Лабораторная работа №04. Основные конструкции языка C#

1 Цель и порядок работы

Цель работы – ознакомиться с основными конструкциями языка C#.

Порядок выполнения работы:
- ознакомиться с описанием лабораторной работы;
- получить задание у преподавателя, согласно своему варианту;
- написать программу и отладить ее на ЭВМ.

2 Краткая теория

2.1 Правила синтаксиса.

При написании программы придерживаются синтаксических правил таких как:

{ } операторные скобки объединяют несколько операторов в один блок,
; конец оператора,
, разделитель при перечислении констант, переменных,
( ) содержат параметры функций или операторов.

2.2 Комментарии.

C# представляет несколько механизмов комментирования кода:
- построчное //
- многострочное /* */
- комментарии, которые позволяют автоматически генерировать документацию в XML – формате. Эти комментарии начинаются с символов ///, за которыми следуют специальные тэги.

2.3 Класс Console. Консольный ввод–вывод.

При обсуждении процедур ввода–вывода следует иметь в виду одно важное обстоятельство. Независимо от типа выводимого значения, в конечном результате выводится, СИМВОЛЬНОЕ ПРЕДСТАВЛЕНИЕ значения. Это становится очевидным при выводе информации в окно приложения, что и обеспечивают по умолчанию методы Console.Write и Console.WriteLine.

Консольный вывод.

Методы WriteLine и Write позволяют отображать информацию на консоль.

```
// Вывод строки.
Console.WriteLine("The Captain is on the board!");
//Символьное представление целочисленного значения.
Console.WriteLine(314);
//Символьное представление значения типа float.
Console.WriteLine(3.14);
```

Могут принимать различное число параметров. Однако:

```
Console.WriteLine("qwerty",314,3.14);
```

Результатом выполнения выражения вызова функции будет строка:

qwerty

Значения второго и третьего параметров не выводятся. Первый строковый параметр выражений вызова функций Write и WriteLine используется как управляющий шаблон для представления выводимой информации. Значения следующих за строковым параметром выражений будут выводиться в окно представления лишь в том случае, если первый параметр–строка будет явно указывать места расположения выводимых значений, соответствующих этим параметрам, маркерами, которые в самом простом случае представляют собой заключённые в фигурные скобки целочисленные литералы.

При этом способ указания места состоит в следующем:

- CLR индексирует все параметры метода WriteLine, следующие за первым параметром–строкой. При этом второй по порядку параметр получает индекс 0, следующий за ним – 1, и т.д. до конца списка параметров.
- В произвольных местах параметра–шаблона размещаются маркеры выводимых значений.
- Значение маркера должно соответствовать индексу параметра, значение которого должно выводиться на экран.
- Значение целочисленного литерала маркера не должно превышать максимального значения индекса параметра.

Таким образом, оператор

```
Console.WriteLine("The sum of {0} and {1} is {2}",314,3.14,314+3.14);
```

обеспечивает вывод следующей строки:

The sum of 314 and 3.14 is 317.3

В последнем операнде выражения вызова WriteLine при вычислении значения выражения используется неявное приведение типа. При вычислении значения суммы операнд типа int без потери значения приводится к типу float. Безопасные преобразования типов проводятся автоматически.

Несмотря на явную абсурдность выводимых утверждений, операторы

```
Console.WriteLine("The sum of {0} and {1} is {0}",314,3.14,314+3.14);
Console.WriteLine("The sum of {2} and {1} is {0}",314,3.14,314+3.14);
```

также отработают вполне корректно.

Форматирование строки вывода.

Помимо индекса параметра маркер выводимого значения может содержать дополнительную информацию относительно формата представления выводимой информации.

Выводимые значения преобразуются к символьному представлению, которое, в свою очередь, при выводе в окно приложения может быть дополнительно преобразовано в соответствии с предопределённым "сценарием преобразования".

Вся необходимая для дополнительного форматирования информация размещается непосредственно в маркерах и отделяется запятой от индекса маркера.

Кроме того, в маркерах вывода могут также размещаться дополнительные строки форматирования (FormatString). При этом маркер приобретает достаточно сложную структуру, внешний вид которой в общем случае можно представить следующим образом (M – значение индекса, N – область позиционирования):

{M,N:FormatString}

Сама же строка форматирования аналогична ранее рассмотренной строке–параметру метода ToString и является комбинацией предопределённых символов форматирования и дополнительных целочисленных значений (см. табл.1).

Таблица 1. Предопределенные символы форматирования.

Символ форматирования	Описание
C	Отображение значения как валюты с использованием принятого по соглашению символа
D	Отображение значения как decimal integer
E	Отображение значения в соответствии с научной нотацией
F	Отображение значения как fixed Point
G	Display the number as a fixed-Point or integer, depending on which is the most compact
N	Применение запятой для разделения порядков
X	Отображение значения в шестнадцатеричной нотации

Непосредственно за символом форматирования может быть расположена целочисленная ограничительная константа, которая в зависимости от типа выводимого значения может определять количество выводимых знаков после точки, либо общее количество выводимых символов. При этом дробная часть действительных значений округляется, либо дополняется нулями справа. При выводе целочисленных значений ограничительная константа игнорируется, если количество выводимых символов превышает её значение. В противном случае выводимое значение слева дополняется нулями.

Следующие примеры иллюстрируют варианты применения маркеров со строками форматирования:

```
Console.WriteLine("Integer fotmating - {0:D3},{1:D5}",12345, 12);
Console.WriteLine("Currency formatting - {0:C},{1:C5}", 99.9, 999.9);
Console.WriteLine("Exponential formatting - {0:E}", 1234.5);
Console.WriteLine("Fixed Point formatting - {0:F3}", 1234.56789);
Console.WriteLine("General formatting - {0:G}", 1234.56789);
Console.WriteLine("Number formatting - {0:N}", 1234567.89);
//Integers only!
Console.WriteLine("Hexadecimal formatting - {0:X7}",12345);
```

В результате выполнения этих операторов в окно консольного приложения будут выведены следующие строки

```
Integer fotmating - 12345,00012
Currency formatting - $99.90,$999.90000
Exponential formatting - 1.234500E+003
Fixed Point formatting - 1234.568
General formatting - 1234.56789
Number formatting - 1,234,567.89
Hexadecimal formatting - 0003039
```

Консольный ввод.

Метод Read читает по одному символу до тех пор, пока все символы в потоке не закончатся, и возвращает код символа либо -1, если символов больше нет в потоке.

Метод Readln читает строку символов.

```
string s = Console.ReadLine();
```

Результатом считывания всегда будет строка. Поэтому, для получения нужного типа данных необходимо использовать преобразование.

2.4 Преобразования и класс Convert.

Класс Convert, определенный в пространстве имен System, играет важную роль, обеспечивая необходимые преобразования между различными типами. Внутри арифметического типа можно использовать явный (скобочный) способ приведения к нужному типу. Но таким способом нельзя привести, например, переменную типа string к типу int, Оператор присваивания: ux = (int)s1; приведет к ошибке периода компиляции. Здесь необходим вызов метода ToInt32 класса Convert.

Методы класса Convert поддерживают общий способ выполнения преобразований между типами. Класс Convert содержит 15 статических методов вида To<Type> (ToBoolean(),…ToUInt64()), где Type может принимать значения от Boolean до UInt64 для всех встроенных типов. Единственным исключением является тип object, – метода ToObject нет по понятным причинам, поскольку для всех типов существует неявное преобразование к типу object.

Также есть возможность преобразования к системному типу DateTime, который хотя и не является встроенным типом языка C#, но допустим в программах, как и любой другой системный тип.

```
//System type: DateTime
System.DateTime dat = Convert.ToDateTime("15.03.2003");
Console.WriteLine("Date = {0}", dat);
```

Результатом вывода будет строка:

Date = 15.03.2003 0:00:00

Все методы To<Type> класса Convert перегружены и каждый из них имеет, как правило, более десятка реализаций с аргументами разного типа. Так что фактически эти методы задают все возможные преобразования между всеми встроенными типами языка C#.

Кроме методов, задающих преобразования типов, в классе Convert имеются и другие методы, например, задающие преобразования символов Unicode в однобайтную кодировку ASCII, преобразования значений объектов и другие методы.

Преобразование типа с использованием класса Convert создает значение нового типа, эквивалентное значению старого типа, однако при этом не обязательно сохраняется идентичность (или точные значения) двух объектов.

Различаются преобразования:

- расширяющие;
- сужающие.

Расширяющее преобразование.

Значение одного типа преобразуется к значению другого типа, которое имеет такой же или больший размер. Например, значение, представленное в виде 32–разрядного целого числа со знаком, может быть преобразовано в 64–разрядное целое число со знаком.

Расширяющее преобразование считается безопасным, поскольку исходная информация при таком преобразовании не искажается (см. табл. 2.1).

Таблица 2.1. Перечень безопасных преобразований к типу.

Тип	Возможно безопасное преобразование к типу…
Byte	UInt16, Int16, UInt32, Int32, UInt64, Int64, Single, Double, Decimal
SByte	Int16, Int32, Int64, Single, Double, Decimal
Int16	Int32, Int64, Single, Double, Decimal
UInt16	UInt32, Int32, UInt64, Int64, Single, Double, Decimal
Char	UInt16, UInt32, Int32, UInt64, Int64, Single, Double, Decimal
Int32	Int64, Double, Decimal
UInt32	Int64, Double, Decimal
Int64	Decimal
UInt64	Decimal
Single	Double

Некоторые расширяющие преобразования типа могут привести к потере точности. Таблица 2.2 описывает варианты преобразований, которые иногда приводят к потере информации.

Таблица 2.2. Перечень преобразований, при которых возможна потеря точности.

Тип	Возможна потеря точности при преобразовании к типу…
Int32	Single
UInt32	Single
Int64	Single, Double
UInt64	Single, Double
Decimal	Single, Double

Сужающее преобразование

Значение одного типа преобразуется к значению другого типа, которое имеет меньший размер (из 64–разрядного в 32–разрядное).

Такое преобразование потенциально опасно потерей значения.

Сужающие преобразования могут приводить к потере информации. Если тип, к которому осуществляется преобразование, не может правильно передать значение источника, то результат преобразования оказывается равен константе PositiveInfinity или NegativeInfinity.

При этом значение PositiveInfinity интерпретируется как результат деления положительного числа на ноль, а значение NegativeInfinity интерпретируется как результат деления отрицательного числа на ноль.

Если сужающее преобразование обеспечивается методами класса System.Convert, то потеря информации сопровождается генерацией исключения, которые рассмотрим позже (см. табл. 3).

Таблица 3. Варианты сужающих преобразований.

Тип	Возможна потеря значения и генерация исключения при преобразовании в:
Byte	Sbyte
SByte	Byte, UInt16, UInt32, UInt64
Int16	Byte, SByte, UInt16
UInt16	Byte, SByte, Int16
Int32	Byte, SByte, Int16, UInt16, UInt32
UInt32	Byte, SByte, Int16, UInt16, Int32

Int64	Byte, SByte, Int16, UInt16, Int32, UInt32, UInt64
UInt64	Byte, SByte, Int16, UInt16, Int32, UInt32, Int64
Decimal	Byte, SByte, Int16, UInt16, Int32, UInt32, Int64, UInt64
Single	Byte, SByte, Int16, UInt16, Int32, UInt32, Int64, UInt64
Double	Byte, SByte, Int16, UInt16, Int32, UInt32, Int64, UInt64

Метод Parse().

Этот метод поддерживают все предопределенные типы значений. Используется для извлечения из строки значения нужного типа.

Например: `int i = Int32.Parse(Console.ReadLine());`

2.5 Класс System.Environment. Окружение.

Класс System.Environment позволяет узнать информацию о текущем окружении и платформе. Основные его свойства и методы сведены в таблицу 4.

Таблица 4. Свойства и методы класса System.Environment.

Название	Описание
Свойства	
CommandLine	возвращает командную строку, с помощью которой была запущена программа
CurrentDirectory	содержит текущую директорию, с которой был запущен процесс
ExitCode	не зависимо от типа возврата Main (int или void), процесс будет возвращать значение
MashineName	позволяет узнать имя машины, на которой запущен процесс
NewLine	возвращает строку, которая содержит символ перехода на новую строку в текущей системе
OSVersion	возвращает специальный объект, который содержит описание текущей платформы и версию
SystemDirectory	возвращает полный путь к системной директории
UserDomainName	возвращает имя домена, с которым ассоциируется текущий пользователь системы
UserName	возвращает имя пользователя системы
Методы	
Exit()	завершает текущий процесс, возвращая системе код выхода, который определен свойством ExitCode
GetCommandLineArgs()	позволяет вернуть массив строк, которые содержат параметры командной строки, передаваемые программе
GetEnvironmentVariale()	позволяет вернуть значение переменной окружения
GetEnvironmentVariales()	позволяет вернуть список всех переменных окружения и их значения
GetLogicalDrives()	позволяет вернуть список всех логических дисков, которые имеются на машине

2.6 Система типов

C# является языком со строгой типизацией данных. Таким образом, IL не допускает никаких действий, которые дают в результате неопределенные типы данных.

.NET Framework предоставляет общую систему типов CTS (Common Type System), использование которой позволяет разрабатывать приложение на любом из языков,

поддерживающих эту среду, и при этом не заботиться о несовместимости типов при повторном использовании разработанных компонентов.

Система типов поддерживает две категории типов, каждая из которых разделена на подкатегории:

- типы значений (типы–значения),
- ссылочные типы (типы–ссылки).

Схема типов представлена на рисунке 1.

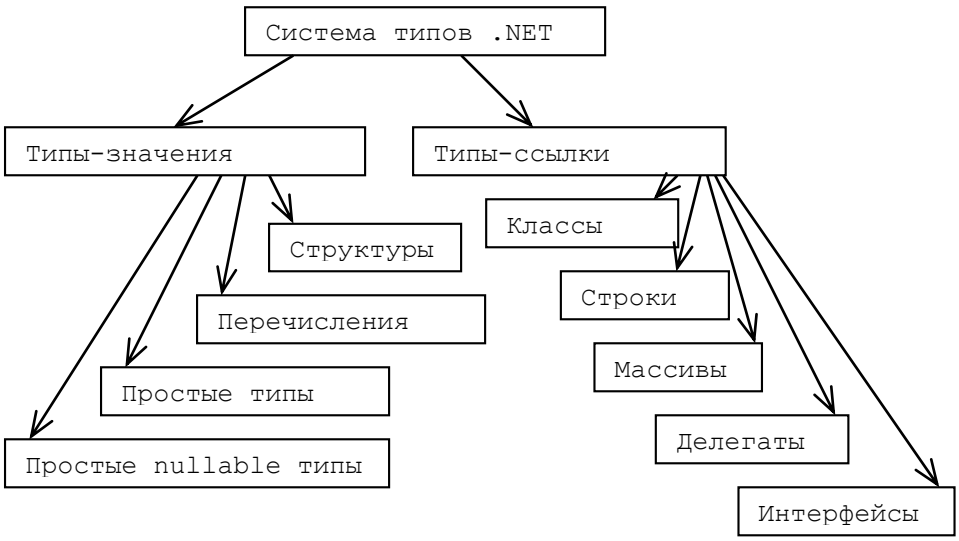


Рис.1. Система типов .Net

Ниже представлены основные отличия ссылочных типов и типов–значений.

