Методические указания

по организации практических занятий и самостоятельных работ

Основы алгоритмизации и программирования

Специальность

09.02.07 Информационные системы и программирование

2023

Методические указания по Основы алгоритмизации и программирования разработаны с учетом ФГОС среднего профессионального образования специальности 09.02.07 Информационные системы и программирование (по отраслям), предназначены для студентов и преподавателей колледжа.

Методические указания определяют этапы выполнения работы на практическом занятии, содержат рекомендации по выполнению индивидуальных заданий и образцы решения задач, а также список рекомендуемой литературы.

Составитель (автор): преподаватель колледжа

Рассмотрены на заседании предметно-цикловой информационных технологий

Протокол № от « 30 » июня 2023 г

Председатель предметно-цикловой комиссии

личная подпись

и одобрены решением учебно-методического совета колледжа.

Протокол № от « 30 » июня 2023 г

Председатель учебно-методического совета колледжа

личная подпись

Рекомендованы к практическому применению в образовательном процессе

Рецензенты:

Оглавление

ПРАКТИЧЕСКАЯ РАБОТА №1
Построение линейных и ветвящихся блок-схем.

Цели:

- *Изучить определение алгоритма и его свойства;*
- *Изучить виды представления алгоритмов;*
- *Изучить объекты построения блок-схем;*
- *Изучить примеры блок-схем.*

Алгоритмом называется строго определенная последовательность действий, определяющих процесс перехода от исходных данных к искомому результату.

Свойства алгоритмов:

Детерминированность (Однозначность). Каждое действие (шаг, этап) должно быть четким, однозначным, исключающим произвольное толкование и не оставляющим места для двусмысленности. Выполнение алгоритма носит, по сути, механический характер и не требует никаких дополнительных указаний.

Результативность. Алгоритм должен приводить к решению задачи или сообщению, что задача решений не имеет за конечное число шагов.

Конечность. Каждое отдельное действие, как и весь алгоритм должны иметь возможность реального исполнения. Поэтому алгоритм имеет придел, т. е. конечен.

Массовость. Алгоритм разрабатывается в общем виде так, чтобы его можно было применять для класса задач, различающихся только исходными данными. При этом исходные данные выбираются из некоторой области, которая называется областью применяемости алгоритма. Например, для решения квадратного уравнения $ax^2+bx+c=0$, коэффициенты действительные числа, $a \neq 0$, и a, в, с – различные.

Блок-схема – это графическая реализация алгоритма, которая представляет собой удобный и наглядный способ записи алгоритма.

Блок-схема состоит из функциональных блоков разной формы, связанных между собой стрелками. В каждом блоке описывается одно или несколько действий.

Способы записи алгоритмов:

Существуют разные способы записи алгоритмов:

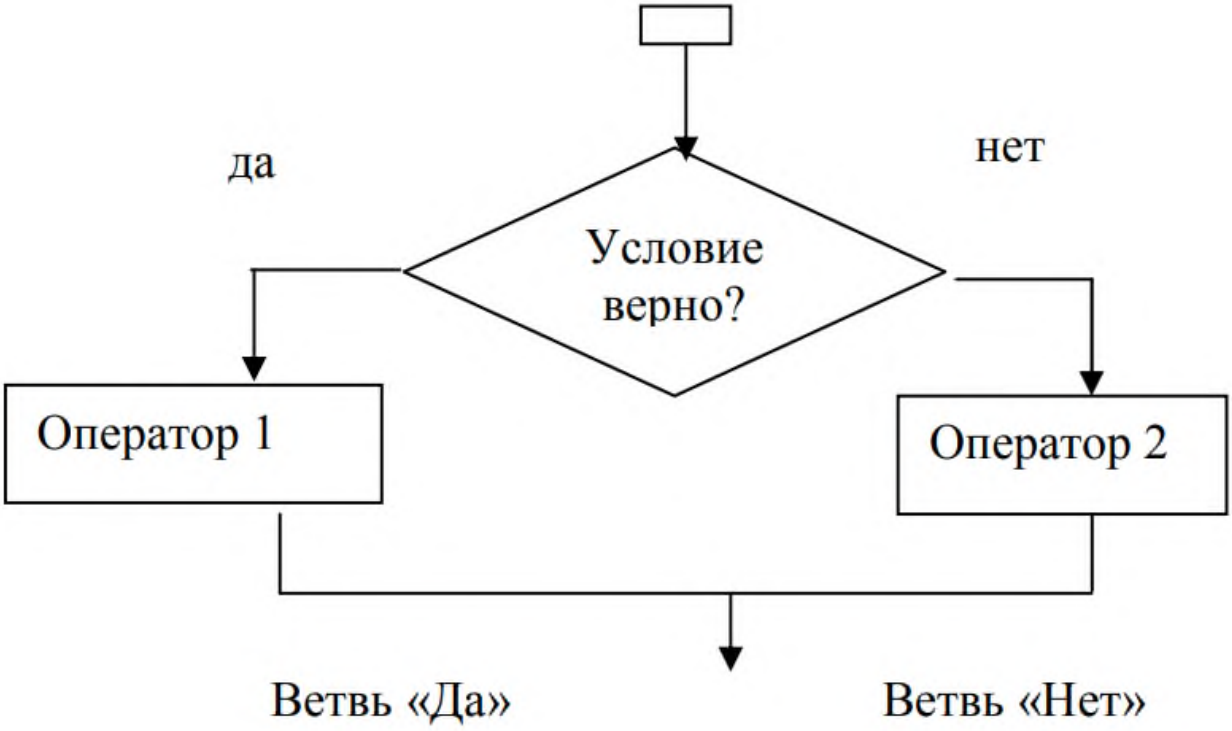
- Словесно-формульный
- Графический
- Операторный или псевдокод (программа на алгоритмическом языке).

Типы алгоритмов:

Алгоритмы бывают линейные, разветвляющиеся и циклические.

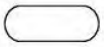
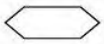
Линейный алгоритм – это алгоритм, в котором действия выполняются только один раз и строго в том порядке, в котором они записаны. Линейные алгоритмы в математике – это, например, вычисление площадей фигур.

Разветвляющийся алгоритм – это алгоритм, в котором то или иное действие выполняется после анализа условия. Процесс анализа условия и выбора одной из ветвей на блок-схеме показывают с помощью логического блока. Логический блок имеет один вход и два выхода (ветвь «да» и ветвь «нет»).



Циклический алгоритм – это алгоритм, в котором группа операторов выполняется несколько раз подряд. Блок-схема цикла обязательно содержит логический блок. Выполняется циклический алгоритм так: сначала проверяется условие, если условие верно (истина), то выполняется тело цикла (действия или группа операторов) и, далее, изменяются значения параметра цикла (управляющей переменной) и снова проверяется условие и т. д. На каком-то шаге условие не выполнится (ложь) и тогда происходит выход из цикла и продолжается выполнение программы.

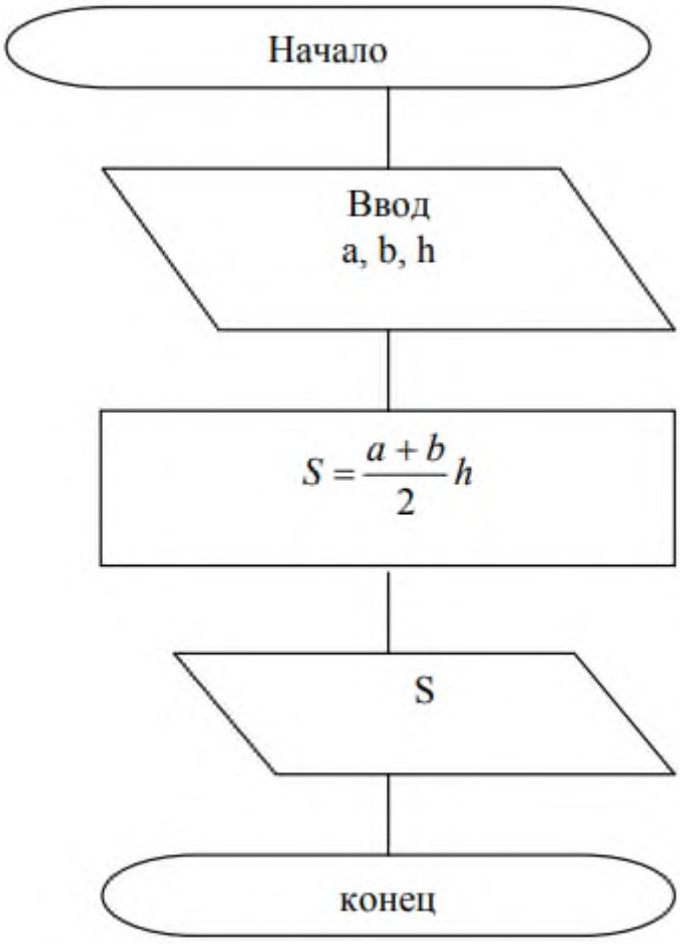
В таблице приведены основные объекты блок-схем:

Название блока	Обозначение	Назначение блока
Терминатор		Начало или завершение программы
Процесс		Обработка данных: вычисления, пересылки и т.п.
Данные		Операции ввода-вывода данных
Решение		Ветвление, выбор, итерационные и поисковые циклы
Подготовка		Счетные циклы
Предопределенный процесс		Вызов процедур, функций
Соединитель		Маркировка разрывов линий
Комментарий		Пояснения к операциям

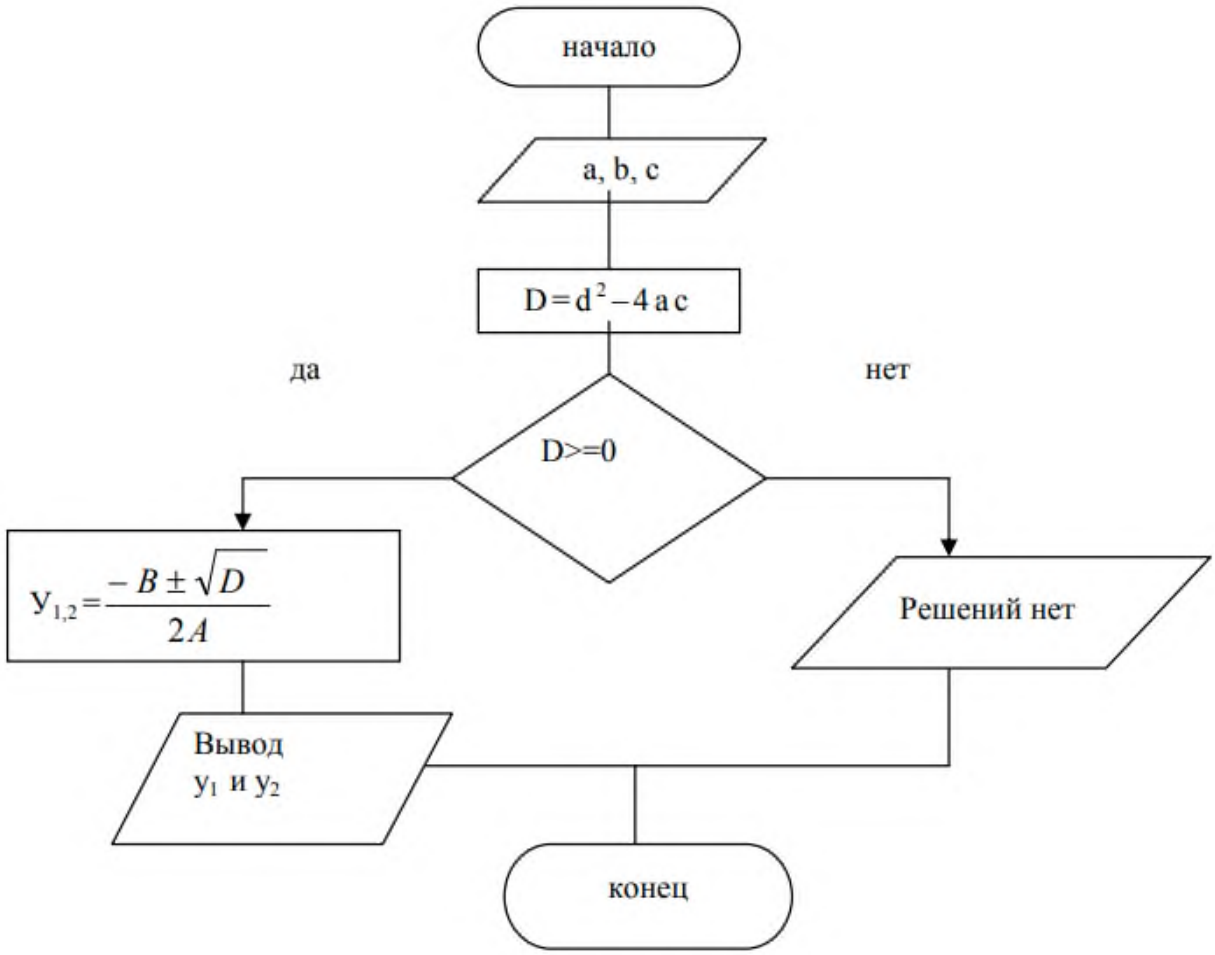
Примеры алгоритмов:

Пример 1 (Линейный алгоритм). Составить алгоритм вычисления площади трапеции с основаниями a, b и высотой h.

Решение:



Пример блок-схемы решения квадратного уравнения $ax^2+bx+c=0$. (ветвящийся алгоритм).



ЗАДАНИЕ

Построить блок-схемы согласно заданию. Задание выполняется в общей тетради по предмету, с обратной стороны.

Вариант 1.

1. Вычислить значения функции $y = 17x^2 – 6x + 13$ при любом значении x;
2. Дана сторона квадрата. Найти его периметр.
3. Даны два целых числа. Найти их среднее арифметическое;
4. Даны катеты прямоугольного треугольника. Найти его периметр.
5. Известна стоимость 1 кг конфет, печенья и яблок. Найти стоимость всей покупки, если купили x кг конфет, y кг печенья и z кг яблок.
6. Даны радиус круга и сторона квадрата. У какой фигуры площадь больше?
7. Проверить, принадлежит ли число, введенное с клавиатуры, интервалу (–5, 3).
8. Определить, является ли треугольник со сторонами a, b, c равносторонним или равнобедренным.

Вариант 2.

1. Вычислить значения функции $y = 3a^2 + 5a – 21$ при любом значении a.
2. Дан радиус окружности. Найти ее диаметр.
3. Даны два целых числа. Найти их среднее геометрическое. (Средним геометрическим называется квадратный корень из произведения этих чисел).
4. Даны катеты прямоугольного треугольника. Найти его гипотенузу.
5. Известна стоимость монитора, системного блока, клавиатуры и мыши. Сколько будут стоить 3 компьютера из этих элементов? N компьютеров?
6. Определить, является ли число A делителем числа B? A наоборот?
7. Дано натуральное число. Определить, является ли оно двузначным.
8. Известны возрасты Мити и Васи. Определить, кто из них старше или они одного возраста.

ПРАКТИЧЕСКАЯ РАБОТА №2
Построение циклических блок-схем.

Цели:

- Изучить виды циклов;
- Изучить представление циклических блок схем.

Циклический алгоритм – это алгоритм, в котором группа операторов выполняется несколько раз подряд. Блок-схема цикла обязательно содержит логический блок. Выполняется циклический алгоритм так: сначала проверяется условие, если условие верно (истина), то выполняется тело цикла (действия или группа операторов) и, далее, изменяются значения параметра цикла (управляющей переменной) и снова проверяется условие и т. д. На каком-то шаге условие не выполнится (ложь) и тогда происходит выход из цикла и продолжается выполнение программы.

В рассмотрении циклического алгоритма следует выделить несколько понятий.

Тело цикла – это набор инструкций, предназначенный для многократного выполнения.

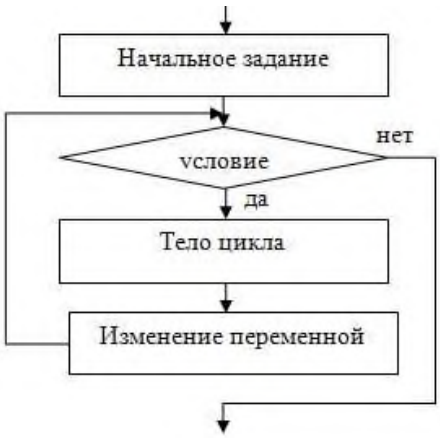
Итерация – это единичное выполнение тела цикла.

Переменная цикла (управляющая переменная) – это величина, изменяющаяся на каждой итерации цикла.

Каждый цикл должен содержать следующие необходимые элементы:

- *первоначальное задание переменной цикла,*
- *проверку условия,*
- *выполнение тела цикла,*
- *изменение переменной цикла.*

Циклы бывают двух видов – с предусловием и с постусловием. В цикле с предусловием сначала проверяется условие входа в цикл, а затем выполняется тело цикла, если условие верно. Цикл с предусловием также может быть задан с помощью счетчика. Это удобно в тех случаях, когда точно известно количество итераций. В общем виде блок-схема, реализующая цикл с предусловием, представлена ниже. Сначала задается начальное значение переменной цикла, затем условие входа в цикл, тело цикла и изменение переменной цикла. Выход из цикла осуществляется в момент проверки условия входа в цикл, когда оно не выполняется, т.е. условие ложно. Цикл с предусловием может ни разу не выполниться, если при первой проверке условия входа в цикл оно оказывается ложным.



В цикле с постусловием сначала выполняется тело цикла, а потом проверяется условие.

Если условие верно, то итерация повторяется, если же неверно, то осуществляется выход из цикла. В отличие от цикла с предусловием, любой цикл с постусловием всегда выполнится хоть раз.

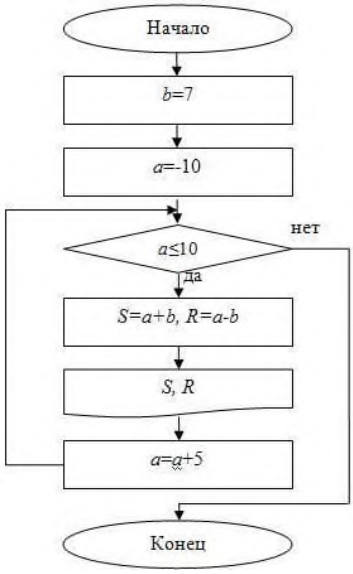
Как видно из представленных блок-схем для циклов с предусловием и постусловием, условие записывается внутри блока условия (формы ромба), как и в разветвляющемся алгоритме. Принципиальная разница между разветвляющимся и циклическим алгоритмами при графической реализации состоит в том, что в циклическом алгоритме в обязательном порядке присутствует стрелка, идущая наверх. Именно эта стрелка обеспечивает многократный повтор тела цикла.

