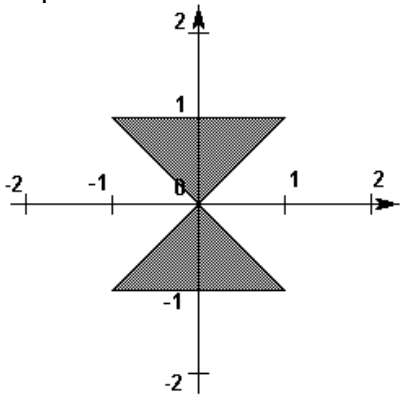
Вариант 2



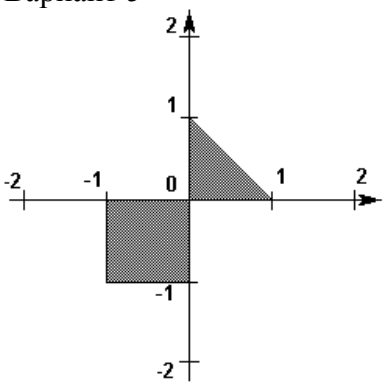
Вариант 3



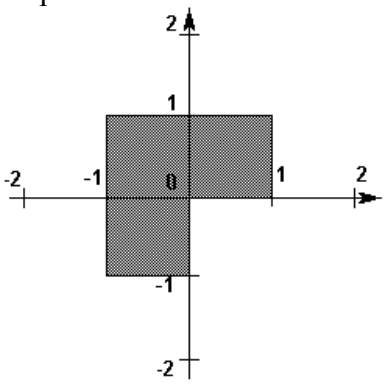
Вариант 4



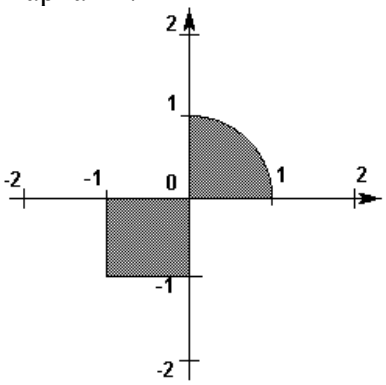
Вариант 5



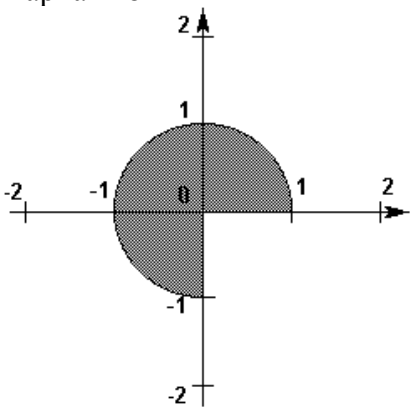
Вариант 6



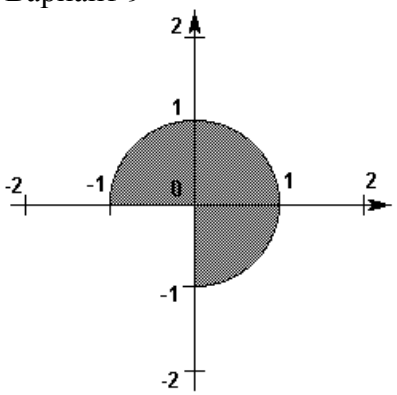
Вариант 7



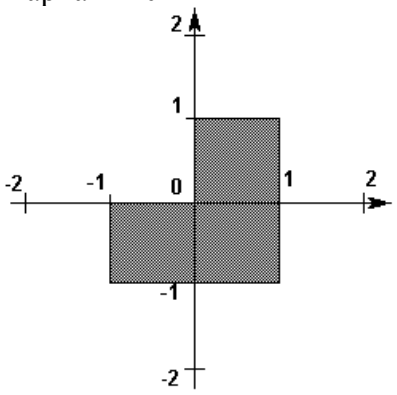
Вариант 8



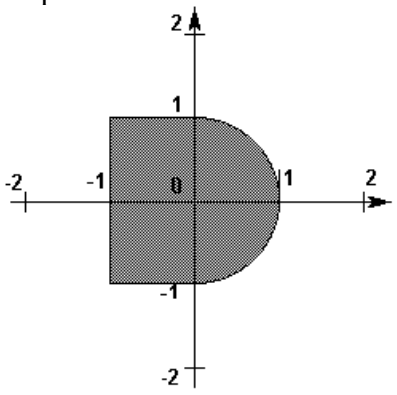
Вариант 9



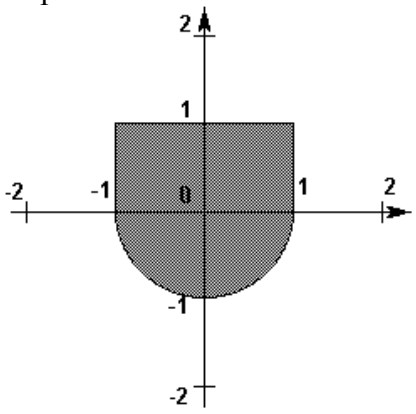
Вариант 10



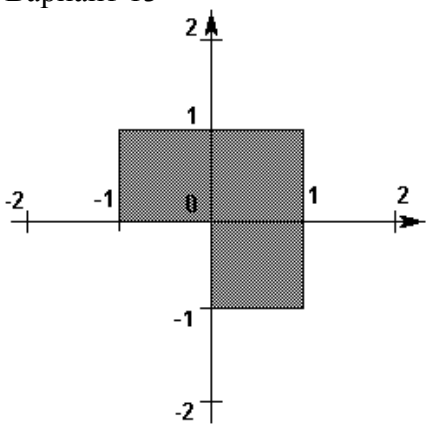
Вариант 11



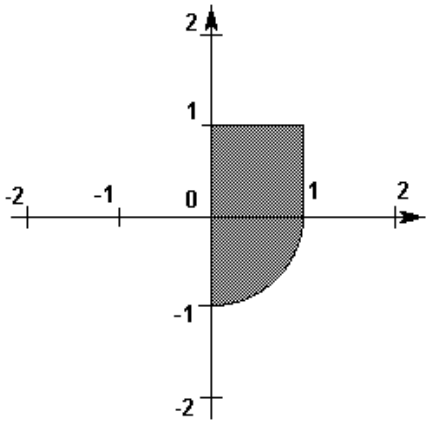
Вариант 12



Вариант 13



Вариант 14



Задание 3. Операторы цикла в языке С

1. Цель работы

Целью лабораторной работы является получение практических навыков в работе с операторами цикла языка С.

