Лабораторная 1. Линейная программа (2 часа)

Целью лабораторного занятия является приобретение навыков практического применения знаний для создания простейших программ с неразветвлённым вычислительным процессом.

В задачи лабораторного занятия входят:

закрепление знаний студентов в процессе выполнения анализа алгоритма решаемой задачи;

приобретение умений и навыков использования современных сред разработки программного обеспечения.

Формой выполнения лабораторной работы является поиск оптимальных алгоритмов решения программных задач с целью изучения основ программирования с формированием выводов и заключений.

Задание на лабораторную: Ознакомиться со средой Microsoft Visual C# , решить 3 задачи по вариантам и оформить отчёт.

Задание №1

Задачи на ввод и вывод данных , оператор присваивания , арифметические операторы , стандартные функции . Все входные и выходные данные в заданиях этой группы являются вещественными числами . Не требуется выполнять проверку введённых пользователем данных.

Задачи по вариантам

1. Даны катеты прямоугольного треугольника a и b. Найти его гипотенузу c и периметр P.
2. Даны два круга с общим центром и радиусами R1 и R2 (R1 > R2). Найти площади этих кругов S1 и S2, а также площадь S3 кольца, внешний радиус которого равен R1, а внутренний радиус равен R2.
3. Дана длина L окружности. Найти ее радиус R и площадь S круга, ограниченного этой окружностью.
4. Даны три точки A, B, C на числовой оси. Точка C расположена между точками A и B. Найти произведение длин отрезков AC и BC.
5. Даны координаты двух противоположных вершин прямоугольника: (x1, y1), (x2, y2). Стороны прямоугольника параллельны осям координат. Найти периметр и площадь данного прямоугольника.
6. Даны координаты трех вершин треугольника: (x1, y1), (x2, y2), (x3, y3). Найти его периметр и площадь, используя формулу для расстояния между двумя точками на плоскости. Для нахождения площади треугольника со сторонами a, b, c использовать формулу Герона:$S = (p \cdot (p - a) \cdot (p - b) \cdot (p - c))1/2$, где $p = (a + b + c)/2$ — полупериметр.
7. Дано значение температуры T в градусах Фаренгейта. Определить значение этой же температуры в градусах Цельсия. Температура по Цельсию TC и температура по Фаренгейту TF связаны следующим соотношением:$TC = (TF - 32) \cdot 5/9$. Градус Фаренгейта
8. Известно, что X кг шоколадных конфет стоит A рублей, а Y кг ирисок стоит B рублей. Определить, сколько стоит 1 кг шоколадных конфет, 1 кг ирисок, а также во сколько раз шоколадные конфеты дороже ирисок.
9. Скорость лодки в стоячей воде V км/ч, скорость течения реки U км/ч (U < V). Время движения лодки по озеру T1 ч, а по реке (против течения) — T2 ч. Определить путь S, пройденный лодкой (путь = время · скорость). Учесть, что при движении против течения скорость лодки уменьшается на величину скорости течения.
10. Скорость первого автомобиля V1 км/ч, второго — V2 км/ч, расстояние между ними S км. Определить расстояние между ними через T часов, если автомобили первоначально движутся навстречу друг другу.

Задание №2

Задачи на целочисленные операции . Все входные и выходные данные в заданиях этой группы являются целыми числами. Все числа, для которых указано количество цифр

(двузначное число, трехзначное число и т. д.), считаются положительными. Не требуется выполнять проверку введённых пользователем данных.

Задачи по вариантам

1. Даны целые положительные числа A и B (A > B). На отрезке длины A размещено максимально возможное количество отрезков длины B (без наложений). Используя операцию деления нацело, найти количество отрезков B, размещенных на отрезке AB.
2. Дано трехзначное число. Найти сумму и произведение его цифр.
3. Дано трехзначное число. Вывести число, полученное при прочтении исходного числа справа налево.
4. Дано трехзначное число. В нем зачеркнули первую слева цифру и приписали ее справа. Вывести полученное число.
5. Дано трехзначное число. В нем зачеркнули первую справа цифру и приписали ее слева. Вывести полученное число.
6. Дано трехзначное число. Вывести число, полученное при перестановке цифр сотен и десятков исходного числа (например, 123 перейдет в 213).
7. Дано целое число, большее 999. Используя только целочисленные операции, найти цифру, соответствующую разряду сотен в записи этого числа.
8. Дни недели пронумерованы следующим образом: 1 — понедельник, 2 — вторник, …, 6 — суббота, 7 — воскресенье. Дано целое число K, лежащее в диапазоне 1–365, и целое число N, лежащее в диапазоне 1–7. Определить номер дня недели для K-го дня года, если известно, что в этом году 1 января было днем недели с номером N.
9. Даны целые положительные числа A, B, C. На прямоугольнике размера A * B размещено максимально возможное количество квадратов со стороной C (без наложений). Найти количество квадратов, размещенных на прямоугольнике, а также площадь незанятой части прямоугольника.
10. Дан номер некоторого года (целое положительное число). Определить соответствующий ему номер столетия, учитывая, что, к примеру, началом 20 столетия был 1901 год.

Задание №3

Задачи на использование логических операторов , операторов отношения . Во всех заданиях данной группы требуется вывести логическое значение True, если приведенное высказывание для предложенных исходных данных является истинным, и значение False в противном случае. Все числа, для которых указано количество цифр (двузначное число, трехзначное число и т. д.), считаются целыми положительными. Не требуется выполнять проверку введённых пользователем данных. Использование IF и оператора "?" недопустимо.

Задачи по вариантам

1. Даны координаты двух различных полей шахматной доски x1, y1, x2, y2 (целые числа, лежащие в диапазоне 1–8). Проверить истинность высказывания: «Данные поля имеют одинаковый цвет». Если пользователь введёт дважды координаты одной и той же клетки считать решение задачи ложью. Ходы в шахматах
2. Даны координаты двух различных полей шахматной доски x1, y1, x2, y2 (целые числа, лежащие в диапазоне 1–8). Проверить истинность высказывания: «Ладья за один ход может перейти с одного поля на другое». Если пользователь введёт дважды координаты одной и той же клетки считать решение задачи ложью. Ходы в шахматах
3. Даны координаты двух различных полей шахматной доски x1, y1, x2, y2 (целые числа, лежащие в диапазоне 1–8). Проверить истинность высказывания: «Король за один ход может перейти с одного поля на другое». Если пользователь введёт дважды координаты одной и той же клетки считать решение задачи ложью. Ходы в шахматах
4. Даны координаты двух различных полей шахматной доски x1, y1, x2, y2 (целые числа, лежащие в диапазоне 1–8). Проверить истинность высказывания: «Слон за один ход