Пример 1

Даны числа a, b. Известно, что число a меняется от -10 до 10 с шагом 5, $b = 7$ и не изменяется. Вычислить сумму S и разность R чисел a и b для всех значений a и b.

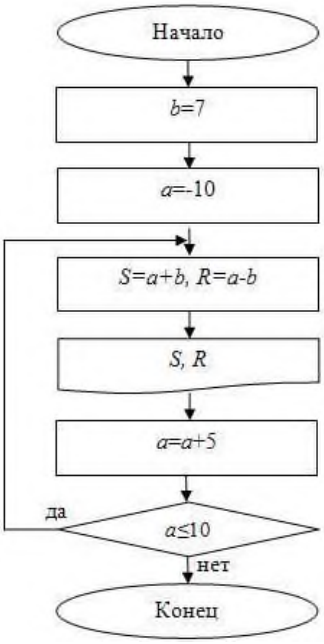
Решение: Здесь число a меняется от -10 до 10 с шагом 5. Это означает, что число a является переменной цикла. Сначала a равно -10 – это первоначальное значение переменной цикла. Далее a будет изменяться с шагом 5, и т.д. пока не будет достигнуто значение 10 – это соответствует

изменению переменной цикла. Итерации надо повторять, пока выполняется условие "$a \leq 10$". Итак, a будет принимать следующие значения: -10, -5, 0, 5, 10. Число b не будет являться переменной цикла, т.к. $b = 7$ и не изменяется по условию задачи.

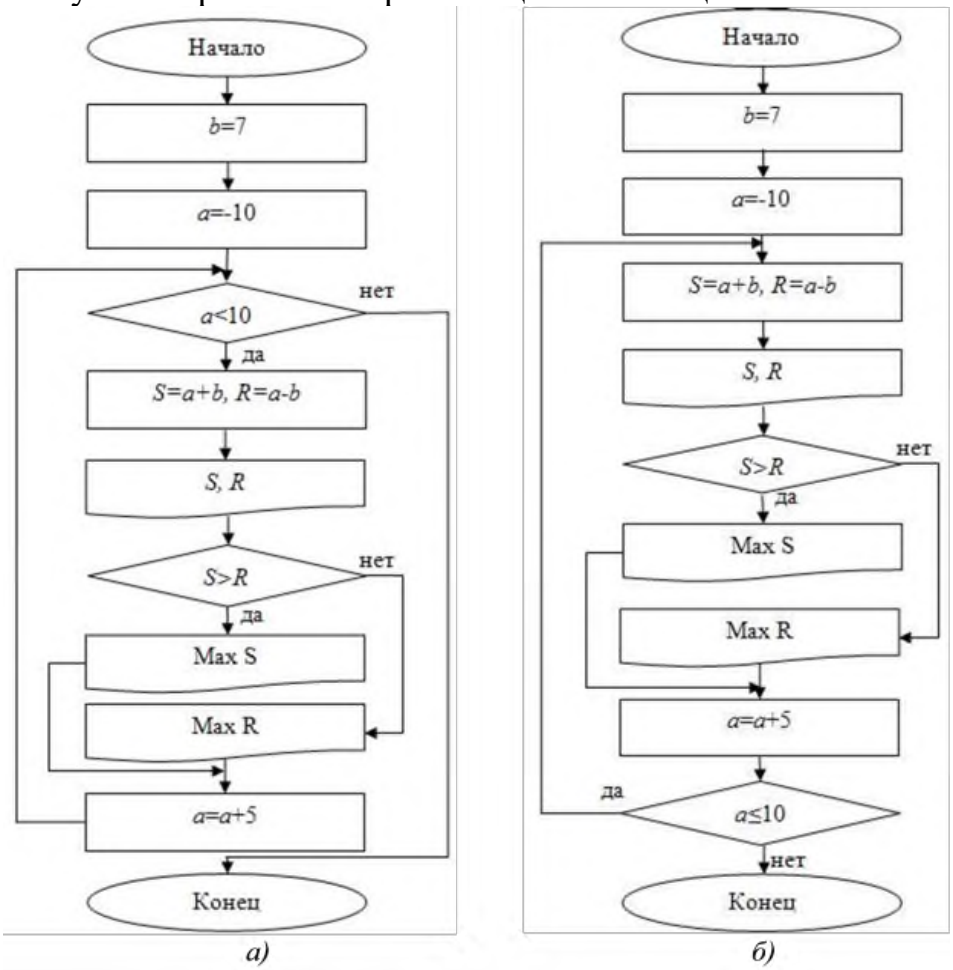


Тело цикла состоит из нескольких действий: вычисление суммы, вычисление разности и вывод полученных данных на экран. Таким образом, у нас получится несколько значений сумм и разностей, т.к. а изменяется. Количество сумм и количество разностей совпадет с количеством различных значений а, т.е. пять.

Данная задача может быть сделана и с циклом с предусловием, и с постусловием. В этом случае тело цикла, условие и изменение переменной цикла будут такими же, как и в цикле с предусловием, но сначала необходимо выполнить тело цикла, а потом проверить условие для выполнения следующей итерации.



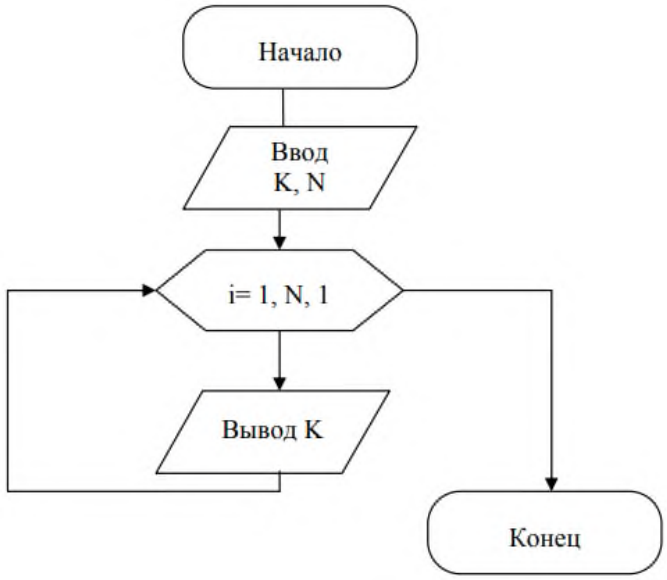
В данной задаче также могут быть соединены циклический и разветвляющийся алгоритмы, если по условию задачи требуется сравнить полученные значения суммы и разности. В этом случае цикл можно реализовать как с предусловием, так и с постусловием, а сравнение суммы и разности добавится внутрь тела цикла, т.к. следует сравнить между собой все полученные суммы и разности. Организация самого цикла останется прежней.



Циклы с параметром (безусловные циклы).

В цикле с параметром число повторений цикла однозначно определено и задается с помощью начального, конечного значений параметра и шагом его изменения.

Пример 2. Составьте блок-схему к следующей задаче: даны целые числа K и N (N > 0). Вывести N раз число K.



Справочная информация!

Оператор	Операция
+	Сложение
-	Вычитание
*	Умножение
/	Деление
%	Остаток от деления
++	Инкремент
--	Декремент

Деление по модулю возвращает остаток от деления: 13 % 2 = 1. Чтобы проверить число на четность, необходимо разделить его на 2 по модулю (%) и проверить остаток от деления. Если остаток равен 0 – то число четное, иначе – нечётное!

Таким же образом проверяется кратность чисел. Если при делении по модулю A%B остаток равен 0, то B является делителем числа A, или же A делится нацело на B!

ЗАДАНИЕ

Построить блок-схемы согласно заданию. Задание выполняется в общей тетради по предмету, с обратной стороны.

Вариант 1.

Использовать цикл с параметром!

1. Напечатать квадраты всех целых чисел от a до 50 (значение a вводится с клавиатуры; $a \leq 50$)

2. Одна штука некоторого товара стоит 20,4 руб. Напечатать таблицу стоимости 2, 3, ..., 20 штук этого товара.

3. Найти произведение всех целых чисел от a до 15 (значение a вводится с клавиатуры.

Использовать цикл с предусловием!

4. Найти n-й член последовательности Фибоначчи.

5. Найти сумму положительных четных чисел, меньших 50.

6. Напечатать те натуральные числа, квадрат которых не превышает заданное число n.

Использовать цикл с постусловием!

7. Напечатать числа 1.0, 1.5, 2.0, …, 13.5.

8. Найти 15 первых натуральных чисел, делящихся нацело на 19 и находящихся в интервале, левая граница которого равна 100.

Вариант 2.

Использовать цикл с параметром!

1. Напечатать все целые числа от a до b (значения a и b вводятся с клавиатуры).

2. Напечатать таблицу перевода 1, 2, ... 20 долларов США в рубли по текущему курсу (значение курса вводится с клавиатуры).

3. Найти произведение всех целых чисел от 1 до b (значение b вводится с клавиатуры

Использовать цикл с предусловием!

4. Получить первые n членов последовательности Фибоначчи.

5. Найти сумму целых положительных нечетных чисел из промежутка от a до b.

6. Подготовьте фрагмент программы, в котором должны вводиться 10 чисел. Если будет введено число 0, ввод должен прекратиться.

Использовать цикл с постусловием!

7. Напечатать все кратные тринадцати натуральные числа, меньшие 100.

8. Найти 20 первых натуральных чисел, делящихся нацело на 13 или на 17 и находящихся в интервале, левая граница которого равна 500.

ПРАКТИЧЕСКАЯ РАБОТА №3
Построение алгоритмов с помощью псевдокода.

Цели:

– *Изучить операторы алгоритмического языка;*

– *Разобрать примеры написания алгоритмов с помощью псевдокода.*

Псевдокод представляет собой описание структуры алгоритма на естественном, частично-формализованном языке, позволяющее выявить основные этапы решения задачи перед точной его записью на языке программирования. В псевдокоде используются некоторые формальные конструкции и общепринятая математическая символика.

Данный способ тесно связан со структурным подходом к программированию. Псевдокод занимает промежуточное положение между естественным языком и языком программирования. Его применяют преимущественно для того, чтобы подробнее объяснить работу программы, что облегчает проверку правильности программы.

Операторы – это инструкции, предписывающие компьютеру выполнить определенное действие. При написании псевдокода мы исходим из того, что все инструкции будут выполняться по порядку, сверху вниз.

Некоторые ключевые слова приведены ниже:

алг (алгоритм)	сим (символьный)	дано	для	да
арг (аргумент)	лит (литерный)	надо	от	нет
рез (результат)	лог (логический)	если	до	при
нач (начало)	таб (таблица)	то	знач	выбор
кон (конец)	нц (начало цикла)	иначе	и	ввод
цел (целый)	кц (конец цикла)	все	или	вывод
вещ (вещественный)	длин (длина)	пока	не	утв

Пример 1.

Определить площадь трапеции по введенным значениям оснований (a и b) и высоты (h). Запись решения задачи на алгоритмическом языке:

```
алг трапеция
        вещ a,b,h,s
нач
        ввод f,b,h
        s:=((a+b)/2)*h
        вывод s
кон
```

Пример 2.

Определить среднее арифметическое двух чисел, если А положительное и частное (a/b) в противном случае. Запись решения задачи на алгоритмическом языке:

```
алг числа
        вещ a,b,c
нач
        ввод a,b
        если a>0 то
                c:=(a+b)/2
        иначе
                c:=a/b
        все
        вывод c
кон
```

Пример 3.

Составить алгоритм нахождения суммы целых чисел в диапазоне от 1 до 10. Запись решения задачи на алгоритмическом языке:

```
алг сумма
        вещ a,s
нач
        S:=0;
        a:=1;
        Нц
                пока a<=10
                S:=S+a;
                A:=a+1;
        Кц
        вывод S
кон
```

Эта же задача, с использованием цикла с параметром:

```
алг сумма
        вещ a,s
нач
        S:=0;
        Нц
                для a от 1 до 10
                S:=S+a;
        Кц
        вывод S
кон
```

Пример 4.

В алгоритме с постусловием сначала выполняется тело цикла, а затем проверяется условие окончания цикла. Решение задачи нахождения суммы первых десяти целых чисел в данном случае будет выглядеть следующим образом:

```
алг сумма
        вещ a,s
нач
        S:=0;
        a:=1;
        нц
                S:=S+a;
                a:=a+1;
                пока a<=10
        кц
        вывод S
кон
```

ЗАДАНИЕ

Написать алгоритм с помощью команд псевдокода согласно заданию. Задание выполняется в общей тетради по предмету, с обратной стороны.

Вариант 1.

Использовать цикл с параметром!

1. Напечатать все целые числа от a до b (значения a и b вводятся с клавиатуры).

2. Напечатать таблицу перевода 1, 2, ... 20 долларов США в рубли по текущему курсу (значение курса вводится с клавиатуры).

3. Найти произведение всех целых чисел от 1 до b (значение b вводится с клавиатуры

Использовать цикл с предусловием!

4. Получить первые n членов последовательности Фибоначчи.

5. Найти сумму целых положительных нечетных чисел из промежутка от a до b.

6. Подготовьте фрагмент программы, в котором должны вводиться 10 чисел. Если будет введено число 0, ввод должен прекратиться.

Использовать цикл с постусловием!

7. Напечатать все кратные тринадцати натуральные числа, меньшие 100.

8. Найти 20 первых натуральных чисел, делящихся нацело на 13 или на 17 и находящихся в интервале, левая граница которого равна 500.

Вариант 2.

Использовать цикл с параметром!

1. Напечатать квадраты всех целых чисел от a до 50 (значение a вводится с клавиатуры; $a \leq 50$)

2. Одна штука некоторого товара стоит 20,4 руб. Напечатать таблицу стоимости 2, 3, ..., 20 штук этого товара.

3. Найти произведение всех целых чисел от a до 15 (значение a вводится с клавиатуры.

Использовать цикл с предусловием!

4. Найти n-й член последовательности Фибоначчи.

5. Найти сумму положительных четных чисел, меньших 50.

6. Напечатать те натуральные числа, квадрат которых не превышает заданное число n.

Использовать цикл с постусловием!

7. Напечатать числа 1.0, 1.5, 2.0, …, 13.5.

8. Найти 15 первых натуральных чисел, делящихся нацело на 19 и находящихся в интервале, левая граница которого равна 100.

ПРАКТИЧЕСКАЯ РАБОТА №4
Построение блок-схем обработки массивов.

Цели:

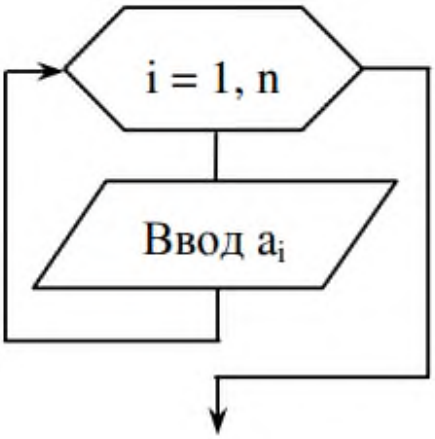
- *Изучить определение массива;*
- *Научиться упорядочивать массив;*
- *Научиться находить минимальные и максимальный элементы.*

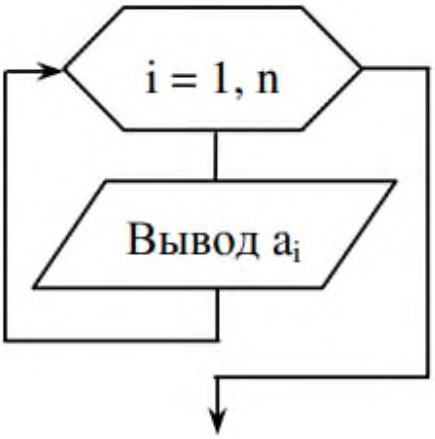
Массивы представляют собой упорядоченную совокупность данных, имеющую одно имя. Каждому элементу массива соответствует выражение порядкового типа (чаще – целое число), определяющее место этого элемента в массиве, которое называется индексом.

Размерность массива – количество индексов, необходимое для однозначного доступа к элементу массива. Если для определения места элемента в массиве используется один индекс, то массив называют одномерным (вектором), два – двумерным (матрицей).

Во многих языка программирования индекс заключается в квадратные скобки. Индекс может быть константой – ***a[5], b[1,1]***; переменной – ***a[i], b[i,j]***; выражением – ***a[i+3], b[i+1,j+1]***.

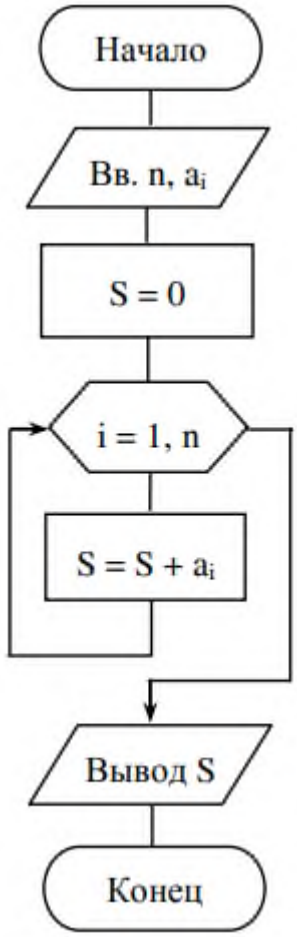
Ввод и вывод элементов одномерного массива представляют собой циклический процесс. Параметром цикла является текущее значение индекса i, изменяющееся в общем случае от начального до конечного значения с шагом 1. Блок-схемы алгоритмов ввода и вывода элементов одномерного массива приведены соответственно ниже:





Вычисление суммы элементов одномерного массива:

Задан одномерный массив A(n), состоящий из n элементов. Вычислить сумму всех его элементов: $S = \sum_{i=1}^{n} A[i]$ Вычисление суммы выполняется по формуле: S = S+A[i]. Решение задачи показано ниже:

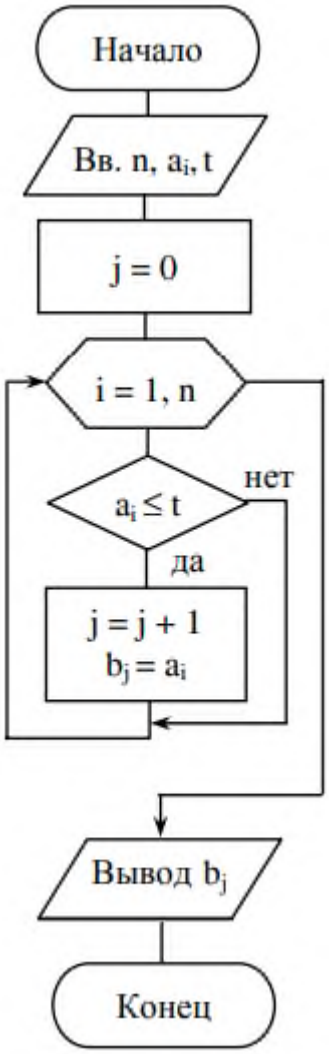


Формирование нового массива из элементов заданного массива, удовлетворяющих заданному условию, и подсчет их количества:

Требуется из заданного массива A(n), состоящего из n элементов, выбрать элементы, удовлетворяющие заданному условию, $a_i \leq t$ и сформировать из них массив B.

