



---

**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ**

**Брянский государственный технический университет**

---

Утверждаю

Ректор университета

\_\_\_\_\_ О.Н. Федонин

«\_\_\_\_\_» \_\_\_\_\_ 2013 г.

## **ЯЗЫКИ ПРОГРАММИРОВАНИЯ**

### **ИЗУЧЕНИЕ УСЛОВНЫХ И ЦИКЛИЧЕСКИХ КОНСТРУКЦИЙ**

Методические указания  
к выполнению лабораторной работы  
для студентов очной формы обучения  
специальностей 090303 – «Информационная безопасность  
автоматизированных систем»,  
090900 – «Информационная безопасность»

**Брянск 2013**

УДК 004.43

Языки программирования. Изучение условных и циклических конструкций: методические указания к выполнению лабораторной работы для студентов очной формы обучения специальностей 090303 – «Информационная безопасность автоматизированных систем», 090900 – «Информационная безопасность». – Брянск: БГТУ, 2013. – 20 с.

*Разработали:*

Ю.А. Леонов, к.т.н., доц.,

Е.А. Леонов, к.т.н., доц.

Научный редактор: Ю.М. Казаков

Редактор издательства: Л.И. Афонина

Компьютерный набор: Ю.А. Леонов

Рекомендовано кафедрой «Компьютерные технологии и системы» БГТУ (протокол № 2 от 19.09.2013)

Темплан 2013 г., п.

---

Подписано в печать

Формат 60x84 1/16. Бумага офсетная.

Офсетная печать.

Усл. печ. л. 1,16 Уч. – изд. л. 1,16 Тираж 20 экз. Заказ Бесплатно

---

Издательство брянского государственного технического университета,  
241035, Брянск, бульвар 50-летия Октября, 7, БГТУ. 58-82-49

Лаборатория оперативной полиграфии БГТУ, ул. Харьковская, 9

## 1. ЦЕЛЬ РАБОТЫ

Целью работы является приобретение практических навыков при построении ветвящихся и повторяющихся процессов с использованием условных и циклических конструкций на примере табулирования функции на заданном числовом отрезке.

Продолжительность работы – 4ч.

## 2. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ



### 2.1. Блок-схемы

*Блок-схемой* – называется графическое представление алгоритма. Блок-схемы должны быть оформлены в соответствии с ГОСТ 19.701-90. «Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения».



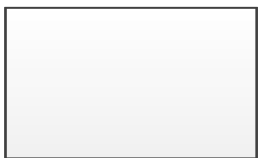


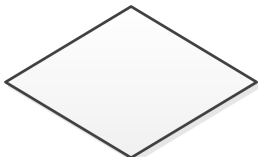
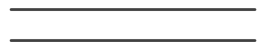
В блок-схеме каждому типу действий (вводу исходных данных, вычислению значений выражений, проверке условий и т.п.) соответствует геометрическая фигура, представленная в виде *блочного символа*. Блочные символы соединяются *линиями переходов*, определяющими очередность выполнения действий. Любая блок-схема обязательно должна иметь блоки «начало», из которого начинается выполнения алгоритма, и «конец», завершающим выполнение алгоритма. Рассмотрим наиболее часто употребляемые символы (табл. 1).

**Таблица 1**







**Основные символы блок-схем**

Название	Обозначение	Пояснение
<i>Символы данных</i>		
Данные		Символ отображает данные, носитель данных не определен.
Запоминаемые данные		Символ отображает хранимые данные в виде, пригодном для обработки, носитель данных не определен.



Продолжение табл. 1

Название	Обозначение	Пояснение
Документ		Символ отображает данные, представленные на носителе в удобочитаемой форме
Ручной ввод		Символ отображает данные, вводимые вручную во время обработки с устройств любого типа
<i>Символы процесса</i>		
Процесс		Символ отображает функцию обработки данных любого вида
Предопределенный процесс		Символ отображает предопределенный процесс, состоящий из одной или нескольких операций или шагов программы, которые определены в другом месте (в подпрограмме, модуле).
Ручная операция		Символ отображает любой процесс, выполняемый человеком.
Решение		Символ отображает решение или функцию переключательного типа, имеющую один вход и ряд альтернативных выходов, один и только один из которых может быть активизирован после вычисления условий, определенных внутри этого символа. Соответствующие результаты вычисления могут быть записаны по соседству с линиями, отображающими эти пути.
Параллельные действия		Символ отображает синхронизацию двух или более параллельных операций.