Таблица 5. Отличия типов-значения от типов-ссылок.

	Типы–значения value-types	Типы–ссылки reference-types
Объект представлен	непосредственно значением	ссылкой в куче
Объект располагается	в стеке	в куче
Значение по умолчанию	0, false, '\0', null	ссылка имеет значение null
При выполнении операции присваивания копируется	значение	ссылка

Для типов-ссылок необходимо явно выделять место в памяти, используя метод New().

В C# объявление любой структуры и класса основывается на объявлении предопределённого класса Object (наследует класс Object). Следствием этого является возможность вызова от имени объектов–представителей любой структуры или класса унаследованных от класса object методов. В частности, метода ToString. Этот метод возвращает строковое (значение типа string) представление объекта.

Все типы (типы-значения и типы-ссылки), за исключением простых типов-значений и пары предопределённых ссылочных типов (string и object), должны определяться (если уже не были ранее специально определены) программистами в рамках объявлений. Подлежащие объявлению типы называются производными типами.

Простые (элементарные) типы

Это типы, имя и основные свойства которых известны компилятору. Относительно элементарных типов компилятору не требуется никакой дополнительной информации. Свойства и функциональность этих типов известны.

Используемые в .NET языки программирования основываются на общей системе типов. Между именами простых типов в C# и именами типов, объявленных в Framework Class Library, существует взаимно однозначное соответствие (см. табл. 6).

Таблица 6. Соответствие простых типов C# FCL-типу.

Тип в C#	Соответствует FCL-типу	Описание
sbyte	*System.SByte*	Целый. 8-разрядное со знаком. Диапазон значений: 128 … 127
byte	*System.Byte*	Целый. 8-разрядное без знака. Диапазон значений: 0 … 255
short	*System.Int16*	Целый. 16-разрядное со знаком. Диапазон значений: -32768 … 32767
ushort	*System.UInt16*	Целый. 16-разрядное без знака. Диапазон значений: 0 … 65535
int	*System.Int32*	Целый. 32-разрядное со знаком. Диапазон значений: -2147483648 … 2147483647
uint	*System.UInt32*	Целый. 32-разрядное без знака. Диапазон значений: -0 … 4294967295
long	*System.Int64*	Целый. 64-разрядное со знаком. Диапазон значений: -9223372036854775808 … 9223372036854775807
ulong	*System.UInt64*	Целый. 64-разрядное без знака. Диапазон значений: 0 … 18446744073709551615
char	*System.Char*	16 (!) разрядный символ UNICODE.
float	*System.Single*	Плавающий. 32 разряда. Стандарт IEEE.
double	*System.Double*	Плавающий. 64 разряда. Стандарт IEEE.
decimal	*System.Decimal*	128-разрядное значение повышенной точности с плавающей точкой.
bool	*System.Boolean*	Значение true или false.

Перечисления

Это тип-значение, состоящий из набора поименованных констант.

Формат определения перечисления:

```
enum Имя [:базовый класс]
{
    Имя0 =  Значение0,…,ИмяN = ЗначениеN
}
```

По умолчанию нумерация начинается с нуля и увеличивается на 1.

В качестве примера опишем перечисление, указывающее день недели:

```
enum DayOfWeek { Monday = 1, Tuesday = 2, Wednesday = 3,
          Thursday = 4, Friday = 5, Saturday = 6, Sunday = 7 };
//Переменная xVal инициализируется значением перечисления.
int xVal = DayOfWeek.Friday;
```

Перечисление является классом, а это означает, что в распоряжении программиста оказываются методы сравнения значений перечисления, методы преобразования значений перечисления в строковое представление, методы перевода строкового представления значения в перечисление, а также (судя по документации) средства для создания объектов–представителей класса перечисления.

Список членов класса перечисления сведен в таблицу 7.

Таблица 7. Соответствие простых типов C# FCL-типу.

Название метода	Описание
Открытые методы	
CompareTo()	Сравнивает этот экземпляр с заданным объектом и возвращает сведения об их относительных значениях.
Equals()	Переопределен. Возвращает значение, показывающее, равен ли данный экземпляр заданному объекту.
Format()	Статический. Преобразует указанное значение заданного перечисляемого типа в эквивалентное строчное представление в соответствии с заданным форматом.
GetHashCode()	Переопределен. Возвращает хеш–код для этого экземпляра.
GetName()	Статический. Выводит имя константы в указанном перечислении, имеющем заданное значение.
GetNames()	Статический. Выводит массив имен констант в указанном перечислении.
GetType() (унаследовано от Object)	Возвращает Type текущего экземпляра.
GetTypeCode()	Возвращает базовый тип TypeCode для этого экземпляра.
GetUnderlyingType()	Статический. Возвращает базовый тип указанного перечисления.
GetValues()	Статический. Выводит массив значений констант в указанном перечислении.
IsDefined()	Статический. Возвращает признак наличия константы с указанным значением в заданном перечислении.
Parse()	Статический. Перегружен. Преобразует строковое представление имени или числового значения одной или нескольких перечисляемых констант в эквивалентный перечисляемый объект.
ToObject()	Статический. Перегружен. Возвращает экземпляр указанного типа перечисления, равный заданному значению.
ToString()	Перегружен. Переопределен. Преобразует значение этого экземпляра в эквивалентное ему строковое представление.
Защищенные методы	
Finalize() (унаследовано от Object)	Переопределен. Позволяет объекту Object попытаться освободить ресурсы и выполнить другие завершающие операции, перед тем как объект Object будет уничтожен в процессе сборки мусора. В языках C# и C++ для функций финализации используется синтаксис деструктора.
MemberwiseClone() (унаследовано от Object)	Создает неполную копию текущего Object.

Структуры

Во многом схожи с классами. Основное их отличие в том, что структуры не являются ссылочным типом. Дополнительно, на структуры накладываются следующие ограничения:

- структура не может иметь деструктор;
- структура не может иметь конструктор без параметров;
- структура не может использоваться как базовый тип.

В качестве примера укажем структуру, описывающую точку.

```
struct Point
{
    public int x, y;
    public Point(int p1, int p2)
    { x = p1; y = p2; }
}
```

2.7 Литералы. Представление значений.

В программах на языках высокого уровня (C# в том числе) литералами называют последовательности входящих в алфавит языка программирования символов, обеспечивающих явное представление значений, которые используются для обозначения начальных значений в объявлении членов классов, переменных и констант в методах класса.

Арифметические литералы

Арифметические литералы кодируют значения различных (арифметических) типов. Тип арифметического литерала определяется следующими интуитивно понятными внешними признаками:

- *стандартным внешним видом.* Значение целочисленного типа обычно кодируется интуитивно понятной последовательностью символов '1',…, '9', '0'. Значение плавающего типа также предполагает стандартный вид (точка–разделитель между целой и дробной частью, либо научная или экспоненциальная нотация – 1.2500E+052). Шестнадцатеричное представление целочисленного значения кодируется шестнадцатеричным литералом, состоящим из символов '0',…,'9', а также 'a',…, 'f', либо 'A',…,'F' с суффиксом '0x',
- *значением – соответствующем типу.* 32768 никак не может быть значением типа short,
- дополнительным суффиксом. Суффиксы l, L соответствуют типу long; ul, UL – unsigned long; f, F – float; d, D – decimal. Значения типа double кодируются без префикса.

Логические литералы

К логическим литералам относятся следующие последовательности символов: true и false. Больше логических литералов в C# нет.

Символьные литералы

Представляют собой заключённые в одинарные кавычки вводимые с клавиатуры одиночные символы: 'X', 'p', 'Q', '7',

В C# char — это 16-разрядный тип без знака, который позволяет представлять значения в диапазоне 0—65 535. Стандартный 8-разрядный набор символов ASCII составляет лишь подмножество Unicode с диапазоном 0—127. Таким образом, ASCII-символы — это действительные C# - символы.

Например, чтобы присвоить значение буквы X переменной ch, нужно выполнить следующие инструкции:

```
char ch;
ch = 'X';
```

Чтобы вывести char-значение, хранимое в переменной ch:

```
Console.WriteLine("Это ch: " + ch) ;
```

Хотя тип char определяется в C# как целочисленный, его нельзя свободно смешивать с целыми числами, т.к. автоматического преобразования целочисленных значений в значения типа char не существует.

Например, следующий фрагмент программы содержит ошибку.

```
//ошибка, это работать не будет.
char ch;
ch = 10;
```

Поскольку 10 — целое число, оно не может быть автоматически преобразовано в значение типа char. При попытке скомпилировать этот код вы получите сообщение об ошибке. Ниже в этой главе мы рассмотрим "обходной путь", позволяющий обойти это ограничение.

Символьные управляющие последовательности.

Эта категория литералов (см. табл. 8) используется для создания дополнительных эффектов (звонок), простого форматирования выводимой информации и кодирования символов при выводе и сравнении (в выражениях сравнения). Заключаются в одинарные кавычки.

Таблица 8. Перечень основных символов управляющих последовательностей.

Управляющая последовательность	Описание
\a	Предупреждение (звонок)
\b	Возврат на одну позицию
\f	Переход на новую страницу
\n	Переход на новую строку
\r	Возврат каретки
\t	Горизонтальная табуляция
\v	Вертикальная табуляция
\0	Ноль
\'	Одинарная кавычка
\"	Двойная кавычка
\\	Обратная косая черта

Строковые литералы

Это последовательность символов и символьных управляющих последовательностей, заключённых в двойные кавычки.

```
…"c:\\My Documents\\sample.txt"…
```

Строковый литерал, интерпретируемый компилятором в таком виде, котором он записан. Управляющие последовательности воспринимаются строго как последовательности символов.

Символы в строковом литерале будут трактоваться так, как они есть, если строковый литерал предварить символом @. Другими словами, не будут интерпретироваться как управляющие последовательности:

```
…@"c:\My Documents\sample.txt"…
```

Оба примера имеют одно и то же значение:

```
c:\My Documents\sample.txt
```

Строковые литералы являются литералами типа string.

2.8 Переменные.

CTS позволяет работать со значениями двух видов:

- переменные, которые непосредственно хранят данные,
- переменные, которые содержат только ссылку на данные (память в этом случае выделяется с использованием оператора new).

Объявление и инициализация.

Выполнение оператора объявления переменной типа-значения в методе класса приводит к созданию в памяти объекта соответствующего типа, возможно проинициализированного определённым значением. Это значение может быть задано в виде литерала соответствующего типа

Например,

```
//определения переменных типов-значений:
int a;
System.Int32 a;

//определение и инициализация переменных типа значения:
int a = 0;
int a = new int();
System.Int32 a = 0;
System.Int32 a = new System.Int32();
```

Следует учитывать: CLR не допускает использования в выражениях неинициализированных локальных переменных, таких, которые объявлены в теле метода.

```
int a;            // Объявление a.
int b;            // Объявление b.
b = 10;       // Инициализация b.
a=b+b;        // Инициализация a.
```

Область видимости.

При определении классов и методов используются фигурные скобки. Содержимое между этими скобками называется блоком. Блоки могут быть вложены друг в друга. Любая переменная, объявленная внутри конкретного блока, называется локальной переменной для этого блока, и она не существует вне, то есть ее нельзя использовать в других блоках, если они не являются вложенными.

```
{
        {
                double a=3;
                . . .
                {
                        Console.Writeln(a);      //правильно
                }
        }
        Console.Writeln(a);                      //ошибка компиляции
}
```

Не допускается объявление переменной во вложенном блоке с тем же именем, что и переменная, объявленная в основном блоке:

```
        {
                double a=3;
                . . .
                {
                        double a;                //ошибка компиляции
                }
                                                  }
```

2.9 Константы

Объявляются с дополнительным спецификатором const. Требуют непосредственной инициализации

```
//константа инициализируется литералом 3.14.
const float Pi = 3.14;
```

2.10 Операции и выражения

Для каждого определённого в C# типа существует собственный набор операций, определённых на множестве значений этого типа.

Эти операции определяют диапазон возможных преобразований, которые могут быть осуществлены над элементами множеств значений типа. Несмотря на специфику разных типов, в C# существует общее основание для классификации соответствующих множеств операций. Каждая операция является членом определённого подмножества операций и имеет собственное графическое представление (см. табл. 9).

Общие характеристики используемых в C# операций представлены ниже.

Таблица 9. Общие характеристики операций, используемых в C#.

Категории операций	**Операции**
Arithmetic	+ – * / %
Логические (boolean и побитовые)	& | ^ ! ~ && ||
Строковые (конкатенаторы)	+
Increment, decrement	++ --

Сдвига	>> <<
Сравнения	== != < > <= >=
Присвоения	= += -= *= /= %= &= |= ^= <<= >>=
Member access	.
Индексации	[]
Cast (приведение типа)	()
Conditional (трёхоперандная)	?:
Delegate concatenation and removal	+ –
Создания объекта	new()
Type information	is sizeof typeof
Overflow exception control (управление исключениями)	checked unchecked
Indirection and Address (неуправляемый код)	* -> [] &

При создании программы на стадии выполнения учитываются следующие обстоятельства:

- приоритет операций (см. табл. 10),
- типы операндов и приведение типов.

Таблица 10. Приоритет операций.

Приоритет	**Операции**
1	() [] . (постфикс)++ (постфикс)-- new sizeof typeof checked unchecked
2	! ~ (имя типа) +(унарный) –(унарный) ++(префикс) --(префикс)
3	* / %
4	+ –
5	<< >>
6	< > <= => is as
7	== !=
8	&
9	^
10	|
11	&&
12	||
13	?:
14	= += -= *= /= %= &= |= ^= <<= >>=

Контроль за переполнением. Checked и unchecked.

Причиной некорректных результатов выполнения арифметических операций является особенность представления значений арифметических типов.

Арифметические типы имеют ограниченные размеры. Поэтому любая арифметическая операция может привести к переполнению. По умолчанию в C# переполнение, возникающее при выполнении операций, никак не контролируется. Возможный неверный результат вычисления остаётся всего лишь результатом выполнения операции.

