



МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ВЫСШАЯ ШКОЛА ПЕЧАТИ И МЕДИАИНДУСТРИИ

*Институт Принтмедиа и информационных технологий
Кафедра Информатики и информационных технологий*

направление подготовки

09.03.02 «Информационные системы и технологии»

ЛАБОРАТОРНАЯ РАБОТА № 12

Дисциплина: Введение в программирование.

Выполнил(а):

студент(ка) группы 191-726

Синельникова К.Т.

(Дата)

(Подпись)

Проверил: асс. Кононенко К.М.

(Дата)

(Подпись)

Замечания: _____

Москва

2019

Оглавление

Теория	3
Задания	14
Блок-схемы.....	15
Код программы.....	22
Результат программы.....	27

Теория

Методы

Метод — это блок кода, содержащий ряд инструкций. Программа инициирует выполнение инструкций, вызывая метод и указывая все аргументы, необходимые для этого метода. В C# все инструкции выполняются в контексте метода. Метод Main является точкой входа для каждого приложения C#, и вызывается общезыковой средой выполнения (CLR) при запуске программы.

Сигнатуры методов

Методы объявляются в class или struct, для чего указывается следующее:

- Уровень доступа (необязательно), например public или private.

Значение по умолчанию — private.

- Необязательные модификаторы, например abstract или sealed.
- Возвращаемое значение или void, если у метода его нет.
- Имя метода.
- Любые параметры методов. Параметры метода заключаются в

скобки и разделяются запятыми. Пустые скобки указывают, что параметры методу не требуются. Вместе все эти части формируют сигнатуру метода. Тип возврата метода не является частью сигнатуры метода в целях перегрузки метода. Однако он является частью сигнатуры метода при определении совместимости между делегатом и методом, который он указывает. Обратите внимание на то, что класс Motorcycle включает перегруженный метод Drive. Оба метода называются одинаково, но различаются по типам параметров.

Вызов метода

Можно использовать метод *instance* или *static*. Для того чтобы вызвать метод instance, необходимо создать экземпляр объекта и вызвать для него метод; метод instance будет применен к этому экземпляру и его данным. Статический метод вызывается путем ссылки на имя типа, к которому относится метод; статические методы не работают с данными экземпляров. При попытке вызвать статический метод с помощью экземпляра объекта возникает ошибка компилятора.

Вызов метода аналогичен доступу к полю. После имени объекта (при вызове метода экземпляра) или имени типа (при вызове метода `static`) добавьте точку, имя метода и круглые скобки. Аргументы перечисляются в этих скобках и разделяются запятыми.

Определение метода задает имена и типы всех необходимых параметров. Когда вызывающий код вызывает метод, он предоставляет конкретные значения, называемые аргументами, для каждого параметра. Аргументы должны быть совместимы с типом параметра, но имя аргумента (если оно используется в вызывающем коде) может не совпадать с именем параметра, указанным в методе. В следующем примере метод `Square` имеет один параметр типа `int` с именем `i`. Первый вызов метода передает методу `Square` переменную типа `int` с именем `nit`; второй — числовую константу, а третий — выражение.

В наиболее распространенной форме вызова методов используются позиционные аргументы; они передаются в том же порядке, что и параметры метода. Таким образом, методы класса `Motorcycle` могут вызываться, как показано в следующем примере. Например, вызов метода `Drive` включает два аргумента, которые соответствуют двум параметрам в синтаксисе метода. Первый становится значением параметра `miles`, а второй — значением параметра `speed`.

При вызове метода вместо позиционных аргументов можно также использовать *именованные аргументы*. При использовании именованных аргументов необходимо указать имя параметра, двоеточие (":"), а затем аргумент. Аргументы для метода могут отображаться в любом порядке, при условии, что все обязательные аргументы присутствуют. В следующем примере для вызова метода `TestMotorcycle.Drive` используются именованные аргументы. В этом примере именованные аргументы передаются из списка параметров метода в обратном порядке.

Метод можно вызывать, используя и позиционные, и именованные аргументы. При этом за именованным аргументом позиционный аргумент идти не может. В следующем примере метод `TestMotorcycle.Drive` из предыдущего

примера вызывается с использованием одного позиционного и одного именованного аргумента.

Унаследованные и переопределенные методы

Помимо членов, определенных в нем явно, тип наследует члены, определенные в его базовых классах. Так как все типы в системе управляемых типов напрямую или косвенно наследуются из класса `Object`, все типы наследуют его члены, такие как `Equals(Object)`, `GetType()` и `ToString()`. В следующем примере определяется класс `Person`, который создает экземпляры двух объектов `Person` и вызывает метод `Person.Equals`, чтобы определить, равны ли эти объекты. При этом метод `Equals` в классе `Person` не определяется, он наследуется из `Object`.

Типы могут переопределять унаследованные члены, используя ключевое слово `override` и обеспечивая реализацию переопределенного метода. Сигнатура метода должна быть такой же, как у переопределенного метода. Следующий пример аналогичен предыдущему за тем исключением, что переопределяет метод `Equals(Object)`. (Он также переопределяет метод `GetHashCode()`, поскольку оба эти метода предназначены для получения согласованных результатов.)