может перейти с одного поля на другое». Если пользователь введёт дважды координаты одной и той же клетки считать решение задачи ложью. Ходы в шахматах
5. Даны координаты двух различных полей шахматной доски x1, y1, x2, y2 (целые числа, лежащие в диапазоне 1–8). Проверить истинность высказывания: «Ферзь за один ход может перейти с одного поля на другое». Если пользователь введёт дважды координаты одной и той же клетки считать решение задачи ложью. Ходы в шахматах
6. Даны координаты двух различных полей шахматной доски x1, y1, x2, y2 (целые числа, лежащие в диапазоне 1–8). Проверить истинность высказывания: «Конь за один ход может перейти с одного поля на другое». Если пользователь введёт дважды координаты одной и той же клетки считать решение задачи ложью. Ходы в шахматах
7. Даны целые числа a, b, c, являющиеся сторонами некоторого треугольника. Проверить истинность высказывания: «Треугольник со сторонами a, b, c является прямоугольным».
8. Дано трехзначное число. Проверить истинность высказывания: «Цифры данного числа образуют возрастающую или убывающую последовательность».
9. Дано целое положительное число. Проверить истинность высказывания: «Данное число является нечетным трехзначным».
10. Даны координаты поля шахматной доски x, y (целые числа, лежащие в диапазоне 1–8). Учитывая, что левое нижнее поле доски (1, 1) является черным, проверить истинность высказывания: «Данное поле является белым».
Отчёт по лабораторной работе
1) титульный лист;
2) цель работы;
3) основную часть. По каждой из решённых задач в отчёте должны быть:
постановка задачи;
словесное объяснение алгоритма
текст программы с комментариями;
не менее десяти тестовых примеров. Один из примеров подтверждается скриншотом исполняемой программы в обязательном порядке.
4) развёрнутые выводы по работе;
5) список использованной литературы и Интернет-ресурсов

Лабораторная 2. Условия и циклы (2 часа)
Задание №1
Задачи на использование операторов условия . Осуществить ввод необходимых данных, выполнить реализацию алгоритма с использованием операторов условия, обеспечить вывод полученных результатов. Для решения задачи предварительно составляется блок-схема. Не допускается использование операторов, прерывающих ход программы (break, goto).
Задачи по вариантам
1.Ввести номер года (положительное целое число). Определить количество дней в этом году, учитывая, что обычный год насчитывает 365 дней, а високосный — 366 дней. Високосным считается год, делящийся на 4, за исключением тех годов, которые делятся на 100 и не делятся на 400 (например, годы 300, 1300 и 1900 не являются високосными, а 1200 и 2000 — являются). См. также: Високосный год
2.Ввести целочисленные координаты трех вершин прямоугольника, стороны которого параллельны координатным осям. Найти координаты его четвертой вершины. Если пользователь введёт координаты точек так, что нельзя получить прямоугольник со сторонами, параллельными координатным осям, вывести соответствующее сообщение.
3.На числовой оси расположены три точки: A, B, C. Определить, какая из двух последних точек (B или C) расположена ближе к A, и вывести эту точку и ее расстояние от точки A. Если пользователь введёт координаты B и C так, что они будут равноотстоящими от A, совпадать с A или между собой - выдать соответствующее сообщение.

4.Ввести четыре целых числа A, B, C, D, одно из которых отлично от трех других, равных между собой. Определить порядковый номер числа, отличного от остальных. Если пользователь введёт числа так, что они не будут соответствовать условию задачи - выдать сообщение об ошибке.
5.Ввести три числа A,B,C. Если среди них имеется хотя бы одно четное вывести максимальное из них, иначе - минимальное. Если пользователь введёт числа так, что среди них нельзя будет определить лишь одно максимальное/минимальное - выдать соответствующее сообщение.
6.Ввести три переменные вещественного типа: A, B, C. Если их значения упорядочены по возрастанию или убыванию, то удвоить их; в противном случае заменить знак каждой переменной на противоположный. Вывести новые значения переменных A, B, C.
7.Ввести три числа A, B, C. Найти сумму двух наибольших из них. Если пользователь введёт числа так, что среди них нельзя будет определить два наибольших - выдать соответствующее сообщение.
8.Ввести три числа A, B, C. Вывести вначале наименьшее, а затем наибольшее из данных чисел. Если пользователь введёт числа так, что среди них нельзя будет определить одно наименьшее/наибольшее - выдать соответствующее сообщение.
9.Ввести три числа A, B, C. Вывести среднее по величине из них (то есть число, расположенное между наименьшим и наибольшим). Если пользователь введёт числа так, что среди них нельзя будет определить среднее - выдать соответствующее сообщение.
10.Ввести три целых числа A, B, C. Найти количество положительных и количество отрицательных чисел в исходном наборе.

Задание №2

Задачи на использование операторов варианта . Осуществить ввод необходимых данных, выполнить реализацию алгоритма с использованием операторов варианта, обеспечить вывод полученных результатов. Для решения задачи предварительно составляется блок-схема. Не допускается использование массивов и операторов goto.

Задачи по вариантам

1.Даны два целых числа: D (день) и M (месяц), определяющие правильную дату невисокосного года. Вывести значения D и M для даты, предшествующей указанной. Если пользователь вводит D и M несоответствующие календарю - выдать сообщение об ошибке. См. также: Григорианский календарь
2.Мастям игральных карт присвоены порядковые номера: 1 — пики, 2 — трефы, 3 — бубны, 4 — червы. Достоинству карт, старших десятки, присвоены номера: 11 — валет, 12 — дама, 13 — король, 14 — туз. Даны два целых числа: N — достоинство (6 <= N <= 14) и M — масть карты (1 <= M <= 4). Вывести название соответствующей карты вида «шестерка бубен», «дама червей», «туз треф» и т. п. Если пользователь введёт данные не соответствующие условию задачи - выдать сообщение об ошибке.
3.Дано целое число в диапазоне 20–69, определяющее возраст (в годах). Вывести строку-описание указанного возраста, обеспечив правильное согласование числа со словом «год», например: 20 — «двадцать лет», 32 — «тридцать два года», 41 — «сорок один год». Если пользователь введёт данные не соответствующие условию задачи - выдать сообщение об ошибке.
4.Дано целое число в диапазоне 10–40, определяющее количество учебных заданий по некоторой теме. Вывести строку-описание указанного количества заданий, обеспечив правильное согласование числа со словами «учебное задание», например: 18 — «восемнадцать учебных заданий», 23 — «двадцать три учебных задания», 31 — «тридцать одно учебное задание». Если пользователь введёт данные не соответствующие условию задачи - выдать сообщение об ошибке.
5.Дано целое число в диапазоне 100–999. Вывести строку-описание данного числа, например: 256 — «двести пятьдесят шесть», 814 — «восемьсот четырнадцать». Если