Учитывая, что не все элементы массива A войдут в массив B, необходимо выполнять подсчет количества j элементов, удовлетворяющих заданному условию.

Особенностью решения этой задачи является то, что индексы элементов массивов A и B не совпадают, поэтому для обозначения индекса элементов массива B используют переменную j, значение которой изменяется на 1 перед занесением в массив B нового значения.



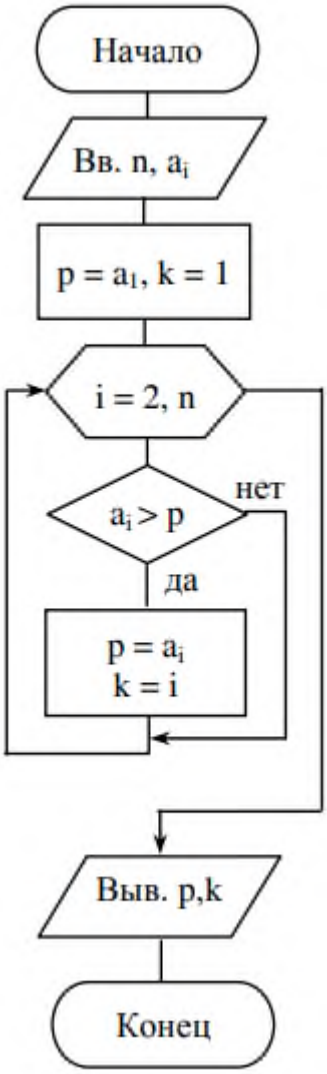
Поиск максимального (минимального) элемента в массиве с запоминанием его индекса:

Требуется из заданного массива A(n), состоящего из n элементов, найти максимальный (минимальный) элемент и его индекс. На блок-схеме алгоритма поиска максимального элемента p – максимальное значение из элементов массива, k – индекс максимального элемента.

При поиске максимального элемента в массиве вначале считаем кандидатом на максимальный первый элемент массива, затем в цикле сравниваем очередной элемент массива с кандидатом на максимальный, если очередной элемент больше кандидата, то меняем значение кандидата на значение данного элемента массива и фиксируем его индекс.

При поиске минимального элемента в массиве необходимо условие $a_i > p$ заменить условием $a_i < p$.

Если в массиве несколько элементов имеют одинаковое максимальное (минимальное) значение, то в k запоминается индекс первого из них. Для того чтобы в k запоминался индекс последнего максимального элемента, нужно условие $\boldsymbol{a_i > p}$ заменить условием $\boldsymbol{a_i \ge p}$, а для нахождения минимального – условием $\boldsymbol{a_i \le p}$.

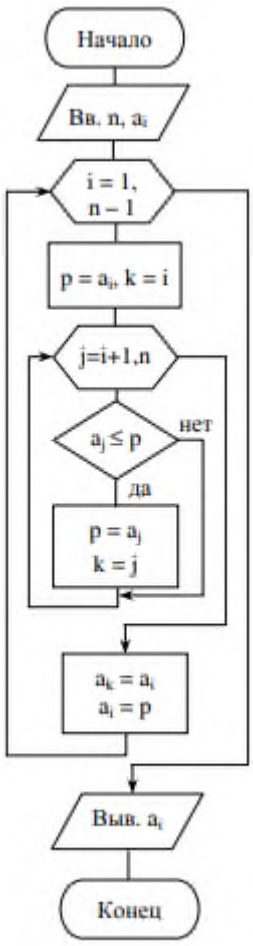


Упорядочивание массива:

Требуется расположить n элементов массива A(n) в порядке возрастания (убывания) их величин. Для решения задачи используется метод, основанный на поиске минимального (максимального) элемента массива или части массива, называемый сортировка выбором.

Для упорядочивания массива по возрастанию вначале находим минимальный элемент из всего массива, меняем его местами с первым элементом. Затем находим минимальный элемент из оставшегося (не считая первого элемента) массива, меняем его местами со вторым элементом и т.д. После нахождения минимального элемента из последних двух элементов и размещения его на предпоследнем месте на последнем остается самый большой элемент – массив упорядочен.

Блок-схема алгоритма упорядочивания массива приведена ниже, где p – переменная для хранения значения минимального элемента на каждом шаге, k – индекс минимального элемента на каждом шаге, i – индекс элемента упорядоченного массива, j – индекс элемента части массива, в которой отыскивается минимальный элемент.



Как поменять значения двух переменных местами?
A = 10
B = 5
Существует несколько способов для выполнения данного действия:

1. *Необходимо добавить третью – резервную переменную C:*
   C = A
   A = B => A = 5
   B = C B = 10
2. *Второй способ решает данную задачу* ***без использования*** *третьей переменной:*
   A = A+B
   B = A – B => A = 5
   A = A – B B = 10

ЗАДАНИЕ

Построить блок-схемы согласно заданию. Задание выполняется в общей тетради по предмету, с обратной стороны.

Вариант 1.

1. Найти сумму элементов одномерного массива, состоящего из 10 элементов. Разделить каждый элемент исходного массива на полученное значение.
2. Найти среднее арифметическое значение элементов массива, состоящего из 5 элементов. Преобразовать исходный массив, вычитая из каждого элемента среднее значение.
3. Задан массив, состоящий из 10 элементов. Сформировать два массива по 5 элементов каждый, включая в первый элементы исходного массива с четными индексами, а во второй – с нечетными.
4. Вычислить минимальный элемент массива, состоящего из 8 элементов, и его номер. Преобразовать исходный массив, вычитая из каждого элемента минимальное значение.
5. Дан массив вещественных чисел. Каждый элемент, больший 10, заменить его квадратным корнем.

Вариант 2.

1. Задан массив, состоящий из 5 элементов. Вычислить значения функции $y = x^2$ при значениях аргумента, заданных в массиве x, и поместить их в массив y.
2. Определить среднее арифметическое значение элементов массива, состоящего из 6 элементов. Найти далее индекс элемента массива, наиболее близкого к среднему значению, т.е. найти минимальную по абсолютной величине разность элементов массива и среднего арифметического значения (|ai – D|, где D – среднее арифметическое значение).
3. Найти максимальный и минимальный элементы массива, состоящего из 8 элементов, и поменять их местами.
4. В массиве хранится информация о количестве побед, одержанных 20 футбольными командами. Определить номера команд, имеющих меньше трех побед.
5. Дан массив вещественных чисел. Все элементы массива с четными номерами заменить их абсолютной величиной.

Примечание:

*Квадратный корень во многих языках программирования обозначается функцией **SQRT()** – sqrt(4) = 2*

*Абсолютная величина (модуль числа) обозначается функцией **ABS()** – abs(-5) = 5*

ПРАКТИЧЕСКАЯ РАБОТА №5
Контрольная работа

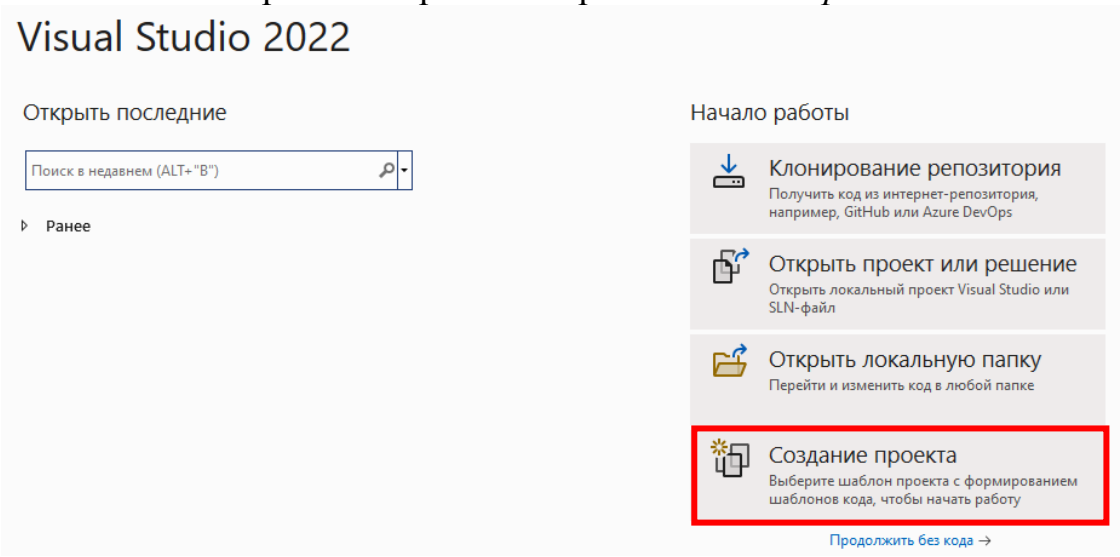
ПРАКТИЧЕСКАЯ РАБОТА №6
Создание линейной программы на языке C#.

Цели:

– *Изучить среду разработки MS Visual Studio;*
– *Научиться создавать и сохранять проект;*
– *Изучить типы данных и арифметические операции;*
– *Научиться выполнять элементарные расчеты.*

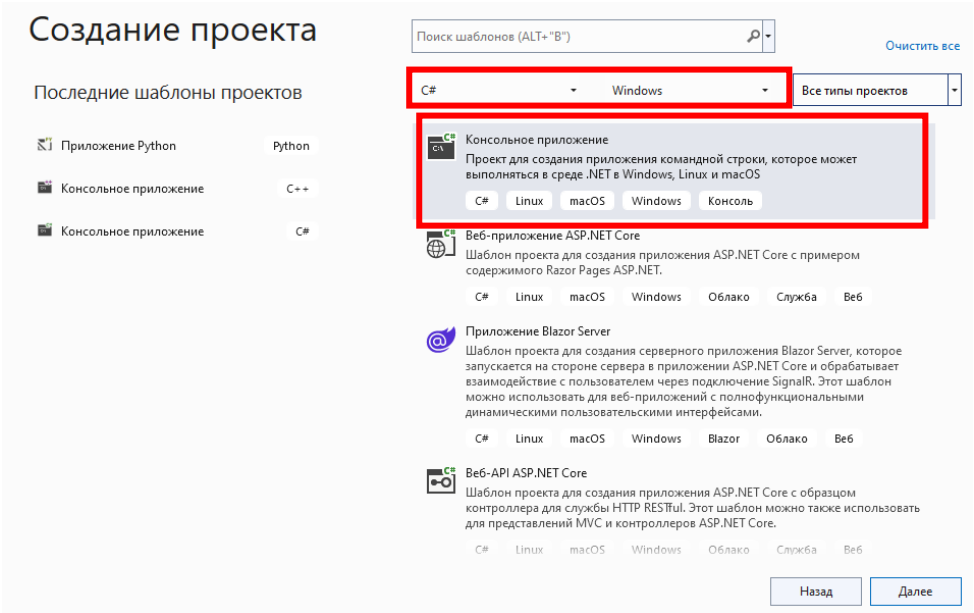
Для создания приложений на C# будем использовать бесплатную и полнофункциональную среду разработки - Visual Studio Community 2022.

Создадим первую программу. Она будет простенькой. Вначале откроем Visual Studio. На стартовом экране выберем *Создание проекта*.

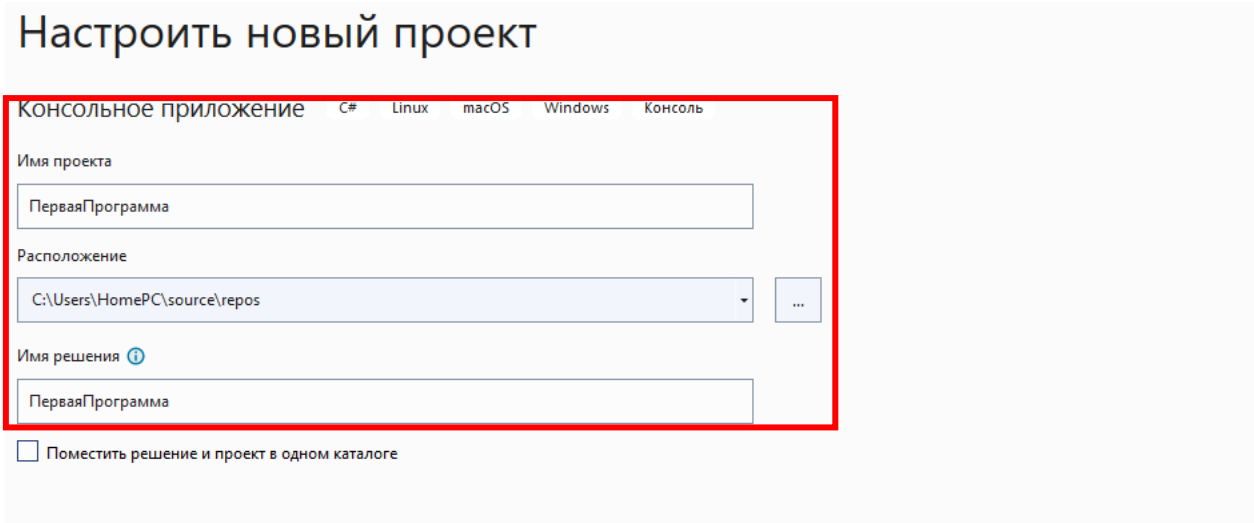


На следующем окне в качестве типа проекта выберем *Консольное приложение*, то есть мы будем создавать консольное приложение на языке C#.

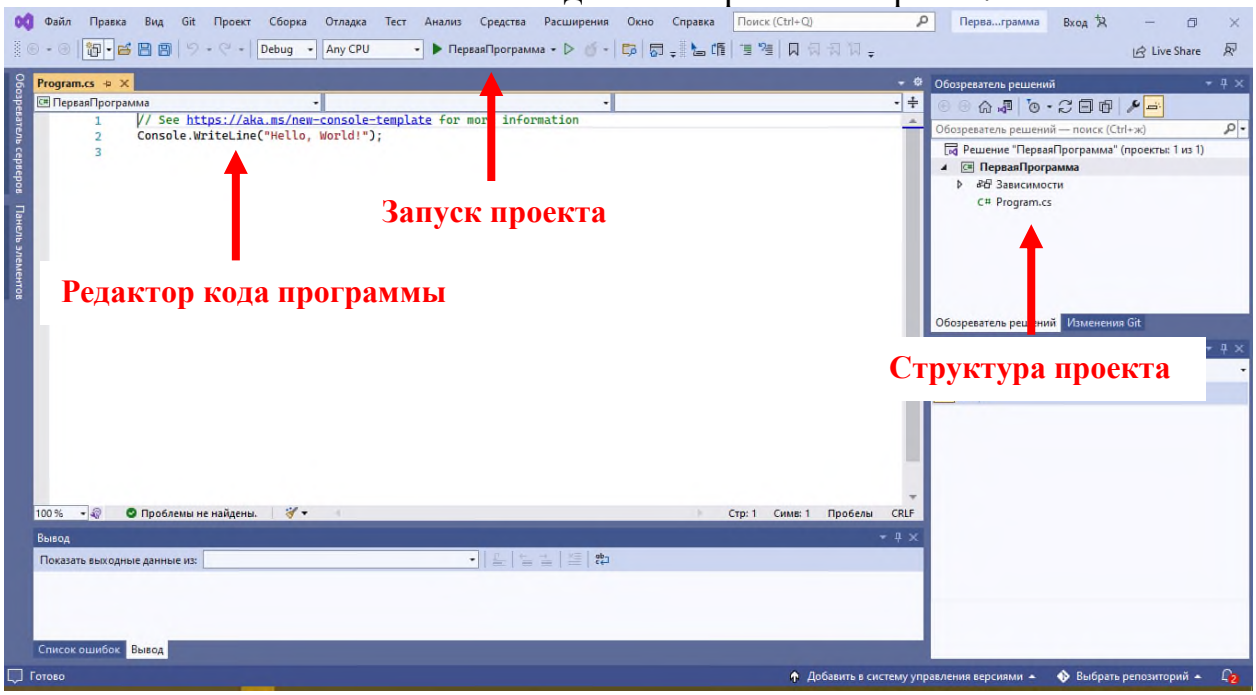
Для быстрого поиска можно выбрать в качестве языка C#, а в качестве платформы – Windows.



Далее на следующем этапе нам будет предложено указать имя проекта и каталог, где будет располагаться проект. *Решение* может объединять в себе несколько связанных приложений.



После этого Visual Studio создаст и откроет нам проект.



В большом поле в центре, которое, по сути, представляет текстовый редактор, находится сгенерированный по умолчанию код C#. Впоследствии мы изменим его на свой.

Справа находится окно *Обозреватель решений*, в котором можно увидеть структуру нашего проекта. В данном случае у нас сгенерированная по умолчанию структура: узел *Зависимости* – этот узел содержит сборки dll, которые добавлены в проект по умолчанию. Эти сборки как раз содержат классы библиотеки .NET, которые будет использовать C#. Однако не всегда все сборки нужны. Ненужные потом можно удалить, в то же время если понадобится добавить какую-нибудь нужную библиотеку, то именно в этот узел она будет добавляться.

Весь код программы на языке C# помещается в файлы с расширением .cs. По умолчанию в проекте, который создается в Visual Studio уже есть один файл с кодом C# - файл Program.cs.