Продолжение табл. 1

Название	Обозначение	Пояснение
<i>Символы линий</i>		
Линия		Символ отображает поток данных или управления. При необходимости или для повышения удобочитаемости могут быть добавлены стрелки-указатели.
Передача управления		Символ отображает непосредственную передачу управления от одного процесса к другому.
Канал связи		Символ отображает передачу данных по каналу связи.
Пунктирная линия		Символ отображает альтернативную связь между двумя или более символами. Кроме того, символ используют для обведения аннотированного участка.
<i>Специальные символы</i>		
Соединитель		Символ отображает выход в часть схемы и вход из другой части этой схемы и используется для обрыва линии и продолжения ее в другом месте. Соответствующие символы-соединители должны содержать одно и то же уникальное обозначение.
Терминатор		Символ отображает выход во внешнюю среду и вход из внешней среды (начало или конец схемы программы, внешнее использование и источник или пункт назначения данных).

Окончание табл. 1

Название	Обозначение	Пояснение
Комментарий		Символ используют для добавления описательных комментариев или пояснительных записей в целях объяснения или примечаний. Пунктирные линии в символе комментария связаны с соответствующим символом или могут обводить группу символов. Текст комментариев или примечаний должен быть помещен около ограничивающей фигуры.
Пропуск		Символ (три точки) используют в схемах для отображения пропуска символа или группы символов, в которых не определены ни тип, ни число символов. Символ используют только в символах линии или между ними. Он применяется главным образом в схемах, изображающих общие решения с неизвестным числом повторений.

**Пример.** Необходимо составить алгоритм решения задачи нахождения значения функции  $R(r)$  в виде блок-схемы (рис. 1).

$$R(r) = \begin{cases} -r^2, & |r| < 1 \\ r/2, & 1 \leq |r| \leq 2 \\ r - F(r), & |r| > 2 \end{cases}, \text{ где } F(r) = \begin{cases} 1, & r \geq 0 \\ 0, & r < 0. \end{cases}$$