Передача параметров

Типы в C# делятся на *типы значений* и *ссылочные типы*. Список встроенных типов значений см. в разделе Типы и переменные. По умолчанию и типы значений, и ссылочные типы передаются в метод по значению.

Передача параметров по значению

При передаче типа значения в метод по значению вместо самого объекта передается его копия. Это значит, что изменения объекта в вызываемом методе не отражаются на исходном объекте, когда управление возвращается вызывающему объекту.

Код в следующем примере передает тип значения в метод по значению, а вызываемый метод пытается изменить значение типа значения. Он определяет переменную типа `int`, который является типом значения, присваивает ему значение 20 и передает его в метод с именем `ModifyValue`, который изменяет значение переменной на 30. Однако, когда метод возвращается, значение переменной остается неизменным.

Если объект ссылочного типа передается в метод по значению, ссылка на этот объект передается по значению. Это значит, что метод получает не сам объект, а аргумент, который указывает расположение объекта. Если с помощью этой ссылки в член объекта вносится изменение, это изменение отражается в объекте, даже если управление возвращается вызывающему объекту. При этом изменения в объекте, переданном в метод, не отражаются на исходном объекте, когда управление возвращается вызывающему объекту.

В следующем примере определяется класс (ссылочного типа) с именем `SampleRefType`. Он создает экземпляр объекта `SampleRefType`, задает в его поле `value` значение 44 и передает объект в метод `ModifyObject`. В этом примере, в сущности, происходит то же самое, что и в предыдущем, — аргумент передается в метод по значению. Однако поскольку здесь используется ссылочный тип, результат будет другим. В данном случае в методе `ModifyObject` изменено поле `obj.value`, при этом поле `value` аргумента `rt` в методе `Main` также изменяется на 33, как видно из результатов в предыдущем примере.

Передача параметров по ссылке

Параметр передается по ссылке, когда нужно изменить значение аргумента в методе и сохранить это изменение после того, как управление вернется вызывающему методу. Для передачи параметра по ссылке используйте ключевое слово [ref](#) или [out](#). Можно также передать значение по ссылке, чтобы предотвратить копирование, и при этом запретить внесение изменений с помощью ключевого слова [in](#).

Следующий пример идентичен предыдущему за тем исключением, что значение передается в метод `ModifyValue` по ссылке. Если значение параметра в методе `ModifyValue` будет изменено, при возвращении управления вызывающему объекту это изменение не сохранится.

Общий шаблон, в котором используются параметры по ссылке, включает замену значений переменных. Когда две переменные передаются в метод по ссылке, он меняет их содержимое местами. В следующем примере меняются местами целочисленные значения.

Передача параметров ссылочного типа позволяет изменить значение самой ссылки, а не отдельных ее элементов или полей.

Массивы параметров

В некоторых случаях требование об указании точного числа аргументов для метода является строгим. Если параметр в массиве параметров указывается с помощью ключевого слова `params`, метод можно вызывать с переменным числом аргументов. Параметр, помеченный ключевым словом `params`, должен быть типом массива и занимать последнюю позицию в списке параметров метода.

После этого вызывающий объект можно вызвать одним из трех способов:

- передавая массив соответствующего типа, содержащий требуемое число элементов;
- передавая в метод список отдельных аргументов соответствующего типа, разделенный запятыми;
- не передавая никакие аргументы в массив параметров.

В следующем примере определяется метод с именем `GetVowels`, возвращающий все гласные из массива параметров. Метод `Main` демонстрирует все три способа вызова метода. От вызывающих объектов не требуются аргументы для параметров, которые включают модификатор `params`. В этом случае значение параметра — `null`.

Необязательные параметры и аргументы

В определении метода может быть указано, являются его параметры обязательными или нет. По умолчанию параметры обязательны. Для определения необязательных параметров значения параметра по умолчанию включаются в определение метода. Если при вызове метода никакие аргументы для необязательного параметра не указываются, вместо них используется значение по умолчанию.

Значение параметра по умолчанию должно быть назначено одним из следующих видов выражений:

- Константа, например, строковый литерал или число.
- Выражение в форме `new ValType`, где `ValType` — это тип значения.

Обратите внимание на то, что при этом вызывается не имеющий параметров неявный конструктор типа значения, который не является фактическим членом типа.

- Выражение в форме `default(ValType)`, где `ValType` — это тип значения.

Если метод содержит как обязательные, так и необязательные параметры, необязательные параметры определяются в конце списка параметров после всех обязательных параметров.

В следующем примере определяется метод `ExampleMethod`, который имеет один обязательный и два необязательных параметра.

Если для вызова метода с несколькими необязательными аргументами используются позиционные аргументы, вызывающий объект должен предоставить аргумент для всех необязательных параметров, для которых предоставлен аргумент, от первого до последнего. Например, если при использовании метода `ExampleMethod` вызывающий объект предоставляет аргумент для параметра `description`, он должен также предоставить его для параметра `optionalInt`. `opt.ExampleMethod(2, 2, "Addition of 2 and 2");` —

допустимый вызов метода; `opt.ExampleMethod(2, , "Addition of 2 and 0");` вызывает ошибку компилятора "Аргумент отсутствует".