пользователь введёт данные не соответствующие условию задачи - выдать сообщение об ошибке.
6.В восточном календаре принят 60-летний цикл, состоящий из 12-летних подциклов, обозначаемых названиями цвета: зеленый, красный, желтый, белый и голубой. При этом каждый цвет следует по два года подряд. В каждом подцикле годы носят названия животных: крысы, коровы, тигра, зайца, дракона, змеи, лошади, овцы, обезьяны, петуха, собаки и свиньи. По номеру года определить его название, если 4 год нашей эры — начало цикла: «год зеленой крысы». См. также: Китайский гороскоп Перевод китайского календаря в годы нашей эры
7.Составить программу, которая бы присваивала переменной T значение true, если дата d1,m1 предшествует(в рамках года) дате d2,m2 и значение false иначе(d1 и d2-дата,m1 и m2-месяц). Переменную T распечатать. Год считать невисокосным. Если введённые даты не соответствуют календарю - выдать сообщение об ошибке.
8.Составить программу, которая бы по введённому значению некоторой длины в метрах выводила бы это значение с использованием наиболее подходящей кратной или дольной приставки (км,м,дм,см,мм,мкм,нм). Подсказка: для нахождения порядка числа использовать десятичный логарифм. См. также: Десятичный логарифм
9.Для натурального числа K напечатать фразу "мы нашли K грибов в лесу", согласовав окончание слова "гриб" с числом K. Обратите внимание на особое согласование в случае когда 10<K<20.
10.Составить программу, которая бы реализовала следующий алгоритм: переменной T присвоить значение true если сочетание D(день) M(месяц) G(год) образует правильную дату, и значение false- иначе(учитывая количество дней в месяце и название месяца). Переменную T распечатать. Примечание: високосным считается год, делящийся на 4, за исключением тех годов, которые делятся на 100 и не делятся на 400 (например, годы 300, 1300 и 1900 не являются високосными, а 1200 и 2000 — являются). См. также: Високосный год

Задание №3

Задачи на использование операторов цикла for . Осуществить ввод необходимых данных, выполнить реализацию алгоритма с использованием операторов цикла for, обеспечить вывод полученных результатов. Не разрешается использовать другие операторы цикла. Для решения задачи предварительно составляется блок-схема. Не допускается использование массивов и операторов, прерывающих ход программы (break, goto).

Задачи по вариантам

1.Ввести целое число N > 1 и две вещественные точки на числовой оси: A, B (A < B). Отрезок [A, B] разбит на N равных отрезков. Вывести H — длину каждого отрезка, а также значения функции f(x) = 1 – sin(x) в точках, разбивающих отрезок [A, B]: f(A), f(A + H), f(A + 2H), ..., f(B).
2.Ввести целое число N > 0 и вещественное a > 0. Последовательность вещественных чисел определяется следующим образом xn+1=(xn+a/xn)/2. Считая x0=a вывести первые N членов последовательности. Такой способ применяли еще в древнем Вавилоне для вычисления квадратного корня числа a. После выдачи последовательности распечатать значение квадратного корня из a, вычисленное стандартной функцией. См. также: Квадратный корень , Итерационная формула Герона
3.Ввести целое число N > 1. Последовательность чисел Фибоначчи FK (целого типа) определяется следующим образом: F1 =1, F2= 1, FK=FK-2 + FK-1, K = 3, 4, ... N. Вывести элементы F1, F2, …, FN. См. также: Числа Фибоначчи .
4.Ввести целое число N > 0. Последовательность вещественных чисел AK определяется следующим образом: A0=1/0!, AK=1/K!, K=1, 2, ...,N. Вывести сумму последовательности. Примечание K! — это K–факториал — обозначает произведение всех целых чисел от 1 до K. См. также: Факториал , Ряд Тейлора

5.Логистическое отображение (также известное, как квадратичное отображение или отображение Фейгенбаума) даётся формулой xn+1=r*xn*(1-xn). Считая x0=0.333 распечатать N первых элементов отображения. Величину r, принадлежащую интервалу (0..4) вводит пользователь. (При r>3.6 должна наблюдаться хаотическая последовательность). В качестве тестового примера построить последовательности при разных значениях r. См. также: Логистическое отображение

6.Ввести целое число N > 2. Последовательность целых чисел AK определяется следующим образом: A1=1, A2=2, A3=3, AK=AK–1 + AK–2 – 2*AK–3, K = 4, 5, ... N. Вывести элементы A1, A2, ..., AN.

7.Ввести вещественное число X и целое число N > 0. Найти значение выражения X – X3/(3!) + X5/(5!) – ... + (–1)N*X2*N+1/((2*N+1)!) , которое является приближенным значением функции sin в точке X. Отобразить сумму ряда и рассчитанное с помощью функции sin значения. См. также: Факториал , Ряд Тейлора

8.Ввести целое число N > 0. Найти квадрат данного числа, используя для его вычисления следующую формулу: N2=1 + 3 + 5 + ... + (2*N – 1). После добавления к сумме каждого слагаемого выводить текущее значение суммы (в результате будут выведены квадраты всех целых чисел от 1 до N).

9.Ввести целое число N > 0. Найти значение выражения 1.1 – 1.2 + 1.3 – ... (N слагаемых, знаки чередуются). Условный оператор не использовать.

10.Ввести целое число N > 0. Среди цифр этого числа выделить только чётные, из которых составить другое число и вывести. Например, при N=3854972 ответом будет число 842.

Задание №4

Задачи на использование операторов цикла с постусловием . Осуществить ввод необходимых данных, выполнить реализацию алгоритма с использованием операторов цикла do - while, обеспечить вывод полученных результатов. Использование других операторов цикла недопустимо. Для решения задачи предварительно составляется блок-схема. Не допускается использование массивов и операторов, прерывающих ход программы (break, goto).

Задачи по вариантам

1.Ввести два целых числа N1 и N2. Если N1 >N2, найти сумму целых чисел в диапазоне N1 ... N2. Если N2 больше N1, найти сумму целых чисел в диапазоне N2... N1. Если N1 равно N2, вывести на экран соответствующее сообщение.

2.Осуществить ввод последовательности целых чисел, определить третье положительное число и подсчитать количество цифр в нем. Последовательность потенциально не ограничена, окончанием последовательности служит третье положительное число.

3.Осуществить ввод последовательности целых чисел, определить максимальное четное число, его порядковый номер и подсчитать сумму его цифр. Последовательность потенциально не ограничена, окончанием последовательности служит число 0. Если окажется, что чётных чисел в последовательности не было, вывести соответствующее сообщение.

4.Осуществить ввод последовательности целых чисел и сравнить, что больше, сумма положительных или произведение отрицательных. Последовательность потенциально не ограничена, окончанием последовательности служит число 0.

5.Осуществить ввод последовательности целых чисел и определить предпоследнее отрицательное число. Последовательность потенциально не ограничена, окончанием последовательности служит число 0. Если окажется, что в последовательности было менее двух отрицательных чисел, вывести соответствующее сообщение.

6.Осуществить ввод целого числа M. На промежутке от 1 до M найти все числа Армстронга. Натуральное число из n цифр называется числом Армстронга, если сумма его цифр, возведенных в n-ю степень, равна самому числу. Примеры: 153=13 + 53 + 33; 1634=14+64+34+44. См. также: Число Армстронга

7.Ввести действительное число x и натуральное число n. Вычислить x*(x - n)*(x - 2*n)(x - 3*n)…(x - n2).
8.Осуществить ввод последовательности целых чисел. Определить, сколько из них и какие принимают наибольшее значение. Последовательность потенциально не ограничена, окончанием последовательности служит число 0.
9.Осуществить ввод последовательности целых чисел в количестве не меньшем двух. Вычислить сумму тех из них, порядковые номера которых - простые числа. Последовательность потенциально не ограничена, окончанием последовательности служит число 0. См. также: Простые числа
10.Осуществить ввод последовательности целых чисел в количестве не меньшем трёх. Определить, сколько из них больше своих "соседей", т.е. предыдущего и последующего чисел. Последовательность потенциально не ограничена, окончанием последовательности служит число 0.