Далее идет непосредственно сам файл кода программы Program.cs, который по умолчанию открыт в центральном окне и который имеет всего две строки:

```
// See https://aka.ms/new-console-template for more information
        Console.WriteLine("Hello, World!");
```

Первая строка открывается символами // и представляет комментарии - пояснения к коду.

Вторая строка, собственно, представляет собой код программы: ***Console.WriteLine("Hello World!");*** Эта строка выводит на консоль строку "Hello World!".

Именно код файла Program.cs выполняется по умолчанию, если мы запустим проект на выполнение. Но при необходимости мы также можем добавлять другие файлы с кодом C#.

Несмотря на то, что программа содержит только одну строку кода, это уже некоторая программа, которую мы можем запустить. Запустить проект мы можем с помощью клавиши F5 или с панели инструментов, нажав на зеленую стрелку. И если вы все сделали правильно, то при запуске приложения на консоль будет выведена строка "Hello World!". Однако вы не успеете ничего увидеть, так как приложение закроется сразу после выполнения данного оператора. Чтобы задержать окно консоли, необходимо дописать одну строчку:

Console.ReadLine();

Данный оператор считывает символы, вводимые с клавиатуры. После ввода информации следует нажать клавишу **Enter** для выполнения оператора. В нашем же случае достаточно просто нажать **Enter** и программа завершится.

Теперь изменим весь этот код на следующий:

```
Console.Write("Введите свое имя: ");
string name = Console.ReadLine();        // вводим имя
Console.WriteLine("Привет" + name);     // выводим имя на консоль
```

Обратите внимание на разницу между операторами ***Write*** и ***WriteLine***. Первый выводит сообщение и оставляет курсор на текущей строке, второй же после вывода сообщения переводит курсор на следующую строку.

После вывода предложения указать свое имя, мы создаем новую переменную ***name*** строкового типа (***string***) и присваиваем ей значение, введенное с клавиатуры, об этом нам говорит оператор ***ReadLine()***. В C# *команда присвоения* осуществляется с помощью символа «=».

Обратите внимание на вывод сообщения: введенная вручную строка «Привет» и значение переменной *name* объединяются с помощью символа «+».

Базовым строительным блоком программы являются **инструкции** (statement). Инструкция или чаще **оператор** представляет некоторое действие, например, арифметическую операцию, вызов метода, объявление переменной и присвоение ей значения. В конце каждой инструкции в C# ставится точка с запятой (;). Данный знак указывает компилятору на конец инструкции. Например, в проекте консольного приложения, который создается по умолчанию, есть такая строка:

Console.WriteLine("Hello, World!");

Данная строка представляет вызов метода Console.WriteLine, который выводит на консоль строку. В данном случае вызов метода является инструкцией и поэтому завершается точкой с запятой.

Набор инструкций может объединяться в блок кода. Блок кода заключается в фигурные скобки, а инструкции помещаются между открывающей и закрывающей фигурными скобками.

```
{
    Console.WriteLine("Привет");
    Console.WriteLine("Добро пожаловать в C#");
}
```

Одни блоки кода могут содержать другие блоки:

```
{
    Console.WriteLine("Первый блок");
    {
        Console.WriteLine("Второй блок");
    }
}
```

Регистрозависимость

C# является регистрозависимым языком. Это значит, что в зависимости от регистра символов какие-то определенные названия могут представлять разные классы, методы, переменные и т.д. Например, для вывода на консоль используется метод WriteLine - его имя начинается именно с большой буквы: "WriteLine". Если мы вместо "Console.WriteLine" напишем "Console.writeline", то программа не скомпилируется, так как данный метод обязательно должен называться "WriteLine", а не "writeline" или "WRITELINE" или как-то иначе.

Комментарии

Важной частью программного кода являются комментарии. Они не являются собственно частью программы, при компиляции они игнорируются. Тем не менее комментарии делают код программы более понятным, помогая понять те или иные его части. Есть два типа комментариев: *однострочный и многострочный*. Однострочный комментарий размещается на одной строке

после двойного слеша //. А многострочный комментарий заключается между символами /* текст комментария */. Он может размещаться на нескольких строках. Например:

```
/*
       первая программа на C#,
       которая выводит приветствие на консоль
*/
Console.WriteLine("Привет");                // Выводим строку "Привет"
Console.WriteLine("Добро пожаловать в C#"); // Выводим строку "Добро пожаловать в C#"
```

Для хранения данных в программе применяются переменные.

Переменная представляет именованную область памяти, в которой хранится значение определенного типа. Переменная имеет тип, имя и значение. Тип определяет, какого рода информацию может хранить переменная.

Перед использованием любую переменную надо определить. Синтаксис определения переменной выглядит следующим образом:

```
тип имя_переменной;
```

Вначале идет тип переменной, потом ее имя. В качестве имени переменной может выступать любое произвольное название, которое удовлетворяет следующим требованиям:

– *имя может содержать любые цифры, буквы и символ подчеркивания, при этом первый символ в имени должен быть буквой или символом подчеркивания*

– *в имени не должно быть знаков пунктуации и пробелов*

– *имя не может быть ключевым словом языка C#. Таких слов не так много, и при работе в Visual Studio среда разработки подсвечивает ключевые слова синим цветом.*

Хотя имя переменой может быть любым, но следует давать переменным описательные имена, которые будут говорить об их предназначении, например, как в случае с переменной *name*.

Отличительной чертой переменных является то, что в программе можно многократно менять их значение. Например, создадим небольшую программу, в которой определим переменную, поменяем ее значение и выведем его на консоль:

```
string name = "Tom";  // определяем переменную и инициализируем ее
Console.WriteLine(name);   // Tom
name = "Bob";      // меняем значение переменной
Console.WriteLine(name);   // Bob
```

Кроме того, в C# есть константы. **Константа** должна быть обязательно инициализирована при определении, и после определения значение константы не может быть изменено

Константы предназначены для описания таких значений, которые не должны изменяться в программе. Для определения констант используется ключевое слово **const**, которое указывается перед типом константы:

```
const string NAME = "Tom";
```

Типы данных

Название	Ключевое слово	Тип .NET	Диапазон	Описание	Размер, битов
Логический тип	bool	Boolean	true, false		
Целые типы	sbyte	SByte	-128..127	Со знаком	8
	byte	Byte	0..255	Без знака	8
	short	Int16	-32768..32767	Со знаком	16
	ushort	UInt16	0..65535	Без знака	16
	int	Int32	-2*10^9..2*10^9	Со знаком	32
	uint	UInt32	0..4*10^9	Без знака	32
	long	Int64	-9*10^18..9*10^18	Со знаком	64
	ulong	UInt64	0..18*10^18	Без знака	64
Символьный тип	char	Char	U+0000..U+ffff	Unicode-символ	16
Вещественные типы	float	Single	1.5*10^-45..3.4*10^38	7 цифр	32
	double	Double	5.0*10^-324..1.7*10^308	15-16 цифр	64
Финансовый тип	decimal	Decimal	1.0*10^-28..7.9*10^28	28-29 цифр	128
Строковый тип	string	String	Длина ограничена объемом доступной памяти	Строка из Unicode-символов	
Тип object	object	Object	Можно хранить все что угодно	Всеобщий порядок	

Консольный ввод-вывод

Нередко возникает необходимость вывести на консоль в одной строке значения сразу нескольких переменных. В этом случае мы можем использовать прием, который называется **интерполяцией**:

```
string name = "Tom";
int age = 34;
double height = 1.7;
Console.WriteLine($"Имя: {name}  Возраст: {age}  Рост: {height}м");
```

Для встраивания отдельных значений в выводимую на консоль строку используются фигурные скобки, в которые заключается встраиваемое значение. Это можем значение переменной (*{name}*) или более сложное выражение (например, операция сложения *{4+7}*). А перед всей строкой ставится знак доллара **$**.

При выводе на консоль вместо помещенных в фигурные скобки выражений будут выводиться их значения.

Есть другой способ вывода на консоль сразу нескольких значений:

```
string name = "Tom";
int age = 34;
double height = 1.7;
Console.WriteLine("Имя: {0}  Возраст: {2}  Рост: {1}м", name, height, age);
```

Здесь мы видим, что строка ***Console.WriteLine*** содержит некие числа в фигурных скобках: {0}, {1}, {2}. Это **плейсхолдеры**, вместо которых при выводе строки на консоль будут подставляться некоторые значения. Подставляемые значения указываются после строки через запятую.

При этом важен порядок подобных плейсхолдеров. Например, в данном случае после строки первой указана переменная name, потом height и потом age. Поэтому значение переменной name будет всавляться вместо первого плейсхолдера - {0} (нумерация начинается с нуля), height - вместо {1}, а age - вместо {2}.

Метод ***Console.ReadLine();***

Особенностью метода ***Console.ReadLine()*** является то, что он может считать информацию с консоли только в виде строки.

Однако, может возникнуть вопрос, как нам быть, если мы хотим ввести возраст в переменную типа int или другую информацию в переменные типа double или decimal? По умолчанию платформа предоставляет ряд методов, которые позволяют преобразовать различные значения к типам int, double и т.д. Некоторые из этих методов:

- ***Convert.ToInt32()*** ***(преобразует к типу int)***
- ***Convert.ToDouble()*** ***(преобразует к типу double)***
- ***Convert.ToDecimal()*** ***(преобразует к типу decimal)***

Примеры:

```
Console.Write("Введите имя: ");
string name = Console.ReadLine();

Console.Write("Введите возраст: ");
int age = Convert.ToInt32(Console.ReadLine());

Console.Write("Введите рост: ");
double height = Convert.ToDouble(Console.ReadLine());

Console.Write("Введите размер зарплаты: ");
decimal salary = Convert.ToDecimal(Console.ReadLine());

    Console.WriteLine($"Имя: {name}  Возраст: {age}  Рост: {height}м  Зарплата: {salary}$");
```

Арифметические операции

Операция	Название	Пример	Описание
+	Операция сложения	A = B + C;	Прибавить C к B и результат присвоить A.
-	Операция вычитания	A = B - C;	Вычесть C из B и результат присвоить A.
+ -	Операции смены знака	A = -A;	Поменять знак A.
*	Операция умножения	A = B * C;	Перемножить B и C и результат присвоить A.
/	Операция деления	A = B / C;	Разделить B на C и результат присвоить A.
%	Операция: остаток от деления	A = A % C;	Найти остаток от деления A на C и результат присвоить A.
++	Операция инкремента	A++;	Увеличить A на 1.
--	Операция декремента	A--;	Уменьшить A на 1.

ЗАДАНИЕ

Написать программы на языке C#. Все задачи поместить в одно решение. В каждой задаче указать условие!

Вариант 1.

1. Вычислить значения функции y = 5a – 21 при любом значении a.

2. Дан радиус окружности. Найти ее диаметр.

3. Даны два целых числа. Найти их среднее геометрическое. (Средним геометрическим называется квадратный корень из произведения этих чисел).

4. Даны катеты прямоугольного треугольника. Найти его гипотенузу.

5. Напишите программу, в которую вводится целое число, после чего на экран выводится следующее и предыдущее целое число. Например, при вводе числа 15 на экран должно быть выведено:

Следующее за числом 15 число – 16.

Для числа 15 предыдущее число – 14.

Вариант 2.

1. Вычислить значения функции y = 6x + 13 при любом значении x;

2. Дана сторона квадрата. Найти его периметр.

3. Даны два целых числа. Найти их среднее арифметическое;

4. Даны катеты прямоугольного треугольника. Найти его периметр.

5. Напишите программу, в которую вводится имя человека и выводится на экран приветствие в виде слова «Привет», после которого должна стоять запятая, введенное имя и восклицательный знак. После запятой должен стоять пробел, а перед восклицательным знаком пробела быть не должно.

ПРАКТИЧЕСКАЯ РАБОТА №7

Создание программ с условным оператором. Тернарная операция.

Цели:

– *Изучить синтаксис условного оператора;*
– *Изучить синтаксис и назначение тернарной операции;*
– *Изучить логические выражения;*
– *Изучить логические операции.*

Преобразования базовых типов данных

При рассмотрении типов данных указывалось, какие значения может иметь тот или иной тип и сколько байт памяти он может занимать. В прошлой теме были рассмотрены арифметические операции. Теперь применим операцию сложения к данным разных типов:

```
byte a = 4;
int b = a + 70;
```

Результатом операции вполне справедливо является число 74, как и ожидается.

Но теперь попробуем применить сложение к двум объектам типа ***byte***:

```
byte a = 4;
byte b = a + 70;  // ошибка
```

Здесь поменялся только тип переменной, которая получает результат сложения - с int на byte. Однако при попытке скомпилировать программу мы получим ошибку на этапе компиляции. И если мы работаем в Visual Studio, среда подчеркнет вторую строку красной волнистой линией, указывая, что в ней ошибка.

При операциях мы должны учитывать диапазон значений, которые может хранить тот или иной тип. Но в данном случае число 74, которое мы ожидаем получить, вполне укладывается в диапазон значений типа byte, тем не менее мы получаем ошибку.

Дело в том, что операция сложения (да и вычитания) возвращает значение типа int, если в операции участвуют целочисленные типы данных с разрядностью меньше или равно int (то есть типы byte, short, int). Поэтому результатом операции `a + 70` будет объект, который имеет длину в памяти 4 байта. Затем этот объект мы пытаемся присвоить переменной b, которая имеет тип byte и в памяти занимает 1 байт.

И чтобы выйти из этой ситуации, необходимо применить операцию преобразования типов. **Операция преобразования** типов предполагает указание в скобках того типа, к которому надо преобразовать значение:

```
byte a = 4;
byte b = (byte)(a + 70);
```

Условные выражения

Отдельный набор операций представляет условные выражения. Такие операции возвращают логическое значение, то есть значение типа bool: true, если выражение истинно, и false, если выражение ложно. К подобным операциям относятся операции сравнения и логические операции.

Условные выражения приведены в таблице ниже:

==	Сравнивает два операнда на равенство. Если они равны, то операция возвращает true, если не равны, то возвращается false:	int a = 10;
		int b = 4;
		bool c = a < b; // false

!=	Сравнивает два операнда и возвращает true, если операнды не равны, и false, если они равны.	int a = 10;
		int b = 4;
		bool c = a != b; // true
		bool d = a != 10; // false

<	Операция "меньше чем". Возвращает true, если первый операнд меньше второго, и false, если первый операнд больше второго:	int a = 10;
		int b = 4;
		bool c = a < 25; // true
		bool d = a < b; // false

>	Операция "больше чем". Сравнивает два операнда и возвращает true, если первый операнд больше второго, иначе возвращает false:	int a = 10;
		int b = 4;
		bool c = a > b; // true
		bool d = a > 25; // false

<=	Операция "меньше или равно". Сравнивает два операнда и возвращает true, если первый операнд меньше или равен второму. Иначе возвращает false.	int a = 10;
		int b = 4;
		bool c = a <= b; // false
		bool d = a <= 25; // true

>=	Операция "больше или равно". Сравнивает два операнда и возвращает true, если первый операнд больше или равен второму, иначе возвращается false:	int a = 10;
		int b = 4;
		bool c = a >= b; // true
		bool d = a >= 25; // false

Логические операции

|	Операция логического сложения или логическое ИЛИ. Возвращает true, если хотя бы один из операндов возвращает true.	bool x1 = (5 > 6) | (4 < 6); // 5 > 6 - false, 4 < 6 - true, поэтому возвращается true bool x2 = (5 > 6) | (4 > 6); // 5 > 6 - false, 4 > 6 - false, поэтому возвращается false
&	Операция логического умножения или логическое И. Возвращает true, если оба операнда одновременно равны true.	bool x1 = (5 > 6) & (4 < 6); // 5 > 6 - false, 4 < 6 - true, поэтому возвращается false bool x2 = (5 < 6) & (4 < 6); // 5 < 6 - true, 4 < 6 - true, поэтому возвращается true
||	Операция логического сложения. Возвращает true, если хотя бы один из операндов возвращает true.	bool x1 = (5 > 6) || (4 < 6); // 5 > 6 - false, 4 < 6 - true, поэтому возвращается true bool x2 = (5 > 6) || (4 > 6); // 5 > 6 - false, 4 > 6 - false, поэтому возвращается false

&&	Операция логического умножения. Возвращает true, если оба операнда одновременно равны true.	bool x1 = (5 > 6) && (4 < 6); // 5 > 6 - false, 4 < 6 - true, поэтому возвращается false bool x2 = (5 < 6) && (4 < 6); // 5 < 6 - true, 4 < 6 - true, поэтому возвращается true
!	Операция логического отрицания. Производится над одним операндом и возвращает true, если операнд равен false. Если операнд равен true, то операция возвращает false:	bool a = true; bool b = !a; // false
^	Операция исключающего ИЛИ. Возвращает true, если либо первый, либо второй операнд (но не одновременно) равны true, иначе возвращает false	bool x5 = (5 > 6) ^ (4 < 6); // 5 > 6 - false, 4 < 6 - true, поэтому возвращается true bool x6 = (50 > 6) ^ (4 / 2 < 3); // 50 > 6 - true, 4/2 < 3 - true, поэтому возвращается false

Здесь у нас две пары операций | и || (а также & и &&) выполняют похожие действия, однако же они не равнозначны.

В выражении z=x|y; будут вычисляться оба значения - x и y.

В выражении же z=x||y; сначала будет вычисляться значение x, и если оно равно true, то вычисление значения y уже смысла не имеет, так как у нас в любом случае уже z будет равно true. Значение y будет вычисляться только в том случае, если x равно false

То же самое касается пары операций &/&&. В выражении z=x&y; будут вычисляться оба значения - x и y.

*В выражении же z=x&&y; сначала будет вычисляться значение x, и если оно равно false, то вычисление значения **y** уже смысла не имеет, так как у нас в любом случае уже z будет равно false. Значение y будет вычисляться только в том случае, если x равно true*

Поэтому операции || и && более удобны в вычислениях, так как позволяют сократить время на вычисление значения выражения, и тем самым повышают производительность. А операции | и & больше подходят для выполнения поразрядных операций над числами.

Конструкция if..else

Условные конструкции – один из базовых компонентов многих языков программирования, которые направляют работу программы по одному из путей в зависимости от определенных условий. Одной из таких конструкций в языке программирования C# является конструкция ***if – else***

Конструкция ***if else*** проверяет истинность некоторого условия и в зависимости от результатов проверки выполняет определенный код.