Если метод вызывается с помощью именованных аргументов или комбинации позиционных и именованных аргументов, вызывающий объект может опустить любые аргументы, следующие за последним позиционным аргументом в вызове метода.

В следующем примере метод `ExampleMethod` вызывается трижды. В первых двух вызовах метода используются позиционные аргументы. В первом пропускаются оба необязательных аргумента, а во втором — последний. Третий вызов метода предоставляет позиционный аргумент для обязательного параметра, но использует именованный аргумент для передачи значения в параметр `description`, в то время как аргумент `optionalInt` опускается.

Использование необязательных параметров влияет на *разрешение перегрузки* или на способ, с помощью которого компилятор C# определяет, какая именно перегрузка должна вызываться при вызове метода, следующим образом:

- Метод, индексатор или конструктор является кандидатом на выполнение, если каждый из его параметров необязателен либо по имени или позиции соответствует одному и тому же аргументу в операторе вызова, и этот аргумент можно преобразовать в тип параметра.
- Если найдено более одного кандидата, правила разрешения перегрузки для предпочтительных преобразований применяются к аргументам, указанным явно. Опущенные аргументы для необязательных параметров игнорируются.
- Если два кандидата определяются как равно подходящие, предпочтение отдается кандидату без необязательных параметров, аргументы которых в вызове были опущены. Это — последовательность определения приоритетов в разрешении перегрузки для кандидатов с меньшим числом параметров.

Возвращаемые значения

Методы могут возвращать значение вызывающему объекту. Если тип возврата, указываемый перед именем метода, не `void`, этот метод может возвращать значение с помощью ключевого слова `return`. Инструкция с ключевым словом `return`, за которым следует переменная, константа или выражение, соответствующие типу возврата, будут возвращать это значение объекту, вызвавшему метод. Методы с типом возврата, отличным от `void`, должны использовать ключевое слово `return` для возврата значения. Ключевое слово `return` также останавливает выполнение метода.

Если тип возврата — `void`, инструкцию `return` без значения по-прежнему можно использовать для завершения выполнения метода. Без ключевого слова `return` этот метод будет останавливать выполнение при достижении конца блока кода.

Использование локальной переменной, в данном случае `result`, для сохранения значения является необязательным. Это может улучшить читаемость кода или может оказаться необходимым, если нужно сохранить исходное значение аргумента для всей области метода.

В некоторых случаях нужно, чтобы метод возвращал больше одного значения. Начиная с C# версии 7.0, это легко можно сделать с помощью *типов кортежей* и *литералов кортежей*. Тип кортежа определяет типы данных для элементов кортежа. Литералы кортежей предоставляют фактические значения возвращаемого кортежа. В следующем примере (`string, string, string, int`) определяет тип кортежа, возвращаемый методом `GetPersonAllInfo`. Выражение (`per.FirstName, per.MiddleName, per.LastName, per.Age`) представляет собой литерал кортежа; метод возвращает имя, отчество и фамилию, а также возраст объекта `PersonInfo`.

Имена могут также назначаться элементам кортежа в определении типа кортежа. В следующих примерах демонстрируется альтернативная версия метода `GetPersonAllInfo`, в котором используются именованные элементы:

Если в качестве аргумента метод получает массив, а затем изменяет значение отдельных элементов, он может не возвращать массив, однако при желании вы можете это изменить для соблюдения правильного стиля или обеспечения эффективного потока передачи значений. Это связано с тем, что C# передает все ссылочные типы по значению, а значением ссылки на массив является указатель на массив. В следующем примере изменения в содержимом массива `values`, сделанные в методе `DoubleValues`, может отслеживать любой код, имеющий ссылку на этот массив.

Методы расширения

Как правило, добавлять методы в существующий тип можно двумя способами:

- Изменение исходного кода для этого типа. Конечно, если вы не владеете исходным кодом этого типа, сделать это невозможно. Если при этом в поддержку метода также добавляются поля закрытых данных, это изменение становится критическим.
- Определение нового метода в производном классе. Нельзя добавить метод этим способом, используя наследование для других типов, таких как структуры и перечисления. Кроме того, оно не позволяет "добавить" метод в запечатанный класс.

Методы расширения позволяют "добавить" метод в существующий тип, не меняя сам тип и не реализуя новый метод в наследуемом типе. Кроме того, метод расширения может не входить в ту же сборку, в которую входит расширяемый им тип. Вызовите метод расширения, как будто он является определенным членом типа.

Дополнительные сведения см. в разделе **Методы расширения**.

Асинхронные методы

С помощью функции `async` можно вызывать асинхронные методы, не прибегая к использованию явных обратных вызовов или ручному разделению кода между несколькими методами или лямбда-выражениями.