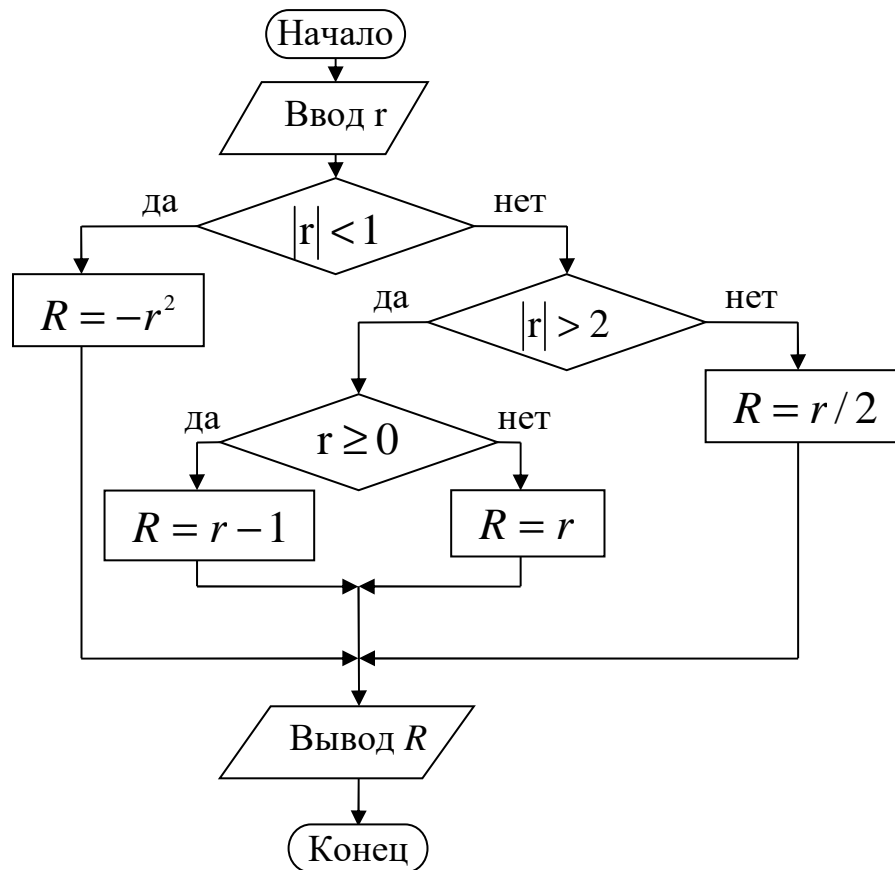


Рис. 1. Представление алгоритма в виде блок-схемы

## 2.2. Синтаксические конструкции условий

Операторы условного перехода позволяют выбрать для исполнения один из нескольких операторов в зависимости от условия.

### 2.2.1. Условная конструкция *if*

Синтаксис конструкции *if*:

***if*** (<выражение>) <конструкция\_1> [***else*** <конструкция\_2>];

В представленной конструкции *if* выражение должно быть булевого (логического) типа данных, т.е. принимающее одно из значений *true* или *false*. Если значение выражения равно *true*, то выполняется *конструкция\_1*, если значение выражения – *false*, то выполняется *конструкция\_2*.

Выражение часто записывается с помощью булевских операций. Пример выражения:  $a > b$ , где  $a$  и  $b$  – это переменные, значение которых определено ранее.

Вместо *конструкция\_1* и *конструкция\_2* может стоять как операторы, так и синтаксические конструкции.

**Например:**

```
if (a > 0 && a <= 10) Console.WriteLine("Число находится в
диапазоне (0,10].");
else Console.WriteLine("Число вне диапазона (0,10].");
```

Также конструкция имеет сокращенную форму:

```
if (<выражение>) <конструкция>;
```

В качестве выражения может быть использовано также только булевское выражение. В этом случае, если значение выражения равно *true*, то указанная после выражения конструкция выполняется. Если значение выражения *false*, то выполнение программы продолжается без выполнения условной конструкции.

### 2.2.2. Конструкция выбора *switch*

Обычно при написании программы не рекомендуется использовать многократно вложенные друг в друга условные конструкции – программа становится громоздкой и трудно понимаемой. В случае если необходимо проверять достаточно много условий и в зависимости от них выполнять те или иные действия, то для этих целей в языке C# существует специальная конструкция *switch*.

Синтаксис оператора *switch*:

```
switch (<выражение>)
{
    case <константа_1>:[<конструкция_1><оператор передачи
управления>]
    [case <константа_2>:[<конструкция_2><оператор пере-
дачи управления>]
    ...
    [case <константа_n>:][<конструкция_n><оператор пере-
дачи управления>]
    [default: <конструкция_d><оператор передачи управле-
ния>]
}
```

В конструкции *switch* осуществляется проверка результата выражения с набором констант. Чаще всего в качестве выражения используется какая-либо переменная. Тип констант в секции *case* и тип значения выражения обязан совпадать. В конструкции *switch* происходит



последовательная проверка результата выражения с константами в каждой из секций *case*, в случае их равенства выполняется указанная конструкция в секции, если для данной секции конструкция не задана, выполняется следующая конструкция в порядке объявления. Таким образом, список секций *case* без выполняемых конструкций задает набор возможных альтернатив значений выражения для выполнения одного набора действий.

Каждая выполняемая конструкция должна заканчиваться оператором передачи управления, например: *break*, *goto case*, *return* или *throw*. Чаще всего применяется оператор *break*, который позволяет прервать выполнение проверки в конструкции *switch*, при совпадении одной из альтернатив, так как при совпадении одной альтернативы проверка других бессмысленна, потому как проверяется только равенство.

### Пример 1:

```
switch (A)
{
    case 2: Console.WriteLine("A равно 2"); break;
    case 5: Console.WriteLine("A равно 5"); break;
    case 10: Console.WriteLine("A равно 10"); break;
    default: Console.WriteLine("A не равно не 2, не 5, не 10");
            break;
}
```

Если значение выражения не совпадает ни с одним из значений констант в списке секций *case*, то выполняется конструкция *d* в секции *default*, которая также должна передавать управление. Секция *default* также является необязательной.