Ее простейшая форма состоит из блока if:

```
If (условие)
{
   выполняемые инструкции
}
```

После ключевого слова if ставится условие. Условие должно представлять значение типа bool. Это может быть непосредственно значение типа bool или результат условного выражения или другого выражения, которое возвращает значение тиа bool. И если это условие истинно (равно true), то срабатывает код, который помещен далее после условия внутри фигурных скобок.

Например:

```
int num1 = 8;
int num2 = 6;
if (num1 > num2)
{
   Console.WriteLine($"Число {num1} больше числа {num2}");
}
```

В данном случае у нас первое число больше второго, поэтому выражение num1 > num2 истинно и возвращает true, следовательно, управление переходит к строке ***Console.WriteLine($"Число {num1} больше числа {num2}");***

Если блок if содержит одну инструкцию, то мы можем его сократить, убрав фигурные скобки.

Также мы можем соединить сразу несколько условий, используя логические операторы:

```
int num1 = 8;
int num2 = 6;
if (num1 > num2 && num1 == 8)
{
   Console.WriteLine($"Число {num1} больше числа {num2}");
}
```

Но что, если мы захотим, чтобы при несоблюдении условия также выполнялись какие-либо действия? В этом случае мы можем добавить блок ***else (иначе)***:

```
int num1 = 8;
int num2 = 6;
if (num1 > num2)
{
```

```
    Console.WriteLine($"Число {num1} больше числа {num2}");
}
else
{
    Console.WriteLine($"Число {num1} меньше числа {num2}");
}
```

Блок ***else*** выполняется, если условие после ***if*** ложно, то есть равно ***false***. Если блок ***else*** содержит только одну инструкцию, то опять же мы можем его сократить, убрав фигурные скобки.

Но в примере выше при сравнении чисел мы можем насчитать три состояния: первое число больше второго, первое число меньше второго и числа равны. Используя конструкцию ***else if***, мы можем обрабатывать дополнительные условия:

```
int num1 = 8;
int num2 = 6;
if (num1 > num2)
{
    Console.WriteLine($"Число {num1} больше числа {num2}");
}
else if (num1 < num2)
{
    Console.WriteLine($"Число {num1} меньше числа {num2}");
}
else
{
    Console.WriteLine("Число num1 равно числу num2");
}
```

При необходимости можно добавить несколько выражений ***else if***.

Тернарная операция

Тернарную операция также позволяет проверить некоторое условие и в зависимости от его истинности выполнить некоторые действия. Она имеет следующий синтаксис:

```
[первый операнд - условие] ? [второй операнд] : [третий операнд]
```

Здесь сразу три операнда. В зависимости от условия тернарная операция возвращает второй или третий операнд: если условие равно true, то возвращается второй операнд; если условие равно false, то третий. Например:

```
int x = 3;
int y = 2;
```

```
int z = x < y ? (x + y) : (x - y);
Console.WriteLine(z);  // 1
```

Здесь первый операнд (то есть условие) представляет выражение x < y. Если оно равно true, то возвращается второй операнд - (x+y), то есть результат операции сложения. Если условие равно false, то возвращается третий операнд – (x-y).

Результат тернарной операции (то есть второй или третий операнд в зависимости от условия) присваивается переменной z.

ЗАДАНИЕ

Написать программы на языке C#. Все задачи поместить в одно решение. В каждой задаче указать условие!

Вариант 1.

1. Дано натуральное число. Определить: а) является ли оно четным; б) оканчивается ли оно цифрой 7

2. Дано двузначное число. Определить: а) является ли сумма его цифр двузначным числом; б) больше ли числа «а» сумма его цифр.

3. Определить, является ли треугольник со сторонами a, b, c: а) равносторонним; б) равнобедренным

4. Даны три целых числа. Вывести на экран те из них, которые являются четными.

5. Даны три вещественных числа. Возвести в квадрат те из них, значения которых неотрицательны.

6. Даны четыре вещественных числа. Определить, сколько из них отрицательных. Оператор цикла не использовать.

7. Определить, в какую из областей – I или II (рис. 4.1) – попадает точка с заданными координатами. Для простоты принять, что точка не попадает на границу областей.

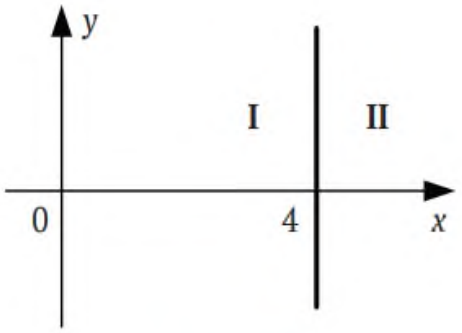


Рис. 4.1

Вариант 2.

1. Дано двузначное число. Определить: а) какая из его цифр больше: первая или вторая; б) одинаковы ли его цифры.

2. Дано двузначное число. Определить: а) кратна ли трем сумма его цифр; б) кратна ли сумма его цифр числу «а».

3. Даны два числа. Если квадратный корень из второго числа меньше первого числа, то увеличить второе число в пять раз.

4. Даны три вещественных числа. Возвести в квадрат те из них, значения которых неотрицательны.

5. Даны три вещественных числа. Вывести на экран те из них, которые принадлежат интервалу [1.6, 3.8].

6. Даны вещественные положительные числа a, b, c. Если существует треугольник со сторонами a, b, c, то определить, является ли он прямоугольным.

7. Определить, в какую из областей – I или II (рис. 4.2) – попадает точка с заданными координатами. Для простоты принять, что точка не попадает на границу областей.

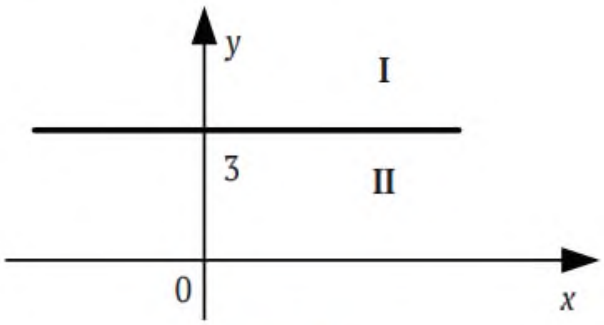


Рис. 4.2

ПРАКТИЧЕСКАЯ РАБОТА №8
Оператор выбора. Цикл со счетчиком.

Цели:

– *Изучить синтаксис и назначение оператора выбора;*

– *Изучить синтаксис и назначение цикла со счетчиком.*

Оператор выбора - Конструкция switch

Конструкция switch/case оценивает некоторое выражение и сравнивает его значение с набором значений. И при совпадении значений выполняет определенный код.

Конструкция switch имеет следующее формальное определение:

```
switch (выражение)
{
   case значение1:
      код,выполняемый если выражение имеет значение1
      break;
   case значение2:
      код,выполняемый если выражение имеет значение1
      break;
   //.............
   case значениеN:
      код, выполняемый если выражение имеет значениеN
      break;
   default:
      код, выполняемый если выражение не имеет ни одно из выше указанных
значений
      break;
}
```

После ключевого слова **switch** в скобках идет сравниваемое выражение. Значение этого выражения последовательно сравнивается со значениями, помещенными после оператора **case**. И если совпадение будет найдено, то будет выполняться определенный блок **case**.

В конце каждого блока **case** должен ставиться один из операторов перехода: **break**, **goto case**, **return** или **throw**. Как правило, используется оператор **break**. При его применении другие блоки **case** выполняться не будут.

Например:

```
string name = "Tom";

switch (name)
{
   case "Bob":
      Console.WriteLine("Ваше имя - Bob");
      break;
   case "Tom":
      Console.WriteLine("Ваше имя - Tom");
```

```
        break;
    case "Sam":
        Console.WriteLine("Ваше имя - Sam");
        break;
}
```

Если значение переменной ***name*** не совпадает ни с каким значением после операторов **case**, то ни один из блоков **case** не выполняется. Однако если даже в этом случае нам все равно надо выполнить какие-нибудь действия, то мы можем добавить в конструкцию **switch** необязательный блок **default**.

Например:

```
string name = "Alex";

switch (name)
{
    case "Bob":
        Console.WriteLine("Ваше имя - Bob");
        break;
    case "Tom":
        Console.WriteLine("Ваше имя - Tom");
        break;
    case "Sam":
        Console.WriteLine("Ваше имя - Sam");
        break;
    default:
        Console.WriteLine("Неизвестное имя");
        break;
}
```

Однако если мы хотим, чтобы, наоборот, после выполнения текущего блока **case** выполнялся другой блок **case**, то мы можем использовать вместо **break** оператор **goto case**:

```
int number = 1;
switch (number)
{
    case 1:
        Console.WriteLine("case 1");
        goto case 5; // переход к case 5
    case 3:
        Console.WriteLine("case 3");
        break;
    case 5:
        Console.WriteLine("case 5");
        break;
    default:
        Console.WriteLine("default");
```

```
    break;
}
```

Циклы являются управляющими конструкциями, позволяя в зависимости от определенных условий выполнять некоторое действие множество раз. В C# имеются следующие виды циклов:

- ***for***
- ***foreach***
- ***while…do***
- ***do...while***

Цикл For

Объявление цикла **for** состоит из трех частей. Первая часть объявления цикла – некоторые действия, которые выполняются один раз до выполнения цикла. Обычно здесь определяются переменные, которые будут использоваться в цикле.

Вторая часть – условие, при котором будет выполняться цикл. Пока условие равно true, будет выполняться цикл.

И третья часть – некоторые действия, которые выполняются после завершения блока цикла. Эти действия выполняются каждый раз при завершении блока цикла.

После объявления цикла в фигурных скобках помещаются сами действия цикла.

Рассмотрим стандартный цикл **for**:

```
for (int i = 1; i < 4; i++)
{
  Console.WriteLine(i);
}
```

Здесь первая часть объявления цикла - ***int i = 1*** - создает и инициализирует переменную i.

Вторая часть - условие ***i < 4***. То есть пока переменная i меньше 4, будет выполняться цикл.

И третья часть - действия, выполняемые после завершения действий из блока цикла - увеличение переменной i на единицу.

Весь процесс цикла можно представить следующим образом:

1. Определяется переменная int i = 1

2. Проверяется условие i < 4. Оно истинно (так как 1 меньше 4), поэтому выполняется блок цикла, а именно инструкция Console.WriteLine(i), которая выводит на консоль значение переменной i

3. Блок цикла закончил выполнение, поэтому выполняется треться часть объявления цикла - i++. После этого переменная i будет равна 2.

4. Снова проверяется условие i < 4. Оно истинно (так как 2 меньше 4), поэтому опять выполняется блок цикла - Console.WriteLine(i)

5. Блок цикла закончил выполнение, поэтому снова выполняется выражение i++. После этого переменная i будет равна 3.

6. Снова проверяется условие i < 4. Оно истинно (так как 3 меньше 4), поэтому опять выполняется блок цикла - Console.WriteLine(i)

7. Блок цикла закончил выполнение, поэтому снова выполняется выражение i++. После этого переменная i будет равна 4.

8. Снова проверяется условие i < 4. Теперь оно возвражает false, так как значение переменной i НЕ меньше 4, поэтому цикл завершает выполнение. Далее уже выполняется остальная часть программы, которая идет после цикла

В итоге блок цикла сработает 3 раза, пока значение i не станет равным 4. И каждый раз это значение будет увеличиваться на 1. Однократное выполнение блока цикла называется **итерацией**. Таким образом, здесь цикл выполнит три итерации.

Если блок цикла **for** содержит одну инструкцию, то мы можем его сократить, убрав фигурные скобки.

При этом необязательно именно в первой части цикла объявлять переменную, а в третий части изменять ее значение – это могут быть любые действия. Например:

```
var i = 1;

for (Console.WriteLine("Начало выполнения цикла"); i < 4; Console.WriteLine($"i = {i}"))
{
   i++;
}
```

ЗАДАНИЕ

Написать программы на языке C#. Все задачи поместить в одно решение. В каждой задаче указать условие!

Вариант 1.

1. Одна штука некоторого товара стоит 20,4 руб. Напечатать таблицу стоимости 2, 3, ..., 20 штук этого товара.

2. Напечатать таблицу умножения на 7.

3. Найти

а) произведение всех целых чисел от 7 до 14;

б) произведение всех целых чисел от a до 15 (значение a вводится с клавиатуры; $1 \le a \le 15$);

4. Последовательность Фибоначчи образуется так: первый и второй члены последовательности равны 1, каждый следующий равен сумме двух предыдущих (1, 1, 2, 3, 5, 8, 13, ...). Дано натуральное число n ($n \ge 3$). Найти n-й член последовательности Фибоначчи.

5. Составить программу для расчета факториала натурального числа n (факториал числа n равен $1 \cdot 2 \cdot \ldots \cdot n$).

6. Вывести на экран все нечетные двузначные числа, у которых последняя цифра равна 3 или 7.

Вариант 2.

1. Напечатать таблицу перевода расстояний в дюймах в сантиметры для значений 10, 11, ..., 22 дюйма (1 дюйм = 25,4 мм).

2. Найти:

а) произведение всех целых чисел от 1 до b (значение b вводится с клавиатуры; $1 \le b \le 10$);

б) произведение всех целых чисел от a до b (значения a и b вводятся с клавиатуры; $b \ge a$).

3. Последовательность Фибоначчи образуется так: первый и второй члены последовательности равны 1, каждый следующий равен сумме двух предыдущих (1, 1, 2, 3, 5, 8, 13, ...). Дано натуральное число n ($n \ge 3$). а) Найти n-й член последовательности Фибоначчи. б) Получить первые n членов последовательности Фибоначчи.

4. Вывести на экран все нечетные двузначные числа, у которых последняя цифра равна 3 или 7

5. Дано натуральное число. Выяснить, является ли оно простым (простым называется натуральное число, большее 1, не имеющее других делителей, кроме единицы и самого себя).

6. Написать программу возведения числа «a» в степень «b» без использования функции возведения в степень.

ПРАКТИЧЕСКАЯ РАБОТА №9
Циклы с предусловием и постусловием. Класс Math.

Цели:

– *Изучить синтаксис и назначение цикла с предусловием;*
– *Изучить синтаксис и назначение цикла с постусловием;*
– *Изучить класс Math;*
– *Изучить операторы continue и break.*

Математические вычисления и класс Math

Для выполнения различных математических операций в библиотеке классов .NET предназначен класс **Math**. Он является статическим, поэтому все его методы также являются статическими.

Рассмотрим основные методы класса **Math**:

`Abs(double value):` возвращает абсолютное значение для аргумента value
`Acos(double value)`: возвращает арккосинус value. Параметр value должен иметь значение от -1 до 1
`Asin(double value)`: возвращает арксинус value. Параметр value должен иметь значение от -1 до 1
`Atan(double value)`: возвращает арктангенс value
`BigMul(int x, int y)`: возвращает произведение x * y в виде объекта long
`Ceiling(double value):` возвращает наименьшее целое число с плавающей точкой, которое не меньше value
`Cos(double d)`: возвращает косинус угла d
`Cosh(double d)`: возвращает гиперболический косинус угла d
`DivRem(int a, int b, out int result):` возвращает результат от деления a/b, а остаток помещается в параметр result
`Exp(double d):` возвращает основание натурального логарифма, возведенное в степень d
`Floor(decimal d)`: возвращает наибольшее целое число, которое не больше d
`IEEERemainder``(double a, double b)`: возвращает остаток от деления a на b
`Log(double d)`: возвращает натуральный логарифм числа d
`Log(double a, double newBase)`: возвращает логарифм числа a по основанию newBase
`Log10(double d)`: возвращает десятичный логарифм числа d
`Max(double a, double b):` возвращает максимальное число из a и b
`Min(double a, double b):` возвращает минимальное число из a и b
`Pow(double a, double b):` возвращает число a, возведенное в степень b
`Round(double d):` возвращает число d, округленное до ближайшего целого числа
`Round(double a, int b):` возвращает число a, округленное до определенного количества знаков после запятой, представленного параметром b
`Sign(double value)`: возвращает число 1, если число value положительное, и -1, если значение value отрицательное. Если value равно 0, то возвращает 0
`Sin(double value)`: возвращает синус угла value
`Sqrt(double value):` возвращает квадратный корень числа value

`Tanh(double value)`: возвращает гиперболический тангенс угла value
`Truncate(double value):` отбрасывает дробную часть числа value, возвращая лишь целое значние

Цикл do … While

В цикле **do** сначала выполняется код цикла, а потом происходит проверка условия в инструкции **while**. И пока это условие истинно, цикл повторяется.

```
do
{
   действия цикла
}
while  (условие)
```

Например:

```
int i = 6;
do
{
   Console.WriteLine(i);
   i--;
}
while (i > 0);
```

Здесь код цикла сработает 6 раз, пока ***i*** не станет равным нулю. Но важно отметить, что цикл ***do*** гарантирует хотя бы единократное выполнение действий, даже если условие в инструкции ***while*** не будет истинно. То есть мы можем написать:

```
int i = -1;
do
{
   Console.WriteLine(i);
   i--;
}
while (i > 0);
```

Хотя у нас переменная ***i*** меньше 0, цикл все равно один раз выполнится.

Цикл while

В отличие от цикла **do** цикл **while** сразу проверяет истинность некоторого условия, и если условие истинно, то код цикла выполняется:

```
int i = 6;
while (i > 0)
{
   Console.WriteLine(i);
   i--;
}
```

Операторы continue и break

Иногда возникает ситуация, когда требуется выйти из цикла, не дожидаясь его завершения. В этом случае мы можем воспользоваться оператором ***break***.

Например:

```
for (int i = 0; i < 9; i++)
{
   if (i == 5)
      break;
   Console.WriteLine(i);
}
```

Хотя в условии цикла сказано, что цикл будет выполняться, пока счетчик ***i*** не достигнет значения 9, в реальности цикл сработает 5 раз. Так как при достижении счетчиком ***i*** значения 5, сработает оператор ***break***, и цикл завершится.

Теперь поставим себе другую задачу. А что, если мы хотим, чтобы при проверке цикл не завершался, а просто пропускал текущую итерацию. Для этого мы можем воспользоваться оператором ***continue***:

```
for (int i = 0; i < 9; i++)
{
   if (i == 5)
      continue;
   Console.WriteLine(i);
}
```

В этом случае цикл, когда дойдет до числа 5, которое не удовлетворяет условию проверки, просто пропустит это число и перейдет к следующей итерации.