Если пометить метод с помощью модификатора `async`, можно использовать в этом методе оператор `await`. Если ожидаемая задача не завершена, то достигнув выражения `await` в асинхронном методе, управление возвращается вызывающему объекту, а выполнение метода с ключевым словом `await` приостанавливается до завершения выполнения ожидаемой задачи. После завершения задачи можно возобновить выполнение в методе.

Примечание

Асинхронный метод возвращается в вызывающий объект, когда он встречает первый ожидаемый объект, выполнение которого еще не завершено, или когда выполнение асинхронного метода доходит до конца — в зависимости от того, что происходит раньше.

Асинхронный метод может иметь тип возвращаемого значения `Task<TResult>`, `Task` или `void`. Тип возвращаемого значения `void` в основном используется для определения обработчиков событий, где требуется возвращать тип `void`. Асинхронный метод, который возвращает тип `void`, не может быть ожидающим. Вызывающий объект метода, возвращающего значение типа `void`, не может перехватывать исключения, которые выдает этот метод. Начиная с C# 7.0 асинхронный метод может возвращать любой тип вида задачи.

В следующем примере `DelayAsync` представляет собой асинхронный метод с оператором `return`, который возвращает целое число. Поскольку этот метод асинхронный, его объявление должно иметь тип возвращаемого значения `Task<int>`. Поскольку тип возврата — `Task<int>`, вычисление выражения `await` в `DoSomethingAsync` создает целое число, как показывает следующий оператор:

```
int result = await delayTask.
```

Асинхронный метод не может объявлять параметры `in`, `ref` или `out`, но может вызывать методы, имеющие такие параметры.

Дополнительные сведения об асинхронных методах см. в разделах Асинхронное программирование с использованием ключевых слов `async` и

await, Поток управления в асинхронных программах и Асинхронные типы возврата.

Элементы, воплощающие выражение

Часто используются определения методов, которые просто немедленно возвращаются с результатом выражения или которые имеют единственную инструкцию в тексте метода. Если метод возвращает void или является асинхронным, текст этого метода должен быть выражением оператора (как и при использовании лямбда-выражений). Свойства и индексаторы должны быть доступны только для чтения, и использовать ключевое слово метода доступа get не следует.

Итераторы

Итератор выполняет настраиваемую итерацию по коллекции, например по списку или массиву. Итератор использует инструкцию yield return для возврата всех элементов по одному. По достижении оператора yield return текущее расположение запоминается, чтобы вызывающий объект мог запросить следующий элемент в последовательности.

Тип возврата итератора может быть IEnumerable, IEnumerable<T>, IEnumerator или IEnumerator<T>.

Задания

1. Описать функцию $\text{PowerA3}(A, B)$, вычисляющую третью степень числа A и возвращающую ее в переменной B (A — входной, B — выходной параметр; оба параметра являются вещественными). С помощью этой функции найти третьи степени пяти данных чисел.

2. Описать функцию $\text{Sign}(X)$ целого типа, возвращающую для вещественного числа X следующие значения:

–1, если $X < 0$; 0, если $X = 0$; 1, если $X > 0$.

С помощью этой функции найти значение выражения $\text{Sign}(A) + \text{Sign}(B)$ для данных вещественных чисел A и B .

3. Описать функцию $\text{RingS}(R1, R2)$ вещественного типа, находящую площадь кольца, заключенного между двумя окружностями с общим центром и радиусами $R1$ и $R2$ ($R1$ и $R2$ — вещественные, $R1 > R2$). С ее помощью найти площади трех колец, для которых даны внешние и внутренние радиусы.

4. Описать функцию $\text{Quarter}(x, y)$ целого типа, определяющую номер координатной четверти, в которой находится точка с ненулевыми вещественными координатами (x, y) . С помощью этой функции найти номера координатных четвертей для трех точек с данными ненулевыми координатами

5. Описать функцию $\text{Fact2}(N)$ вещественного типа, вычисляющую двойной факториал:

$N!! = 1 \cdot 3 \cdot 5 \cdot \dots \cdot N$, если N — нечетное;

$N!! = 2 \cdot 4 \cdot 6 \cdot \dots \cdot N$, если N — четное ($N > 0$ — параметр целого типа;

вещественное возвращаемое значение используется для того, чтобы избежать целочисленного переполнения при больших значениях N).