Задание №5

Задачи на использование операторов цикла с предусловием . Осуществить ввод необходимых данных, выполнить реализацию алгоритма с использованием операторов цикла while, обеспечить вывод полученных результатов. Использование других операторов цикла недопустимо. Для решения задачи предварительно составляется блок-схема. Не допускается использование массивов и операторов, прерывающих ход программы (break, goto).

Задачи по вариантам

1.Ввести целое число N > 0, являющееся некоторой степенью числа 2: N=2K. Найти целое число K — показатель этой степени. Не разрешается использовать логарифм. Если пользователь введёт число не являющееся степенью числа 2 - вывести соответствующее сообщение.
2.Ввести целое число N > 0. Используя операции деления нацело и взятия остатка от деления, найти число, полученное при прочтении числа N справа налево
3.Ввести целое число N > 1. Если оно является простым, то есть не имеет положительных делителей, кроме 1 и самого себя, то вывести это число, иначе вывести ближайшее большее простое число. См. также: Простые числа
4.Ввести целое число N > 1. Последовательность чисел Фибоначчи FK определяется следующим образом: F1=1, F2=1, FK=FK–2 + FK–1, K=3, 4, ... Проверить, является ли число N числом Фибоначчи. Если является, то вывести True, если нет — вывести False. См. также: Числа Фибоначчи .
5.Ввести вещественное число e > 0. Последовательность вещественных чисел AK определяется следующим образом: A1=1, A2=2, AK=(AK–2 + 2·AK–1)/3, K = 3, 4, … . Найти первый из номеров K, для которых выполняется условие |AK – AK–1| < e, и вывести этот номер, а также числа AK–1 и AK.
6.Ввести положительные числа A, B, C. На прямоугольнике размера A x B размещено максимально возможное количество квадратов со стороной C (без наложений). Найти количество квадратов, размещенных на прямоугольнике. Операции умножения и деления не использовать.
7.Ввести два целых числа a и b. Вычислить НОД (a,b) - наибольший общий делитель a и b. Делителями называются числа, которые делят без остатка заданное число, кроме единицы и самого этого числа. См. также: Наибольший общий делитель
8.Ввести натуральное (целое неотрицательное) число a и целое положительное число d. Вычислить частное q и остаток r при делении a на d, не используя операций целочисленного деления и нахождения остатка. Разрешается использовать только целые переменные и целочисленные операции.
9.Ввести целые положительные числа A и B. Найти их наибольший общий делитель (НОД), используя алгоритм Евклида: НОД(A, B)=НОД(B, A mod B), если B ≠ 0; НОД(A,

0)=A, где «mod» обозначает операцию взятия остатка от деления. См. также: Наибольший общий делитель , Алгоритм Евклида
10.Ввести целое число N > 1. Найти первое из чисел Фибоначчи, большее чем N. Последовательность чисел Фибоначчи FK (целого типа) определяется следующим образом: F1 =1, F2= 1, FK=FK-2 + FK-1, K = 3, 4, ... N. См. также: Числа Фибоначчи .
Отчёт по лабораторной работе
По каждой из решённых задач в отчёте должны быть:
Постановка задачи
Текст программы с комментариями
Не менее 2 тестовых примеров. Один из примеров подтверждается скриншотом исполняемой программы в обязательном порядке.

Лабораторная 3. Одномерные массивы (2 часа)
Задание №1
Задачи на использование одномерных целочисленных массивов . Условие вида "дан массив" означает, что пользователем вводится величина размерности и все элементы массива с клавиатуры. Осуществить ввод необходимых данных, выполнить реализацию алгоритма, обеспечить вывод полученных результатов. Для решения задачи предварительно составляется блок-схема. Если по ходу решения задачи требуется создание дополнительных массивов, размерность которых изначально неизвестна, необходимо выполнить предварительную обработку исходного массива, для выяснения размерности вновь создаваемого. Не допускается использование операторов, прерывающих ход программы (break, goto). Ввод массивов, обработка и вывод результатов реализуется отдельными методами .
Задачи по вариантам
1 Даны два массива A и B одинакового размера N. Сформировать новый массив C того же размера, каждый элемент которого равен максимальному из элементов массивов A и B с тем же индексом.
2 Дан целочисленный массив A размера N. Переписать в новый целочисленный массив B все четные числа из исходного массива (в том же порядке) и вывести размер полученного массива B и его содержимое.
3 Дан массив A размера N. Сформировать новый массив B того же размера по следующему правилу: элемент BK равен сумме элементов массива A с номерами от 0 до K.
4 Дан массив A размера N. Сформировать новый массив B того же размера по следующему правилу: элемент BK равен среднему арифметическому элементов массива A с номерами от 0 до K.
5 Дан массив A размера N. Сформировать два новых массива B и C: в массив B записать все положительные элементы массива A, в массив C — все отрицательные (сохраняя исходный порядок следования элементов). Вывести вначале размер и содержимое массива B, а затем — размер и содержимое массива C.
6 Даны два массива A и B, элементы которых упорядочены по возрастанию. Объединить эти массивы так, чтобы результирующий массив C остался упорядоченным по возрастанию
7 Даны два массива A и B. Распечатать те элементы, которые присутствуют в обоих массивах.
8 Даны два массива A и B. Распечатать те элементы массива A, которых нет в массиве B. Распечатать те элементы массива B, которых нет в массиве A.
9 Даны два массива A и B. Определить которых из них имеет больший диапазон, т.е. разницу между самым большим и самым меньшим значением.

10 Дан целочисленный массив A размера N. Переписать в новый целочисленный массив B все элементы с порядковыми номерами, кратными трем (3, 6, …), и вывести размер полученного массива B и его содержимое. Условный оператор не использовать.

Задание №2

Задачи на исследование серий в одномерных целочисленных массивах . Условие вида "дан массив" означает, что пользователем вводится величина размерности и все элементы массива с клавиатуры. Осуществить ввод необходимых данных, выполнить реализацию алгоритма, обеспечить вывод полученных результатов. Для решения задачи предварительно составляется блок-схема. Если по ходу решения задачи требуется создание дополнительных массивов, размерность которых изначально неизвестна, необходимо выполнить предварительную обработку исходного массива, для выяснения размерности вновь создаваемого. Не допускается использование операторов, прерывающих ход программы (break, goto). Ввод массивов, обработка и вывод результатов реализуется отдельными методами .

Задачи по вариантам

1 Дан целочисленный массив A размера N. Назовем серией группу подряд идущих одинаковых элементов, а длиной серии — количество этих элементов (длина серии может быть равна 1). Сформировать два новых целочисленных массива B и C одинакового размера, записав в массив B длины всех серий исходного массива, а в массив C — значения элементов, образующих эти серии.

2 Дан целочисленный массив размера N. Вставить перед каждой его серией элемент с нулевым значением. Серия - это группа подряд идущих одинаковых элементов, длина серии — количество этих элементов (длина серии может быть равна 1).

3 Дан целочисленный массив размера N. Вставить после каждой его серии элемент с нулевым значением. Серия - это группа подряд идущих одинаковых элементов, длина серии — количество этих элементов (длина серии может быть равна 1).