2. Темы для предварительной проработки

- Операторы цикла языка С.

3. Задания для выполнения

Для ряда, члены которого вычисляются по формуле, соответствующей Вашему индивидуальному заданию, подсчитать сумму членов ряда с точностью до 0.000001 и сумму первых 10 членов ряда. Если Вы считаете это необходимым, можете упростить или преобразовать выражение.

4. Пример решения задачи

Дан ряд.

$$a_n = (-1^n)\frac{n+1}{n^2+2^n}$$

Посчитать сумму членов этого ряда.

4.1. Разработка алгоритма решения.

Очевидно, что процесс подсчета суммы членов ряда должен быть итерационным: следует повторять вычисления по одной и той же формуле при значениях n=0, 1, 2, ... В каждой итерации цикла следует выполнять вычисления по заданной формуле для текущего значения n, т.е. подсчитывать очередной член ряда. Полученное значение следует прибавлять к переменной, которая представляет сумму. Эта переменная в каждой итерации будет содержать в себе сумму всех уже обработанных членов ряда, следовательно, ее начальное значение (когда ни один член еще не обработан) должно быть 0. После вычисления суммы при значении n=9 следует вывести значение суммы - это один из результатов программы в соответствии с заданием (берется значение 9, т.к. первый член ряда вычисляется при n=0, таким образом, девятый - при n=9). После вычисления каждого члена ряда (но до прибавления его значения к сумме) следует сравнить полученное значение с заданным пределом точности. Из-за того, что значение члена ряда может быть как положительным, так и отрицательным, при сравнении следует использовать абсолютное значение. Если абсолютное значения члена ряда не превышает предела точности, следует закончить вычисления: выйти из цикла напечатать значение суммы и завершить программу.

Алгоритм вычисления 2n

Для выполнения возведения в степень можно применить библиотечную функцию С pow(x,y) . Но есть возможность получения этого значения более эффективным способом. В каждой следующей итерации цикла значение этого выражения вдвое больше, чем в предыдущей. Так что, будет целесообразно выделить отдельную переменную для сохранения значения 2n. Ее начальное значение должно быть 20 = 1, а в конце каждой итерации оно должно удваиваться.

Алгоритм вычисления (-1)n

В этом случае также нецелесообразно применять функцию возведения в степень. Значение этого выражения будет 1 при четных значениях n и -1 - при нечетных. Так что, можно выделить переменную для сохранения значения этого выражения. Ее начальное значение должно быть (-1)0=1, а в конце каждой итерации оно должно менять знак на противоположный.

Схема алгоритма

Результирующий вариант схемы алгоритма показан на рисунке.

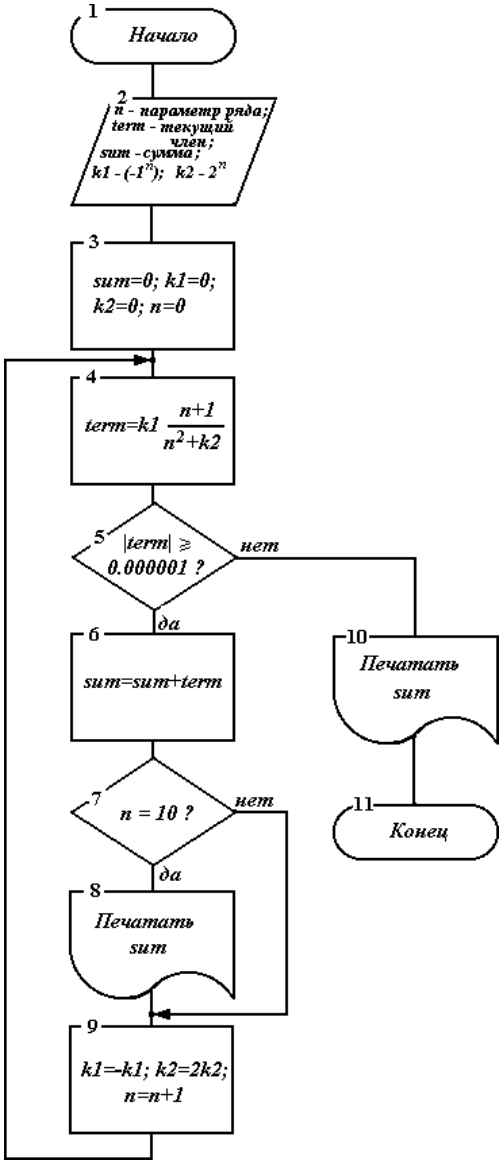


Рис.9.

4.2. Определение переменных программы

Все переменные нашей программы соответствуют переменным, которые введены в схеме алгоритма. Дополнительно определим их типы.

n - параметр ряда; по условиям задачи это - целое число; поскольку заранее неизвестно, сколько итераций цикла будет нужно для достижения предела точности, объявим его как "длинное целое":

```
long n;
```

Отметим, что хотя по условию задания n - целое, результаты выражений, в которых фигурирует n, будут иметь дробную часть. Преобразование типов можно выполнять непосредственно при вычислении выражения, но чтобы заранее исключить ошибки, введем еще одну переменную - dbln, которая будет представлять значение n как числа с плавающей точкой:

```
double dbln;
```

term - значение текущего члена ряда. Объявим его как:

```
double term;
```

sum - текущее значение суммы ряда с начальным значением 0. Объявим его как:

```
double sum=0;
```

k2 - переменная для сохранения текущего значения 2n с начальным значением 1 (см. пп.5.1.2). Она должна быть "длинным целым" - из тех же соображений, что и n:

```
long k2=1;
```

k1 - переменная для сохранения текущего значения (-1)n с начальным значением 1 (см. пп.5.1.3). Для нее достаточно быть "коротким целым":

```
short k1=1;
```

eps - переменная для представления заданного предела точности. Она должна иметь начальное (неизменное) значение 0.000001, поэтому может быть объявлена как const. Использование переменной для представления константы вместо употребления константы непосредственно в операторах программы может повысить гибкость программы:

```
const double eps=0.000001;
```