Механизм контроля за переполнением, возникающим при выполнении арифметических операций, обеспечивается ключевыми словами checked (включить контроль за переполнением) и unchecked (отключить контроль за переполнением), которые используются в составе выражений. CLR выполняет контроль переполнения и, в случае обнаружения такового, генерирует исключение OverflowException. Конструкции управления контролем за переполнением имеют две формы:

- операторную, которая обеспечивает контроль над выполнением одного выражения:

```
:::::
short x = 32767;
short y = 32767;
short z = 0;

try
{
 z = checked(x + unchecked(x+y));
}
catch (System.OverflowException e)
{
 Console.Writeline("Переполнение при выполнении сложения");
}

return z;
:::::
```

При этом контролируемое выражение может быть произвольной формы и сложности и может содержать другие вхождения как контролируемых, так и неконтролируемых выражений,

- блочную, которая обеспечивает контроль над выполнением операций в блоке операторов:

```
:::::
short x = 32767;
short y = 32767;
short z = 0, w = 0;

try
{
unchecked
{
 w = x+y;
}

checked
{
 z = x+w;
}

}
catch (System.OverflowException e)
{
 Console.Writeline("Переполнение при выполнении сложения");
}

return z;
```

Операция is

Позволяет проверить, совместим ли (относится к данному типу либо к типу, унаследованному от данного) объект с определенным типом. Результат использования логического типа.

```
int i = 10;
if (i is object)
{
    Console.WriteLine(" i is object");
     }
```

Операция as

Применяется для явного преобразования типа ссылочных переменных. Если конвертируемый тип совместим с указанным, преобразование осуществляется успешно. Иначе операция возвращает значение NULL.

```
object o1 = "my str";
object o2 = 5;
string s1 = o1 as string; // s1="my str"
string s2 = o2 as string; // s2= NULL
```

Операция as позволяет выполнить безопасное преобразование типа за один шаг, без необходимости первоначальной проверки его на совместимость с помощью операции is до собственно преобразования.

Особенности выполнения арифметических операций

Особенности выполнения операций над целочисленными операндами и операндами с плавающей точкой связаны с особенностями выполнения арифметических операций и с ограниченной точностью переменных типа float и double.

Представление величин:

float – 7 значащих цифр.

double – 16 значащих цифр.

1000000*100000==1000000000000, но максимально допустимое положительное значение для типа System.Int32 составляет 2147483647. В результате переполнения получается неверный результат –727379968.

Ограниченная точность значений типа System.Single проявляется при присвоении значений переменной типа System.Double. Приводимый ниже простой программный код иллюстрирует некоторые особенности арифметики .NET.

```
using System;

class Class1
{
const double epsilon = 0.00001D;
static void Main(string[] args)
{
int valI = 1000000, resI;
resI = (valI*valI)/valI;

// -727379968/1000000 == -727
Console.WriteLine
("The result of action (1000000*1000000/1000000) is {0}", resI);
```

```
float valF00 = 0.2F, resF;
double valD00 = 0.2D, resD;

// Тест на количество значащих цифр для значений типа double и float.
resD = 12345678901234567890; Console.WriteLine(">>>>> {0:F10}",resD);
resF = (float)resD; Console.WriteLine(">>>>> {0:F10}",resF);
resD = (double)(valF00 + valF00); // 0.400000005960464
if (resD == 0.4D) Console.WriteLine("Yes! {0}",resD);
else Console.WriteLine("No! {0}",resD);

resF = valF00*5.0F;
resD = valD00*5.0D;

resF = (float)valD00*5.0F;
resD = valF00*5.0D; //1.0000000149011612

if (resD == 1.0D) Console.WriteLine("Yes! {0}",resD);
else              Console.WriteLine("No! {0}",resD);

resF = valF00*5.0F;
resD = valF00*5.0F; //1.0000000149011612

if (resD.Equals(1.0D))  Console.WriteLine("Yes! {0}",resD);
else                    Console.WriteLine("No! {0}",resD);

if (Math.Abs(resD - 1.0D) < epsilon)
      Console.WriteLine("Yes! {0:F7}, {1:F7}",resD - 1.0D, epsilon);
else
      Console.WriteLine("No! {0:F7}, {1:F7}",resD - 1.0D, epsilon);
}
}
```

В результате выполнения программы выводится такой результат:

```
The result of action (1000000*1000000/1000000) is -727
>>>>> 12345678901234600000,0000000000
>>>>> 12345680000000000000,0000000000
No! 0,400000005960464
No! 1,00000001490116
No! 1,00000001490116
Yes! 0,0000000, 0,0000100
```

Особенности арифметики с плавающей точкой

- Если переменной типа float присвоить величину x из интервала
- –1.5E–45 < x < 1.5E–45 (x != 0),
- результатом операции окажется положительный (x > 0) или отрицательный (x < 0) нуль (+0, –0),
- Если переменной типа double присвоить величину x из интервала
- –5E–324 < x < 5E–324 (x != 0), результатом операции окажется положительный
- (x > 0) или отрицательный (x < 0) нуль (+0, –0),
- Если переменной типа float присвоить величину x, которая
- –3.4E+38 > x или x < 3.4E+38, результатом операции окажется положительная
- (x > 0) или отрицательная (x < 0) бесконечность (+Infinity, – Infinity),
- Если переменной типа double присвоить величину x, для которой
- –1.7E+308 > x или x < 1.7E+308, результатом операции окажется положительная (x > 0) или отрицательная (x < 0) бесконечность (+Infinity, – Infinity),

- Выполнение операции деления над значениями типами с плавающей точкой (0.0/0.0) даёт NaN (Not a Number).

2.11 Управляющие операторы

Управляющие операторы применяются в рамках методов. Они определяют последовательность выполнения операторов в программе и являются основным средством реализации алгоритмов.

Различаются следующие категории управляющих операторов:

- операторы выбора. Вводятся ключевыми словами if, if … else …, switch,
- итеративные операторы. Вводятся ключевыми словами while, do … while, for, foreach,
- операторы перехода (в рамках методов). Вводятся ключевыми словами goto, break, continue.

if, if … else …

После ключевого слова if располагается взятое в круглые скобки условное выражение (логического типа), следом за которым располагается оператор (блок операторов) произвольной сложности.

Далее в операторе if … else … после ключевого слова else размещается ещё один оператор.

В силу того, что в C# отсутствуют предопределённые алгоритмы преобразования значений к булевскому типу, в условное выражение должно быть выражением типа bool – переменной, константой, либо выражением на основе операций сравнения и логических операций.

```
        if (…)
         {

         }
        else
         {
}
```

Оператор if часто сам в свою очередь является условным оператором произвольной сложности.

```
        if (…) if (…)
         {

         }
        else
         {

         }
        else
         {

         }
```

Первый Else всегда относится к ближайшему if.

```
        if (true) { int XXX = 125; }
        if (true) { int XXX = 125; } else { int ZZZ = 10; }
```

switch

Представляет собой оператор, организующий множественный выбор.

Описание:

```
switch (выражение)
{
case значение1: оператор1; break;
::::::::::
case значениеN: операторN; break;
default: операторN+1; break;
}
```

Каждое значение Case в обязательном порядке ЗАВЕРШАЕТСЯ оператором break.

```
char val;
:::::
switch (val)
{
case 'A': Console.WriteLine(" A");break;
case 'B': Console.WriteLine(" B");break;

default: Console.WriteLine(" unknown");break;
}
```

while

```
while (выражение)
{
операторы
}
```

Правило выполнения этого итеративного опера состоит в следующем: сначала проверяется условие продолжения оператора и в случае, если значение условного выражения равно true, соответствующий оператор (блок операторов) выполняется.

Невозможно построить оператор WHILE на основе одиночного оператора объявления. Оператор

```
while (true) int XXX = 0;
```

С самого первого момента своего существования (ещё до начала трансляции!) сопровождается предупреждением:

```
Embedded statement cannot be a declaration or labeled statement.
```

do … while

```
do
{
операторы
} while (выражение)
```

Разница с ранее рассмотренным оператором цикла состоит в том, что здесь сначала выполняется оператор (блок операторов), а затем проверяется условие продолжения оператора.

for

```
for (Инициализация;УсловиеПродолжения;изменение переменных)
Оператор
```

Инициализация, УсловиеПродолжения, изменение переменных в заголовке оператора цикла for могут быть пустыми. Однако наличие пары символов ';' в заголовке цикла for обязательно.

Список выражений представляет собой разделённую запятыми последовательность выражений.

Следует иметь в виду, что оператор объявления также строится на основе списка выражений (выражений объявления), состоящих из спецификаторов типа, имён и, возможно, инициализаторов. Этот список завершается точкой с запятой, что позволяет рассматривать список выражений инициализации как самостоятельный оператор в составе оператора цикла for. При этом область видимости имён переменных, определяемых этим оператором, распространяется только на операторы, относящиеся к данному оператору цикла. Это значит, что переменные, объявленные в операторе инициализации данного оператора цикла НЕ МОГУТ быть использованы непосредственно после оператора до конца блока, содержащего этот оператор. А следующие друг за другом в рамках общего блока операторы МОГУТ содержать в заголовках одни и те же выражения инициализации.

foreach

```
foreach (тип имя__переменной in коллекция) инструкци
```

имя__переменной обозначает переменную которая представляет элемент коллекции.

коллекция объект, представляющий массив или коллекцию.

Этим оператором обеспечивается повторение множества операторов, составляющих тело цикла, для каждого элемента массива или коллекции. После перебора ВСЕХ элементов массива или коллекции и применения множества операторов для каждого элемента массива или коллекции, управление передаётся следующему за ОператорFOREACH оператору (разумеется, если таковые имеются).

Область видимости имён переменных, определяемых этим оператором, распространяется только на операторы, относящиеся к данному оператору цикла.

```
 // Объявили и определили массив
int[] array = new int[10];
// Для каждого элемента массива надо сделать…
foreach (int i in array) {/*:::::*/};
```

Специализированный оператор, приспособленный для работы с массивами и коллекциями. Обеспечивает повторение множества (единичного оператора или блока операторов) операторов для КАЖДОГО элемента массива или коллекции.

Конструкция экзотическая и негибкая.

Предполагает выполнение примитивных последовательностей действий над массивами и коллекциями (начальная инициализация или просмотр ФИКСИРОВАННОГО количества

элементов). Действия, связанные с изменениями размеров и содержимого коллекций в рамках этого оператора могут привести к непредсказуемым результатам.

goto, break, continue

Объявляется метка (правильнее, оператор с меткой). Оператор может быть пустым. Метка – идентификатор отделяется от оператора двоеточием. В качестве дополнительного разделителя могут быть использованы пробелы, символы табуляции и перехода на новую строку. Метка, как и любое другое имя, подчиняется правилам областей видимости. Она видна в теле метода только в одном направлении: из внутренних (вложенных) блоков. Поэтому оператор перехода

```
goto ИмяПомеченногоОператора;
```

позволяет ВЫХОДИТЬ из блоков, но не входить в них.

Обычно об этом операторе обычно говорится много нехороших слов как о главном разрушителе структурированного программного кода и его описание сопровождается рекомендациями к его НЕИСПОЛЬЗОВАНИЮ.

Операторы

```
break;
```

и

```
continue;
```

используются как вспомогательные средства управления в операторах цикла.

2.12 Массив.

Массив – множество однотипных элементов. Любой массив является производным от класса System.Array.

В отличие от других языков программирования, при объявлении массива в C# нельзя указать его размер, т.к. при объявлении не создается сам массив, а только ссылка на будущий массив. Поэтому после объявления необходима инициализация массива.

Объявление массивов

Одномерные массивы

Объявление одномерного массива выглядит следующим образом:

<тип>[] < имя массива >;

Заметьте, в отличие от языка C++ квадратные скобки приписаны не к имени переменной, а к типу. Они являются неотъемлемой частью определения класса, так что запись T[] следует понимать как класс **одномерный массив с элементами типа** T.

В данном случае речь идет об отложенной инициализации. **При объявлении с отложенной инициализацией сам массив не создается, а создается только ссылка на массив,** имеющая неопределенное значение Null. Поэтому **пока массив не будет реально создан и его элементы инициализированы, использовать его в вычислениях нельзя.**

int[] a, b, c; // пример объявления трех массивов с отложенной инициализацией

Чаще всего, при объявлении массива используется имя с инициализацией. И опять таки, как и в случае простых переменных, могут быть два варианта инициализации:

а) инициализация является явной и задается константным массивом:

double[] x= {5.5, 6.6, 7.7};

Синтаксически, элементы константного массива следует заключать в фигурные скобки.

б) массив создается и инициализируется (выделяется место в памяти с указанным числом элементов массива) массив из 5 элементов типа int:

int[] d= new int[5];

При создании массива можно указывать число элементов, которое хранит переменная, инициализируемая в результате работы программы:

```
string s = Console.ReadLine();
int size = int.Parse(s);

double[] rr = new double[size];
```

Доступ к отдельному элементу массива осуществляется посредством индекса. Индекс описывает позицию элемента внутри массива. Первый элемент имеет нулевой индекс.

Выход за границы массивов в C# расценивается как динамическая ошибка. Будет сгенерирована исключительная ситуация типа IndexOutOfRangeException, и программа прекратит выполнение.

Многомерные массивы

Объявление *многомерного массива* в общем случае:

<тип>[, … ,] <имя массива>;

Число запятых, увеличенное на единицу, и задает размерность массива. Можно лишь отметить, что хотя явная инициализация с использованием многомерных константных массивов возможна, но применяется редко из-за громоздкости такой структуры.

Простейший многомерный массив — **двумерный**. В двумерном массиве позиция любого элемента определяется двумя индексами. Если представить двумерный массив в виде таблицы данных, то один индекс означает строку, а второй — столбец.

Чтобы объявить двумерный массив целочисленных значений размером 10x20 с именем table, достаточно записать следующее:

```
int [ , ]  table  = new int[10, 20];
```

Обратите особое внимание на то, что значения размерностей отделяются запятой. Синтаксис первой части этого объявления означает, что создается ссылочная переменная двумерного массива. Для реального выделения памяти для этого массива с помощью оператора new используется более конкретный синтаксис:

```
int[10, 20]
```

Чтобы получить доступ к элементу двумерного массива, необходимо указать оба индекса, разделив их запятой. Например, чтобы присвоить число 10 элементу массива table, позиция которого определяется координатами 3 и 5, можно использовать следующую инструкцию:

```
table [ 3 , 5] = 10;
```

Пример использования массивов.

Рассмотрим классическую задачу умножения прямоугольных матриц. Нам понадобится три динамических массива для представления матриц и три процедуры, одна из которых будет

заполнять исходные матрицы случайными числами, другая выполнять умножение матриц, третья – печатать сами матрицы. Вот тестовый пример:

```
public void TestMultiMatr()
    {
      int n1, m1, n2, m2,n3, m3;
          Arrs.GetSizes("MatrA",out n1,out m1);
          Arrs.GetSizes("MatrB",out n2,out m2);
          Arrs.GetSizes("MatrC",out n3,out m3);
          int[,]MatrA = new int[n1,m1], MatrB = new int[n2,m2];
          int[,]MatrC = new int[n3,m3];
          Arrs.CreateTwoDimAr(MatrA); Arrs.CreateTwoDimAr(MatrB);
          Arrs.MultMatr(MatrA, MatrB, MatrC);
          Arrs.PrintAr2("MatrA",MatrA); Arrs.PrintAr2("MatrB",MatrB);
          Arrs.PrintAr2("MatrC",MatrC);
    }//TestMultiMatr
```

Три матрицы MatrA, MatrB и MatrC имеют произвольные размеры, выясняемые в диалоге с пользователем, и использование для их описания динамических массивов представляется совершенно естественным. Метод CreateTwoDimAr заполняет случайными числами элементы матрицы, переданной ему в качестве аргумента, метод PrintAr2 – выводит матрицу на печать. Я не буду приводить их код, похожий на код их одномерных аналогов.