Стоит отметить, что операторы **break** и **continue** можно применять **в любом типе циклов.**

Вложенные циклы

Одни циклы могут быть вложенными в другие. Например:

```
for (int i = 1; i < 10; i++)
{
   for (int j = 1; j < 10; j++)
   {
      Console.Write($"{i * j} \t");
   }
   Console.WriteLine();
}
```

В данном случае цикл ***for (int i = 1; i < 10; i++)*** выполняется 9 раз, то есть имеет 9 итераций. Но в рамках каждой итерации выполняется девять раз вложенный цикл ***for (int j = 1; j < 10; j++)***. В итоге данная программа выведет таблицу умножения. Вложенный циклы используются, например, для обработки двумерных массивов.

ЗАДАНИЕ

Написать программы на языке C#. Все задачи поместить в одно решение. В каждой задаче указать условие!

Вариант 1.

1. Даны натуральные числа a и b (a > b). Определить результат целочисленного деления a на b, не используя стандартную операцию целочисленного деления.

2. Вывести на экран натуральные числа, не превышающие заданное число n. Оператор цикла с параметром не использовать.

3. Напечатать минимальное число, большее 190, которое нацело делится на 17.

4. Напечатать те натуральные числа, квадрат которых не превышает заданное число n

5. Разработать фрагмент программы, в котором пользователь должен ввести четное число. В случае ввода нечетного числа на экран должно выводиться сообщение об ошибке, после чего действия должны повторяться до ввода правильного значения.

6. Подготовьте фрагмент программы, в котором должны вводиться 10 чисел. Если будет введено число 0, ввод должен прекратиться.

7. Дано натуральное число. Определить:

а) количество цифр 3 в нем;

б) сколько раз в нем встречается последняя цифра;

в) количество четных цифр в нем.

Вариант 2.

1. Даны натуральные числа a и b (a > b). Определить остаток от деления a на b, не используя стандартную операцию вычисления остатка.

2. Напечатать все нечетные числа из интервала [10, 100]. Оператор цикла с параметром не использовать.

3. Найти максимальное из натуральных чисел, не превышающих 5000, которое нацело делится на 139.

4. Напечатать те натуральные числа, квадрат которых не превышает заданное число n

5. Подготовить фрагмент программы, в которой пользователь должен ввести установленный пароль в виде целого числа. В случае ввода неправильного пароля на экран должно выводиться сообщение об ошибке, после чего действия должны повторяться до ввода правильного значения. После этого на экран должно выводиться некоторое приветствие.

6. Напечатать числа 1.0, 1.5, 2.0, …, 13.5. Оператор цикла с параметром не использовать.

7. Дано натуральное число. Определить:

а) сумму его цифр, больших пяти;

б) произведение его цифр, больших семи;

в) сколько раз в нем встречаются цифры 0 и 5 (всего).

ПРАКТИЧЕСКАЯ РАБОТА №10
Решение задач.

Цели:

– *Закрепить полученные ранее навыки программирования на языке C#.*

Известны оценки по физике каждого из 20 учеников класса. Определить среднюю оценку.

Дано натуральное число.

1) Определить его максимальную цифру.
2) Определить его минимальную цифру.
3) Определить, на сколько его максимальная цифра превышает минимальную.
4) Найти сумму его максимальной и минимальной цифр.

Дано натуральное число. Определить:

1) Количество цифр 3 в нем
2) Сколько раз в нем встречается последняя цифра
3) Количество четных цифр в нем
4) Сумму его цифр, больших пяти
5) Произведение его цифр, больших семи

Дано натуральное число. Определить:

1) Сколько раз в нем встречается цифра A
2) Количество его цифр, кратных z (значение z вводится с клавиатуры; z= 2, 3, 4)
3) Сумму его цифр, больших a (значение a вводится с клавиатуры)
4) Сколько раз в нем встречаются цифры x и y

ПРАКТИЧЕСКАЯ РАБОТА №11
Массивы.

Цели:

- *Изучить способы определения одномерных и двумерных массивов;*
- *Изучить синтаксис и назначение цикла foreach;*
- *Изучить генератор случайных чисел Random;*
- *Изучить некоторые методы обработки массивов.*

ЧАСТЬ 1
ОДНОМЕРНЫЕ МАССИВЫ

Цикл foreach

Цикл ***foreach*** предназначен для перебора набора или коллекции элементов. Его общее определение:

```
foreach(тип_данных переменная in  коллекция)
{
   // действия цикла
}
```

После оператора ***foreach*** в скобках сначала идет определение переменной. Затем ключевое слово ***in*** и далее коллекция, элементы которой надо перебрать.

При выполнении цикл последовательно перебирает элементы коллекции и помещает их в переменную, и таким образом в блоке цикла мы можем выполнить с ними некоторые действия.

Например, возьмем строку. Строка по сути – это коллекция символов. И .NET позволяет перебрать все элементы строки - ее символы с помощью цикла ***foreach***.

```
foreach (char c in "Tom")
{
   Console.WriteLine(c);
}
```

Здесь цикл ***foreach*** пробегается по всем символам строки "Tom" и каждый символ помещает в символьную переменную ***c***. В блоке цикла значение переменной ***c*** выводится на консоль. Поскольку в строке "Tom" три символа, то цикл выполнится три раза.

Стоит отметить, что определяемая в объявлении цикла переменная должна по типу соответствовать типу элементов перебираемой коллекции. Так, элементы строки - значения типа ***char*** - символы. Поэтому переменная **c** имеет тип ***char***.

Генератор случайных чисел RANDOM

Для генерации случайных чисел в программах, написанных на C#, предназначен класс ***Random***.

Синтаксис использования весьма прост:

```
//Создание объекта для генерации чисел
Random rnd = new Random();

//Получить очередное (в данном случае - первое) случайное число
int value = rnd.Next();

//Вывод полученного числа в консоль
Console.WriteLine(value);
```

Красным цветом выделены названия переменных (*они могут быть любыми!*), голубым – название метода для генерации нового числа.

А теперь, давайте усложним задачу. Представим, что нам нужно получить случайное число в определенном диапазоне. В данном случае все также выполняется довольно просто:

```
//Создание объекта для генерации чисел
Random rnd = new Random();

//Получить случайное число (в диапазоне от 0 до 10)
int value = rnd.Next(0, 10);

//Вывод числа в консоль
Console.WriteLine(value);
```

Чтобы задать диапазон генерируемых случайных чисел, необходимо в параметрах метода ***Next*** указать через запятую левую и правую границу интервала.

Но это ещё не всё, есть еще один важный нюанс, на самом деле генератор случайных чисел является генератором псевдослучайных чисел, т.е. числа, которые он возвращает, не являются чисто случайными, они «вырабатываются» по определенным и четким законам, а «случайность» зависит от инициализации объекта, который генерирует числа. В примерах, приведенных выше, мы использовали конструктор по умолчанию, и в таком случае «начальное значение» задается системой, и на основании системного времени. Но мы может задавать это самое «начальное значение» и сами:

```
//Создание объекта для генерации чисел (с указанием начального значения)
Random rnd = new Random(245);
//Получить случайное число
int value = rnd.Next();
```

Обратите внимание, начальное значение используется для генерации чисел, а не для возвращения первого числа, т.е. первое сгенерированное число не будет тем, что указано при создании объекта.

Одномерные массивы

Массив представляет набор однотипных данных. Объявление массива похоже на объявление переменной за тем исключением, что после указания типа ставятся квадратные скобки:

```
тип_переменной[] название_массива;
```

Например, определим массив целых чисел:

```
int[] numbers;
```

После определения переменной массива мы можем присвоить ей определенное значение:

```
int[] nums = new int[4];
```

Здесь вначале мы объявили массив ***nums***, который будет хранить данные типа ***int***. Далее используя операцию new, мы выделили память для 4 элементов массива: ***new int[4].*** Число 4 еще называется длиной массива. При таком определении все элементы получают значение по умолчанию, которое предусмотрено для их типа. Для типа ***int*** значение по умолчанию - 0. Также мы сразу можем указать значения для этих элементов:

```
int[] nums2 = new int[4] { 1, 2, 3, 5 };

int[] nums3 = new int[] { 1, 2, 3, 5 };

int[] nums4 = new[] { 1, 2, 3, 5 };

int[] nums5 = { 1, 2, 3, 5 };
```

Все перечисленные выше способы будут равноценны.

Подобным образом можно определять массивы и других типов, например, массив значений типа ***string***:

```
string[] people = { "Tom", "Sam", "Bob" };
```

Индексы и получение элементов массива

Для обращения к элементам массива используются индексы. Индекс представляет номер элемента в массиве (*указывается в квадратных скобках*),

при этом ***нумерация начинается с нуля***, поэтому индекс первого элемента будет равен 0, индекс четвертого элемента - 3.

Используя индексы, мы можем получить элементы массива:

```
int[] numbers = { 1, 2, 3, 5 };

// получение элемента массива
Console.WriteLine(numbers[3]);  // 5

// получение элемента массива в переменную
var n = numbers[1];    // 2
Console.WriteLine(n);  // 2
```

Также мы можем изменить элемент массива по индексу:

```
int[] numbers = { 1, 2, 3, 5 };

// изменим второй элемент массива
numbers[1] = 505;

Console.WriteLine(numbers[1]);  // 505
```

И так как у нас массив определен только для 4 элементов, то мы не можем обратиться, например, к шестому элементу. Если мы попытаемся так сделать, то мы получим ошибку во время выполнения.

Свойство Length и длина массива

Каждый массив имеет свойство ***Length***, которое хранит длину массива. Например, получим длину выше созданного массива ***numbers***:

```
int[] numbers = { 1, 2, 3, 5 };

Console.WriteLine(numbers.Length);  // 4
```

Для получения длины массива после названия массива через точку указывается свойство *Length:* ***numbers.Length***.

Получение элементов с конца массива

Благодаря наличию свойства ***Length***, мы можем вычислить индекс последнего элемента массива – это *длина массива - 1*. Например, если длина массива - 4 (то есть массив имеет 4 элемента), то индекс последнего элемента будет равен 3. И, используя свойство Length, мы можем легко получить элементы с конца массива:

```
int[] numbers = { 1, 2, 3, 5 };
```

```
Console.WriteLine(numbers[numbers.Length - 1]);  // 5 - первый с конца или последний элемент
Console.WriteLine(numbers[numbers.Length - 2]);  // 3 - второй с конца или предпоследний элемент
Console.WriteLine(numbers[numbers.Length - 3]);  // 2 - третий элемент с конца
```

Однако при подобном подходе выражения типа ***numbers.Length - 1***, смысл которых состоит в том, чтобы получить какой-то определенный элемент с конца массива, утяжеляют код. И, начиная, с версии C# 8.0 в язык был добавлен специальный оператор ^, с помощью которого можно задать индекс относительно конца коллекции.

Перепишем предыдущий пример, применяя оператор ^:

```
int[] numbers = { 1, 2, 3, 5 };

Console.WriteLine(numbers[^1]);  // 5 - первый с конца или последний элемент
Console.WriteLine(numbers[^2]);  // 3 - второй с конца или предпоследний элемент
Console.WriteLine(numbers[^3]);  // 2 - третий элемент с конца
```

Перебор массивов

Для перебора массивов мы можем использовать различные типы циклов. Например, цикл foreach:

```
int[] numbers = { 1, 2, 3, 4, 5 };
foreach (int i in numbers)
{
   Console.WriteLine(i);
}
```

Здесь в качестве контейнера выступает массив данных типа ***int***. Поэтому мы объявляем переменную с типом ***int***

Подобные действия мы можем сделать и с помощью цикла ***for***:

```
int[] numbers = { 1, 2, 3, 4, 5 };
for (int i = 0; i < numbers.Length; i++)
{
   Console.WriteLine(numbers[i]);
}
```

В то же время цикл ***for*** более гибкий по сравнению с ***foreach***. Если ***foreach*** последовательно извлекает элементы контейнера и только для чтения, то в цикле ***for*** мы можем перескакивать на несколько элементов вперед в зависимости от приращения счетчика, а также можем изменять элементы:

```
int[] numbers = { 1, 2, 3, 4, 5 };
for (int i = 0; i < numbers.Length; i++)
{
```

```
    numbers[i] = numbers[i] * 2;
    Console.WriteLine(numbers[i]);
}
```

Также можно использовать и другие виды циклов, например, ***while***:

```
int[] numbers = { 1, 2, 3, 4, 5 };
int i = 0;
while (i < numbers.Length)
{
    Console.WriteLine(numbers[i]);
    i++;
}
```

ЗАДАНИЕ

Написать программы на языке C#. Все задачи поместить в одно решение. В каждой задаче указать условие!

Вариант 1.

1. Массив предназначен для хранения значений ростов двенадцати человек. С помощью датчика случайных чисел заполнить массив целыми значениями, лежащими в диапазоне от 163 до 190 включительно.
2. Заполнить массив десятью первыми членами арифметической прогрессии с известным первым членом прогрессии A и ее разностью P;
3. Заполнить массив двадцатью первыми натуральными числами, делящимися нацело на 13 или на 17 и находящимися в интервале, левая граница которого равна 300;
4. Найти минимальный элемент массива.
5. Дан массив вещественных чисел, из всех положительных элементов вычесть элемент с номером k[1], из всех отрицательных – число n. Нулевые элементы оставить без изменения.
6. Дан массив целых чисел. Все элементы, оканчивающиеся цифрой 4, уменьшить вдвое.
7. Найти:

а) сумму всех элементов массива;

б) произведение всех элементов массива;

Вариант 2.

1. Массив предназначен для хранения значений весов двадцати человек. С помощью датчика случайных чисел заполнить массив целыми значениями, лежащими в диапазоне от 50 до 100 включительно.
2. Вывести элементы массива на экран в обратном порядке.
3. Заполнить массив двадцатью первыми членами геометрической прогрессии с известным первым членом прогрессии A и ее знаменателем Z.
4. Найти максимальный элемент массива.
5. Дан массив вещественных чисел, ко всем нулевым элементам прибавить n, из всех положительных элементов вычесть A, ко всем отрицательным прибавить B.
6. Дан массив целых чисел. Все четные элементы заменить на их квадраты, а нечетные удвоить.
7. Найти:

а) сумму квадратов всех элементов массива;

б) среднее арифметическое всех элементов массива;

ЧАСТЬ 2
ДВУМЕРНЫЕ МАССИВЫ

Многомерным называется такой массив, который содержит два или более измерения, причем доступ к каждому элементу такого массива осуществляется с помощью определенной комбинации двух или более индексов. Многомерный массив индексируется двумя и более целыми числами.

Двумерные массивы

Простейшей формой многомерного массива является двумерный массив. Местоположение любого элемента в двумерном массиве обозначается двумя индексами. Такой массив можно представить в виде таблицы, на строки которой указывает один индекс, а на столбцы – другой.

Все возможные способы объявления двумерного массива:

```
int[,] nums1;
int[,] nums2 = new int[2, 3];
int[,] nums3 = new int[2, 3] { { 0, 1, 2 }, { 3, 4, 5 } };
int[,] nums4 = new int[,] { { 0, 1, 2 }, { 3, 4, 5 } };
int[,] nums5 = new[,] { { 0, 1, 2 }, { 3, 4, 5 } };
int[,] nums6 = { { 0, 1, 2 }, { 3, 4, 5 } };
```

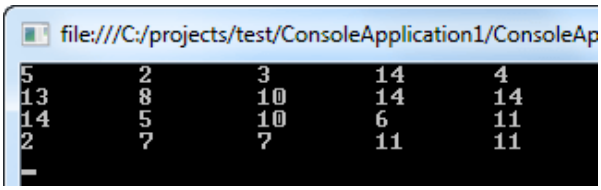
Пример объявления и инициализации двумерного массива показан ниже:

```
// Объявляем двумерный массив
int[,] myArr = new int[4, 5];

Random ran = new Random();

// Инициализируем данный массив
for (int i = 0; i < 4; i++)
{
   for (int j = 0; j < 5; j++)
   {
      myArr[i, j] = ran.Next(1, 15);
      Console.Write("{0}\t", myArr[i, j]);
   }
   Console.WriteLine();
}
```

В результате выполнения данного кода мы получили:



Схематическое представление массива ***myArr*** показано ниже:

	0	1	2	3	4
0	5	2	3	14	4
1	13	8	**10**	14	14
2	14	5	10	6	11
3	2	7	7	11	11

10 – myArr[1,2]

Разберем пример. Необходимо посчитать сумму элементов двумерного массива.

Шаг 1. Необходимо объявить и инициализировать массив;

Шаг 2. Вывод массива;

Шаг 3. Подсчет суммы элементов и вывод результата.

```
//объявление и заполнение массива
int[,] aray = new int[4, 4];
Random r = new Random();
for (int i = 0; i < 4; i++)
   for (int j = 0; j < 4; j++)
      aray[i, j] = r.Next(1, 9);
//Вывод массива
for (int i = 0; i < 4; i++)
{
   for (int j = 0; j < 4; j++)
      Console.Write(aray[i, j] + "\t");
   Console.WriteLine();
}
//подсчет суммы
int sum = 0;
for (int i = 0; i < 4; i++)
{
   for (int j = 0; j < 4; j++)
      sum += aray[i, j];
}
Console.WriteLine("Сумма элементов = {0}",sum);
Console.ReadLine();
```

Стоит сказать, что данный код неоправданно длинный для такой задачи и его можно сократить:

```
int[,] aray = new int[4, 4];
Random r = new Random();
int sum = 0;
for (int i = 0; i < 4; i++)
{
```

```
   for (int j = 0; j < 4; j++)
   {
      aray[i, j] = r.Next(1, 9);
      sum += aray[i, j];
      Console.Write(aray[i, j] + "\t");
   }
   Console.WriteLine();
}
Console.WriteLine("Сумма элементов = {0}", sum);
Console.ReadLine();
```

Пример 2. Посчитать сумму элементов для каждой строки.