4 Дан целочисленный массив размера N. Преобразовать массив, увеличив каждую его серию на один элемент. Серия - это группа подряд идущих одинаковых элементов, длина серии — количество этих элементов (длина серии может быть равна 1).

5 Дан целочисленный массив размера N. Преобразовать массив, уменьшив каждую его серию на один элемент. Серия - это группа подряд идущих одинаковых элементов, длина серии — количество этих элементов (длина серии может быть равна 1).

6 Дано целое число K и целочисленный массив размера N. Удалить из массива серию с номером K. Если серий в массиве меньше K, то вывести массив без изменений. Серия - это группа подряд идущих одинаковых элементов, длина серии — количество этих элементов (длина серии может быть равна 1).

7 Дано целое число K и целочисленный массив размера N. Поменять местами первую серию массива и его серию с номером K. Если серий в массиве меньше K, то вывести массив без изменений. Серия - это группа подряд идущих одинаковых элементов, длина серии — количество этих элементов (длина серии может быть равна 1).

8 Дано целое число L и целочисленный массив размера N. Заменить каждую серию массива, длина которой меньше L, на один элемент с нулевым значением. Серия - это группа подряд идущих одинаковых элементов, длина серии — количество этих элементов (длина серии может быть равна 1).

9 Дан целочисленный массив размера N. Преобразовать массив, увеличив его первую серию наибольшей длины на один элемент. Серия - это группа подряд идущих одинаковых элементов, длина серии — количество этих элементов (длина серии может быть равна 1).

10 Дан целочисленный массив размера N. Преобразовать массив, увеличив все его серии наибольшей длины на один элемент. Серия - это группа подряд идущих одинаковых элементов, длина серии — количество этих элементов (длина серии может быть равна 1).

Задание №3

Задачи на обработку одномерных целочисленных массивов . Условие вида "дан массив" означает, что пользователем вводится величина размерности и все элементы массива с клавиатуры. Осуществить ввод необходимых данных, выполнить реализацию алгоритма, обеспечить вывод полученных результатов. Для решения задачи предварительно составляется блок-схема. Если по ходу решения задачи требуется создание дополнительных массивов, размерность которых изначально неизвестна, необходимо выполнить предварительную обработку исходного массива, для выяснения размерности вновь создаваемого. Не допускается использование операторов, прерывающих ход программы (break, goto). Ввод массивов, обработка и вывод результатов реализуется отдельными методами .

1 Дан одномерный целочисленный массив из n элементов. Найти количество различных чисел среди элементов этого массива. Например, если задан массив, состоящий из чисел 10,13,10,18,5,10,5, то ответ будет 4, поскольку различные числа это 10,13,18,5. Рекомендуется использовать ещё один массив для хранения различных чисел.

2 Расставить по возрастанию одномерный целочисленный массив из n элементов. При упорядочивании разрешается менять местами только два соседних элемента. Результат распечатать.

3 Дан одномерный целочисленный массив из n элементов. Определить и распечатать все локальные экстремумы в нём. Экстремумами не могут быть крайние элементы.

4 Дан массив A размера N. Упорядочить его по возрастанию методом сортировки простым выбором: найти максимальный элемент массива и поменять его местами с последним элементом; выполнить описанные действия N – 1 раз, каждый раз уменьшая на 1 количество анализируемых элементов и выводя содержимое массива.

5 Дан целочисленный массив размера N. Удалить из массива все одинаковые элементы, оставив их первые вхождения.

6 Дан целочисленный массив размера N. Удалить из массива все элементы, встречающиеся менее трех раз, и вывести размер полученного массива и его содержимое.

7 Дан целочисленный массив размера N. Удалить из массива все элементы, встречающиеся ровно два раза, и вывести размер полученного массива и его содержимое.

8 Дан целочисленный массив размера N. Удалить из массива все соседние одинаковые элементы, оставив их первые вхождения

9 Дан массив A размера N. Не изменяя данный массив, вывести номера его элементов в том порядке, в котором соответствующие им элементы образуют возрастающую последовательность.

10 Дан целочисленный массив размера N. Удалить из массива все одинаковые элементы, оставив их последние вхождения.

Лабораторная 4. Двумерные массивы (2 часа)

Задание №1

Задачи на двухмерные массивы . Условие вида "дана матрица" означает, что пользователем вводится с клавиатуры размерность и все элементы. Осуществить ввод необходимых данных, выполнить реализацию алгоритма, обеспечить вывод полученных результатов. Для решения задачи предварительно составляется блок-схема. Не допускается использование операторов, прерывающих ход программы (break, goto). Ввод многомерных массивов, обработка и вывод результатов реализуется отдельными методами .

1 Дана целочисленная матрица размера M x N. Найти номер первого из ее столбцов, содержащих максимальное количество одинаковых элементов.

2 Дана матрица размера M x N. Найти ее строки, элементы которых упорядочены по возрастанию.

3 Дана целочисленная матрица размера M x N. Различные строки матрицы назовем похожими, если совпадают множества чисел, встречающихся в этих строках. Найти строки, похожие на первую строку данной матрицы.
4 Дана целочисленная матрица размера M x N. Найти ее строки, все элементы которых различны.
5 Дана квадратная целочисленная матрица размера M. Написать программу, которая проверяет, является ли введенная с клавиатуры матрица магическим квадратом. Магическим квадратом называется матрица, сумма элементов которой в каждой строке, в каждом столбце и по каждой диагонали одинакова.
6 Дана матрица размера M x N. Упорядочить ее столбцы так, чтобы их последние элементы образовывали убывающую последовательность.
7 Дана матрица размера M x N. Элемент матрицы называется ее локальным максимумом, если он больше всех окружающих его элементов. Поменять знак всех локальных максимумов данной матрицы на противоположный. При решении допускается использовать вспомогательную матрицу.
8 Дана матрица размера M x N. Упорядочить ее строки так, чтобы их минимальные элементы образовывали убывающую последовательность.
9 Дана целочисленная матрица размера M x N. Найти ее строки, содержащие равное количество положительных и отрицательных элементов (нулевые элементы матрицы не учитываются). Если таких строк нет, то вывести соответствующее сообщение.
10 Дана матрица размером m*n, определить количество и координаты особых элементов матрицы. Элемент считается особым, если он больше суммы остальных элементов своего столбца и при этом в его строке слева от него находятся элементы меньше него, а справа больше него. (Особый элемент может быть крайним в строке)

Лабораторная 5. Строки (2 часа)
Задание №1
Задачи на обработку строк . Условие вида "дана строка" означает, что пользователем вводится строка с клавиатуры. Осуществить ввод необходимых данных, выполнить реализацию алгоритма, обеспечить вывод полученных результатов. Для решения задачи предварительно составляется блок-схема. Не допускается использование операторов, прерывающих ход программы (break, goto). Ввод строк, обработка и вывод результатов реализуется отдельными методами .
1 Даны строки S и S0. Проверить, содержится ли строка S0 в строке S. Не использовать стандартные средства для поиска подстрок.
2 Даны строки S и S0. Найти количество вхождений строки S0 в строку S. Не использовать стандартные средства для поиска подстрок.
3 Дана строка S. Разделить строку на отдельные слова не используя стандартные средства для разбиения строк.
4 Дана строка S. Из строки требуется удалить текст, заключенный в фигурные скобки. В строке может быть несколько фрагментов, заключённых в фигурные скобки. Возможно использование вложенных фигурных скобок и, следовательно необходимо, чтобы программа это учитывала.
5 Дана строка S. Найти количество различных букв в ней. Программа должна работать без учёта регистра букв.
6 Дана строка S. Найти количество различных слов в ней. Программа должна работать без учёта регистра букв.
7 Дана строка S. Определить есть ли в строке удвоенные буквы (пара соседствующих одинаковых букв), напечатать слова, содержащие их.
8 Дана строка S (предложение). Найти самое длинное слово в строке не используя стандартные средства для разбиения строк.