Метод MultMatr выполняет умножение прямоугольных матриц. Текст этого метода:

```
public void MultMatr(int[,]A, int[,]B, int[,]C)
    {
      if (A.GetLength(1) != B.GetLength(0))
                Console.WriteLine("MultMatr: ошибка размерности!");
      else
        for(int i = 0; i < A.GetLength(0); i++)
          for(int j = 0; j < B.GetLength(1); j++)
          {
            int s=0;
            for(int k = 0; k < A.GetLength(1); k++)
              s+= A[i,k]*B[k,j];
                        C[i,j] = s;
          }
    }//MultMatr
```

Массивы массивов

Еще одним видом массивов C# являются массивы массивов, называемые также ступенчатыми массивами (jagged arrays). Такой массив массивов можно рассматривать как одномерный массив, элементы которого являются массивами, элементы которых, в свою очередь снова могут быть массивами и так может продолжаться до некоторого уровня вложенности.

Следующий фрагмент программы при объявлении массива jagged выделяет память для его первой размерности, а память для его второй размерности выделяется "вручную".

```
int [][] jagged = new int [ 3 ] [ ] ;
        jagged[0] = new int [ 4 ] ;
        jagged[1] = new int [ 3 ] ;
        jagged[2] = new int [ 5 ] ;
```

После выполнения этого фрагмента кода массив jagged выглядит так:

jagged[0][0]	jagged[0][1]	jagged[0][2]	jagged[0][3]	
jagged[1][0]	jagged[1][1]	jagged[1][2]		
jagged[2][0]	jagged[2][1]	jagged[2][2]	jagged[2][3]	jagged[2][4]

В каких ситуациях может возникать необходимость в таких структурах данных? Такие массивы могут применяться для представления деревьев, у которых узлы могут иметь произвольное число потомков. Это может быть, например, генеалогическое дерево. Вершины первого уровня – Fathers, представляющие отцов, могут задаваться одномерным массивом, так что Fathers[i] – это i-й отец. Вершины второго уровня представляются массивом массивов – Children, так что Children[i] – это массив детей i-го отца, а Children[i][j] – это j-й ребенок i-го отца. Для представления внуков понадобится третий уровень, так что GrandChildren [i][j][k] будет представлять к-го внука j-го ребенка i-го отца.

Есть некоторые особенности в объявлении и инициализации таких массивов. Если при объявлении типа многомерных массивов для указания размерности использовались запятые, то для изрезанных массивов используется более ясная символика – совокупности пар квадратных скобок, например int[][] задает массив, элементы которого одномерные массивы элементов типа int.

Сложнее с созданием самих массивов и их инициализацией. Здесь нельзя вызвать конструктор new int[3][5], поскольку он не задает изрезанный массив. Фактически нужно вызывать конструктор для каждого массива на самом нижнем уровне. В этом и состоит сложность объявления таких массивов. Начну с формального примера:

//массив массивов - формальный пример

```
//объявление и инициализация
int[][] jagger = new int[3][]
        {
           new int[] {5,7,9,11},
           new int[] {2,8},
           new int[] {6,12,4}
        };
```

Массив jagger имеет всего два уровня. Можно считать, что у него три элемента, каждый из которых является массивом. Для каждого такого массива необходимо вызвать конструктор new, чтобы создать внутренний массив. В данном примере элементы внутренних массивов получают значение, будучи явно инициализированы константными массивами.

Встроенный сервис по обслуживанию массивов

Массивам соответствует свой класс **System.Array**, который позволяет с ними работать. Этот класс характеризуется специальным набором свойств и методов для создания, управления, поиска, и сортировки, элементов массива представленных в таблице 4.1.

Для использования свойств операнд принимает вид:

Имя_массива. Свойство;

Для использования методов оператор:

Array. Метод(параметры);

либо **Имя_массива. Метод(параметры);**

Таблица 11. Набор свойств и методов для работы с массивами.

Название	Описание
Свойство Length	Возвращает целое число представляющее общее количество элементов во всех измерениях массива.
Свойство Rank	Возвращает целое число представляющее количество измерений (размерность) массива.
Метод Array CreateInstance()	Статический метод (один из вариантов), создаёт массив элементов заданного типа и определённой размерности.
Метод GetLength ()	Возвращает количество элементов в указной размерности массива.
Метод SetValue()	Присваивает элементу массива значение, представленное первым параметром (один из вариантов).
Метод GetValue()	Извлекает значение из двумерного массива по индексам (один из вариантов).
Метод Sort()	Позволяет сортировать одномерный массив, **массив передается как параметр**
Метод Clear()	Устанавливает элементы массива в 0 для массивов, содержащих значения, null – содержащих ссылки, false – логического типа
Метод Clone()	Для создания копии массива, при этом копируются лишь ссылки, сами объекты копироваться не будут

2.13 Символы и строки.

Обработка текстовой информации является, вероятно, одной из самых распространённых задач в программировании, и C# предоставляет для ее решения широкий набор средств: отдельные символы, массивы символов, изменяемые и неизменяемые строки и регулярные выражения.

Символы

Символьный тип char предназначен для хранения символов в кодировке Unicode. Символьный тип относится к встроенным типам данных C# и соответствует стандартному классу Char библиотеки .NET из пространства имен System. В этом классе определены статические методы, позволяющие задать вид и категорию символа, а также преобразовать символ в верхний или нижний регистр и в число. Основные методы приведены в табл. 12.

Таблица 12. Основные методы класса System.Char.

Название	Описание
GetNumericValue	Возвращает числовое значение символа, если он является цифрой, и -1 в противном случае
GetUnicodeCategory	Возвращает категорию Unicode-символа. Вее Unicode-символы разделены на категории, например, десятичные цифры (Decimal-DigitNumber), римские цифры (LetterNumber), разделители строк (LineSeparator), буквы в нижнем регистре (LowercaseLetter) и т. д.
IsControl	Возвращает true, если символ является управляющим
IsDigit	Возвращает true, если символ является десятичной цифрой
IsLetter	Возвращает true, если символ является буквой
IsLetterOrDigit	Возвращает true, если символ является буквой или цифрой

IsLower	Возвращает true, если символ задан в нижнем регистре
IsNumber	Возвращает true, если символ является числом (десятичным или шестнадцатеричным)
IsPunctuation	Возвращает true, если символ является знаком препинания
IsSeparator	Возвращает true, если символ является разделителем
IsUpper	Возвращает true, если символ записан в верхнем регистре
IsWhiteSpace	Возвращает true, если символ является пробельным (пробел, перевод строки и возврат каретки)
Parse	Преобразует строку в символ (строка должна состоять из одного символа)
ToLower	Преобразует символ в нижний регистр
ToUpper	Преобразует символ в верхний регистр
MaxValue, MinValue	Возвращают символы с максимальным и минимальным кодами (эти символы не имеют видимого представления)

В примере продемонстрировано использование этих методов.

```
using System;
namespace ConsoleApplication
{
  class Class1
  {
        static void Main()
    {
        try
        {
            char b = 'B', c = '\x63', d = '\u0032';                    //1
            Console.WriteLine("{0} {1} {2}", b, c, d);
            Console.WriteLine("{0} {1} {2}",
                 char.ToLower(b), char.ToUpper(c), char.GetNumericValue(d));
            char a;
            do                                                          //2
            {
                Console.Write("Введите символ: ");
                a = char.Parse(Console.ReadLine());
                Console.WriteLine("Введен символ {0} . его код - {1} ",
                          a, (int)a);
                if (char.IsLetter(a)) Console.WriteLine("Буква");
                if (char.IsUpper(a)) Console.WriteLine("Верхий рег.");
                if (char.IsLower(a)) Console.WriteLine("Нижний рег.");
                if (char.IsControl(a)) Console.WriteLine("Управляющий");
                if (char.IsNumber(a)) Console.WriteLine("Число");
                if (char.IsPunctuation(a)) Console.WriteLine("Разделитель");
            } while (a != 'q');
        }
        catch
        {
            Console.WriteLine("Возникло исключение");
            return;
        }
    }
  }
}
```

В операторе 1 описаны три символьных переменных. Они инициализируются символьными литералами в различных формах представления. Далее выполняются вывод и преобразование символов.

В цикле 2 анализируется вводимый с клавиатуры символ. Можно вводить и управляющие символы, используя сочетание клавиши Ctrl с латинскими буквами. При вводе использован метод Parse, преобразующий строку, которая должна содержать единственный символ, в символ типа char. Поскольку вводится строка, ввод каждого символа следует завершать нажатием клавиши Enter. Цикл выполняется, пока пользователь не введет символ q.

Вывод символа сопровождается его кодом в десятичном виде. Для вывода кода используется явное преобразование к целому типу. Явное преобразование из символов в строки и обратно в C# не существует, неявным же образом любой объект, в том числе и символ, может быть преобразован в строку, например:

```
string s = 'к' + 'о' + 'т'; // результат - строка "кот"
```

При вводе и преобразовании могут возникать исключительные ситуации, например, если пользователь введет пустую строку. Для «мягкого» завершения программы предусмотрена обработка исключений.

Массивы символов

Массив символов, как и массив любого иного типа, построен на основе базового класса Array, некоторые свойства и методы которого были перечислены в табл. 11. Применение этих методов позволяет эффективно решать некоторые задачи. Пример использования массива символов:

```
using System;
namespace ConsoleApplication
{
  class Classl
  {
      static void Main()
        {
          char[] a = { 'm', 'a', 's', 's', 'i', 'v' }; // 1
          char[] b = "а роза упала на лапу азора".ToCharArray(); // 2
          PrintArray("Исходный массив a:", a);
          int pos = Array.IndexOf(a, 'm');
          a[pos] = 'M';
          PrintArray("Измененный массив a:", a);
          PrintArray("Исходный массив b:", b);
          Array.Reverse(b);
          PrintArray("Измененный массив b:", b);
        }
      public static void PrintArray(string header, Array a )
        {
          Console.WriteLine(header);
          foreach ( object x in a ) Console.Write(x);
          Console.WriteLine("\n");
        }
   }
}
```

Результат работы программы:

Исходный массив a:

massiv

Измененный массив a:
Massiv
Исходный массив b
а роза упала на лапу азора
Измененный массив b:
ароза упал ан алапу азор а

Символьный массив можно инициализировать, либо непосредственно задавая его элементы (оператор 1), либо применяя метод ToCharArray класса string, который разбивает исходную строку на отдельные символы (оператор 2).

Строки типа String

Тип string, предназначенный для работы со строками символов в кодировке Unicode, является встроенным типом C#. Ему соответствует базовый класс System.String библиотеки .NET.

Создать строку можно несколькими способами:

```
string s; // инициализация отложена
string t = "qqq"; // инициализация строковым литералом
string u = new string(' ',20); // конструктор создает строку из 20
пробелов
char[] a = { 'O' , 'O' , 'O' }; // массив для инициализации строки
string v = new string( a ); // создание из массива символов
```

Для строк определены следующие операции:

- присваивание (=);
- проверка на равенство (==);
- проверка на неравенство (! =) ;
- обращение по индексу ([ ]) ;
- сцепление (конкатенация) строк (+) .

Несмотря на то что строки являются ссылочным типом данных, на равенство и неравенство проверяются не ссылки, а значения строк. Строки равны, если имеют одинаковое количество символов и совпадают посимвольно.

Обращаться к отдельному элементу строки по индексу можно только для получения значения, но не для его изменения. Это связано с тем, что строки типа string относятся к так называемым неизменяемым типам данных Методы, изменяющие содержимое строки, на самом деле создают новую копию строки. Неиспользуемые ≪старые≫ копии автоматически удаляются сборщиком мусора.

В классе System.String предусмотрено множество методов, полей и свойств, позволяющих выполнять со строками практически любые действия. Основные элементы класса приведены в табл. 13.

Таблица 13. Основные элементы класса System.String

Название	Описание
Compare	Сравнение двух строк в лексикографическом (алфавитном) порядке. Разные реализации метода позволяют сравнивать строки и подстроки с учетом и без учета регистра и особенностей национального представления дат и т. д.
CompareOrdinal	Сравнение двух строк по кодам символов. Разные реализации метода позволяют сравнивать строки и подстроки
CompareTo	Сравнение текущего экземпляра строки с другой строкой
Concat	Конкатенация строк. Метод допускает сцепление произвольного числа строк
Copy	Создание копии строки

Empty	Пустая строка (только для чтения)
Format	Форматирование в соответствии с заданными спецификаторами формата
IndexOf, IndexOfAny, LastlndexOf, LastIndexOfAny	Определение индексов первого и последнего вхождения заданной подстроки или любого символа из заданного набора
Insert	Вставка подстроки в заданную позицию
Intern, Islnterned	Возвращает ссылку на строку, если такая уже существует. Если строки нет, Intern добавляет строку во внутренний пул, IsInterned возвращает null
Join	Слияние массива строк в единую строку. Между элементами массива вставляются разделители
Length	Длина строки (количество символов)
PadLeft, PadRight	Выравнивание строки по левому или правому краю путем вставки нужного числа пробелов в начале или в конце строки
Remove	Удаление подстроки из заданной позиции
Replace	Замена всех вхождений заданной подстроки или символа новыми подстрокой или символом
Split	Разделяет строку на элементы, используя заданные разделители. Результаты помещаются в массив строк
StartsWith, EndsWith	Возвращает true или false в зависимости оттого, начинается или заканчивается строка заданной подстрокой
Substring	Выделение подстроки, начиная с заданной позиции
ToCharArray	Преобразование строки в массив символов
ToLower, ToUlpper	Преобразование символов строки к нижнему или верхнему регистру
Tim, TrimStart, TrimEnd	Удаление пробелов в начале и конце строки или только с одного ее конца (обратные по отношению к методам PadLeft и PadRight действия)

Пример применения методов для работы со строкой типа string:

```
using System;
namespace ConsoleApplication
{
  class Classl
  {
      static void Main()
        {
            string s = "прекрасная королева Изольда";
          Console.WriteLine(s);
          string sub = s.Substring(3).Remove(12, 2); //1
          Console.WriteLine(sub);
          string [ ] mas = s . Split(' ');          //2
          string joined = string.Join("! ", mas);
          Console.WriteLine(joined);
          Console.WriteLine("Введите строку");
          string x = Console.ReadLine();           // 3
          Console.WriteLine("Вы ввели строку " + x);
          double a = 12.234;
          int b = 29;
```

```
            Console.WriteLine("a = {0,6:C} b = {1,2:X}", a, b); //4
            Console.WriteLine("a = {0,6:0.##} b={1,5:0.#' руб.'}", a, b );//5
            Console.WriteLine("a = {0:F3} b={1:D3}", a, b );//6
          }
    }
}
```

Результат работы программы:
прекрасная королева Изольда
красная корова Изольда
прекрасная! королева! Изольда
Введите строку
не хочу!
Вы ввели строку не хочу!
a = 12,23р. b = 1D
a = 12,23 b = 29 руб.
a = 12,234 b = 029

В операторе 1 выполняются два последовательных вызова методов: метод Substring возвращает подстроку строки s, которая содержит символы исходной строки, начиная с четвертого символа. Для этой подстроки вызывается метод Remove, удаляющий из нее два символа, начиная с 12-го. Результат работы метода присваивается переменной sub.