Задача очень похожа, разница в том, что после выполнения всех итераций внутреннего цикла необходимо выводить сумму, а затем обнулять переменную, хранящую эту сумму:

```
int[,] aray = new int[4, 4];
Random r = new Random();
int sum = 0;
for (int i = 0; i < 4; i++)
{
   for (int j = 0; j < 4; j++)
   {
      aray[i, j] = r.Next(1, 9);
      Console.Write(aray[i, j] + "\t");
   }
   Console.WriteLine();
}

for (int i = 0; i < 4; i++)
{
   sum = 0;
   for (int j = 0; j < 4; j++)
      sum+=aray[i, j];
   Console.WriteLine("Сумма элементов {0}-й строки = {1}",i+1,sum);
}
Console.ReadLine();
```

В данном случае вывод будет выглядеть следующим образом:

```
6       1       6       2
2       2       4       1
5       7       7       4
5       7       3       2
Сумма элементов 1-й строки = 15
Сумма элементов 2-й строки = 9
Сумма элементов 3-й строки = 23
Сумма элементов 4-й строки = 17
```

Можно также изменить вывод и сократить код:

```
int[,] aray = new int[4, 4];
Random r = new Random();
int sum = 0;
for (int i = 0; i < 4; i++)
{
   sum = 0;
   for (int j = 0; j < 4; j++)
   {
      aray[i, j] = r.Next(1, 9);
      Console.Write(aray[i, j] + "\t");
      sum+=aray[i, j];
   }
   Console.Write("|Сумма = {0}", sum);
   Console.WriteLine();
}
Console.ReadLine();
```

Результат выполнения кода можете проверить сами.

ЗАДАНИЕ

Написать программы на языке C#. Все задачи поместить в одно решение. В каждой задаче указать условие!

Вариант 1.

1. Найти максимальный элемент матрицы.
2. Найти сумму столбцов матрицы.
3. Найти сумму элементов побочной диагонали квадратной матрицы.
4. Найти количество положительных четных элементов произвольной матрицы.
5. Подсчитать количество строк, содержащих отрицательные элементы.
6. Найти максимальные элементы во всех столбцах матрицы.
7. Заполнить квадратную матрицы значениями, введенными с клавиатуры. Найти номер первой строки, содержащей максимальный элемент в матрице.

Вариант 2.

1. Найти минимальный элемент матрицы.
2. Заполнить матрицу элементами от -10 до +10. Найти сумму модулей элементов для каждого столбца матрицы.
3. Найти сумму элементов главной диагонали квадратной матрицы.
4. Найти количество отрицательных нечетных элементов произвольной матрицы.

5. Подсчитать количество столбцов, содержащих отрицательные элементы.

6. Найти минимальные элементы во всех строках матрицы.

7. Заполнить квадратную матрицу значениями, введенными с клавиатуры. Найти номер последнего столбца, содержащего минимальный элемент матрицы.

ПРАКТИЧЕСКАЯ РАБОТА №12
Процедуры и функции.

Цели:

– *Изучить синтаксис и назначение процедур;*
– *Изучить синтаксис и назначение функций;*
– *Изучить виды параметров.*

ЧАСТЬ 1
МЕТОДЫ (ПРОЦЕДУРЫ)

Если переменные хранят некоторые значения, то методы содержат в себе набор инструкций, которые выполняют определенные действия. По сути метод – это именованный блок кода, который выполняет некоторые действия.

Общее определение методов выглядит следующим образом:

```
[модификаторы] тип_возвращаемого_значения название_метода ([параметры])
{
  // тело метода
}
```

Модификаторы и параметры необязательны.

Ранее мы уже использовали как минимум один метод – ***Console.WriteLine()***, который выводит информацию на консоль. Теперь рассмотрим, как мы можем создавать свои методы.

Определим один метод:

```
void SayHello()
{
  Console.WriteLine("Hello");
}
```

Здесь определен метод ***SayHello***, который выводит некоторое сообщение. К названиям методов предъявляются те же требования, что и к названиям переменных. Однако, как правило, названия методов начинаются с большой буквы.

Перед названием метода идет возвращаемый тип данных. Здесь это тип ***void***, который указывает, что фактически метод ничего не возвращает, он просто производит некоторые действия *(является процедурой).*

После названия метода в скобках идет перечисление параметров. Но в данном случае скобки пустые, что означает, что метод не принимает никаких параметров.

После списка параметров, в фигурных скобках идет блок кода, который представляет набор выполняемых методом инструкций. В данном случае блок метода ***SayHello*** содержит только одну инструкцию, которая выводит строку на консоль.

Но если мы запустим данный проект, то мы не увидим никакой строки, которую должен выводить метод ***SayHello***. Потому что после определения метод еще нужно вызвать, чтобы он выполнил свою работу.

Вызов методов

Чтобы использовать метод ***SayHello***, нам надо его вызвать. Для вызова метода указывается его имя, после которого в скобках идут значения для его параметров (если метод принимает параметры).

```
void SayHello()
{
   Console.WriteLine("Hello");
}

SayHello(); // Hello
SayHello(); // Hello
```

Преимуществом методов является то, что их можно повторно и многократно вызывать в различных частях программы. Например, в примере выше два раза вызывается метод ***SayHello***.

При этом в данном случае нет разницы, сначала определяется метод, а потом вызывается или наоборот.

Определим и вызовем еще несколько методов:

```
void SayHelloRu()
{
   Console.WriteLine("Привет");
}
void SayHelloEn()
{
   Console.WriteLine("Hello");
}
void SayHelloFr()
{
   Console.WriteLine("Salut");
}
//Основная программа
string language = "en";
switch (language)
{
   case "en":
      SayHelloEn();
      break;
   case "ru":
      SayHelloRu();
      break;
   case "fr":
      SayHelloFr();
```

```
    break;
}
```

Здесь определены три метода ***SayHelloRu(), SayHelloEn() и SayHelloFr()***, которые также имеют тип ***void***, не принимают никаких параметров и также выводит некоторую строку на консоль. Условно говоря, они выводят приветствие на определенном языке.

В конструкции ***switch*** проверяется значение переменной ***language***, которая условно хранит код языка, и в зависимости от ее значения вызывается определенный метод.

Параметры методов.

Параметры позволяют передать в метод некоторые входные данные. Параметры определяются через запятую в скобках после названия метода. Определение параметра состоит из двух частей: сначала идет тип параметра и затем его имя.

Например, определим метод PrintMessage, который получает извне выводимое сообщение:

```
void PrintMessage(string message)
{
   Console.WriteLine(message);
}

PrintMessage("Hello work");         // Hello work
PrintMessage("Hello METANIT.COM");   // Hello METANIT.COM
PrintMessage("Hello C#");           // Hello C#
```

Здесь метод ***PrintMessage***() принимает один параметр, который называется ***message*** и имеет тип ***string***.

Чтобы выполнить метод, который имеет параметры, при вызове после имени метода в скобках ему передаются значения для его параметров, например:

```
PrintMessage("Hello work");
```

Здесь параметру ***message*** передается строка "Hello work". Значения, которые передаются параметрам, еще называются *аргументами*. То есть передаваемая строка "Hello work" в данном случае является аргументом.

Иногда можно встретить такие определения как формальные параметры и фактические параметры. Формальные параметры – это собственно параметры метода (в данном случае ***message***), а фактические параметры - значения, которые передаются формальным параметрам. То есть фактические параметры – это и есть аргументы метода.

Определим еще один метод, который складывает два числа:

```
void Sum(int x, int y)
{
   int result = x + y;
   Console.WriteLine($"{x} + {y} = {result}");
}

Sum(10, 15);    // 10 + 15 = 25
```

Метод ***Sum*** имеет два параметра: ***x*** и ***y.*** Оба параметра представляют тип ***int***. Поэтому при вызове данного метода нам обязательно нужно передать на место этих параметров два числа. Внутри метода вычисляется сумма переданных чисел и выводится на консоль.

При вызове метода ***Sum*** значения передаются параметрам по позиции. Например, в вызове ***Sum(10, 15)*** число 10 передается параметру x, а число 15 – параметру y.

Необязательные параметры

По умолчанию при вызове метода необходимо предоставить значения для всех его параметров. Но C# также позволяет использовать *необязательные параметры*. Для таких параметров нам необходимо объявить значение по умолчанию. ***Также следует учитывать, что после необязательных параметров все последующие параметры также должны быть необязательными***:

```
void PrintPerson(string name, int age = 1, string company = "Undefined")
{
   Console.WriteLine($"Name: {name}  Age: {age}  Company: {company}");
}
```

Здесь параметры ***age*** и ***company*** являются необязательными, так как им присвоены значения. Поэтому при вызове метода мы можем не передавать для них данные:

```
void PrintPerson(string name, int age = 1, string company = "Undefined")
{
   Console.WriteLine($"Name: {name}  Age: {age}  Company: {company}");
}

PrintPerson("Tom", 37, "Microsoft");  // Name: Tom  Age: 37  Company: Microsoft
PrintPerson("Tom", 37); // Name: Tom  Age: 37  Company: Undefined
PrintPerson("Tom"); // Name: Tom  Age: 1   Company: Undefined
```

Именованные параметры

В предыдущих примерах при вызове методов значения для параметров передавались в порядке объявления этих параметров в методе. То есть аргументы передавались параметрам по позиции. Но мы можем нарушить подобный порядок, используя именованные параметры.

Для передачи значений параметрам по имени при вызове метода указывается имя параметра и через двоеточие его значение: name:"Tom"

```
void PrintPerson(string name, int age = 1, string company = "Undefined")
{
   Console.WriteLine($"Name:{name} Age: {age} Company: {company}");
}

PrintPerson("Tom", company: "Microsoft", age: 37);  // Name: Tom  Age: 37  Company: Microsoft
PrintPerson(age: 41, name: "Bob");// Name: Bob  Age: 41  Company: Undefined
PrintPerson(company: "Google", name: "Sam"); // Name: Sam  Age: 1   Company: Google
```

ЗАДАНИЕ

Написать программы на языке C#. Все задачи поместить в одно решение. В каждой задаче указать условие!

Вариант 1.

1. Написать процедуру, которая в качестве параметров принимает основание и высоту треугольника и рассчитывает его площадь.

2. Составить процедуру, которая определяет сколько цифр в введенном числе (сколько-значное число?).

3. Написать процедуру возведения числа A в степень B. **Функцию возведения в степень не использовать!**

4. Написать процедуру, выводящую последовательность чисел Фибоначчи до N-го числа.

5. Составить процедуру, принимающую в качестве параметров коэффициенты квадратного уравнения и находящую его корни.

6. Написать процедуру, которая определяет, является ли число B делителем числа A.

Вариант 2.

1. Написать процедуру, которая в качестве параметров принимает размеры трех сторон треугольника и определяет, является ли данный треугольник прямоугольным.

2. Составить процедуру, рассчитывающую сумму цифр введенного числа.

3. Написать процедуру умножения числа A на число B. **Операцию умножения не использовать!**

4. Написать процедуру, выводящую N чисел последовательности Фибоначчи.

5. Составить процедуру, которая выводит все делители введенного числа.

6. Написать процедуру, которая в качестве параметров принимает координаты точки на плоскости и в результате выводит сообщение, в какой четверти находится точка.

ЧАСТЬ 2
МЕТОДЫ (ФУНКЦИИ)

Возвращение значения и оператор *return*.

Метод может возвращать значение, какой-либо результат. В примерах выше были определены методы, которые имели тип ***void***. Методы с таким типом не возвращают никакого значения. Они просто выполняют некоторые действия.

Но методы также могут возвращать некоторое значение. Для этого применяется оператор ***return***, после которого идет возвращаемое значение. Например, определим метод, который возвращает значение типа ***string***:

```
string GetMessage()
{
   return "Hello";
}
```

Метод ***GetMessage*** имеет тип ***string***, следовательно, он должен возвратить строку. Поэтому в теле метода используется оператор ***return***, после которого указана возвращаемая строка.

При этом методы, которые в качестве возвращаемого типа имеют любой тип, кроме ***void***, обязательно должны использовать оператор return для возвращения значения. Например, следующее определение метода некорректно:

```
string GetMessage()
{
   Console.WriteLine("Hello");
}
```

Также между возвращаемым типом метода и возвращаемым значением после оператора return должно быть соответствие.

Результат методов, который возвращают значение, мы можем присвоить переменным или использовать иным образом в программе:

```
string GetMessage()
{
   return "Hello";
}

string message = GetMessage();  // получаем результат метода в переменную message
Console.WriteLine(message);     // Hello
```

Метод ***GetMessage()*** возвращает значение типа ***string***. Поэтому мы можем присвоить это значение какой-нибудь переменной типа ***string***: ***string message = GetMessage();***

Либо даже передать в качестве значения параметру другого метода:

```
string GetMessage()
{
   return "Hello";
}
void PrintMessage(string message)
{
   Console.WriteLine(message);
}
PrintMessage(GetMessage());
```

После оператора ***return*** также можно указывать сложные выражения или вызовы других методов, которые возвращают определенный результат. Например, определим метод, который возвращает сумму чисел:

```
int Sum(int x, int y)
{
   return x + y;
}

int result = Sum(10, 15);   // 25
Console.WriteLine(result);   // 25

Console.WriteLine(Sum(5, 6));   // 11
```

Выход из метода

Оператор ***return*** не только возвращает значение, но и производит выход из метода. Поэтому он должен определяться после остальных инструкций. Однако мы можем использовать оператор ***return*** и в методах с типом ***void***. В этом случае после оператора return не ставится никакого возвращаемого значения (ведь метод ничего не возвращает). Типичная ситуация – в зависимости от определённых условий произвести выход из метода:

```
void PrintPerson(string name, int age)
{
   if (age > 120 || age < 1)
   {
      Console.WriteLine("Недопустимый возраст");
      return;
   }
   Console.WriteLine($"Имя: {name}  Возраст: {age}");
}
```

```
PrintPerson("Tom", 37);       // Имя: Tom  Возраст: 37
PrintPerson("Dunkan", 1234);  // Недопустимый возраст
```

Здесь метод ***PrintPerson()*** в качестве параметров принимает имя и возраст пользователя. Однако в методе вначале мы проверяем, соответствует ли возраст некоторому диапазону (меньше 120 и больше 0). Если возраст находится вне этого диапазона, то выводим сообщение о недопустимом возрасте и с помощью оператора ***return*** выходим из метода. После этого метод заканчивает свою работу.

Однако если возраст корректен, то выводим информацию о пользователе на консоль.

Передача параметров по ссылке и значению. Выходные параметры.

Существует два способа передачи параметров в метод в языке C#: по значению и по ссылке.

Передача параметров по значению

Наиболее простой способ передачи параметров представляет передача по значению, по сути, это обычный способ передачи параметров:

```
void Increment(int n)
{
   n++;
   Console.WriteLine($"Число в методе Increment: {n}");
}

int number = 5;
Console.WriteLine($"Число до метода Increment: {number}");
Increment(number);
Console.WriteLine($"Число после метода Increment: {number}");
```

В результате выполнения кода, мы получим следующее:

```
Число до метода Increment: 5
Число в методе Increment: 6
Число после метода Increment: 5
```

При передаче аргументов параметрам по значению параметр метода получает не саму переменную, а ее копию и далее работает с этой копией независимо от самой переменной.

Так, выше при вызове метод ***Increment*** получает копию переменной ***number*** и увеличивает значение этой копии. Поэтому в самом методе ***Increment*** мы видим, что значение параметра ***n*** увеличилось на 1, но после выполнения метода переменная ***number*** имеет прежнее значение - 5. То есть изменяется копия, а сама переменная не изменяется.

Передача параметров по ссылке и модификатор ref.

При передаче параметров по ссылке перед параметрами используется модификатор ***ref***:

```
void Increment(ref int n)
{
   n++;
   Console.WriteLine($"Число в методе Increment: {n}");
}

int number = 5;
Console.WriteLine($"Число до метода Increment: {number}");
Increment(ref number);
Console.WriteLine($"Число после метода Increment: {number}");
```

Результат:

```
Число до метода Increment: 5
Число в методе Increment: 6
Число после метода Increment: 6
```

При передаче значений параметрам по ссылке метод получает адрес переменной в памяти. И, таким образом, если в методе изменяется значение параметра, передаваемого по ссылке, то также изменяется и значение переменной, которая передается на его место.

Так, в метод ***Increment*** передается ссылка на саму переменную ***number*** в памяти. И если значение параметра n в ***Increment*** изменяется, то это приводит и к изменению переменной ***number***, так как и параметр ***n*** и переменная ***number*** указывают на один и тот же адрес в памяти.

Обратите внимание, что модификатор ***ref*** указывается как перед параметром при объявлении метода, так и при вызове метода перед аргументом, который передается параметру.

Выходные параметры. Модификатор out.

Выше мы использовали входные параметры. Но параметры могут быть также выходными. Чтобы сделать параметр выходным, перед ним ставится модификатор ***out***:

```
void Sum(int x, int y, out int result)
{
   result = x + y;
}
```

Здесь результат возвращается не через оператор ***return***, а через выходной параметр ***result***. Использование в программе:

```
void Sum(int x, int y, out int result)
{
   result = x + y;
}

int number;

Sum(10, 15, out number);

Console.WriteLine(number);   // 25
```

Причем, как и в случае с ***ref*** ключевое слово ***out*** используется как при определении метода, так и при его вызове.

Также обратите внимание, что методы, использующие такие параметры, обязательно должны присваивать им определенное значение.

Преимущество использования подобных параметров состоит в том, что, по сути, мы можем вернуть из метода не одно значение, а несколько. Например:

```
void GetRectangleData(int width, int height, out int rectArea, out int rectPerimetr)
{
   rectArea = width * height;        // площадь прямоугольника - произведение ширины на высоту
   rectPerimetr = (width + height) * 2; // периметр прямоугольника - сумма длин всех сторон
}

int area;
int perimetr;

GetRectangleData(10, 20, out area, out perimetr);

Console.WriteLine($"Площадь прямоугольника: {area}");      // 200
Console.WriteLine($"Периметр прямоугольника: {perimetr}");   // 60
```

Здесь у нас есть метод ***GetRectangleData***, который получает ширину и высоту прямоугольника (параметры width и height). А два выходных параметра мы используем для подсчета площади и периметра прямоугольника.