4.3. Разработка текста программы

Начинаем разработку текста программы с заголовка главной функции main():

```
int main(void)
```

Далее открывается тело функции, и в него включаются определения переменных (см. п.5.2). Определения переменных реализуют блок 2 схемы алгоритма. В дополнение к переменным, определенным в схеме алгоритма, объявляем рабочую переменную dbln и константу eps. Поскольку правила языка С позволяют присваивать переменным начальные значения, в объявлении переменных отчасти реализован также и блок 3. Блоки 4 - 9 схемы алгоритма образуют цикл. В языке С есть два "простых" оператора цикла: цикл с предусловием - while и цикл с постусловием - do-while. Но по схеме алгоритма видно, что выход из цикла происходит в середине тела цикла (блок 5): после вычисления члена ряда, но до добавления его значения к сумме. Следовательно, "простые" циклы применить нельзя. "Сложный" оператор цикла - for - является циклом с предусловием, но задавать условие в самом операторе цикла необязательно. К тому же этот оператор дает возможность определить действия, которые нужно выполнить до начала цикла и действия, которые выполняются в конце каждой итерации. Таким образом, открытие цикла будет иметь вид:

```
for (n=0; ; n++, k2*=2, k1=-k1) {
```

где первый параметр for - присваивание начального значения переменной n (остаток блока 3), второй параметр - условие выхода из цикла - пустой, третий параметр реализует блок 8 схемы алгоритма. В теле цикла содержится несколько отдельных действий, значит, будет несколько операторов, поэтому тело цикла берется в операторные скобки.

В теле цикла первый оператор:

```
dbln=n;
```

это действие не предусмотрено в схеме алгоритма.

Далее вычисляется значение текущего члена ряда:

```
term=k1*(dbln+1)/(dbln*dbln+k2);
```

этот оператор полностью реализует блок 4 и формулу из индивидуального задания за исключением того, что вместо операции возведения в степень 2 мы применяем умножение.

Следующее действие - проверка достижения предела точности - на схеме алгоритма представлена блоком 5 и выполняется условным оператором, который начинается с:

```
if (fabs(term)>=eps)
```

Следует отметить, что сравнивается абсолютное значение term, а функция для получения абсолютного значения переменной типа double - fabs(). Эта функция описана в файле math.h, так что мы должны включить этот файл в начало программы:

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

Продолжение условного оператора, действие, которое выполняется при выполнении условия, - добавление значения члена к сумме (блок 6):

```
sum+=term;
```

При невыполнении условия мы должны выйти из цикла, так что условный оператор требует и второй части:

```
else break;
```

По схеме алгоритма нам надо проверить, не достигла ли переменная n значения 9 (блок 7), и, если да, - печатать значение суммы (блок 8). Это реализуется условным оператором:

```
  if (n==9)
 printf("Сумма 10 членов ряда = %10.7lf\n",sum);
```

Поскольку при невыполнении условия не делается ничего, часть else для этого оператора не нужна.

При вызове функции printf() всегда возникает проблема форматизации вывода. Мы включили в формат текст, который поясняет вывод, а значение переменной sum выводим по спецификации %10.7lf. Тип спецификации соответствует типу переменной - double, а числовые параметры мы выбрали произвольно, поскольку в задании не оговорены требования к точности вывода. Перед точкой мы оставляем 2 позиции - для знака и целой части (которая должна быть 0), после точки - 7 позиций, что на 1 превышает точность заданного предела точности. Функция printf() описана в файле stdio.h, поэтому мы включаем этот файл в начало программы:

```
  #include <stdio.h>
```

Поскольку блок 9 схемы алгоритма реализован нами в заголовке цикла, то цикл на этом заканчивается, и мы ставим операторную скобку, закрывающую тело цикла.

Выход из цикла происходит по оператору break в составе условного оператора. После выхода из цикла мы должны напечатать окончательное значение суммы (блок 10), что мы и делаем оператором:

```
  printf("Полная сумма ряда = %10.7lf\n",sum);
```

При определении формата вывода работают те же соображения, что и в предыдущем случае.

Реализация алгоритма закончена, мы ставим операторную скобку, которая закрывает тело функции main().

Полный текст программы приведен ниже.

```
#include <stdio.h>
#include <math.h>
int main(void) {
 long n;             /* параметр ряда */
 double dbln;        /* параметр ряда в форме с плавающей точкой */
 double sum=0;       /* сумма членов ряда */
 double term;        /* значение текущего члена */
 const double eps=0.000001;  /* предел точности */
 long k2=1;          /* pow(2,n)*/
 short k1=1;         /* pow(-1,n)*/
  /* основной цикл; в модификациях вычисляются
     следующие значения pow(2,n) и pow(-1,n)*/
  for (n=0; ; n++, k2*=2, k1=-k1) {
   /* преобразование n в форму с плавающей точкой */
   dbln=n;
   /* вычисление очередного члена */
   term=k1*(dbln+1)/(dbln*dbln+k2);
   /* проверка достижения предела точности */
   if (fabs(term)>=eps)
     /* если не достигнут - накопление суммы */
     sum+=term;
   /* если достигнут - выход из цикла */
   else break;
   /* если 10 членов - вывод суммы */
   if (n==9)
     printf("Сумма 10 членов ряда = %10.7lf\n",sum);
   }
  /* конец основного цикла */
  /* вывод окончательной суммы */
```

```
   printf("Полная сумма ряда = %10.7lf\n",sum);
  return 0;
} /* конец программы */
```

Компиляция и запуск проекта программы выполняется также как в первом задании.