Аргументом метода Split (оператор 2) является разделитель, в данном случае - символ пробела. Метод разделяет строку на отдельные слова, которые заносятся в массив строк mas. Статический метод Join (он вызывается через имя класса) объединяет элементы массива mas в одну строку, вставляя между каждой парой слов строку "! ".

В операторе 5 используются так называемые пользовательские шаблоны форматирования. Если приглядеться, в них нет ничего сложного: после двоеточия задается вид выводимого значения посимвольно, причем на месте каждого символа может стоять либо #, либо 0. Если указан знак #, на этом месте будет выведена цифра числа, если она не равна нулю. Если указан 0, будет выведена любая цифра, в том числе и 0.

Пользовательский шаблон может также содержать текст, который в общем случае заключается в апострофы.

3 Контрольные вопросы

1. Что понимается под термином «.NET Framework»?
2. Зависят ли приложения, разрабатываемые в .NET, от платформы?
3. Возможно ли создание гетерогенных приложений в среде .NET?
4. Что означает аббревиатура «CLR»?
5. Является ли среда CLR многоязычной?
6. Приведите обобщенный синтаксис объявления переменной на языке C#.
7. Приведите обобщенный синтаксис инициализации переменной на языке C#.
8. Какая дисциплина (вариант контроля) типов принята в языке C#?
9. Каковы основные категории типов в языке C#? Перечислите пять простых типов языка C#.
10. Что понимается под областью видимости переменной в языке C#?
11. Как обозначается область видимости переменной в языке C#?
12. Как соотносится время жизни переменной и область видимости?
13. Приведите синтаксис условного оператора в общем виде. Проиллюстрируйте его фрагментом программы на языке C#.

14. Приведите синтаксис оператора выбора в общем виде. Проиллюстрируйте его фрагментом программы на языке C#.
15. Что понимается под термином «пространство имен»?
16. В чем состоит назначение пространств имен в языке C#?
17. Благодаря какому механизму удается избежать коллизий имен в языке C#?
18. Какое пространство имен использует системная библиотека .NET Framework?
19. Какое пространство имен использует системная библиотека C#?
20. В чем состоит назначение директивы using?
21. Какой символ используется для указания полного имени объекта в языке C#?
22. Приведите синтаксис директивы using в общем виде. Проиллюстрируйте его фрагментом программы на языке C#.
23. Приведите синтаксис описания пространства имен в общем виде. Проиллюстрируйте его фрагментом программы на языке C#.
24. Опишите структуру программы на языке C#.
25. Какие группы символов входят в алфавит языка C#.
26. Что такое управляющие последовательности, и каким образом они задаются?
27. Как задаются идентификаторы?
28. Перечислите ключевые слова языка C#.
29. Как определить константу?
30. Опишите возможности ввода-вывода данных.
31. Опишите известные вам манипуляторы ввода-вывода.
32. Опишите оператор выбора case.
33. Опишите условный оператор if.
34. Какие операции отношения вы знаете.
35. Каков приоритет логических выражений.
36. Какие виды операторов цикла существуют.
37. Опишите оператор цикла с предусловием.
38. Опишите оператор цикла с постусловием.
39. Опишите оператор цикла с параметром.
40. Перечислите операторы передачи управления.
41. Какое назначение оператора break.
42. Какое назначение оператора continue.
43. Какое назначение оператора return.
44. Какое назначение оператора goto.
45. Как определить массив?
46. Как проинициализировать массив?
47. Какие варианты объявления с инициализацией вы знаете?
48. Как обратиться к элементу массива?
49. Как объявить многомерный массив?
50. Как проинициализировать многомерный массив?
51. Какие виды строк существуют в C#?
52. Как объявить C-строку?
53. Как осуществляется ввод-вывод строк?
54. Какие операции над строками вы знаете?
55. Перечислите операции над символами?

4 Задание

1. Создать новый проект.
2. Написать программы в соответствии с вариантом задания из пункта 5.1 и 5.2.
3. Отладить и протестировать программу.

4. Вариант определяется согласно номеру студента в списке группы.

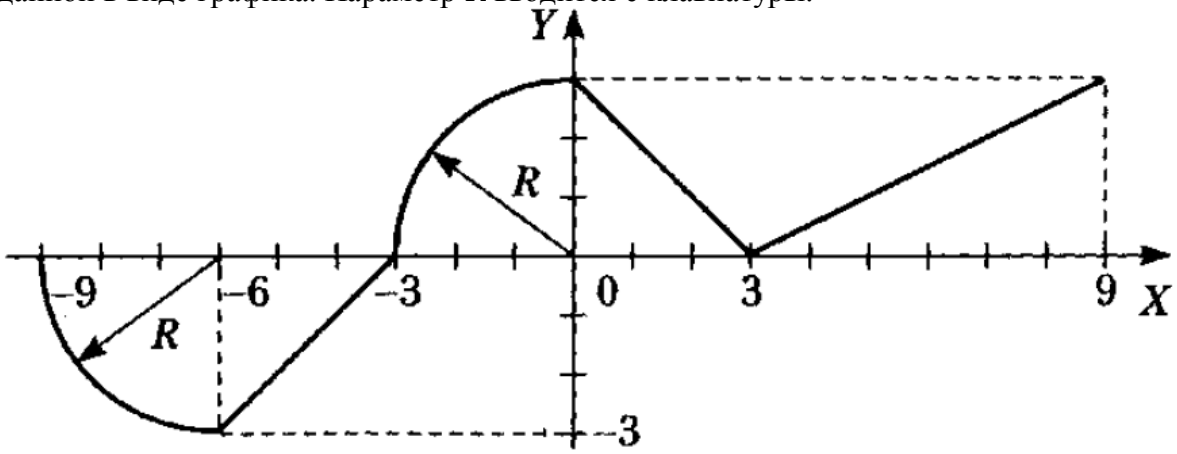
5 Варианты заданий

5.1 Управляющие операторы и операторы цикла

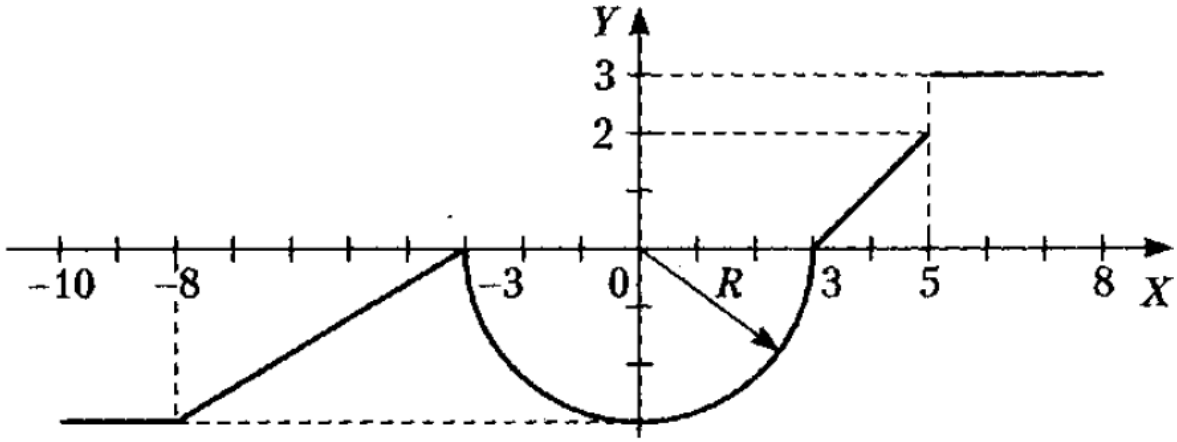
Управляющие операторы

Написать программу, которая по введенному значению аргумента вычисляет значение функции, заданной в виде графика. Параметр R вводится с клавиатуры.