Блок-схемы

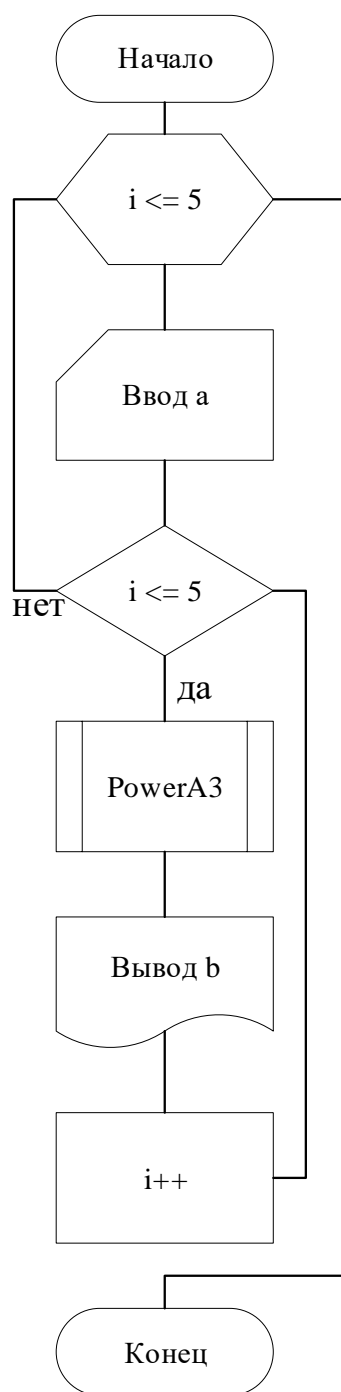


Рисунок 1 — Блок-схема к заданию 1

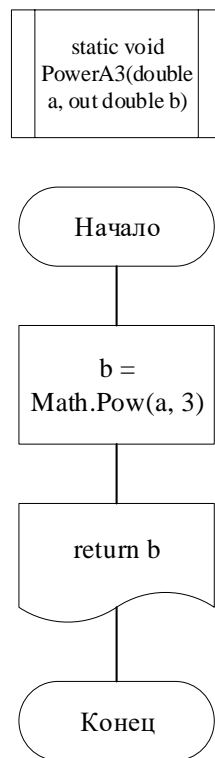


Рисунок 2 — Блок-схема к заданию 1

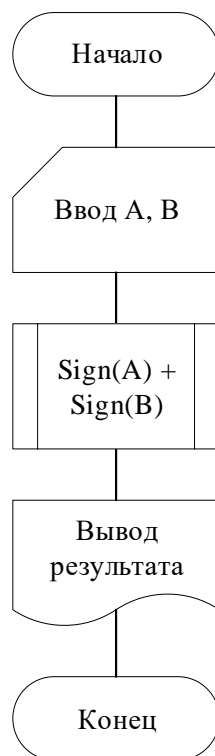


Рисунок 3 — Блок-схема к заданию 2

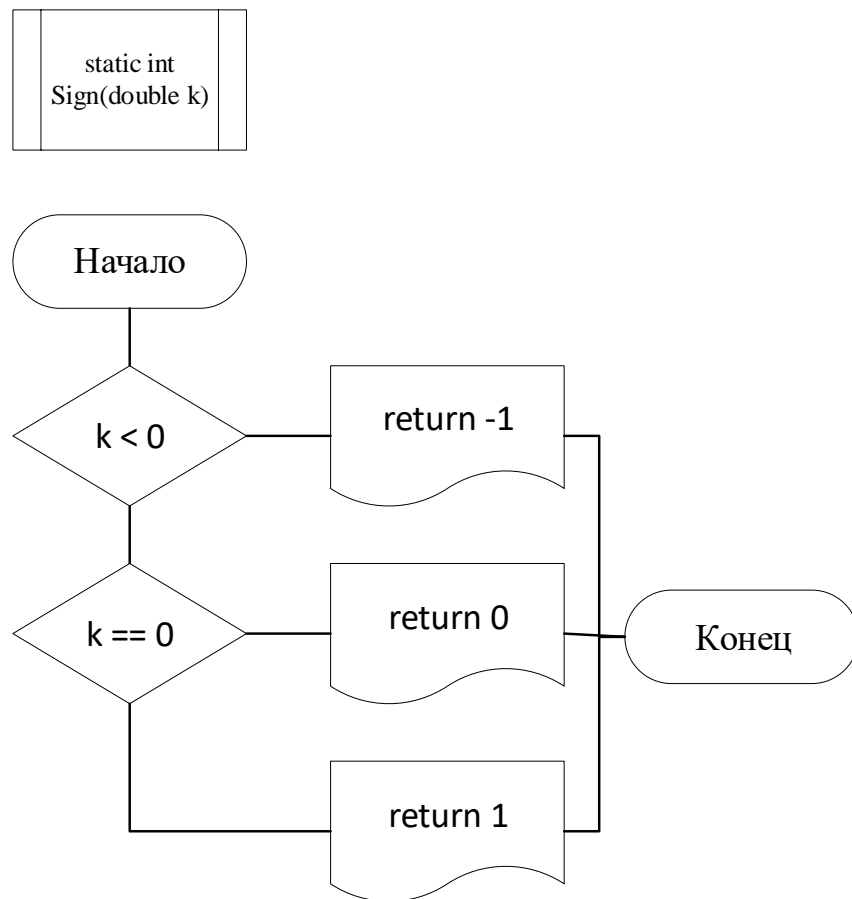


Рисунок 4 — Блок-схема к заданию 2

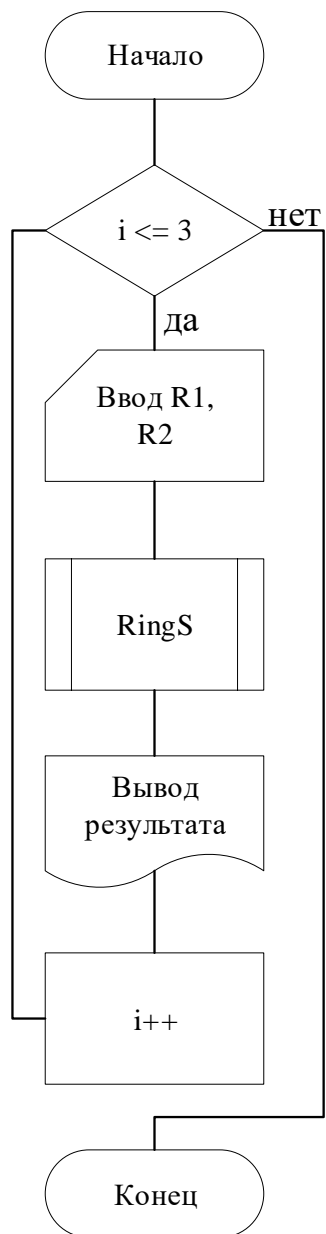


Рисунок 5 — Блок-схема к заданию 3

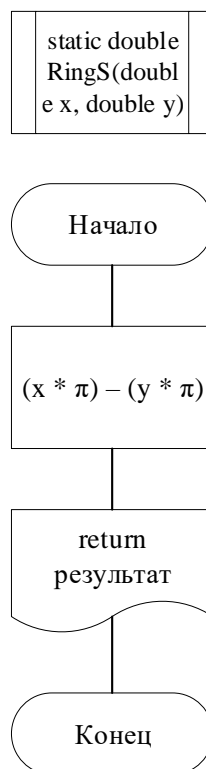


Рисунок 6 — Блок-схема к заданию 3

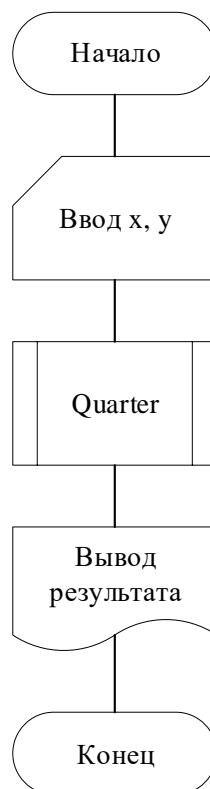


Рисунок 7 — Блок-схема к заданию 4

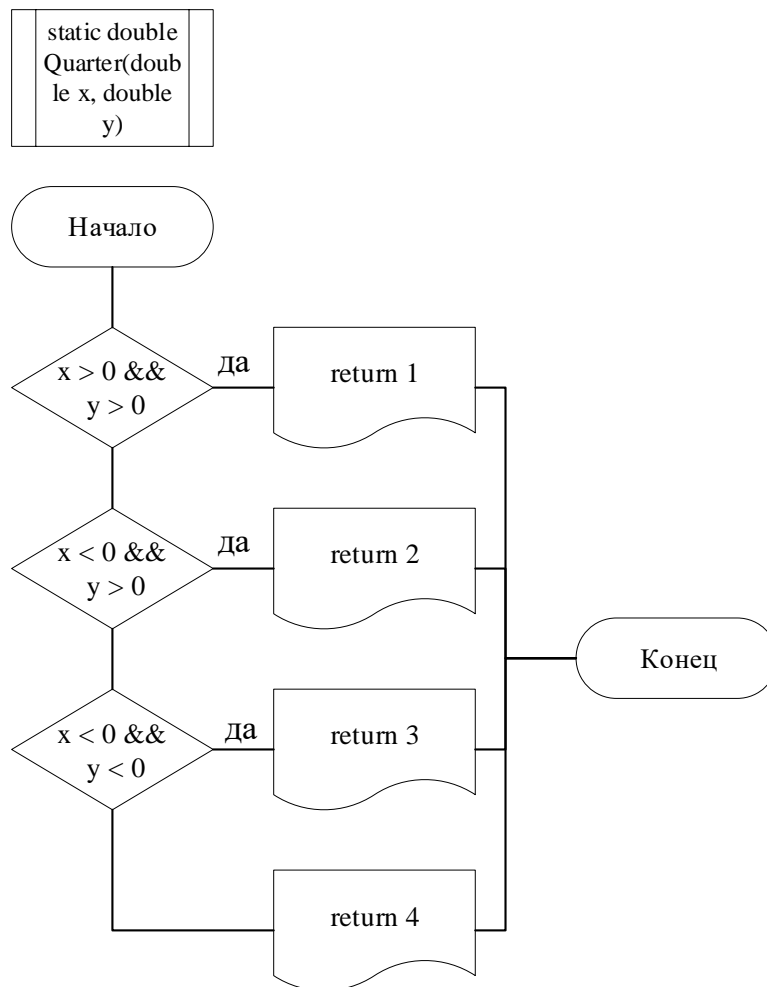


Рисунок 8 — Блок-схема к заданию 4

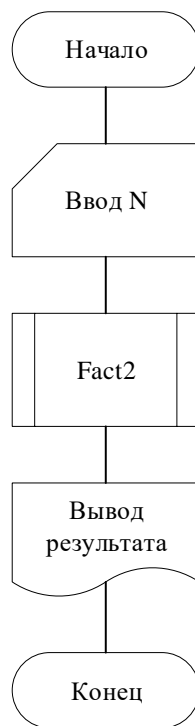


Рисунок 9 — Блок-схема к заданию 5

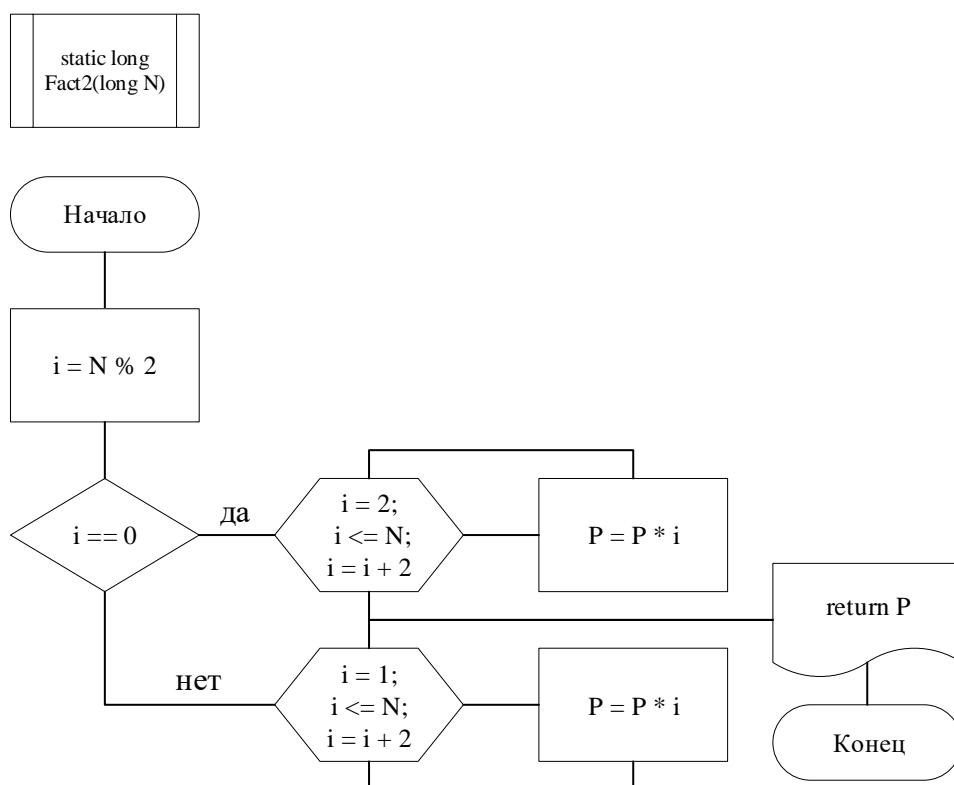


Рисунок 10— Блок-схема к заданию 5

Код программы

Листинг 1 — Задание 1 (PowerA3)

```
1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Text;
5. using System.Threading.Tasks;
6. namespace Код_Лабораторной_12__1_5_