5. Варианты индивидуальных заданий

Вариант 1

$$a_n = (-1^n)\frac{1}{(n+1)(n+2)(n+3)}$$

Вариант 2

$$a_n = (-1^n)\frac{n+1}{n^3+2}$$

Вариант 3

$$a_n = (-1^n)(1-\frac{2n-1}{2(n+1)})$$

Вариант 4

$$a_n = (-1^n)\frac{n^2+1}{n^3+3}$$

Вариант 5

$$a_n = (-1^n)\frac{n+1}{3^n+2^n}$$

Вариант 6

$$a_n = (-1^n)(1-\frac{(n+1)^2}{(n+2)^2})$$

Вариант 7

$$a_n = (-1^n)\frac{2^n}{n^{n+1}+1}$$

Вариант 8

$$a_n = (-1^n)(1-\frac{2^n}{2^n+1})$$

Вариант 9

$$a_n = (-1^n)\frac{n+1}{2^{n-1}}$$

Вариант 10

$$a_n = (-1^n)\frac{n+1}{n^3-n^2+1}$$

Вариант 11

$$a_n = (-1^n)\frac{2^{n+1}}{2^{2n}+1}$$

Вариант 12

$$a_n = (-1^n)\frac{1}{n^2+2^n}$$

Вариант 13

$$a_n = (-1^n)\frac{1+3n}{3^n}$$

Вариант 14

$$a_n = (-1^n)\frac{n+1}{n^3+1}$$

Задание 4. Работа с массивами

1. Цель работы

Целью лабораторной работы является получение практических навыков в работе с массивами в языке С.

2. Темы для предварительной проработки

Операторы цикла языка С. Вложенные циклы.

Условный оператор языка С.

Массивы.

3. Задание

Объявить массив целых чисел и заполнить его случайными значениями. В индивидуальных заданиях указано, какую обработку массива следует произвести.

4. Пример решения задачи

Дан массив из 100 элементов. Во всех последовательностях отрицательных чисел ограничить значения тех элементов, абсолютное значение которых превышает абсолютное среднее для этой последовательности

4.1. Разработка алгоритма.

Схема алгоритма показана на рисунке ниже.

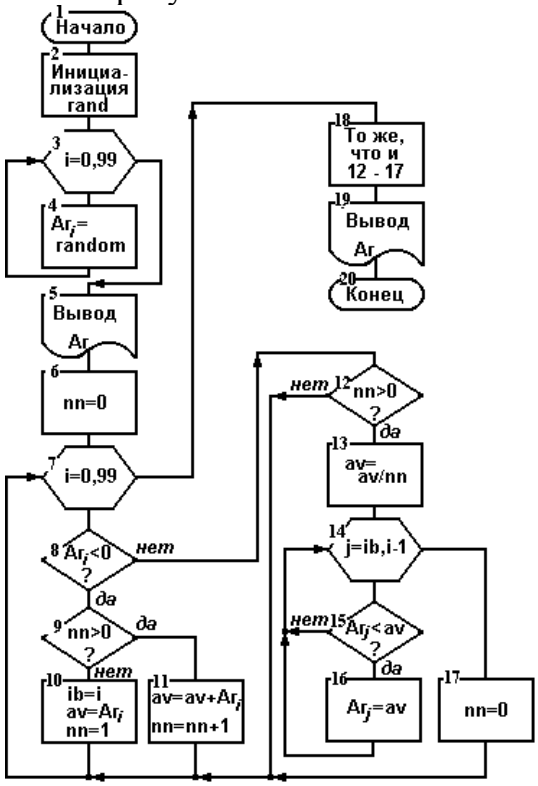


Рис. 10.

В первой фазе выполнения программы нам необходимо будет сформировать массив случайных чисел. Перед тем как мы будем обращаться к датчику случайных чисел, необходимо его проинициализировать (блок 2). Далее организуем цикл со счетчиком (блок 3), в каждой итерации которого генерируется следующее случайное число и записывается в следующий элемент массива (блок 4). После окончания цикла заполнения массива выводим массив на экран (блок 5).

Нам необходимо будет вычислять среднее значение последовательности, следовательно - подсчитывать количество элементов в ней. Для этого мы вводим переменную nn - счетчик элементов, нулевое значение этой переменной будет показывать, что у нас нет последовательности для обработки. В начале обработки мы устанавливаем nn=0 (блок 6).

Далее организуем цикл со счетчиком (блок 7), в котором перебираем элементы массива. Для каждого элемента в первую очередь проверяется его знак (блок 8). Если это отрицательный элемент, то это может быть первый или не первый элемент последовательности. Это можно определить, проверяя значение переменной nn: если она 0 - это первый элемент (блок 9). Для первого элемента мы запоминаем в переменной ib индекс начала последовательности, устанавливаем счетчик элементов nn в 1, а в переменную av записываем значение этого элемента (блок 10). Для не первого элемента мы увеличиваем счетчик на 1, а значение элемента суммируем со значением переменной av (блок 11). Таким образом, переменная av у нас играет роль накопителя суммы элементов последовательности.

Если же очередной элемент последовательности положительный, то возникает вопрос - не является ли этот элемент первым положительным элементом после отрицательной последовательности? Это можно проверить по счетчику nn. Если элемент первый, то значение nn должно быть больше 0 (блок 12). Если нет, то нам необходимо обработать ту отрицательную последовательность, которая только что закончилась. Для обработки мы в первую очередь получаем среднее значение (блок 13). Потом организуем цикл (блок 14) со счетчиком j, который изменяется от ib (индекс начала отрицательной последовательности, который мы сохранили раньше) до i-1 (i - это индекс первого положительного элемента после отрицательной - это индекс первого положительного элемента после отрицательной последовательности, следовательно i-1 - индекс последнего элемента отрицательной последовательности). В каждой итерации этого цикла мы сравниваем j-й элемент массива со средним значением av (блок 15). Если значение элемента меньше среднего (т.е. больше по абсолютному значению), то среднее значение записывается в j-й элемент (блок 16), если же нет - ничего не происходит. По выходу из цикла мы устанавливаем счетчик nn в 0 (блок 17), как признак того, что у нас нет необработанной последовательности. Для не первого положительного элемента нет необходимости что-либо делать.