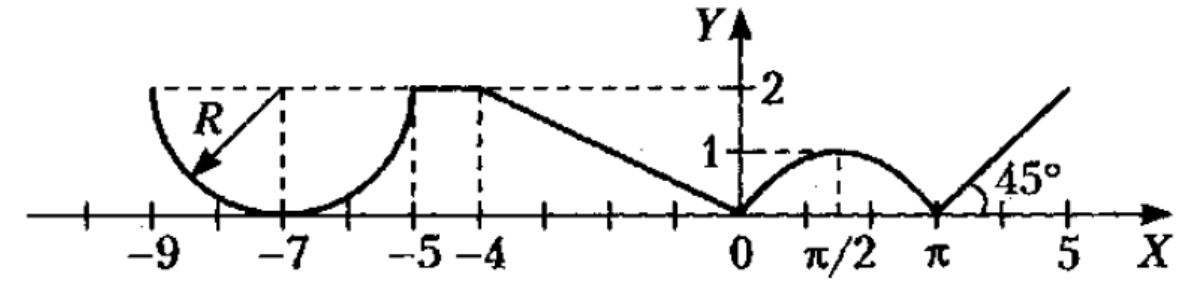
1



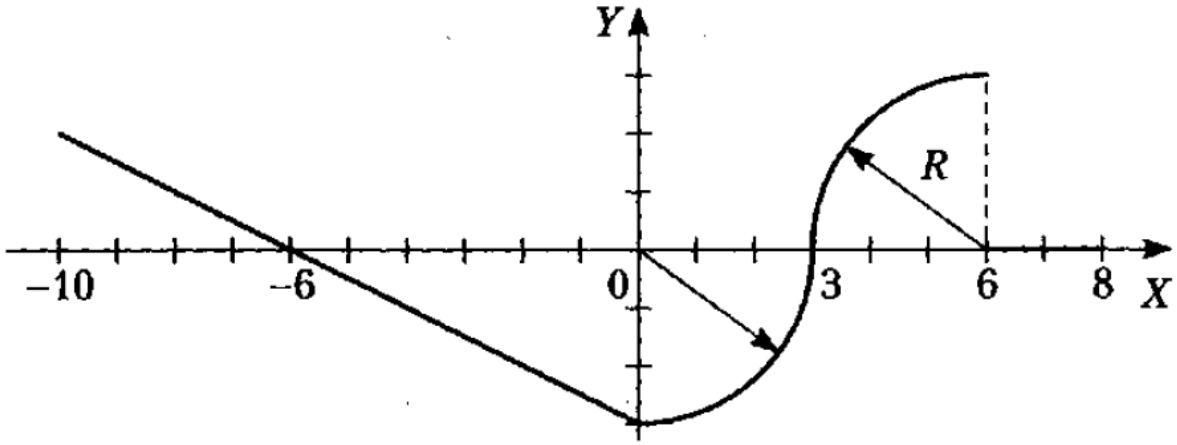
2



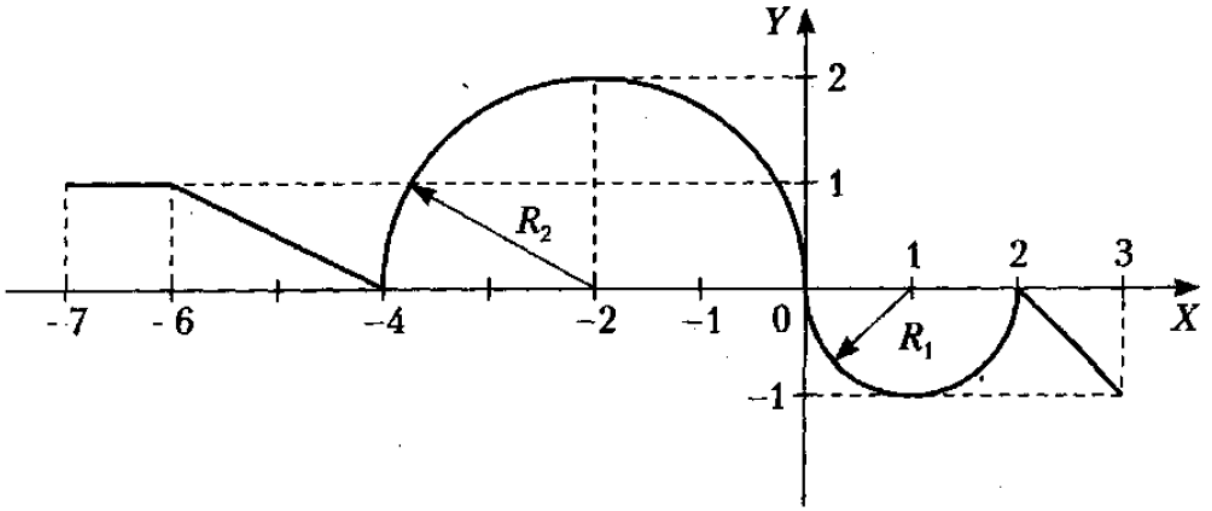
3



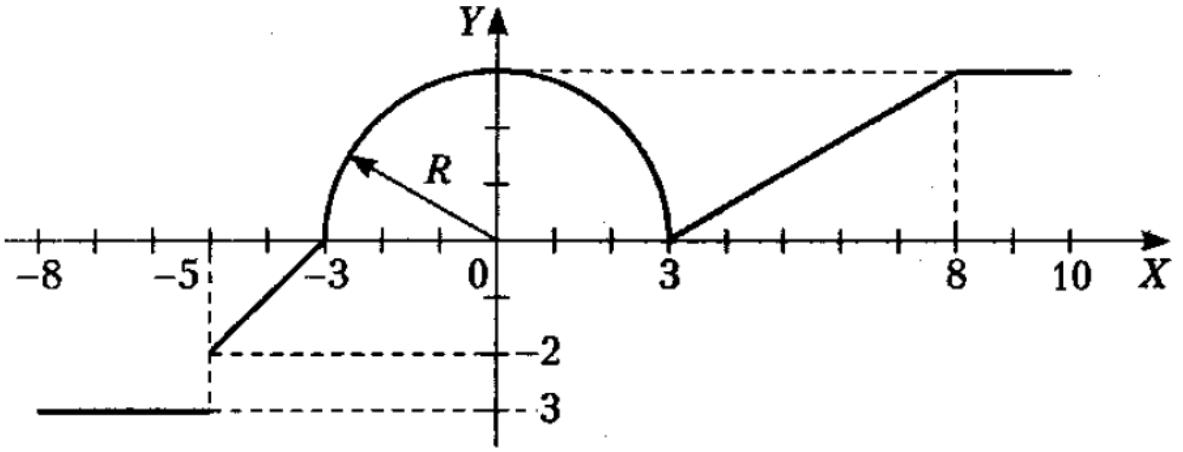
4



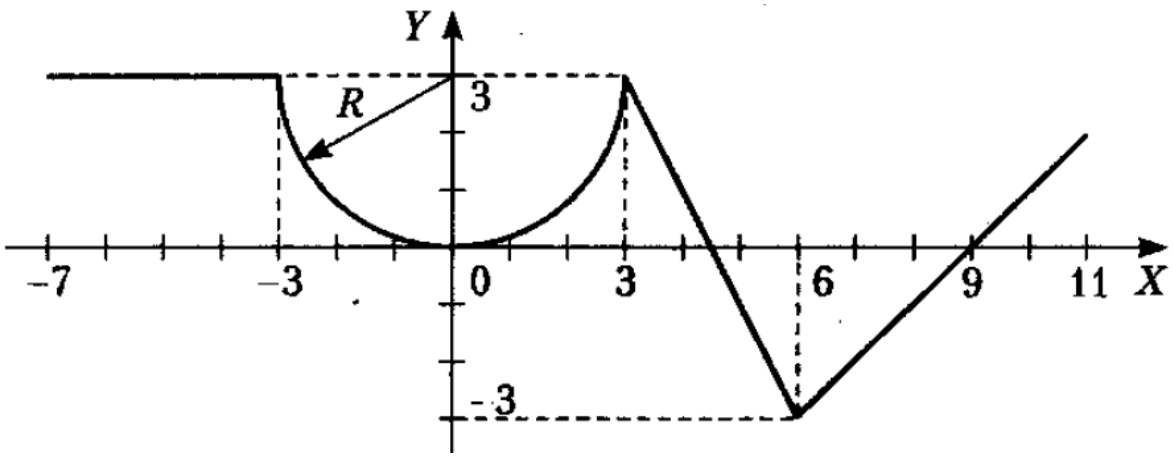
5



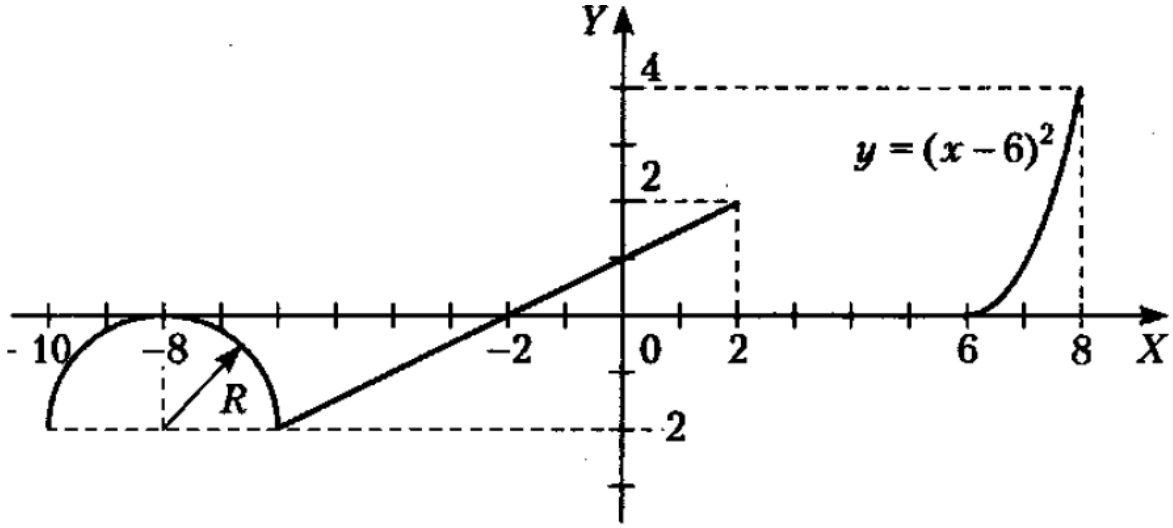
6



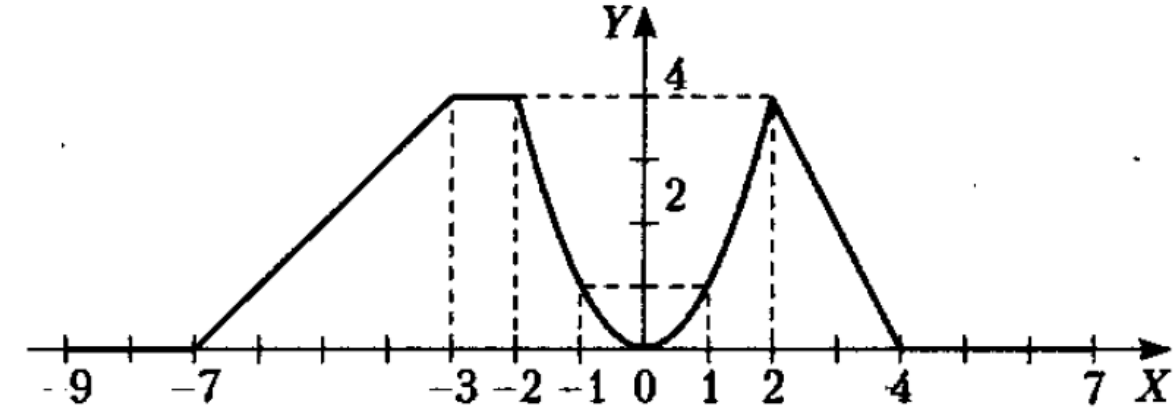
7


Y
R
3
-3
-7
-3
0
3
6
9
11
X

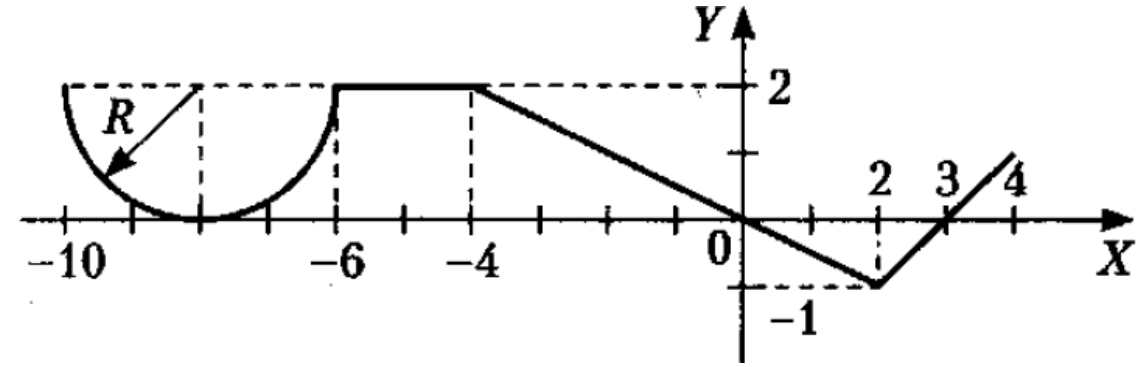
8


Y
4
2
-2
$y = (x-6)^2$
-10
-8
R
-2
0
2
6
8
X

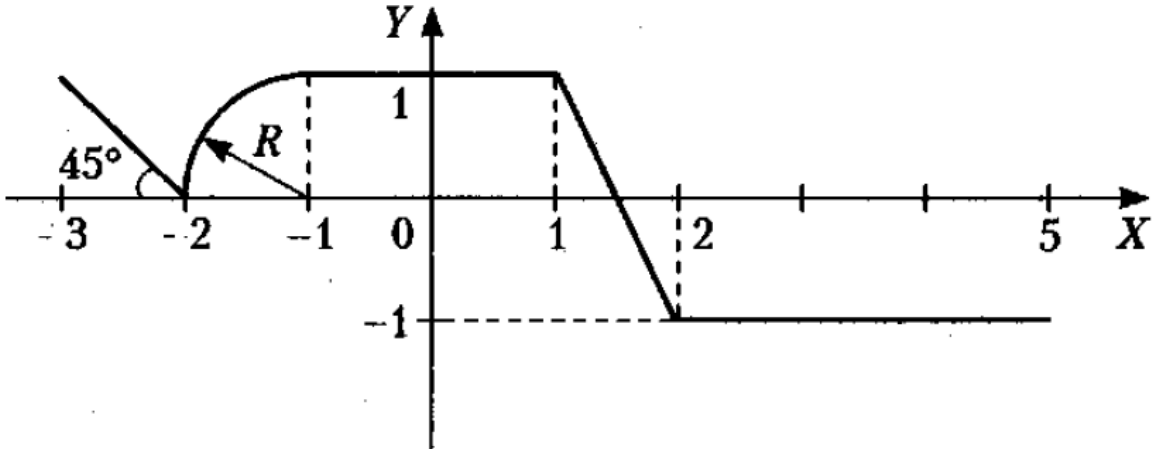
9


Y
4
2
-9
-7
-3
-2
-1
0
1
2
4
7
X

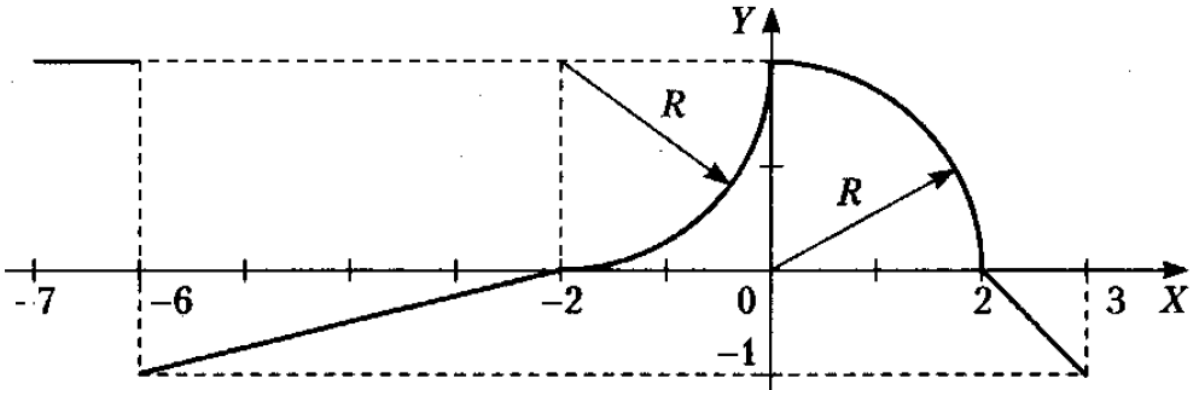
10


Y
R
2
-1
-10
-6
-4
0
2
3
4
X

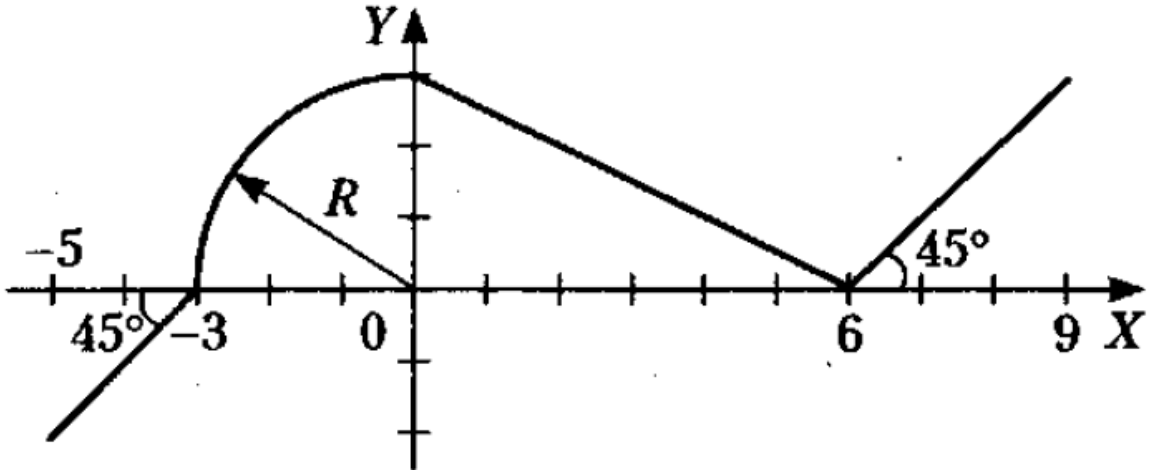
11


Y
1
R
45°
-3
-2
-1
0
1
2
5
X
-1

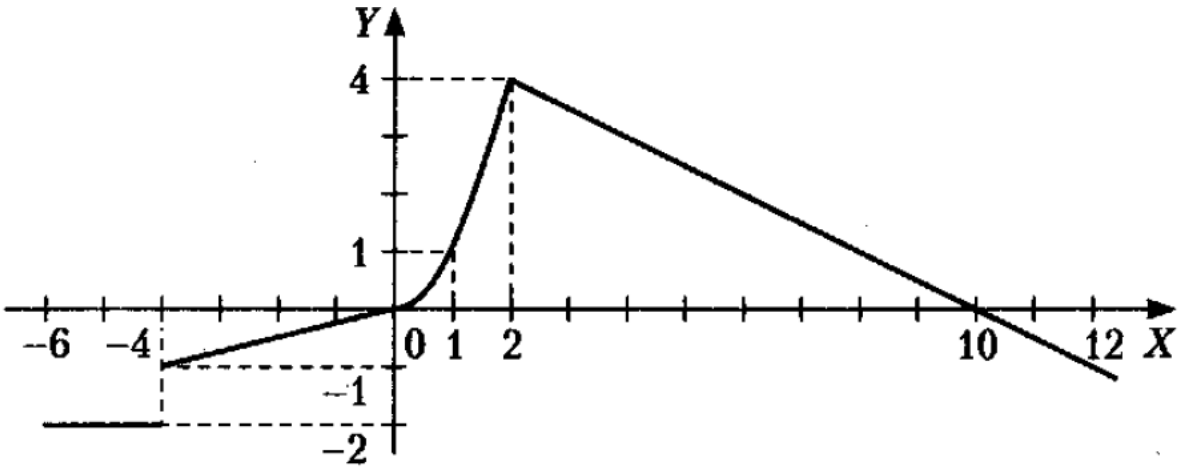
12


Y
R
R
-7
-6
-2
0
2
3
X
-1

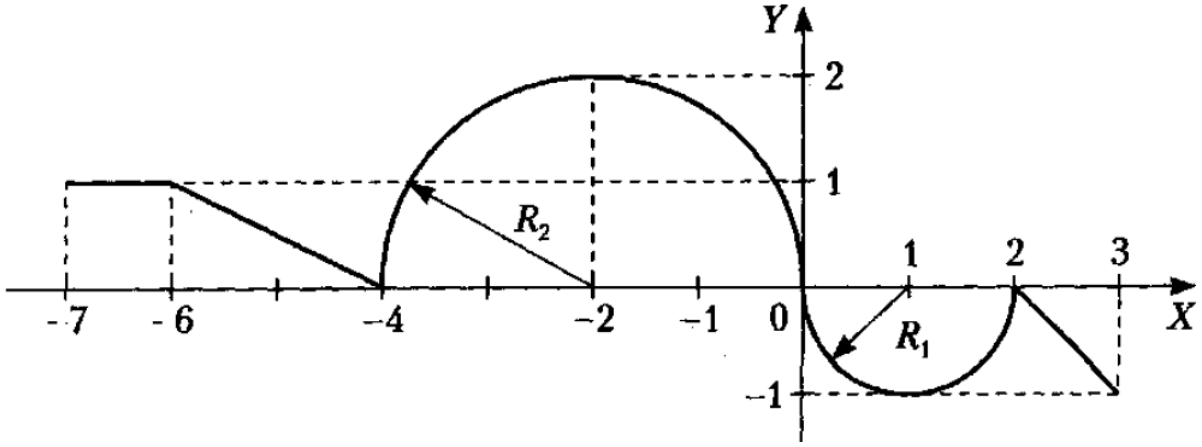
13


Y
R
-5
45°
45°
-3
0
6
9
X

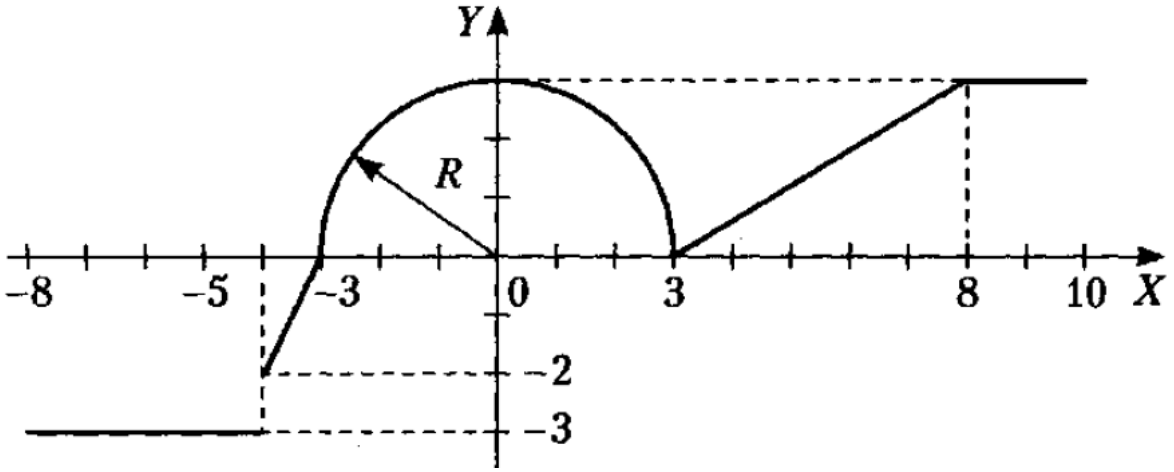
14


Y
4
1
-6
-4
0
1
2
10
12
X
-1
-2

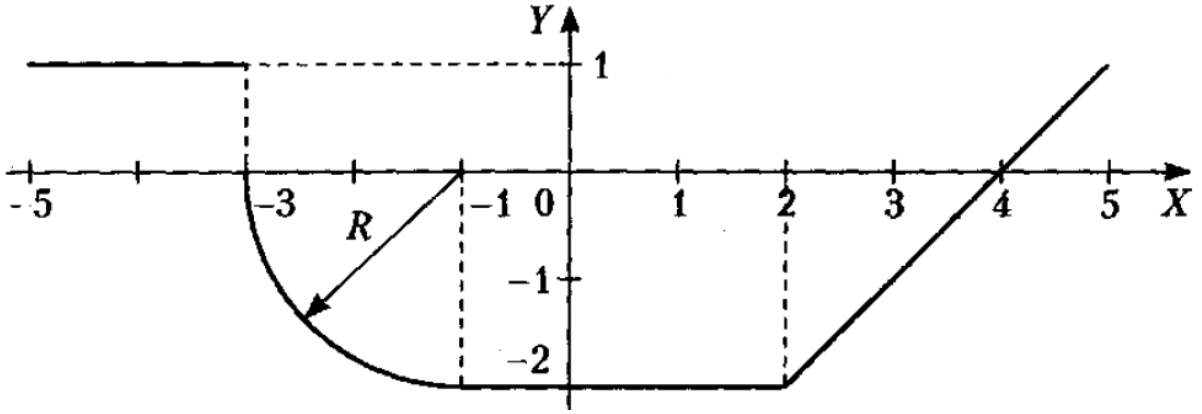
15


Y
2
1
R_2
1
2
3
-7
-6
-4
-2
-1
0
R_1
X
-1

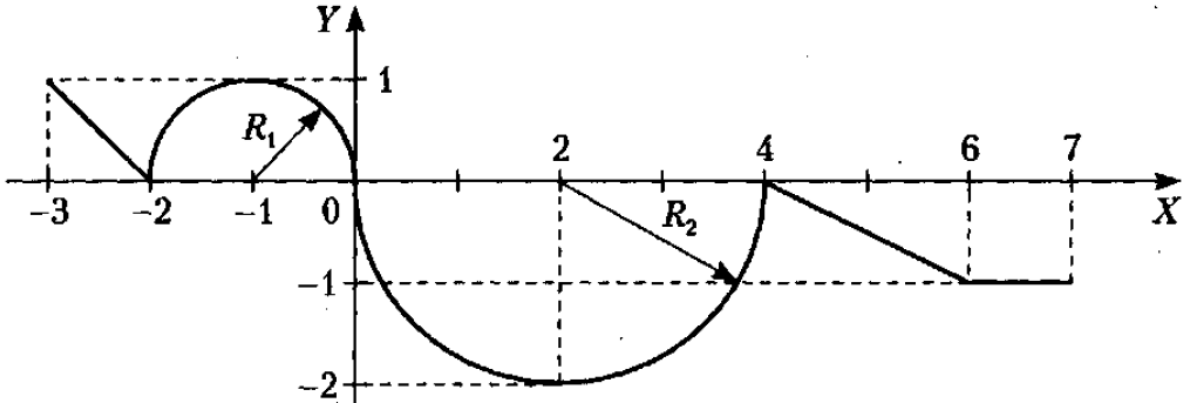
16


Y
R
-8
-5
-3
0
3
8
10
X
-2
-3

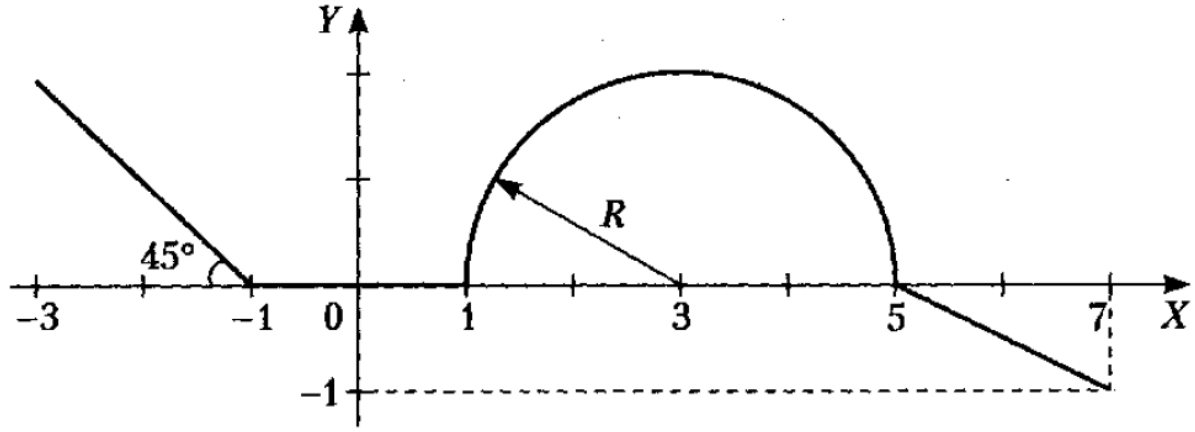
17


Y
1
-5
-3
R
-1
0
1
2
3
4
5
X
-1
-2

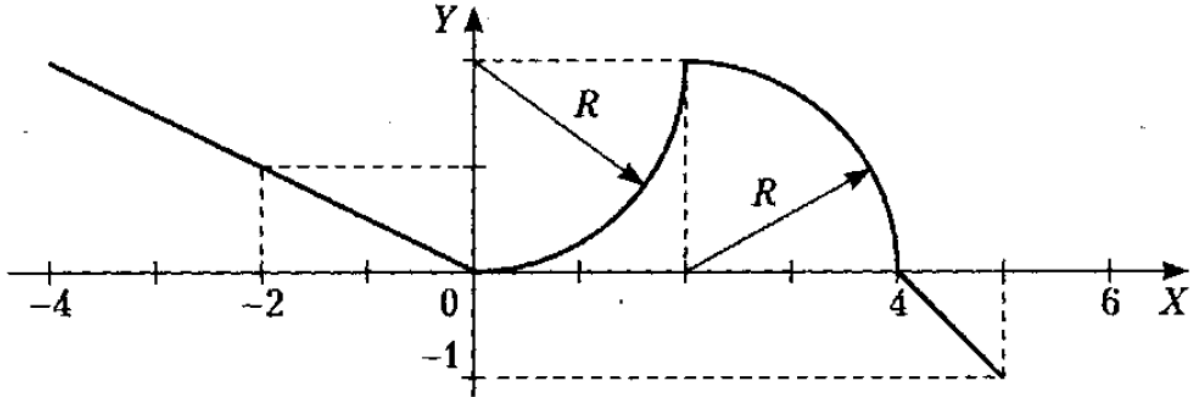
18


Y
1
R_1
2
4
6
7
-3
-2
-1
0
R_2
X
-1
-2

19



20



Операторы цикла:

1. Имеется серия измерений элементов треугольника. Группы элементов пронумерованы. В серии в произвольном порядке могут встречаться такие группы элементов треугольника:
   a. основание и высота;
   b. две стороны и угол между ними (угол задан в радианах);
   c. три стороны.

   Разработать программу, которая запрашивает номер группы элементов, вводит соответствующие элементы и вычисляет площадь треугольника. Вычисления прекратить, если в качестве номера группы введен 0.
2. Начав тренировки, спортсмен в первый день пробежал 10 км. Каждый день он увеличивал дневную норму на 10% нормы предыдущего дня. Какой суммарный путь пробежит спортсмен за 7 дней?
3. Одноклеточная амеба каждые 3 часа делится на 2 клетки. Определить, сколько амеб будет через 3, 6, 9, 12,…, 24 часа.
4. Около стены наклонно стоит палка длиной x м. Один ее конец находится на расстоянии y м от стены. Определить значение угла a между палкой и полом для значений x = k м и y, изменяющегося от 2 до 3 м с шагом h м.
5. У гусей и кроликов вместе 64 лапы. Сколько может быть кроликов и гусей (указать все сочетания)?
6. Составить алгоритм решения задачи: сколько можно купить быков, коров и телят, платя за быка 10 руб., за корову — 5 руб., а за теленка — 0,5 руб., если на 100 руб. надо купить 100 голов скота?
7. Составить программу для проверки утверждения: «Результатами вычислений по формуле $x^2 + x + 17$ при $0 \geq x \geq 15$ являются простые числа». Все результаты вывести

на экран.

8. Покупатель должен заплатить в кассу 5253 руб. У него имеются купюры по 1, 5, 10, 50, 100, 500 и 1000 руб. Сколько купюр разного достоинства отдаст покупатель, если он начинает платить с самых крупных купюр?
9. Ежемесячная стипендия студента составляет A руб., а расходы на проживание превышают стипендию и составляют B руб. в месяц. Рост цен ежемесячно увеличивает расходы на 3%. Составьте программу расчета суммы денег, которую необходимо единовременно попросить у родителей, чтобы можно было прожить учебный год (10 месяцев), используя только эти деньги и стипендию.
10. Составить программу, которая печатает таблицу умножения и сложения натуральных чисел в двоичной системе счисления.
11. Составить программу, которая печатает таблицу умножения и сложения натуральных чисел в четырехричной системе счисления.