### Пример 2:

```
switch (action)
{
    case "delete":
    case "create":
    case "edit": neededAccessLevel="user"; break;
    case "read": neededAccessLevel="guest"; break;
    default: neededAccessLevel="administrator"; break;
}
if (User.accessLevel!= neededAccessLevel)
throw new AccessDenied();
```

**Примечание:**

В случае объявления переменных или их первичной инициализации в условных конструкциях (*if*, *switch*), и использования этих же переменных вне конструкции в которой они были объявлены, возникает ошибка компиляции. В связи с тем, что перед использованием переменная должна быть обязательно инициализирована, а инициализация в условных конструкциях может быть не выполнена. В таком случае инициализация должна проводиться до использования и без условий, даже если созданные условия охватывают все возможные варианты.

**2.3. Циклические конструкции**

Циклические конструкции (циклы) обычно используют для многократного повторения некоторых действий.

В С# имеется четыре различных циклических конструкции:

- цикл с предусловием (*while*);
- цикл с постусловием (*do*);
- цикл со счетчиком (*for*);
- цикл перечисления (*foreach*).

**2.3.1. Конструкция с предусловием *while***

Цикл *while* используется, как правило, в тех случаях, когда заранее неизвестно количество повторений цикла.

Синтаксис оператора *while*:

***while*** (<выражение>) <конструкция>;

Результат вычисления выражения должен быть логического типа, его значение вычисляется каждый раз перед выполнением конструкции (тела цикла), поэтому цикл *while* называют еще циклом с предусловием. Выполнение заданной конструкции будет выполняться до тех пор, пока выражение равно *true*. Если значение выражения *false*, то происходит выход из цикла.

Для предотвращения бесконечного повторения цикла (зацикливания), необходимо, чтобы результат выражения мог измениться на *false*, для этого в теле цикла должна изменяться хотя бы одна переменная, входящая в выражение.

**Пример:** вычисление суммы ряда, состоящего из 50 элементов.

```
int s = 0, n = 1;
while (n <= 50)
{
    s += 1 / n;
    n++;
}
```

### 2.3.2. Конструкция с постусловием *do..while*

Цикл *do..while*, как правило, используется в тех случаях, когда заранее неизвестно количество повторений (итераций) тела цикла, но необходимо выполнить тело цикла хотя бы один раз. Синтаксис цикла *do..while*:

***do*** <конструкция> ***while*** (<выражение>);

В цикле *do..while* вначале выполняется тело цикла, представленное произвольной конструкцией, затем вычисляется значение выражения; если его значение равно *true*, то вновь выполняется оператор, если значение выражения *false*, то цикл заканчивается. Если значение выражения равно *false* с самого начала, то конструкция выполняется лишь один раз. Если выражение никогда не принимает значение *false*, то оператор выполняется бесконечное число раз, т.е. происходит «зацикливание».

**Пример:** вычисление суммы ряда, состоящего из 50 элементов.

```
double s = 0; int n = 1;
do
{
    s += 1.0 / n; n++;
} while (n <= 50);
```

### 2.3.3. Конструкция со счетчиком *for*

Оператор цикла *for* служит для организации цикла в тех случаях, когда заранее известно, сколько раз должен повториться какой-то алгоритм. Описание цикла *for* состоит из трех основных секций и имеет следующий синтаксис:

***for*** (<инициализация>; <условие>; <модификация>) <конструкция>;

Секция «инициализация» служит для инициализации переменных используемых в цикле. Данный участок программного кода выполня-

ется всегда и лишь один раз вне зависимости от того сколько раз исполняется тело цикла. Инициализация выполняется так же, как если бы она была размещена до цикла с той лишь разницей, что объявленные переменные являются локальными для цикла и за его пределами не видны, что позволяет избежать логических ошибок при использовании временных локальных переменных с одинаковыми именами.

В данной секции могут быть объявлены несколько переменных, при этом они должны быть разделены запятой и иметь одинаковый тип. Объявление переменных различных типов в данной секции не допускается.