После выхода из того цикла, который начался в блоке 7, необходимо проверить, не осталась ли у нас необработанная последовательность и, если да, обработать ее. На схеме алгоритма мы показали это одним блоком 18, действия, которые выполняются в этом блоке тождественны действиям, которые детально показаны в блоках 12 - 17. По окончанию обработки мы выводим массив-результат (блок 19).

4.2. Определение переменных программы

Для реализации алгоритма нам будут необходимы следующие переменные.

Массив целых чисел, который будет обрабатываться:

```
int Ar[100];
```

Массив должен располагаться в статической памяти.

Индексы элементов массива для внешнего (блоки 3 - 17) и внутреннего (блоки 14 - 16) циклов:

```
int i, j;
```

Переменные, в которых будут храниться параметры очередной последовательности: сумма элементов, а потом среднее значение - av, количество элементов в последовательности - nn, индекс начала последовательности - ib:

```
int av;
int nn;
int ib;
```

Отметим, что для большинства данных, которые представляются переменными программы, достаточно было бы и типа short или char, т.к. их значения укладываются в диапазон: -128 - 128. Мы выбираем тип int согласно с общим стилем программирования на языке С. Отдельного рассмотрения требует переменная av, т.к. в ней накапливается сумма, а значит, ее значение может значительно превысить значения остальных переменных. Но в наихудшем случае (когда все элементы массива будут иметь максимальные значения) ее значение не превысит 100*50=5000, следовательно типа int достаточно и для нее.

4.3. Разработка текста программы

В начале текста программы включаем в него файлы, содержащие описания тех функций, к каким мы будем обращаться:

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
```

В файле stdio.h описания функций стандартного ввода-вывода, в файле stdlib.h - функции генерации случайных чисел. В описании функции randomize() мы нашли примечание, что она является макросом, который обращается к функции time(), следовательно, перед файлом stdlib.h в программу должен быть включен файл time.h.

Мы обусловили, что массив должен размещаться в статической памяти. Если мы объявим массив до открытия тела главной функции, то он будет размещен именно в статической памяти. Поэтому далее у нас идет объявление массива.

```
int Ar[100];
```

Далее ставим заголовок главной функции main и открываем ее тело:

```
int main(void){
```

Главная функция начинается с объявления остальных переменных программы.

Кодовая часть программы начинается с обращения к функции инициализации датчика случайных чисел (блок 2):

```
randomize(100);
```

Далее организуем простой цикл со счетчиком (блок 3), в каждой итерации которого в следующий элемент массива записывается случайное число (блок 4). Обращение к функции rand() возвращает нам число в пределах 0 - 100; вычитая из него 50, мы приводим его к диапазону -50 - +50.

```
for (i=0; i<100; Ar[i++]=random(101)-50 );
```

Заполненный таким образом массив в цикле выводим на экран (блок 5). Формат вывода каждого элемента - %3d - обеспечивает отображение числа из двух цифр со знаком. Между числами задаем еще два пробела, таким образом, каждый элемент занимает 5 позиций, а в одной строке экрана разместится целое число (16) элементов, что обеспечит удобное представление массива.

```
printf("Начальный массив:\n");
for (i=0; i<100; printf("%3d  ",Ar[i++]));
putchar('\n');
putchar('\n');
```

Далее идет заголовок цикла перебора массива (блок 7), в котором мы также присваиваем начальное значение счетчику nn (блок 6):

```
for (nn=i=0; i<100; i++) {
```

Все тело цикла состоит из одного условного оператора. В этом операторе проверяем (блок 8) знак i-го элемента массива:

```
if (Ar[i]<0)
```

Если это условие - истина, то проверяем, не равен ли 0 счетчик элементов последовательности nn (блок 9):

```
if (!nn)
```

При выполнении этого условия выполняется ряд действий (блок 10), которые мы берем в операторные скобки:

```
{ ib=i; av=Ar[i]; nn=1; }
```

Если же условие ложно, то выполняется сложный оператор (блок 11):

```
else { av+=Ar[i]; nn++; }
```

Если же элемент положительный, то выполняется часть else первого условного оператора, в которой анализируется nn - нет ли у нас необработанной отрицательной последовательности (блок 12):

else if (nn) {

Если есть необработанная последовательность, выполняем усреднение значения av (блок 13):

av/=nn;

и организуем цикл с параметром j, который изменяется от ib до i-1(блок 14):

for (j=ib; j<i; j++)

В каждой итерации этого цикла j-й элемент массива сравнивается с средним значением (блок 15) и, если он больше, заменяется на среднее значение (блок 16). Это реализуется одним условным оператором:

if (Ar[j]<av) Ar[j]=av;

При выходе из цикла записываем 0 в счетчик nn (блок 17):

nn=0;

На этом заканчивается и внешний цикл.

Несколько операторов, которые следуют после выхода из цикла (блок 18), обеспечивают обработку последней последовательности и в основном являются копией тех операторов, которые реализуют блоки 12 - 17. Разница состоит в том, что тут мы внесли усреднение в начальное действие цикла и убрали присваивание nn=0, т.к. значение nn нам больше не понадобится.

Операторы вывода массива-результата - копия вывода начального массива.

Полный текст программы приведен ниже.

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
 int Ar[100];   /* массив, который обрабатывается */
int main(void) {
 int i, j;    /* индексы в массиве */
 int av;      /* среднее значение */
 int nn; /* количество эл-тов в последовательности */
```

```
int ib;      /* индекс начала последовательности */
 randomize();  /* инициализация rand */
 /* заполнение массива случайными числами */
 for (i=0; i<100; Ar[i++]=random(101)-50 );
 /* вывод начального массива */
 printf("Начальный массив:\n");
 for (i=0; i<100; printf("%3d  ",Ar[i++]));
 putchar('\n');
 putchar('\n');
 for (nn=i=0; i<100; i++) { /* перебор массива */
  if (Ar[i]<0)
    /* обработка отрицательного элемента */
    if (!nn) {
      /* начало последовательности */
      /* запомнить индекс начала,
         начальное значение накопителя суммы
         и счетчика элементов */
      ib=i; av=Ar[i]; nn=1;
      }
    else {
      /* накопление суммы,
         подсчет количества */
      av+=Ar[i]; nn++;
      }
  /* конец обработки отрицательного элемента */
  else /* обработка положительного элемента */
    if (nn) {
      /* если есть необработанная
         отрицательная последовательность */
      av/=nn;   /* усреднение */
      /* перебор всей последовательности
         с ограничением */
      for (j=ib; j<i; j++)
        if (Ar[j]>av) Ar[j]=av;
          nn=0; /* последовательность обработана */
      } /* конец если есть необработанная... */
  }  /* конец перебор массива */
 if (nn) /* если не обработана последняя
           отрицательная последовательность */
   for (av/=nn, j=ib; j<i; j++)
     if (Ar[j]>av) Ar[j]=av;
 /* вывод результатов */
 printf("Массив-результат:\n");
 for (i=0; i<100; printf("%3d  ",Ar[i++]));
 putchar('\n');
 return 0;
}
```

Компиляция и запуск проекта программы выполняется также как в первом задании.