12. Составить программу, которая печатает таблицу умножения и сложения натуральных чисел в восьмеричной системе счисления.
13. Составить программу, которая печатает таблицу умножения и сложения натуральных чисел в шестнадцатеричной системе счисления.
14. Найти сумму всех n-значных чисел, кратных k ($1 \le n \le 4$).
15. Показать, что для всех n = 1, 2, 3, …, N
    (15 + 25 + ... + n5) + (17 + 27 + ... + n7) = 2(1 + 2 + ... + n)4.
16. Заменить буквы цифрами так, чтобы соотношение оказалось верным (одинаковым буквам соответствуют одинаковые цифры, разным — разные):
    ХРУСТ · ГРОХОТ = РРРРРРРРРРР.
17. Составить программу, которая находит наибольшее значение отношения трехзначного числа к сумме его цифр.
18. Вычислить сумму кодов всех символов, которые в цикле вводятся с клавиатуры до нажатия на клавишу Esc.
19. Вычислить количество точек с целочисленными координатами, находящихся в круге радиуса R (R > 0).
20. Напечатать в возрастающем порядке все трехзначные числа, в десятичной записи которых нет одинаковых цифр (операции деления и нахождения остатка от деления не использовать).

Ряды

1. Дано натуральное число *N*. Вычислить

$$S = 1 - \frac{1}{2} + \frac{1}{4} - \frac{1}{8} + \cdots + (-1)^n \cdot \frac{1}{2^n}.$$

2. Дано натуральное число *N*. Вычислить:

$$S = \frac{1}{\sin 1} + \frac{1}{\sin 1 + \sin 2} + \cdots + \frac{1}{\sin 1 + \sin 2 + \cdots + \sin N}.$$

3. Дано натуральное число *N*. Вычислить произведение первых *N* сомножителей

$$P = \frac{2}{3} \cdot \frac{4}{5} \cdot \frac{6}{7} \cdots \frac{2N}{2N+1}.$$

4. Дано натуральное число *N*. Вычислить

$$\frac{\cos 1}{\sin 1} \cdot \frac{\cos 1 + \cos 2}{\sin 1 + \sin 2} \cdots \frac{\cos 1 + \cos 2 + \ldots + \cos N}{\sin 1 + \sin 2 + \ldots + \sin N}.$$

5. Дано действительное число *x*. Вычислить

$$x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \frac{x^{11}}{11!} + \frac{x^{13}}{13!}.$$

6. Даны натуральное число n и действительное число x. Вычислить

$$S = \sin x + \sin\sin x + \ldots + \underbrace{\sin\sin\ldots\sin x}_{n \text{ раз}}.$$

7. Даны действительное число a и натуральное число n. Вычислить

$$P = a(a+1)\ldots(a+n-1).$$

8. Даны действительное число a и натуральное число n. Вычислить

$$P = a(a-n)(a-2n)\ldots(a-n^2).$$

9. Даны действительное число a и натуральное число n. Вычислить

$$S = \frac{1}{a} + \frac{1}{a^2} + \frac{1}{a^4} + \ldots + \frac{1}{a^{2n-2}}.$$

10. Дано действительное число x. Вычислить

$$\frac{(x-1)(x-3)(x-7)\ldots(x-63)}{(x-2)(x-4)(x-8)\ldots(x-64)}.$$

11. Вычислить

$$(1+\sin 0,1)(1+\sin 0,2)\ldots(1+\sin 10).$$

12. Даны натуральное число n и действительное число x. Вычислить

$$\sin x + \sin x^2 + \ldots + \sin x^n.$$

13. Дано натуральное число n. Вычислить

$$S = 1\cdot 2 + 2\cdot 3\cdot 4 + \ldots + n\cdot(n+1)\ldots 2n.$$

14. Дано натуральное число n. Вычислить

$$P = \left(1-\frac{1}{2^2}\right)\left(1-\frac{1}{3^2}\right)\ldots\left(1-\frac{1}{n^2}\right), \text{ где } n>2.$$

15. Дано натуральное число n. Вычислить

$$P = \left(1-\frac{1}{2}\right)\left(1-\frac{1}{4}\right)\left(1-\frac{1}{6}\right)\ldots\left(1-\frac{1}{2n}\right).$$

16. Дано натуральное число n. Вычислить

$$S = 1! + 2! + 3! + \ldots + n! \ (n>1).$$

17. Дано натуральное число n. Вычислить

$$S = \frac{1}{3^2} + \frac{1}{5^2} + \frac{1}{7^2} + \ldots + \frac{1}{(2n+1)^2}.$$

18. Для данного действительного числа x вычислить по схеме Горнера

$$y = x^{10} + 2x^9 + 3x^8 + \ldots + 10x + 11.$$

19. Числа Фибоначчи (f_n) определяются формулами

$$f_0 = f_1 = 1, \ f_n = f_{n-1} + f_{n-2} \text{ при } n = 2, 3, \ldots$$

Определить f_{40}.