Секция «условие» предназначена для описания условного выражения определяющего окончание выполнения цикла. В случае если результат вычисления выражения равен *true*, то цикл продолжает выполняться, а в случае *false* выполнение цикла завершается и управление передается программному коду написанному следом за телом цикла.

Секция «модификация» определяет действия, которые должны выполняться по окончании каждой итерации. В данной секции могут быть использованы любые выражения, разделенные запятой, которые будут вычислены по завершении исполнения тела цикла. Чаще всего используется инкремент или декремент для объявленных переменных в секции «инициализация».

В качестве тела цикла может быть использована произвольная конструкция программного кода. В случае если необходимо выполнять в цикле более одного действия, исполняемая конструкция заключается в фигурные скобки.

**Пример:** вычисление суммы ряда, состоящего из 50 элементов с использованием *for*.

```
double s = 0;
for (int n = 1; n <= 50; n++)
{
    s += 1.0 / n;
    Console.WriteLine("n={0}, s={1}", n, s);
}
```

## 2.4. Математические функции и константы

В языке C# все доступные математические константы и функции размещены в классе *Math*. Доступ к членам класса *Math* осуществля-

ется с использованием имени класса, после которого ставится точка и дальше указывается необходимый метод или константа.

**Пример:** вывод на экран значения числа  $\Pi$ .

```
Console.WriteLine(Math.PI);
```

В классе *Math* имеется лишь две константы к значениям, которых можно обращаться без предварительного их определения:  $\text{PI}=3,14159265358979$  и  $\text{E}=2,71828182845905$ .

Также в данном классе имеются следующие математические методы:

*Abs* – возвращает абсолютное значение аргумента (модуль числа);

*Acos* – возвращает угол косинус которого равен аргументу (арк-косинус);

*Asin* – возвращает угол синус которого равен аргументу (арксинус);

*Atan* – возвращает угол тангенс которого равен аргументу (арктангенс);

*Atan2* – возвращает угол образуемым между осью  $x$  и вектором с координатами аргумента;

*BigMul* – умножает два 32-битных числа, необходим для перемножения больших целых чисел, так как простое умножение вызывает переполнения типа данных и результат может быть некорректным;

*Cos* – возвращает косинус указанного угла;

*Cosh* – возвращает гиперболический косинус указанного угла (используется в физических расчетах);

*DivRem* – вычисляет частное двух 32 или 64 разрядных знаковых целых чисел и возвращает остаток в выходном параметре;

*Exp* – возвращает число  $e$ , возведенное в указанную степень;

*Floor* – возвращает наибольшее целое число, которое меньше или равно указанному десятичному числу;

*IEEERemainder* – возвращает остаток от деления одного указанного числа на другое указанное число в соответствии со стандартом IEEE 754 (используется в сетевых технологиях);

*Log* – возвращает натуральный логарифм (с основанием  $e$ ) указанного числа, в случае использования двух аргументов возвращает логарифм указанного числа по указанному основанию, но вычисляется значительно дольше;

*Log10* – возвращает логарифм с основанием 10 указанного числа;

*Max* – сравнивает два числа и возвращает больше из них;  
*Min* – сравнивает два числа и возвращает меньшее из них;  
*Pow* – возвращает результат возведения в степень указанного числа в указанную степень;  
*Round* – округляет десятичное значение до ближайшего целого;  
*Sign* – возвращает значение, определяющее знак десятичного числа;  
*Sin* – возвращает синус указанного угла;  
*Sinh* – возвращает гиперболический синус указанного угла (используется в физических расчетах);  
*Sqrt* – возвращает квадратный корень из указанного числа;  
*Tan* – возвращает тангенс указанного угла;  
*Tanh* – возвращает гиперболический тангенс указанного угла (используется в физических расчетах);  
*Truncate* – вычисляет целую часть заданного числа.

Для вычисления значений других математических функций следует пользоваться общеизвестными тождествами, например арктангенс можно найти как:

```
double ArcCtg = Math.PI / 2 - Math.Atan(x);
```

Для вычисления факториала можно использовать следующий цикл.

```
int f = 1, n = 5;
for (int i = 2; i <= n; i++) f *= i;
```

После вычисления данной подпрограммы  $f$  содержит результат вычисления факториала для числа  $n = 5$ , то есть вычисляется выражение  $f=n!$ .