7. {
8.     class Program
9.     {
10.         static void Main(string[] args)
11.         {
12.             double b, a;
13.             int i = 1;
14.             while (i <= 5)
15.             {
16.                 Console.Write("Введите число: ");
17.                 a = double.Parse(Console.ReadLine());
18.                 PowerA3(a, out b);
19.                 Console.WriteLine("Куб вашего числа: " + b);
20.                 i++;
21.             }
22.             Console.ReadKey();
23.         }
24.     }
25. }
```

Листинг 2 — Задание 1 (PowerA3)

```
1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Text;
5. using System.Threading.Tasks;
6. namespace Код_Лабораторной_12__1_5_
7. {
8.     class Program
9.     {
10.         static void PowerA3(double a, out double b)
11.         {
12.             b = Math.Pow(a, 3);
13.         }
14.     }
15. }
```

Листинг 3 — Задание 2 (Sign)

```
1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Text;
5. using System.Threading.Tasks;
6. namespace Код_Лабораторной_12__1_5_
7. {
8.     class Program
9.     {
10.         static void Main(string[] args)
11.         {
12.             double A, B;
13.             Console.Write("Введите число A: ");
14.             A = double.Parse(Console.ReadLine());
15.             Console.Write("Введите число B: ");
16.             B = double.Parse(Console.ReadLine());
17.             Console.WriteLine("Значение выражения Sign(A)+Sign(B)= " + (Sign(A)
+ Sign(B)));
18.             Console.ReadKey();
19.         }
20.     }
21. }
```

Листинг 4 — Задание 2 (Sign)

```
1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Text;
5. using System.Threading.Tasks;
6. namespace Код_Лабораторной_12__1_5_
7. {
8.     class Program
9.     {
10.         static int Sign(double k)
11.         {
12.             if (k < 0) return -1;
13.             else if (k == 0) return 0;
14.             else return 1;
15.         }
16.     }
17. }
```