5. Варианты индивидуальных заданий

Вариант 1

Дан массив. Заменить все элементы с отрицательным значением на значение минимального не равного 0 положительного элемента.

Вариант 2

Дан массив. Подсчитать количество пар соседних элементов с одинаковыми значениями

Вариант 3

Дан массив. Подсчитать количество участков, которые образуют непрерывные последовательности чисел с неуменьшающимися значениями

Вариант 4

Дан массив. Подсчитать количество пар соседних элементов, которые имеют противоположные знаки.

Вариант 5

Дан массив. Вывести начальные индексы всех непрерывных последовательностей неотрицательных чисел, длина которых больше 5.

Вариант 6

Дан массив. Найти ту непрерывную последовательность положительных чисел, сумма элементов в которой максимальная.

Вариант 7

Дан массив. Разместить все элементы с положительными значениями в левой части массива, элементы с отрицательными значениями - в правой, а нули - между ними.

Вариант 8

Дан массив. Заменить все элементы с отрицательными значениями средним арифметическим значением всех положительных элементов.

Вариант 9

Дан массив. Найти непрерывный участок из 10 элементов, сумма которых максимальна.

Вариант 10

Дан массив. Найти значение 3-го по величине элемента и значения всех элементов массива, которые его превышают, заменить на найденное значение.

Вариант 11

Дан массив. Найти количество пар соседних элементов, которые имеют одинаковые абсолютные значения, но противоположные знаки.

Вариант 12

Дан массив. Во всех последовательностях положительных чисел заменить значения элементов с максимальным и минимальным значением на среднее для этой последовательности.

Вариант 13

Дан массив. Найти непрерывный участок из 10 элементов, который имеет наибольшее среднее значение элементов.

Вариант 14

Дан массив. Во всех последовательностях положительных чисел изменить порядок элементов на противоположный.

Задание 5. Работа с матрицами

1. Цель работы

Целью лабораторной работы является получение практических навыков в работе с матрицами в языке С.

2. Темы для предварительной проработки

Операторы цикла языка С. Вложенные циклы.
Условный оператор языка С.
Матрицы.

3. Задания для выполнения

Создать квадратную матрицу целых чисел размером 9x9. В индивидуальных заданиях указано, какую обработку матрицы требуется выполнить.

Если по условию задания матрицу следует заполнить случайными числами, рекомендуем выбирать эти числа из диапазона 0 - 99. Если по условию задания в матрицу следует записать ЛП - линейную последовательность чисел, имеется в виду последовательность: 1, 2, 3, ...

4. Пример решения задачи

Дана матрица. Заполнить секторы матрицы, которые лежат выше и ниже главной и побочной диагоналей, ЛП, от левого верхнего угла вниз - вправо. Остаток матрицы заполнить нулями.

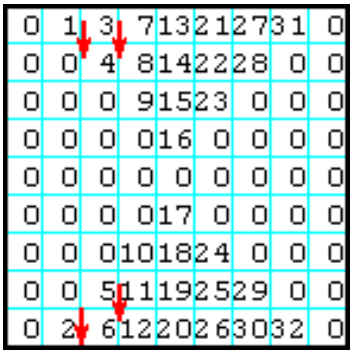


Рис. 11.

5.1. Разработка алгоритма решения.

Если ми обозначим размерность матрицы как ***S***, номер строки как ***L***, а номер столбца как ***R***, и (имея в виду, что реализация алгоритма будет выполнена на языке С) договоримся, что нумерация строк и столбцов будет начинаться с 0, то можно определить, что в строке с номером ***L*** ненулевые элементы в верхней части матрицы лежат на столбцах с номерами ***R1=L < R < R2=S-L***, а в нижней - ***R1=S-L-1 < R < R2=L***. Следовательно, алгоритм может состоять из перебора матрицы строка за строкой с определением для каждого элемента, удовлетворяют ли его индексы вышеприведенным условиям. Если да - элементу присваивается следующее значение из ЛП, если нет - 0.

Но можно несколько упростить алгоритм, обойдя вычисления граничных значений для каждого элемента и необходимости определения, в верхнюю или нижнюю часть матрицы ми попадаем. Обратим внимание на то, что для первой строки ***(L=0) R1=1, R2=S-2***. Для каждой следующей строки ***R1*** увеличивается на 1, а ***R2*** уменьшается на 1. Когда мы пересекаем середину матрицы, то направление модификации изменяется на противоположное: теперь для каждой следующей строки ***R1*** уменьшается на 1, а ***R2*** увеличивается на 1. Признаком пересечения середины может быть условие ***R1 > R2***, оно выполняется в момент пересечения. Схема последнего алгоритма показана на рис.12.