## 2.5. Примеры

**Пример 1.** Разложение целого числа на простые множители.

```
class Program
{
    static void Main(string[] args)
    {
        int x, m; string t;
        Console.Write("Введите целое число. ");
        t = Console.ReadLine();
        x = Convert.ToInt32(t);
        Console.WriteLine("Разложение числа {0} на простые множители", x);
    }
}
```

```

m = 2;
Console.Write("{0}=", x);
bool isFirst = true;
while (m <= x)
{
    if (x % m == 0)
    {
        if (!isFirst) Console.Write("*");
        else isFirst = false;
        Console.Write(m);
        x /= m;
    } else m++;
}
Console.Read();
}

```

**Пример 2.** Программа, моделирующая калькулятор.

```

class Program
{
    static void Main(string[] args)
    {
        double n, Result = 0;
        string t, operation = "+";
        Console.WriteLine("Вводите арифметическое выраже-  
ние");
        Console.WriteLine("каждый операнд или операцию с  
новой строки ");
        do
        {
            t = Console.ReadLine();
            n = Convert.ToDouble(t);
            switch (operation)
            {
                case "+": Result += n; break;
                case "-": Result -= n; break;
                case "*": Result *= n; break;
                case "/": Result /= n; break;
            }
            operation = Console.ReadLine();
        } while (operation != "=");
        Console.WriteLine(Result);
        Console.Read();
    }
}

```

```
    }
}
```

**Пример 3.** Найти все простые числа на заданном отрезке.

```
class Program
{
    static void Main(string[] args)
    {
        int M, N;
        Console.Write("Введите нижнюю границу отрезка. ");
        M = Convert.ToInt32(Console.ReadLine());
        Console.Write("Введите верхнюю границу отрезка. ");
        N = Convert.ToInt32(Console.ReadLine());
        Console.WriteLine("Все простые числа из отрезка [{0},{1}]", M, N);
        bool isSimple = true;
        for (int i = M; i <= N; i++, isSimple = true)
        {
            for (int j = 2; j <= Math.Round(Math.Sqrt(i)); j++)
            {
                if (i % j == 0) isSimple = false;
                if (isSimple) Console.Write(i + " ");
            }
            Console.Read();
        }
    }
}
```

### 3. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

Работа выполняется студентом самостоятельно и состоит из следующих этапов:

- 1) изучение методических указаний по выполнению лабораторной работы и получение индивидуального задания;
- 2) составление блок-схемы алгоритма программы;
- 3) разработка программы;
- 3) отладка программы;
- 4) защита лабораторной работы.



### Общие требования к программе:

- текст программы представляется в электронном виде и должен включать постановку задачи, сведения об авторе и подробные комментарии;
- названия переменных должны быть логически обоснованы и давать понятие о том, что в них предполагается хранить или обрабатывать;
- программа должна запрашивать входные данные и выводить итоговый результат с пояснениями.

## 4. СПИСОК ЗАДАНИЙ

Для выполнения данной лабораторной работы необходимо описать алгоритм поставленной задачи в виде блок-схемы и написать программу на языке С#, которая будет реализовывать табулирование функции для заданной системы уравнений на числовом промежутке  $[a, b]$  с шагом  $p$ . Данные должны выводиться в табличной форме, где каждому значению аргумента соответствует подсчитанное значение функции. Задание необходимо выбрать из табл. 2 согласно номеру варианта.