20. Даны натуральные числа n и k. Вычислить

$$\sqrt{k+\sqrt{2k+\ldots+\sqrt{k(n-1)+\sqrt{kn}}}}.$$

5.2 Массивы и строки

Одномерные массивы:

1. В одномерном массиве, состоящем из n вещественных элементов, вычислить:
   • сумму отрицательных элементов массива;
   • произведение элементов массива, расположенных между максимальным и минимальным элементами.
   Упорядочить элементы массива по возрастанию.
2. В одномерном массиве, состоящем из n вещественных элементов, вычислить:
   • сумму положительных элементов массива;
   • произведение элементов массива, расположенных между максимальным по модулю и минимальным по модулю элементами.
   Упорядочить элементы массива по убыванию.
3. В одномерном массиве, состоящем из n целочисленных элементов, вычислить:
   • произведение элементов массива с четными номерами;
   • сумму элементов массива, расположенных между первым и последним нулевыми элементами.
   Преобразовать массив таким образом, чтобы сначала располагались все положительные элементы, а потом — все отрицательные (элементы, равные нулю, считать положительными).
4. В одномерном массиве, состоящем из n вещественных элементов, вычислить:
   • сумму элементов массива с нечетными номерами;
   • сумму элементов массива, расположенных между первым и последним отрицательными элементами.
   Сжать массив, удалив из него все элементы, модуль которых не превышает единицу. Освободившиеся в конце массива элементы заполнить нулями.
5. В одномерном массиве, состоящем из n вещественных элементов, вычислить:
   • максимальный элемент массива;
   • сумму элементов массива, расположенных до последнего положительного элемента.
   Сжать массив, удалив из него все элементы, модуль которых находится в интервале [a, b]. Освободившиеся в конце массива элементы заполнить нулями.
6. В одномерном массиве, состоящем из n вещественных элементов, вычислить:
   • минимальный элемент массива;
   • сумму элементов массива, расположенных между первым и последним положительными элементами.
   Преобразовать массив таким образом, чтобы сначала располагались все элементы, равные нулю, а потом — все остальные.
7. В одномерном массиве, состоящем из n целочисленных элементов, вычислить:
   • номер максимального элемента массива;
   • произведение элементов массива, расположенных между первым и вторым нулевыми элементами.
   Преобразовать массив таким образом, чтобы в первой его половине располагались элементы, стоявшие в нечетных позициях, а во второй половине — элементы, стоявшие в четных позициях.
8. В одномерном массиве, состоящем из n вещественных элементов, вычислить:
   • номер минимального элемента массива;
   • сумму элементов массива, расположенных между первым и вторым отрицательными элементами.
   Преобразовать массив таким образом, чтобы сначала располагались все элементы, модуль которых не превышает единицу, а потом — все остальные.

9. В одномерном массиве, состоящем из n вещественных элементов, вычислить:
   • максимальный по модулю элемент массива;
   • сумму элементов массива, расположенных между первым и вторым положительными элементами.
   Преобразовать массив таким образом, чтобы элементы, равные нулю, располагались после всех остальных.
10. В одномерном массиве, состоящем из n целочисленных элементов, вычислить:
   • минимальный по модулю элемент массива;
   • сумму модулей элементов массива, расположенных после первого элемента, равного нулю.
   Преобразовать массив таким образом, чтобы в первой его половине располагались элементы, стоявшие в четных позициях, а во второй половине — элементы, стоявшие в нечетных позициях.
11. В одномерном массиве, состоящем из n вещественных элементов, вычислить:
   • номер минимального по модулю элемента массива;
   • сумму модулей элементов массива, расположенных после первого отрицательного элемента.
   Сжать массив, удалив из него все элементы, величина которых находится в интервале [a, b]. Освободившиеся в конце массива элементы заполнить нулями.
12. В одномерном массиве, состоящем из n вещественных элементов, вычислить:
   • номер максимального по модулю элемента массива;
   • сумму элементов массива, расположенных после первого положительного элемента.
   Преобразовать массив таким образом, чтобы сначала располагались все элементы, целая часть которых лежит в интервале [a, b], а потом — все остальные.
13. В одномерном массиве, состоящем из n вещественных элементов, вычислить:
   • количество элементов массива, лежащих в диапазоне от A до B;
   • сумму элементов массива, расположенных после максимального элемента.
   Упорядочить элементы массива по убыванию модулей.
14. В одномерном массиве, состоящем из n вещественных элементов, вычислить:
   • количество элементов массива, равных нулю;
   • сумму элементов массива, расположенных после минимального элемента.
   Упорядочить элементы массива по возрастанию модулей.
15. В одномерном массиве, состоящем из n вещественных элементов, вычислить:
   • количество элементов массива, больших C;
   • произведение элементов массива, расположенных после максимального по модулю элемента.
   Преобразовать массив таким образом, чтобы сначала располагались все отрицательные элементы, а потом — все положительные (элементы, равные нулю, считать положительными).
16. В одномерном массиве, состоящем из n вещественных элементов, вычислить:
   • количество отрицательных элементов массива;
   • сумму модулей элементов массива, расположенных после минимального по модулю элемента.
   Заменить все отрицательные элементы массива их квадратами и упорядочить элементы массива по возрастанию.
17. В одномерном массиве, состоящем из n целочисленных элементов, вычислить:
   • количество положительных элементов массива;
   • сумму элементов массива, расположенных после последнего элемента, равного нулю.

Преобразовать массив таким образом, чтобы сначала располагались все элементы, целая часть которых не превышает единицу, а потом — все остальные.

18. В одномерном массиве, состоящем из и вещественных элементов, вычислить:
    • количество элементов массива, меньших С;
    • сумму целых частей элементов массива, расположенных после последнего отрицательного элемента.
    Преобразовать массив таким образом, чтобы сначала располагались все элементы, отличающиеся от максимального не более чем на 20%, а потом — все остальные.
19. В одномерном массиве, состоящем из п вещественных элементов, вычислить:
    • произведение отрицательных элементов массива;
    • сумму положительных элементов массива, расположенных до максимального элемента.
    Изменить порядок следования элементов в массиве на обратный.
20. В одномерном массиве, состоящем из п вещественных элементов, вычислить:
    • произведение положительных элементов массива;
    • сумму элементов массива, расположенных до минимального элемента.
    Упорядочить по возрастанию отдельно элементы, стоящие на четных местах, и элементы, стоящие на нечетных местах.

Многомерные массивы:

1. Дана целочисленная прямоугольная матрица. Определить:
    • количество строк, не содержащих ни одного нулевого элемента;
    • максимальное из чисел, встречающихся в заданной матрице более одного раза.
2. Дана целочисленная прямоугольная матрица. Определить количество столбцов, не содержащих ни одного нулевого элемента.
    Характеристикой строки целочисленной матрицы назовем сумму ее положительных четных элементов. Переставляя строки заданной матрицы, расположить их в соответствии с ростом характеристик.
3. Дана целочисленная прямоугольная матрица. Определить:
    • количество столбцов, содержащих хотя бы один нулевой элемент;
    • номер строки, в которой находится самая длинная серия одинаковых элементов.
4. Дана целочисленная квадратная матрица. Определить:
    • произведение элементов в тех строках, которые не содержат отрицательных элементов;
    • максимум среди сумм элементов диагоналей, параллельных главной диагонали матрицы.
5. Дана целочисленная квадратная матрица. Определить:
    • сумму элементов в тех столбцах, которые не содержат отрицательных элементов;
    • минимум среди сумм модулей элементов диагоналей, параллельных побочной диагонали матрицы.
6. Дана целочисленная прямоугольная матрица. Определить:
    • сумму элементов в тех строках, которые содержат хотя бы один отрицательный элемент;
    • номера строк и столбцов всех седловых точек матрицы.

Матрица A имеет седловую точку A_{ij}, если A_{ij} является минимальным элементом в i-й строке и максимальным — в j-м столбце.

7. Для заданной матрицы размером 8x8 найти такие k, при которых k-я строка матрицы совпадает с k-м столбцом.
   Найти сумму элементов в тех строках, которые содержат хотя бы один отрицательный элемент.
8. Характеристикой столбца целочисленной матрицы назовем сумму модулей его отрицательных нечетных элементов. Переставляя столбцы заданной матрицы, расположить их в соответствии с ростом характеристик.
   Найти сумму элементов в тех столбцах, которые содержат хотя бы один отрицательный элемент.
9. Соседями элемента A_{ij} в матрице назовем элементы A_{kl}, где $i-1 \le k \le i+1, j-1 \le l \le j+1, (k,l) \ne (i,j)$. Операция сглаживания матрицы дает новую матрицу того же размера, каждый элемент которой получается как среднее арифметическое имеющихся соседей соответствующего элемента исходной матрицы. Построить результат сглаживания заданной вещественной матрицы размером 10x10.
   В сглаженной матрице найти сумму модулей элементов, расположенных ниже главной диагонали.
10. Элемент матрицы называется локальным минимумом, если он строго меньше всех имеющихся у него соседей (определение соседних элементов см. в варианте 9). Подсчитать количество локальных минимумов заданной матрицы размером 10 x 10.
    Найти сумму модулей элементов, расположенных выше главной диагонали.
11. Коэффициенты системы линейных уравнений заданы в виде прямоугольной матрицы. С помощью допустимых преобразований привести систему к треугольному виду.
    Найти количество строк, среднее арифметическое элементов которых меньше заданной величины.
12. Уплотнить заданную матрицу, удаляя из нее строки и столбцы, заполненные нулями.
    Найти номер первой из строк, содержащих хотя бы один положительный элемент.
13. Осуществить циклический сдвиг элементов прямоугольной матрицы на n элементов вправо или вниз (в зависимости от введенного режима), n может быть больше количества элементов в строке или столбце.
14. Осуществить циклический сдвиг элементов квадратной матрицы размером MxN вправо на k элементов таким образом: элементы первой строки сдвигаются в последний столбец сверху вниз, из него — в последнюю строку справа налево, из нее — в первый столбец снизу-вверх, из него — в первую строку; для остальных элементов — аналогично.
15. Дана целочисленная прямоугольная матрица. Определить номер первого из столбцов, содержащих хотя бы один нулевой элемент.
    Характеристикой строки целочисленной матрицы назовем сумму ее отрицательных четных элементов. Переставляя строки заданной матрицы, расположить их в соответствии с убыванием характеристик.
16. Упорядочить строки целочисленной прямоугольной матрицы по возрастанию количества одинаковых элементов в каждой строке.
    Найти номер первого из столбцов, не содержащих ни одного отрицательного элемента.
17. Путем перестановки элементов квадратной вещественной матрицы добиться того, чтобы ее максимальный элемент находился в левом верхнем углу,

следующий по величине — в позиции (2, 2), следующий по величине — в позиции (3, 3) и т. д., заполнив таким образом всю главную диагональ.

Найти номер первой из строк, не содержащих ни одного положительного элемента.

18. Дана целочисленная прямоугольная матрица. Определить:
    • количество строк, содержащих хотя бы один нулевой элемент;
    • номер столбца, в котором находится самая длинная серия одинаковых элементов.
19. Дана целочисленная квадратная матрица. Определить:
    • сумму элементов в тех строках, которые не содержат отрицательных элементов;
    • минимум среди сумм элементов диагоналей, параллельных главной диагонали матрицы.
20. Дана целочисленная прямоугольная матрица. Определить:
    • количество отрицательных элементов в тех строках, которые содержат хотя бы один нулевой элемент;
    • номера строк и столбцов всех седловых точек матрицы.

    Матрица A имеет седловую точку A_{ij}, если A_{ij} является минимальным элементом в i-й строке и максимальным — в j-м столбце.

Строковый тип

1. Написать программу, которая считывает из текстового файла три предложения и выводит их в обратном порядке.
2. Написать программу, которая считывает текст из файла и выводит на экран только предложения, содержащие введенное с клавиатуры слово.
3. Написать программу, которая считывает текст из файла и выводит на экран только строки, содержащие двузначные числа.
4. Написать программу, которая считывает английский текст из файла и выводит на экран слова, начинающиеся с гласных букв.
5. Написать программу, которая считывает текст из файла и выводит его на экран, меняя местами каждые два соседних слова.
6. Написать программу, которая считывает текст из файла и выводит на экран только предложения, не содержащие запятых.
7. Написать программу, которая считывает текст из файла и определяет, сколько в нем слов, состоящих не более чем из четырех букв.
8. Написать программу, которая считывает текст из файла и выводит на экран только цитаты, то есть предложения, заключенные в кавычки.
9. Написать программу, которая считывает текст из файла и выводит на экран только предложения, состоящие из заданного количества слов.
10. Написать программу, которая считывает английский текст из файла и выводит на экран слова текста, начинающиеся и оканчивающиеся на гласные буквы.
11. Написать программу, которая считывает текст из файла и выводит на экран только строки, не содержащие двузначных чисел.
12. Написать программу, которая считывает текст из файла и выводит на экран только предложения, начинающиеся с тире, перед которым могут находиться только пробельные символы.
13. Написать программу, которая считывает английский текст из файла и выводит его на экран, заменив прописной каждую первую букву слов, начинающихся с гласной буквы.
14. Написать программу, которая считывает текст из файла и выводит его на экран, заменив цифры от 0 до 9 словами «ноль», «один», «девять», начиная каждое предложение с новой строки.

15. Написать программу, которая считывает текст из файла, находит самое длинное слово и определяет, сколько раз оно встретилось в тексте.
16. Написать программу, которая считывает текст из файла и выводит на экран сначала вопросительные, а затем восклицательные предложения.
17. Написать программу, которая считывает текст из файла и выводит его на экран, после каждого предложения добавляя, сколько раз встретилось в нем введенное с клавиатуры слово.
18. Написать программу, которая считывает текст из файла и выводит на экран все его предложения в обратном порядке.
19. Написать программу, которая считывает текст из файла и выводит на экран сначала предложения, начинающиеся с однобуквенных слов, а затем все остальные.
20. Написать программу, которая считывает текст из файла и выводит на экран предложения, содержащие максимальное количество знаков пунктуации.