9 Дана строка S (предложение). Составить программу, определяющую является ли текст перевёртышем без учёта пробелов.
10 Дана строка. Вывести все слова, у которых первая и последняя буквы одинаковые не используя стандартные средства для разбиения строк.

Лабораторная 6. Реализация класса и операций по индивидуальному заданию (4 часа)

Задание №1

Задачи на использование классов и объектов, в которых данные описаны в качестве полей. Реализовать класс заданной структуры. В нём предусмотреть конструктор для установки начальных значений полей. Создать объект на основе созданного класса. Осуществить использование объекта в программе.
1. Класс для решения линейного уравнения y=kx+b. Коэффициенты уравнения k, b реализовать с помощью полей вещественного типа. Для решения уравнения предусмотреть метод Root.
2. Элемент ai геометрической прогрессии вычисляется по формуле: ai=a0qi. Реализовать поля a0 и q - вещественного типа. Определить метод Elementi() для вычисления заданного элемента прогрессии.
3. Угол задан с помощью целочисленных полей gradus - градусов, min - угловых минут, sec - угловых секунд. Реализовать класс, в котором предусмотреть метод ToRadians для перевода в радианы.
4. Дата задана с помощью целочисленных полей day , month, year. Предусмотреть метод IsValid, проверяющий возможна ли заданная дата.
5. В классе создать два целочисленных поля a и b. Реализовать метод NOD для нахождения наибольшего общего делителя для a и b.
6.Целочисленные поля a, b, c, являются сторонами некоторого треугольника. Реализовать метод, проверяющий истинность высказывания: «Треугольник со сторонами a, b, c является прямоугольным»
7. Целочисленные поля x и y представляют собой координаты клетки шахматной доски. Учитывая, что левое нижнее поле доски (1, 1) является черным, реализовать метод, проверяющий истинность высказывания: «Данное поле является белым».
8. Поле left - вещественное число, левая граница диапазона. Поле right - вещественное число, правая граница диапазона. Пара этих числе представляет полуоткрытый интервал [left,right). Реализовать класс, в котором предусмотреть метод rangecheck() - проверку заданного числа на принадлежность диапазону.
9. Комплексное число (a+jb) в алгебраической форме задано полями a и b с помощью метода Polar получить запись комплексного числа в показательной форме.
10. Время задано тремя целочисленными полями hour, min, sec. Реализовать метод увеличения времени на 1 секунду.

Задание №2

Задачи на использование классов и объектов, в которых данные описаны в качестве свойств. Реализовать класс заданной структуры. В нём предусмотреть конструктор для установки начальных значений полей. Создать объект на основе созданного класса. Осуществить использование объекта в программе.
1. Реализовать класс для нахождения площади треугольника. Вещественные свойства a,b,c - стороны треугольника. Метод Square находит площадь.
2. Реализовать класс для проверки исходных данных. Вещественные свойства a,b,c - стороны треугольника. Метод IsValid проверяет корректность введённых данных.
3. Круг на плоскости имеет координаты центра x0,y0 - вещественные свойства. Радиус круга r0 - также задан вещественным свойством. Реализовать метод проверяющий принадлежность точки с координатами (x,y) данному кругу.

4. Вещественное свойство sum представляет собой сумму вложения под процент. Свойство proc - процентную ставку годовых. Реализовать метод, определяющий сумму через n лет.
5. Шахматный король находится на клетке с координатами (x,y) - целочисленные свойства, которые могут принимать значения от 0 до 8. Реализовать метод, возвращающий все возможные ходы.
6. Шахматный ферзь находится на клетке с координатами (x,y) - целочисленные свойства, которые могут принимать значения от 0 до 8. Реализовать метод, возвращающий число возможных ходов.
7. Реализовать класс для нахождения углов треугольника. Вещественные свойства a,b,c - стороны треугольника. Метод Angles находит углы.

Лабораторная 7. Реализация наследования (2 часа)
Задание №1
Задачи на наследование классов, в которых данные описаны в качестве свойств. Реализовать базовый класс заданной структуры, на основе него создать наследующий класс. В нём предусмотреть конструктор для установки начальных значений полей. Создать объект на основе созданного класса. Осуществить использование объекта в программе.
1. Создать класс Angle для работы с углами на плоскости. Предусмотреть перевод из градусной меры в радианную, сложение и вычитание углов с учётом приведения к диапазону 0-360. На основе класса Angle создать класс Triangle для работы с прямоугольным треугольником. Предусмотреть нахождение его площади.
2. Создать класс Money представляющий количество банкнот достоинством 10, 50, 100, 500, 1000, 5000. Предусмотреть метод summa, для вычисления общей суммы. На основе класса Money создать класс Bankomat предусматривающий снятие любой возможной суммы, пополнение запаса.
3. Создать класс Money для работы с денежными суммами в котором для рублей и копеек предусмотрены независимые целочисленные данные. Реализовать метод вывода суммы на экран. На основе класса Money создать класс Good для работы с товаром. Предусмотреть метод, осуществляющий уменьшение цены на заданное число процентов.
4. Создать класс Board для описания шахматной доски. В нём предусмотреть массив 8x8 элементов и метод для перевода цифр 1-8 в буквы A-H и обратно. На основе класса Board создать класс Composition для составления шахматной композиции. В нём предусмотреть возможность добавления/удаления фигур на доску, распечатку композиции.
5. Создать класс Points для хранения координат четырёх точек A, B, C и D на плоскости. В классе предусмотреть возможность распечатки координат каждой точки по отдельности и всех разом. На основе класса Points создать класс Quadrilateral для работы с четырёхугольником. Предусмотреть методы для проверки существования четырёхугольника, нахождения площади и диагоналей.

Лабораторная 8. Использование блочных лямбда-выражений (2 часа)
Задание №1
Задачи на использование блочных лямбда-выражений. Описать делегат с требуемой сигнатурой. Используя блочное лямбда-выражение реализовать основной алгоритм задачи. Осуществить использование делегата в программе с применением введённых пользователем исходных данных.
1.Введены целые положительные числа A и B, такие, что A > B. На отрезке длины A размещено максимально возможное количество отрезков длины B (без наложений). Используя операцию деления нацело, найти количество отрезков B, размещенных на отрезке AB.
2.Введено трехзначное число. Найти сумму и произведение его цифр.