Таблица 2

Список заданий

№	Система уравнений	№	Система уравнений
1	$f(x) = \begin{cases} \ln x - \sqrt{x^2 + \left \frac{1}{x}\right }, & \text{если } x < 0; \\ 20, & \text{если } x = 0; \\  1 - x^7  - \sqrt{x}, & \text{если } x > 0; \end{cases}$	16	$f(x) = \begin{cases} \sqrt{1 + x^2}, & \text{если } x < 0; \\ 21, & \text{если } x = 0; \\ \sqrt{x}, & \text{если } x > 0; \end{cases}$
2	$f(x) = \begin{cases} \sin x - \sqrt{x^2 + 3}, & \text{если } x < 0; \\ 5, & \text{если } x = 0; \\  1 - x^2  - x, & \text{если } x > 0; \end{cases}$	17	$f(x) = \begin{cases} \sin^2 2x - 6 \cos^3 x, & \text{если }  x  < 5; \\ \operatorname{tg} 2x + 1, & \text{если }  x  \geq 5; \end{cases}$
3	$f(x) = \begin{cases} \ln x, & \text{если } x < -2; \\ \cos x, & \text{если } -2 < x \leq 2; \\  1 - x^7  - \sqrt{x}, & \text{если } x > 3; \end{cases}$	18	$f(x) = \begin{cases} \sin^3 5x + 2 \cos^2 x, & \text{если }  x  < 2; \\ \operatorname{ctg} 2x + 1, & \text{если }  x  \geq 2; \end{cases}$
4	$f(x) = \begin{cases} \sqrt{1 + x^2}, & \text{если } x < 0; \\ 21, & \text{если } x = 0; \\ \sqrt{x}, & \text{если } x > 0; \end{cases}$	19	$f(x) = \begin{cases} \sin^3 5x, & \text{если }  x  < 2; \\ e^{2x}, & \text{если }  x  \geq 2; \end{cases}$

## Продолжение табл. 2

№	Система уравнений	№	Система уравнений
5	$f(x) = \begin{cases} \sqrt{3+x^5}, & \text{если } x < 0; \\ 325, & \text{если } 0 \leq x \leq 3; \\ \sin x - \cos x, & \text{если } x > 3; \end{cases}$	20	$f(x) = \begin{cases} \sin^2 2x + 7 \operatorname{tg}^4 x, & \text{если }  x  < 3; \\ \sqrt{ \operatorname{arctg} 3x - 5 }, & \text{если }  x  \geq 3; \end{cases}$
6	$f(x) = \begin{cases} \sqrt{1+x^2}, & \text{если } x < 0; \\ 1024, & \text{если } x = 0; \\ 2x^{5x-1}, & \text{если } x > 0; \end{cases}$	21	$f(x) = \begin{cases} e^{3x}, & \text{если } x < 1; \\ \arccos(1-2x), & \text{если } x \geq 1; \end{cases}$
7	$f(x) = \begin{cases} 3 \sin^5 5x, & \text{если }  x  < 2; \\ e^{5x}, & \text{если }  x  \geq 2; \end{cases}$	22	$f(x) = \begin{cases} \sin^3 2x - \sqrt{x^5 + 10}, & \text{если } x < 0; \\ 100, & \text{если } x = 0; \\  8 - x^3  - 10x, & \text{если } x > 0; \end{cases}$
8	$f(x) = \begin{cases} x - \sqrt{x^4 + \left  \frac{1}{x} \right }, & \text{если } x < 0; \\ \cos 2x, & \text{если } x = 0; \\  1 + x^5  - \sqrt{2x}, & \text{если } x > 0; \end{cases}$	23	$f(x) = \begin{cases} 7x + 4 \arccos^2(2x), & \text{если }  x  < 5; \\ \frac{\operatorname{arctg} x}{5x}, & \text{если }  x  \geq 5; \end{cases}$
9	$f(x) = \begin{cases} 2x - 4 \arccos^3 x, & \text{если }  x  < 5; \\ \operatorname{tg} x - 5, & \text{если }  x  \geq 5; \end{cases}$	24	$f(x) = \begin{cases} 5 \sin^3 2x + \arcsin 3x, & \text{если }  x  < 2; \\ 2e^{5x} - \sin x, & \text{если }  x  \geq 2; \end{cases}$
10	$f(x) = \begin{cases} \operatorname{ctg}^2 6x + 2 \cos x, & \text{если }  x  < 2; \\ \ln(2x) - 7, & \text{если }  x  \geq 2; \end{cases}$	25	$f(x) = \begin{cases} 5 - x^{e^2}, & \text{если } x < 0; \\ \cos 3x, & \text{если } x = 0; \\ \operatorname{arctg} 5x^{5x}, & \text{если } x > 0; \end{cases}$
11	$f(x) = \begin{cases} \operatorname{ctg}^2 4x, & \text{если }  x  < 2; \\ e^{2x}, & \text{если }  x  \geq 2; \end{cases}$	26	$f(x) = \begin{cases} \ln x \cdot e^{2x}, & \text{если } x < 1; \\ \arccos(3-x) + \operatorname{tg} x, & \text{если } x \geq 1; \end{cases}$
12	$f(x) = \begin{cases} 5 + \ln 3x, & \text{если } x < -2; \\ \cos x - \sin x, & \text{если } -2 < x \leq 2; \\  4 - 2x^5  - \sqrt{2x}, & \text{если } x > 2; \end{cases}$	27	$f(x) = \begin{cases} \operatorname{arctg}^2 x + 5 \arccos 2x, & \text{если }  x  < 2; \\ 5 + \frac{\ln(2x) + 100}{5 \cos 5x}, & \text{если }  x  \geq 2; \end{cases}$
13	$f(x) = \begin{cases} \sin 2x - \sqrt{x^5 + 4}, & \text{если } x < 0; \\ 10, & \text{если } x = 0; \\  8 - x^3  - 10x, & \text{если } x > 0; \end{cases}$	28	$f(x) = \begin{cases} x - \sqrt{x^4 + \left  \frac{1}{x-2} \right }, & \text{если } x < 0; \\ \cos 2x, & \text{если } x = 0; \\  x^5  - \sqrt{2x}, & \text{если } x > 0; \end{cases}$
14	$f(x) = \begin{cases} \cos^2 2x + 7 \operatorname{tg}^4 x, & \text{если }  x  < 3; \\ \sqrt{ \operatorname{ctg} 3x - 5 }, & \text{если }  x  \geq 3; \end{cases}$	29	$f(x) = \begin{cases} \sin^2 6x + 2 \cos x, & \text{если }  x  < 2; \\ \ln(2x) - 10, & \text{если }  x  \geq 2; \end{cases}$