При этом можно определять переменные, которые передаются ***out***-параметрам непосредственно при вызове метода. То есть мы можем сократить предыдущий пример следующим образом:

```
void GetRectangleData(int width, int height, out int rectArea, out int rectPerimetr)
{
   rectArea = width * height;
   rectPerimetr = (width + height) * 2;
}

GetRectangleData(10, 20, out int area, out int perimetr);

Console.WriteLine($"Площадь прямоугольника: {area}");      // 200
Console.WriteLine($"Периметр прямоугольника: {perimetr}");   // 60
```

Массив в качестве параметра

```
void Sum(int[] numbers, int initialValue)
{
   int result = initialValue;
   foreach (var n in numbers)
   {
      result += n;
   }
   Console.WriteLine(result);
}

int[] nums = { 1, 2, 3, 4, 5 };
Sum(nums, 10);

// Sum(1, 2, 3, 4);     // так нельзя - нам нужно передать массив
```

ЗАДАНИЕ

Написать программы на языке C#. Все задачи поместить в одно решение. В каждой задаче указать условие!

Вариант 1.

1. Создайте функцию заполнения одномерного массива случайными числами из диапазона, введенного с клавиатуры.

2. Создайте метод, который меняет местами значения двух переменных.

3. Напишите метод, который по значениям двух катетов вычисляет гипотенузу и площадь треугольника. Использовать метод с выходными параметрами.

4. Напишите функцию, которая возвращает количество положительных элементов массива.

5. Вычислить наименьшее значение $Y(B[i])=5*B[i]^3+3*B[i]^2-B[i]$, если B[i] задано массивом *int[] B = new int[N]*.

Вариант 2.

1. Создайте функцию для нахождения суммы всех элементов одномерного массива.

2. Создайте функцию, которая возвращает логический тип и проверяет, кратна ли сумма цифр двузначного числа числу A, введенному с клавиатуры. Проверить возвращаемое значение и вывести соответствующее сообщение.

3. Напишите метод, который возвращает максимальный и минимальный элементы одномерного массива. Использовать выходные параметры.

4. Напишите функцию, которая возвращает количество нечетных элементов массива.

5. Даны числа S, T. Получить с использованием функции пользователя с параметрами:

$G(1, \sin(S))+2G(T*S, 24)-G(5, -S)$, где $G(A,B) = (2A+B^2)/(2A*B+5B)$

ПРАКТИЧЕСКАЯ РАБОТА №13
Обработка символов и строк. Класс String.

Цели:

– *Изучить класс String;*
– *Изучить операции над символами и строками;*
– *Изучить способы форматирования строк.*

Строки и класс string.

Довольно большое количество задач, которые могут встретиться при разработке приложений, так или иначе связано с обработкой строк - парсинг веб-страниц, поиск в тексте, какие-то аналитические задачи, связанные с извлечением нужной информации из текста и т.д. Поэтому в этом плане работе со строками уделяется особое внимание.

В языке C# строковые значения представляет тип ***string***, а вся функциональность работы с данным типом сосредоточена в классе ***System.String***. Собственно, ***string*** является псевдонимом для класса ***String***. Объекты этого класса представляют текст как последовательность символов Unicode. Максимальный размер объекта ***String*** может составлять в памяти 2 ГБ, или около 1 миллиарда символов.

Создавать строки можно, как используя переменную типа ***string*** и присваивая ей значение, так и применяя один из конструкторов класса ***String***:

```
string s1 = "hello";
string s2 = new String('a', 6); // результатом будет строка "аааааа"
string s3 = new String(new char[] { 'w', 'o', 'r', 'l', 'd' });
string s4 = new String(new char[] { 'w', 'o', 'r', 'l', 'd' }, 1, 3); // orl

Console.WriteLine(s1);  // hello
Console.WriteLine(s2);  // aaaaaaa
Console.WriteLine(s3);  // world
Console.WriteLine(s4);  // orl
```

Так как строка хранит коллекцию символов, в ней определен индексатор для доступа к этим символам. Применяя индексатор, мы можем обратиться к строке как к массиву символов и получить по индексу любой из ее символов:

```
string message = "hello";
// получаем символ
char firstChar = message[1]; // символ 'e'
Console.WriteLine(firstChar);   //e

Console.WriteLine(message.Length);  // длина строки
```

Используя свойство ***Length***, как и в обычном массиве, можно получить длину строки.

Перебор строк

Строку можно перебрать в цикле ***foreach*** как набор объектов ***char***. Также можно с помощью других типов циклов перебрать строку, применяя обращение к символам по индексу:

```
string message = "hello";

for (var i = 0; i < message.Length; i++)
{
   Console.WriteLine(message[i]);
}
foreach (var ch in message)
{
   Console.WriteLine(ch);
}
```

Основная функциональность класса ***String*** раскрывается через его методы, среди которых можно выделить следующие:

Compare	Сравнивает две строки с учетом текущей культуры (локали) пользователя
CompareOrdinal	Сравнивает две строки без учета локали
Contains	Определяет, содержится ли подстрока в строке
Concat	Соединяет строки
CopyTo	Копирует часть строки, начиная с определенного индекса в массив
EndsWith	Определяет, совпадает ли конец строки с подстрокой
Format	Форматирует строку
IndexOf	Находит индекс первого вхождения символа или подстроки в строке
Insert	Вставляет в строку подстроку
Join	Соединяет элементы массива строк
LastIndexOf	Находит индекс последнего вхождения символа или подстроки в строке
Replace	Замещает в строке символ или подстроку другим символом или подстрокой
Split	Разделяет одну строку на массив строк
Substring	Извлекает из строки подстроку, начиная с указанной позиции
ToLower	Переводит все символы строки в нижний регистр
ToUpper	Переводит все символы строки в верхний регистр
Trim	Удаляет начальные и конечные пробелы из строки

Объединение строк

Конкатенация строк или объединение может производиться как с помощью операции +, так и с помощью метода ***Concat***:

```
string s1 = "hello";
string s2 = "world";
string s3 = s1 + " " + s2; // результат: строка "hello world"
string s4 = string.Concat(s3, "!!!"); // результат: строка "hello world!!!"
```

Для объединения строк также может использоваться метод ***Join***:

```
string s5 = "apple";
string s6 = "a day";
string s7 = "keeps";
string s8 = "a doctor";
string s9 = "away";
string[] values = new string[] { s5, s6, s7, s8, s9 };

string s10 = string.Join(" ", values);
Console.WriteLine(s10); // apple a day keeps a doctor away
```

Метод ***Join*** получает два параметра: строку-разделитель (в данном случае пробел) и массив строк, которые будут соединяться и разделяться разделителем.

Сравнение строк

Для сравнения строк применяется статический метод ***Compare***:

```
string s1 = "hello";
string s2 = "world";

int result = string.Compare(s1, s2);
if (result < 0)
{
   Console.WriteLine("Строка s1 перед строкой s2");
}
else if (result > 0)
{
   Console.WriteLine("Строка s1 стоит после строки s2");
}
else
{
   Console.WriteLine("Строки s1 и s2 идентичны");
}
// результатом будет "Строка s1 перед строкой s2"
```

Данная версия метода ***Compare*** принимает две строки и возвращает число. Если первая строка по алфавиту стоит выше второй, то возвращается число меньше нуля. В противном случае возвращается число больше нуля. И третий случай - если строки равны, то возвращается число 0.

В данном случае так как символ h по алфавиту стоит выше символа w, то и первая строка будет стоять выше.

Поиск в строке

С помощью метода ***IndexOf*** мы можем определить индекс первого вхождения отдельного символа или подстроки в строке:

```
string s1 = "hello world";
char ch = 'o';
int indexOfChar = s1.IndexOf(ch); // равно 4
Console.WriteLine(indexOfChar);

string substring = "wor";
int indexOfSubstring = s1.IndexOf(substring); // равно 6
Console.WriteLine(indexOfSubstring);
```

Подобным образом действует метод ***LastIndexOf***, только находит индекс последнего вхождения символа или подстроки в строку.

Еще одна группа методов позволяет узнать начинается или заканчивается ли строка на определенную подстроку. Для этого предназначены методы ***StartsWith*** и ***EndsWith***. Например, в массиве строк хранится список файлов, и нам надо вывести все файлы с расширением exe:

```
var files = new string[]
{
    "myapp.exe",
    "forest.jpg",
    "main.exe",
    "book.pdf",
    "river.png"
};

for (int i = 0; i < files.Length; i++)
{
    if (files[i].EndsWith(".exe"))
        Console.WriteLine(files[i]);
}
```

Разделение строк

С помощью функции ***Split*** мы можем разделить строку на массив подстрок. В качестве параметра функция ***Split*** принимает массив символов или

строк, которые и будут служить разделителями. Например, подсчитаем количество слов в сроке, разделив ее по пробельным символам:

```
string text = "И поэтому все так произошло";
string[] words = text.Split(new char[] { ' ' });

foreach (string s in words)
{
   Console.WriteLine(s);
}
```

Обрезать определенную часть строки позволяет функция ***Substring***:

```
string text = "Хороший день";
// обрезаем начиная с третьего символа
text = text.Substring(2);
// результат "роший день"
Console.WriteLine(text);
// обрезаем сначала до последних двух символов
text = text.Substring(0, text.Length - 2);
// результат "роший де"
Console.WriteLine(text);
```

Функция ***Substring*** также возвращает обрезанную строку. В качестве параметра первая использованная версия применяет индекс, начиная с которого надо обрезать строку. Вторая версия применяет два параметра - индекс начала обрезки и длину вырезаемой части строки.

Вставка

Для вставки одной строки в другую применяется функция ***Insert***:

```
string text = "Хороший день";
string substring = "замечательный ";

text = text.Insert(8, substring);
Console.WriteLine(text);    // Хороший замечательный день
```

Первым параметром в функции ***Insert*** является индекс, по которому нужно вставлять подстроку, а второй параметр - собственно подстрока.

Удаление строк

Удалить часть строки помогает метод ***Remove***:

```
string text = "Хороший день";
// индекс последнего символа
int ind = text.Length - 1;
```

```
// вырезаем последний символ
text = text.Remove(ind);
Console.WriteLine(text);    // Хороший ден

// вырезаем первые два символа
text = text.Remove(0, 2);
Console.WriteLine(text);    // роший ден
```

Первая версия метода ***Remove*** принимает индекс в строке, начиная с которого надо удалить все символы. Вторая версия принимает еще один параметр - сколько символов надо удалить.

Замена

Чтобы заменить один символ или подстроку на другую, применяется метод ***Replace***:

```
string text = "хороший день";

text = text.Replace("хороший", "плохой");
Console.WriteLine(text);    // плохой день

text = text.Replace("о", "");
Console.WriteLine(text);    // плхй день
```

Во втором случае применения функции ***Replace*** строка из одного символа "о" заменяется на пустую строку, то есть фактически удаляется из текста. Подобным способом легко удалять какой-то определенный текст в строках.

Спецификаторы форматирования

C / c	Задает формат денежной единицы, указывает количество десятичных разрядов после запятой
D / d	Целочисленный формат, указывает минимальное количество цифр
E / e	Экспоненциальное представление числа, указывает количество десятичных разрядов после запятой
F / f	Формат дробных чисел с фиксированной точкой, указывает количество десятичных разрядов после запятой
G / g	Задает более короткий из двух форматов: F или E
N / n	Также задает формат дробных чисел с фиксированной точкой, определяет количество разрядов после запятой
P / p	Задает отображения знака процентов рядом с числом, указывает количество десятичных разрядов после запятой
X / x	Шестнадцатеричный формат числа

Форматирование валюты

Для форматирования валюты используется описатель "C":

```
double number = 23.7;
```

```
string result = string.Format("{0:C0}", number);
Console.WriteLine(result); // 24 р.
string result2 = string.Format("{0:C2}", number);
Console.WriteLine(result2); // 23,70 р.
```

Число после описателя указывает, сколько чисел будет использоваться после разделителя между целой и дробной частью. При выводе также добавляется обозначение денежного знака для текущей культуры компьютера. В зависимости от локализации текущей операционной системы результат может различаться. Также обратите внимание на округление в первом примере.

Форматирование целых чисел

Для форматирования целочисленных значение применяется описатель "d":

```
int number = 23;
string result = string.Format("{0:d}", number);
Console.WriteLine(result); // 23
string result2 = string.Format("{0:d4}", number);
Console.WriteLine(result2); // 0023
```

Число после описателя указывает, сколько цифр будет в числовом значении. Если в исходном числе цифр меньше, то к нему добавляются нули.

Форматирование дробных чисел

Для форматирования дробны чисел используется описатель F, число после которого указывает, сколько знаков будет использоваться после разделителя между целой и дробной частью. Если исходное число - целое, то к нему добавляются разделитель и нули.

```
int number = 23;
string result = string.Format("{0:f}", number);
Console.WriteLine(result); // 23,00

double number2 = 45.08;
string result2 = string.Format("{0:f4}", number2);
Console.WriteLine(result2); // 45,0800

double number3 = 25.07;
string result3 = string.Format("{0:f1}", number3);
Console.WriteLine(result3); // 25,1
```

Формат процентов

Описатель "P" задает отображение процентов. Используемый с ним числовой спецификатор указывает, сколько знаков будет после запятой:

```
decimal number = 0.15345m;
Console.WriteLine("{0:P1}", number);// 15,3%
```

Настраиваемые форматы

Используя знак #, можно настроить формат вывода. Например, нам надо вывести некоторое число в формате телефона +x (xxx)xxx-xx-xx:

```
long number = 19876543210;
string result = string.Format("{0:+# (###) ###-##-##}", number);
Console.WriteLine(result); // +1 (987) 654-32-10
```

ЗАДАНИЕ

Написать программы на языке C#. Все задачи поместить в одно решение. В каждой задаче указать условие!

Задание 1.

1. Вставить в предложение с 3-го символа слово «кит».
2. Удалить из предложения 3 символа, начиная с 6-го.
3. Скопировать из предложения 3 символа, начиная со 2-го.
4. Подсчитать сколько раз встречается буква «м», предлог «не» в предложении.
5. Выяснить, есть ли в предложении хотя бы одна пара одинаковых символов.
6. Удалить из предложения все пробелы.
7. Вставить в предложение пробелы после каждой буквы «а».
8. Проверить, есть ли в предложении запятые.
9. Определить, сколько фамилий в списке начинаются с буквы «А».
10. Определить, сколько фамилий в списке имеют окончание «ов».
11. Определить, сколько фамилий в списке заканчиваются на букву «н».
12. Определить, сколько букв в самой длинной фамилии списка.
13. Определить, сколько букв в самой короткой фамилии списка.
14. Найти самую длинную фамилию в списке. Если таких фамилий несколько, то распечатать их в одну строку.
15. Найти в списке все фамилии, начинающиеся с букв «В» или «Г».
16. Найти в списке все фамилии, имеющие окончание «ев».
17. Найти в списке все фамилии, начинающиеся со слога «Ма».
18. Определить, сколько фамилий в списке состоят менее чем из 6 букв.
19. Определить, сколько фамилий в списке состоят из 8 или 9 букв.
20. Выяснить, имеется ли в списке фамилия «Ганеев». Если имеется, то исправить ее на «Ганиев».
21. Выяснить, сколько раз в списке встречается фамилия «Иванов».
22. Поменять местами первую и последнюю фамилии в списке.
23. Заменить вторую фамилию в списке на «Нигматуллин» и удалить из списка третью фамилию.
24. Выяснить, имеются ли в списке фамилии «Петров». Если имеются, то удалить их из списка.

25. Упорядочить список в алфавитном порядке.
26. Определить, имеются ли в списке однофамильцы.
27. Определить, четна ли сумма цифр, занимающих во введенном числе нечетные позиции.

Задание 2.

1. Задан список из десяти городов. Подсчитать количество названий, которые оканчиваются буквой В.
2. Даны два слова одинаковой длины. Присвоить переменной k число, равное количеству попарно одинаковых букв.
3. Даны два слова. Сколько раз во втором слове встречается первая буква первого слога.
4. Задан список из десяти городов. Поменять местами названия двух городов, названия которых оканчиваются сочетанием букв «град».
5. Имеется некоторая последовательность символов. Образовать новую последовательность, включив в нее символы исходной, кроме символов «ы» и «э».
6. Задан список из десяти городов. Подсчитать количество названий, в которых есть по две буквы «а».
7. Задан список из десяти городов. Поменять местами названия любых двух городов, заканчивающихся буквой «а».
8. Даны два слова разной длины. Присвоить переменной m число, равное количеству попарно различных букв.
9. Имеется некоторый текст. Образовать из него новый, в который включить информацию, заключенную между пробелом и запятой.
10. Задан список из десяти городов. Присвоить переменной t название последнего из городов, которое содержит более 4-х букв.
11. Задан список из 5 имен девочек. Присвоить переменной d имя с наименьшим числом букв.
12. Задан список из десяти городов. Присвоить переменной s название города с максимальным числом букв.
13. Задан текст из 20 символов латинского алфавита. Подсчитать в нем количество гласных букв.
14. Задан список из десяти городов. Поменять местами названия первого города и любого другого, которое содержит более семи букв.
15. Из двух восьмибуквенных слов образовать последовательность букв, в которой должны чередоваться буквы первого и второго слова.
16. Задан список из десяти городов. Поменять местами названия последнего города и любого из городов, название которого оканчивается на букву «к».
17. Имеется некоторая последовательность символов. Образовать новую последовательность, включив в нее символы исходной в обратном порядке.
18. Задан список из десяти городов. Подсчитать количество названий, в которых есть буква «Д».
19. Задан список из десяти городов. Поменять местами названия самого длинного и самого короткого слова.
20. Образовать последовательность символов, включив в нее символы данной последовательности, расположенные на нечетных позициях.
21. Задан текст из символов латинского алфавита, содержащий букву a. Написать все символы, расположенные за первой буквой «а» до ее второго вхождения или до конца текста.

22. Вводится строка – фамилия, имя и отчество учащегося. Вывести на экран преобразованную строку: оставить только фамилию и инициалы.
23. Задан список из десяти городов. Присвоить переменной t название последнего из городов, которое содержит более четырех букв.
24. Задан список из 10 городов. Поменять местами названия первого и последнего города.
25. Определить наибольшую цифру введенного натурального числа.
26. Определить количество символов введенной строки, ASCII-коды которых больше 70.
27. Изменить введенную строку цифр, разместив сначала цифры занимающие нечетные места, потом четные.

Информационное обеспечение обучения

Перечень рекомендуемых учебных изданий, Интернет-ресурсов, дополнительной литературы

Каждому обучающемуся обеспечен доступ к следующим электронным библиотечным системам и профессиональным базам данных:

- ЭБС «Университетская библиотека онлайн».

Электронная библиотека ежегодно обновляется и пополняется.