3.Введено трехзначное число. В нем зачеркнули первую слева цифру и приписали ее справа. Вывести полученное число.
4.Дни недели пронумерованы следующим образом: 1 — понедельник, 2 — вторник, …, 6 — суббота, 7 — воскресенье. Дано целое число K, лежащее в диапазоне 1–365, и целое число N, лежащее в диапазоне 1–7. Определить номер дня недели для K-го дня года, если известно, что в этом году 1 января было днем недели с номером N.
5.Введены целые положительные числа A, B, C. На прямоугольнике размера A * B размещено максимально возможное количество квадратов со стороной C (без наложений). Найти количество квадратов, размещенных на прямоугольнике, а также площадь незанятой части прямоугольника.
6.Введен некоторый год (целое положительное число). Определить соответствующий ему номер столетия, учитывая, что, к примеру, началом 20 столетия был 1901 год.

Лабораторная 9. Реализация интерфейсов. Создание Windows приложения. (12 часов)
Задание №1
Задачи на использование базовых компонентов Windows Forms. С помощью визуального конструктора создать обычную форму в которую включить необходимые элементы управления (Label, TextBox, Button, CheckBox, RadioButton, ListBox, ComboBox). Требуется предусмотреть обработку введённых данных с проверкой их корректности и выдачу результата или сообщения об ошибке.
1. Программа для перевода градусов температуры из шкалы Цельсия в шкалу Фаренгейта и наоборот.
2. Программа для расчёта ежемесячного платежа по кредиту, если вводится ставка % годовых, сумма кредита и срок кредита в месяцах.
3. Программа отображающая список простых чисел, поиск которых начинается от введённого пользователем числа. Количество чисел также задаётся пользователем.
4. Программа, раскладывающая введённое натуральное число на простые сомножители.
5. Программа для нахождения наибольшего общего делителя двух натуральных чисел.
6. Программа отображающая текстовое представление введённого числа.
Задание №2
Задачи на использование стандартных диалоговых окон Windows Forms. С помощью визуального конструктора создать обычную форму в которую включить необходимые элементы управления (Label, TextBox, Button, CheckBox, RadioButton, ListBox, ComboBox). Требуется предусмотреть действия с помощью стандартных диалоговых окон (ColorDialog, FontDialog, FolderBrowserDialog, OpenFileDialog, SaveFileDialog) и выдачу данных с учётом выбора.
1. Программа выводящая пример набранного текста с заданной стандартными диалогами гарнитурой шрифта, размером и цветом.
2. Программа, составляющая список файлов по заданному шаблону из указанной папки.
3. Программа составляющая градиентное изображение в графическом поле. Параметры цвета задаются стандартным диалоговым окном.
4. Программа для отображения графических файлов, выбираемых стандартным диалоговым окном.
5. Программа для добавления, редактирования, удаления текстового списка, хранящегося в ListBox с возможностью сохранения и загрузки текста в файле.
6. Программа для создания цветовой настройки приложения (цвет фона, текста, кнопок и т.д.) с возможностью сохранения в выбранном файле и загрузки из него.
Задание №3
Задачи на создание многооконных приложений Windows. С помощью визуального конструктора создать главную форму в которую включить главное меню. Требуется предусмотреть действия по созданию нового документа, загрузке сохранённого

документа, сохранению документа с помощью стандартных диалоговых окон (OpenFileDialog, SaveFileDialog) обработку и отображение данных.
1. Программа для транспонирования матриц
2. Программа - многодокументный текстовый редактор на основе объекта RichTextBox.
3. Программа для просмотра, поворота и отражения растровых изображений в многодокументном интерфейсе.
4. Программа для выполнения графиков данных из строк исходного изображения.
5. Программа для просмотра и масштабирования растровых изображений.
6. Программа - многооконный web-браузер.
7. Справочник функции или команд, открывающий примеры в новом дочернем окне.

Задание №4

Задачи на использование буфера обмена и технологий перетаскивания данных в Windows. С помощью визуального конструктора создать форму в которую включить основной рабочий элемент и необходимые к нему элементы управления. Требуется предусмотреть действия по копированию данных в буфер обмена и вставке из него, перетаскивание данных с помощью Drag and Drop.
1. Программа для транспонирования матриц с возможностью копирования матрицы из одного окна в другое через буфер обмена
2. Текстовый редактор с возможностью открытия файлов перетаскиванием из Проводника.
4. Многодокументный текстовый редактор на основе объекта TextBox с возможностью копирования/вставки фрагмента текста через буфер обмена

Задание №5

Задачи на создание приложений WPF. С помощью визуального конструктора создать форму приложения WPF в которую включить необходимые элементы управления (Label, Button, CheckBox, RadioButton, ListBox, ComboBox). Дизайн внешнего вида формы должен использовать оформление, существенно отличающее его вид от приложений Windows Forms, рекомендовано применение графических и мультимедийных элементов для оформления формы. Требуется предусмотреть обработку введённых данных с проверкой их корректности и выдачу результата или сообщения об ошибке.
1. Программа для нахождения простых чисел в заданном пользователем диапазоне
2. Программа для деления двух чисел с заданной пользователем точностью, которая может существенно превышать возможности встроенных вещественных типов данных.
3. Программа генерирующая последовательность Морса-Туэ
4. Программа для получения последовательности чисел трибоначчи
5. Программа генерирующая изображение кривой Леви

Задание №6

Задачи на создание приложений XBAP - разновидности приложений WPF, ещё называемой браузерными приложениями XAML. С помощью визуального конструктора создать форму приложения XBAP в которую включить необходимые элементы управления (Label, Button, CheckBox, RadioButton, ListBox, ComboBox). Требуется предусмотреть обработку введённых данных с проверкой их корректности и выдачу результата или сообщения об ошибке.
1. Программа для получения последовательности Падована . Число элементов задаётся пользователем.
2. Программа для получения числовой последовательности Каталана . Число элементов задаётся пользователем.
3. Программа для получения последовательности Дийкстры
4. Программа для получения последовательности чисел Шрёдера

Задание №7

Задачи на создание многопоточных программ с интерфейсом Windows Forms. С помощью визуального конструктора создать обычную форму в которую включить необходимые

элементы управления (Label, TextBox, Button, CheckBox, RadioButton, ListBox, ComboBox, ProgressBar). Основной поток программы осуществляет выполнение главной задачи в программе, фоновый поток должен обеспечивать выведение прогресса исполнения главной задачи с помощью ProgressBar. Указанный функционал реализуется с помощью пространства имён System.Threading. Требуется предусмотреть обработку введённых данных с проверкой их корректности и выдачу результата или сообщения об ошибке.
1. Программа для получения списка простых чисел в указанном пользователем интервале.
2. Программа, упорядочивающая содержимое файла по возрастанию.
4. Программа для получения разложения на простые сомножители чисел в указанном пользователем интервале.

Лабораторная 10. Создание простейшего сайта. Web элементы управления. Работа с данными. (2 часа)
Задание: Разработать простейший сайт по выбранной тематике.