№	Система уравнений	№	Система уравнений
15	$f(x) = \begin{cases} 3 \cos^5 5x, & \text{если }  x  < 2; \\ e^{5x}, & \text{если }  x  \geq 2; \end{cases}$	30	$f(x) = \begin{cases} \arcsin x - \sqrt{x^2 + 10}, & \text{если } x < 0; \\ 50, & \text{если } x = 0; \\  10 - x^2  - x, & \text{если } x > 0; \end{cases}$

## 5. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что называют алгоритмом программы?
2. Какие типы данных вы знаете?
3. Что такое блок-схема и как она строится?
4. Какие вы знаете символы (блоки), предназначенные для представления алгоритма в виде блок-схемы?
5. Для чего предназначены условные и циклические конструкции?
6. Опишите синтаксис условных и циклических конструкций.
7. В каких случаях применяется конструкция *switch*?
8. Какой тип данных может использоваться в проверяемом выражении конструкции *switch*?
9. Когда удобно использовать цикл *for*?
10. Какие секции могут быть объявлены в цикле *for*?
11. Какая последовательность исполнения секций при работе цикла *for*?
12. В чем отличия между циклами *while* и *do..while*?
13. Какие стандартные математические функции описаны в классе *Math*?

## 6. СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. Павловская Т. А. С#. Программирование на языке высокого уровня. – Изд.: Питер, 2009. – 432с.
2. Эндрю Троелсен. Язык программирования С# 2010 и платформа .NET 4. – Изд.: Вильямс, 2011. – 1392с.
3. Кристиан Нейгел, Билл Ивсен, Джей Глинн, Карли Уотсон, Морган Скиннер. С# 4.0 и платформа .NET 4 для профессионалов. – Изд.: Питер, 2011. – 1440с.

*Дополнительная*

4. Тыртышников Е.Е. Методы численного анализа. Учебное пособие. – Изд.: МГУ, 2006. – 281с.
5. Джесс Либерти. Программирование на С#. – Изд.: КноРус, 2003. – 688с.
6. Харви Дейтел. С# в подлиннике. Наиболее полное руководство. – Изд.: БХВ-Петербург, 2006. – 1056с.