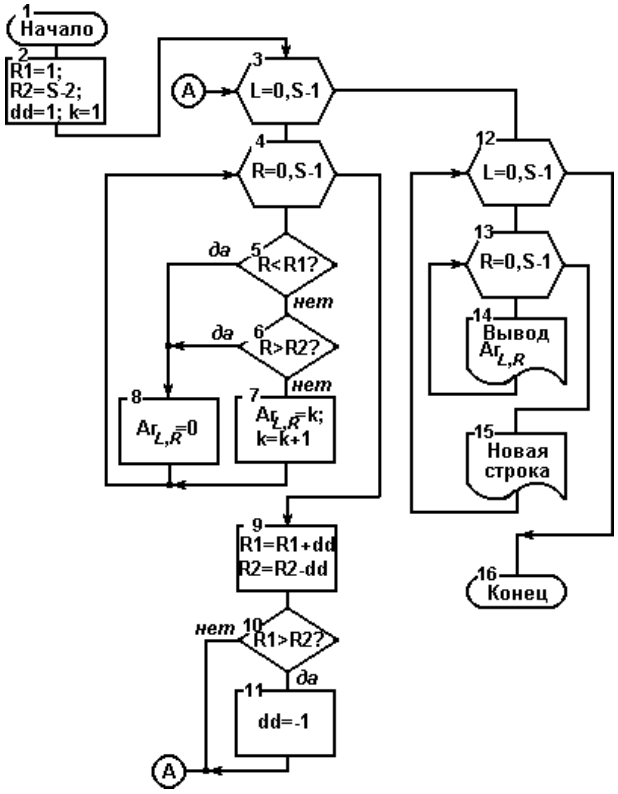


Рис. 12.

Вместе с описанными выше переменными ***R1*** и ***R2***, которые получают начальные значения для первой строки матрицы, ми вводим переменную ***dd*** с начальным значением 1 - это то значение, которое будет модифицировать ***R1*** и ***R2*** для каждой следующей строки, и переменную ***k*** - в которой будет значение текущего члена ЛП, начальное значение - 1 (блок 2). Далее организуются вложенные циклы. Во внешнем цикле перебираются строки (блок 3), а во внутреннем - столбцы матрицы (блок 4). В каждой итерации внутреннего цикла номер столбца ***R*** сравнивается с граничными значениями ***R1***, ***R2*** (блоки 5,6). Если он лежит в пределах от ***R1*** до ***R2***, то текущему члену матрицы присваивается значение ***k*** - текущего члена ЛП, а затем ***k*** увеличивается на 1 (блок 7). Если нет, текущему члену присваивается значение 0 (блок 8).

После выхода из внутреннего цикла модифицируются граничные значения: ***R1*** увеличивается на ***dd***, а ***R2*** уменьшается на ***dd*** (блок 9). Напомним, что начальное значение ***dd=1***. Когда выполняется условие ***R1 > R2*** (блок 10) мы присваиваем ***dd*** значение -1, далее модификация границ будет соответствовать правилам для нижней части матрицы.

После выхода из внешнего цикла, который начался в блоке 3, вновь организуются вложенные циклы перебора строк (блок 12) и столбцов (блок 13). В каждой итерации внутреннего цикла выводится значение одного элемента матрицы (блок 14), после выхода из внутреннего цикла начинается новая строка вывода (блок 15).

5.2. Определение переменных программы

Для реализации алгоритма нам будут нужны такие переменные.

Матрица представляется в памяти как 2-мерний массив (должен быть размещен в статической памяти):

```
int Ar[S][S];
```

Переменные для представления текущих номеров строки (***l***) и столбца (***r***):

```
short l, r;
```

Переменные для представления граничных номеров столбцов:

```
short r1,r2;
```

Переменная - модификатор граничных номеров:

```
short dd;
```

Переменная - текущий член ЛП:

```
short k;
```

Всем скалярным переменным назначаем тип ***short***, т.к. их значения никак не могут выходить из диапазона -128 - 128.

5.3. Разработка текста программы

Текст программы начинаем с включения файла ***stdio.h*** и определения макроконстанты ***S*** - размера матрицы (хотя по условию задания можно было бы использовать просто константу 9 в тексте программы, определение размера через макроконстанту более соответствует стилю программирования на языке С++).

Массив-матрицу ***Ar*** объявляем до открытия тела главной функции, что обеспечивает его размещение в статической памяти.

Открываем тело главной функции и объявляем переменные в соответствии с п.5.2. Присваиваем переменным ***r1***, ***r2***, ***dd***, ***k*** начальные значения (это можно было сделать и при их объявлении). Открываем цикл перебора строк с изменением ***l*** от ***0*** до ***S-1*** и цикл перебора столбцов с изменением ***r*** от ***0*** до ***S-1***. Внутренний цикл состоит из одного условного оператора, так что нет необходимости брать его тело в операторные скобки. Тело внешнего цикла берется в скобки.

В условном операторе проверяем сразу оба условия (блоки 5 и 6). Поскольку для выхода за пределы должно выполняться хотя бы одно из них, они соединены операцией "ИЛИ". При выполнении условия значение ***k*** записывается в элемент массив с индексами ***[l,r]*** и сразу же увеличивается. При невыполнении - в элемент массива записывается 0.

После выхода из внутреннего цикла, но еще в теле внешнего модифицируются значения ***r1*** и ***r2***. Потом условным оператором проверяется условие ***r1>r2*** и, если он выполняется знак модификатора ***dd*** меняется на противоположный. Потом открываются два цикла для вывода. В каждой итерации внутреннего цикла выводится значение одного элемента массива. Формат вывода обеспечивает вывод положительного числа из 2 цифр и пробела перед ним. После каждого выхода из внутреннего цикла выводится символ перехода на новую строку. Таким образом, матрица будет выведена в наглядном представлении. Полный текст программы приведен ниже.

```
#include <stdio.h>
#define S 9
int Ar[S][S]; /* матрица */
int main(void) {
 short l, r;  /* текущие индексы */
 short r1,r2; /* граничные номера столбцов */
 short dd;    /* модификатор граничных номеров */
 short k;     /* текущий член ЛП */
  /* начальные значения переменных */
  r1=1; r2=S-2; dd=1; k=1;
  for (l=0; l<S; l++) {  /* перебор строк */
    for (r=0; r<S; r++)  /* перебор столбцов */
      /* условие ненулевого значения */
      if ((r<r1)||(r>r2)) Ar[l][r]=0;
      else Ar[l][r]=k++;
      /* конец перебора строк */
    /* модификация границ */
    r1+=dd; r2-=dd;
    /* уловие перехода в нижнюю часть */
    if (r1>r2) dd=-dd;
    } /* конец перебора столбцов */
  /* вывод матрицы */
  for (l=0; l<S; l++) {
    for (r=0; r<S; r++) {
      printf("%3d",Ar[l][r]);
      }
    printf("\n");
    }
  return 0;
}
```

Компиляция и запуск проекта программы выполняется также как в первом задании.