Листинг 5 — Задание 3 (RingS)

```
1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Text;
5. using System.Threading.Tasks;
6. namespace Код_Лабораторной_12__1_5_
7. {
8.     class Program
9.     {
10.         static void Main(string[] args)
11.         {
12.             double R1, R2;
13.             int i = 1;
14.             while (i <= 3)
15.             {
16.                 Console.Write("Введите больший радиус: ");
17.                 R1 = double.Parse(Console.ReadLine());
18.                 Console.Write("Введите меньший радиус: ");
19.                 R2 = double.Parse(Console.ReadLine());
20.                 Console.WriteLine("Площадь кольца = " + RingS(R1, R2));
21.                 i++;
22.             }
23.             Console.ReadKey();
24.         }
25.     }
26. }
```

Листинг 6 — Задание 3 (RingS)

```
1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Text;
5. using System.Threading.Tasks;
6. namespace Код_Лабораторной_12__1_5_
7. {
8.     class Program
9.     {
10.         static double RingS(double x, double y)
11.         {
12.             return ((x * Math.PI) - (y * Math.PI));
13.         }
14.     }
15. }
```


Листинг 7 — Задание 4 (Quarter)

```
1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Text;
5. using System.Threading.Tasks;
6. namespace Код_Лабораторной_12__1_5_
7. {
8.     class Program
9.     {
10.         static void Main(string[] args)
11.         {
12.             double x, y;
13.             Console.Write("Введите первую координату : ");
14.             x = double.Parse(Console.ReadLine());
15.             Console.Write("Введите вторую координату: ");
16.             y = double.Parse(Console.ReadLine());
17.             Console.WriteLine(Quarter(x, y) + " координатная четверть ");
18.             Console.ReadKey();
19.         }
20.     }
21. }
```

Листинг 8 — Задание 4 (Quarter)

```
1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Text;
5. using System.Threading.Tasks;
6. namespace Код_Лабораторной_12__1_5_
7. {
8.     class Program
9.     {
10.         static double Quarter(double x, double y)
11.         {
12.             if (x > 0 && y > 0) return 1;
13.             else if (x < 0 && y > 0) return 2;
14.             else if (x < 0 && y < 0) return 3;
15.             else return 4;
16.         }
17.     }
18. }
```

Листинг 9 — Задание 5 (Fact2)

```

1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Text;
5. using System.Threading.Tasks;
6. namespace Код_Лабораторной_12__1_5_
7. {
8.     class Program
9.     {
10.         static void Main(string[] args)
11.         {
12.             long N;
13.             Console.Write("Введите число: ");
14.             N = long.Parse(Console.ReadLine());
15.             Console.Write("Двойной факториал вашего числа = " + Fact2(N));
16.             Console.ReadKey();
17.         }
18.     }
19. }

```


Листинг 10 — Задание 5 (Fact2)

```

1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Text;
5. using System.Threading.Tasks;
6. namespace Код_Лабораторной_12__1_5_
7. {
8.     class Program
9.     {
10.         static long Fact2(long N)
11.         {
12.             long P = 1;
13.             long i;
14.             i = N % 2;
15.             if (i == 0)
16.                 for (i = 2; i <= N; i = i + 2)
17.                     P = P * i;
18.             else
19.                 for (i = 1; i <= N; i = i + 2)
20.                     P = P * i;
21.             return P;
22.         }
23.     }
24. }


```

Результат программы

 C:\Лабы С#\Лаб.раб. 12\Код Лабораторной 12 (1-5)


```
Введите число: 5
Куб вашего числа: 125
Введите число: 2
Куб вашего числа: 8
Введите число: 6
Куб вашего числа: 216
Введите число: 3
Куб вашего числа: 27
Введите число: 4
Куб вашего числа: 64
```

Рисунок 11 — Результат выполнения программы 1

 C:\Лабы С#\Лаб.раб. 12\Код Лабораторной 12 (1-5)


```
Введите число A: 27
Введите число B: -18
Значение выражения Sign(A)+Sign(B)= 0
```

Рисунок 12 — Результат выполнения программы 2

 C:\Лабы С#\Лаб.раб. 12\Код Лабораторной 12 (1-5)


```
Введите больший радиус: 21
Введите меньший радиус: 9
Площадь кольца = 37,6991118430775
Введите больший радиус: 3
Введите меньший радиус: 1
Площадь кольца = 6,28318530717959
Введите больший радиус: 7
Введите меньший радиус: 4
Площадь кольца = 9,42477796076938
```

Рисунок 13 — Результат выполнения программы 3

 C:\Лабы С#\Лаб.раб. 12\Код Лабораторной 12 (1-5)

```
Введите первую координату : 5
Введите вторую координату: -6
4 координатная четверть
```

Рисунок 14 — Результат выполнения программы 4

 C:\Лабы С#\Лаб.раб. 12\Код Лабораторной 12 (1-5)

```
Введите число: 5
Двойной факториал вашего числа = 15
```

Рисунок 15 — Результат выполнения программы 5