5. Варианты индивидуальных заданий

Вариант 1

Заполнить матрицу случайными числами. Развернуть матрицу на 90^o по часовой стрелке.

Вариант 2

Заполнить матрицу случайными числами. Отобразить матрицу симметрично относительно главной диагонали

Вариант 3

Заполнить матрицу ЛП, от левого верхнего угла по спирали: вправо - вниз - влево - вверх.

Вариант 4

Заполнить матрицу ЛП, от центра по спирали: влево - вниз - вправо - вверх.

Вариант 5

Заполнить матрицу случайными числами. На главной диагонали разместить суммы элементов, которые лежат на той же строке и том же столбце.

Вариант 6

Заполнить матрицу ЛП, от левого верхнего угла по диагонали: вправо - вверх.

Вариант 7

Заполнить секторы матрицы, которые лежат влево и вправо от главной и побочной диагоналей, ЛП, от левого верхнего угла вниз - вправо. Остаток матрицы заполнить нулями.

Вариант 8

Заполнить матрицу случайными числами. Отобразить симметрично относительно вертикальной оси секторы матрицы, которые лежат влево и вправо от главной и побочной диагоналей.

Вариант 9

Заполнить матрицу ЛП, от левого нижнего угла по диагонали: влево - вверх.

Вариант 10

Заполнить матрицу случайными числами. Отобразить главную и побочную диагонали симметрично относительно вертикальной оси.

Вариант 11

Заполнить матрицу случайными числами. Разместить на главной диагонали суммы элементов, которые лежат на диагоналях, перпендикулярных к главной.

Вариант 12

Заполнить матрицу случайными числами. Отобразить верхнюю половину матрицы на нижнюю зеркально симметрично относительно горизонтальной оси.

Вариант 13

Заполнить матрицу случайными числами. Розбити матрицу на квадраты размером 3х3. В центре каждого квадрата поместить сумму остальных элементов квадрата.

Вариант 14

Заполнить матрицу случайными числами. Отобразить правую половину матрицы на левую зеркально симметрично относительно вертикальной оси.

Приложение 1.

СПИСОК СТУДЕНТОВ

Номер варианта	ФИО студента
1	Агутов Евгений Александрович
2	Алатов Егор Олегович
3	Васильчев Михаил Алексеевич
4	Денисова Татьяна Вячеславовна
5	Еремин Юрий Владимирович
6	Ерошкина Светлана Александровна
7	Иванов Андрей Владимирович
8	Кудашова Валентина Сергеевна
9	Любченко Сергей Николаевич
10	Макаров Михаил Юрьевич
11	Николаев Евгений Александрович
12	Платонов Александр Александрович
13	Смирнов Алексей Петрович
14	Цапыгин Евгений Анатольевич

Название типа	Пояснения	Диапазон значений
`short`	Краткое целое число	-128 . . . 127
`unsigned short`	Краткое целое число без знака	0 . . . 255
`int`	Целое число	-32768 . . . 32767
`unsigned int`	Целое число	0 . . . 65535
`long`	Длинное целое число	-2^{30} . . . $2^{30}-1$
`unsigned long`	Длинное целое число без знака	0 . . . $2^{31}-1$
`char`	Один символ	символы кода ASCII
`char[ ]`	Строка	
`float`	Число с плавающей точкой	$3.4\ 10^{-38}$ . . . $3.4\ 10^{+38}$
`double`	Число с плавающей точкой двойной точности	$1.7\ 10^{-308}$ . . . $1.7\ 10^{+308}$

Функции стандартного ввода - вывода описаны в файле ***stdio.h***.

printf() *- форматный вывод на экран:*

```
int printf(char *format, <список вывода>);
```

Первый параметр является символьной строкой, которая задает спецификации формата. Остальные параметры - перечисление переменных и выражений, значения которых выводятся. Каждая спецификация формата имеет вид (параметры в квадратных скобках необязательны):

```
%[flags][width][.prec][F|N|h|l]type
```

где `type` - тип спецификации

`d` или `i`	целое десятичное число со знаком
`u`	десятичное число без знака
`x`	целое 16-ричное число без знака
`f`	число с плавающей точкой
`e`	число в Е-форме
`g`	число с плавающей точкой или в Е-форме
`c`	один символ
`s`	строка
`%`	символ %

`flags` - признак выравнивания:

`+` или `пусто`	выравнивание по правому краю
`–`	выравнивание по левому краю

`width` -	целое число - общая ширина поля. Если это число начинается с цифры 0, вывод дополняется слева нулями до заданной ширины. В заданную ширину входят все символы вывода, включая знак, дробную часть и т.п.
`prec` -	целое число, количество знаков после точки при выводе чисел с плавающей точкой
`F` -	соответственный элемент списка вывода является дальним указателем

N -	соответственный елемент списка вывода является близким указателем
l -	соответствующий елемент списка вывода является `long int` или `double`

scanf() *- форматный ввод с клавиатуры:*

```
int scanf(char *format, <список ввода>);
```

*Первый параметр является символьной строкой, которая задает спецификации формата (см. функцию **printf()**). Остальные параметры - перечисление адресов переменных, в которые вводятся данные. В этом списке перед именами всех переменных, кроме тех, которые вводятся по спецификации типа **%s**, должен стоять символ **&**.*

putchar() *- вывод одного символа на экран:*

```
int putchar(int ch);
```

*Параметр функции - код символа, который выводится. При успешном выполнении функция возвращает этот же код, при неуспешном - **EOF**.*

getchar() *- ввод одного символа с клавиатуры:*

```
int getchar(void);
```

Функция возвращает код введенного символа.

puts() *- вывод строки символов на экран:*

```
int puts(char *string);
```

Параметр функции - указатель на начало той строки, из которой выводятся данные. Функция возвращает количество выведенных символов.

gets() *- ввод строки символов с клавиатуры:*

```
 char *gets(char *string);
```