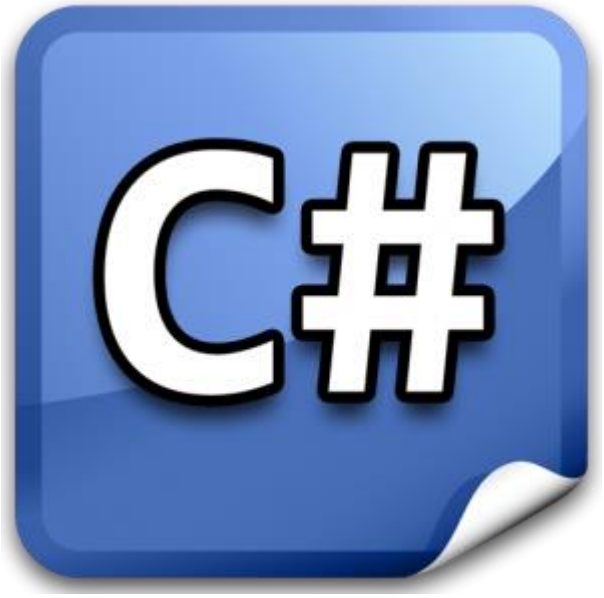
МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Армавирская государственная педагогическая академия

Кафедра информатики и ИТО

Лапшин Н.А.

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА C#



УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

АРМАВИР 2014

Печатается по решению редакционно-издательского совета Армавирской государственной педагогической академии

Лапшин Н.А.
Основы программирования на C#. Учебно-методическое пособие – Армавир: РИО АГПА, 2014, с.

Пособие предназначено для приобретения студентами практических навыков программирования на языке C# в среде MS Visual Studio. В пособии рассматриваются основные конструкции языка программирования, приводятся необходимые теоретические сведения, а также стандартные алгоритмы обработки данных. Все это позволяет рекомендовать его при изучении курсов, включающих программирование на языке C#.

Пособие может быть использовано студентами как очной, так и заочной формы обучения для организации самостоятельной работы. В пособии дан список литературы для дополнительного изучения.

Рецензент:

СОДЕРЖАНИЕ

1. Линейные программы

Типы данных. Описание переменных. Константы: именованные и неименованные

Базовые типы

К базовым типам относятся:

тип целых чисел

int -2147483648 до 2147483647

Int16 -32768..32768

Int32 -2млрд..2млрд

Int64 -9223372036854775808 до 9223372036854775807

Byte 0..255

тип действительных (вещественных) чисел (то есть - с дробной частью). Примеры обозначения действительного числа: -25.000452, 0.24, 4.854E-12

при вводе констант в программе требуют использования специальных суффиксов в конце.

double от $\pm 5{,}0 \times 10^{-324}$ до $\pm 1{,}7 \times 10^{308}$

```
double x = 3.7D;
double m = 8.2;
```

float от $\pm 1{,}5 \times 10^{-45}$ до $\pm 3{,}4 \times 10^{38}$ float x = 3.5F;

символьный тип - Char

Ключевое слово char используется для объявления символа Юникода в диапазоне, указанном в следующей таблице. Символы Юникода — это 10-разрядные символы, которые используются для представления большинства известных письменных языков мира содержит внутри себя всего один символ например 'w' или '#'

Константы типа char могут быть записаны в виде символьных литералов, шестнадцатеричной escape-последовательности или представления Юникода. Кроме того, можно привести коды целых символов. Все следующие операторы объявляют переменную char и инициализируют ее символом X:

```
char c1 = 'Z';        // Буквенный символ
char c2 = '\x0058';   // Шестнадцатеричный код символа
```

строковый тип - string по умолчанию до 2Гбайт например " iit "

Тип данных string — это последовательность, содержащая ни одного или любое число знаков Юникода. В платформе.NET Framework string является псевдонимом для String.

```
string a = "hello";
string b = "h";
```

логический тип - bool (Может принимать два значения Истинно-true Ложно-false)

```
bool f = true;
```

Физически типы данных отличаются друг от друга количеством ячеек памяти (байтов), отводимых для хранения соответствующей переменной. Логическое же отличие проявляется в интерпретации хранящейся информации. Например, пере-

менные типа Char и типа Byte занимают в памяти по одному байту. Однако в первом случае содержимое ячейки памяти интерпретируется как целое беззнаковое число, а во втором - как код (ASCII) символа.

Константы

Константа - это объект, значение которого известно еще до начала работы программы. Константы необходимы для оформления наглядных программ, незаменимы при использовании в тексте программы многократно повторяемых значений, удобны в случае необходимости изменения этих значений сразу во всей программе.

Константа хранит значение, присваиваемое по завершении компиляции программы, и никогда после этого не изменяется. Константы объявляются с помощью ключевого слова const; их использование способствует повышению удобочитаемости кода. В языке существует два вида констант:

неименованные константы (цифры и числа, символы и строки, множества);

именованные константы;

Неименованные константы

Неименованные константы не имеют имен, и потому их не нужно описывать. Тип неименованной константы определяется автоматически, по умолчанию:

любая последовательность цифр (возможно, предваряемая знаком "-" или "+" или разбиваемая одной точкой) воспринимается компилятором как неименованная константа - число (целое или вещественное);

любая последовательность символов, заключенная в апострофы, воспринимается как неименованная константа - строка;

любая последовательность целых чисел, либо символов через запятую, обрамленная квадратными скобками, воспринимается как неименованная константа - множество.

Кроме того, существуют две специальные константы true и false, относящиеся к логическому типу данных. Примерами использования неименованных констант могут послужить следующие операторы:

```
a = -10;
b = 12.075 + x;
c = 'z';
d = "abc" + string44;
```

Именованные константы

Именованные константы, как следует из их названия, должны иметь имя. Стало быть, эти имена необходимо сообщить компилятору, то есть описать в специальном разделе const.

Если не указывать тип константы, то по ее внешнему виду компилятор сам определит, к какому (базовому) типу ее отнести. Любую уже описанную константу можно использовать при объявлении других констант, переменных и типов данных. Вот несколько примеров описания нетипизированных именованных констант:

```
const int speedLimit = 55;
```

```
const double pi = 3.14159265358979323846264338327950;
```

Переменные

В C# переменные объявляются с определенным типом данных и надписью. Если ранее вам приходилось работать со слабо типизированными языками, например JScript, вы привыкли использовать один тип "var" для всех переменных, то в C# необходимо указать тип переменной: int, float, byte, short или другой из более чем 20 различных типов данных. Тип указывает, помимо всего прочего, точный объем памяти, который следует выделить для хранения значения при выполнении приложения. Пример:

```
int i;
int answer = 42;
string greeting = "Hello, World!";
double bigNumber = 1e100;
Console.WriteLine("{0} {1} {2}", answer, greeting, bigNumber);
```

Идентификаторы

Имена операторов, переменных, констант, типов величин, имя самой программы назначаются программистом и называются идентификаторами. Существуют правила, которым должны отвечать все идентификаторы:

идентификатор должен быть уникальным, то есть одним и тем же именем разные объекты не могут быть названы;

идентификатор имеет ограничение по длине (зависит от конкретной реализации языка на компьютере);

идентификатор может состоять только из символов латинского алфавита, цифр и знака подчеркивания ("_"), регистр букв имеет значение;

идентификатор не может начинаться с цифры.

```
//допустимые имена
x  ,  s5  ,  Ax6 ,  D_f  , _Fm5 , xM_5 , _128
```

Имена - это идентификаторы. Любая случайным образом составленная последовательность букв, цифр и знаков подчёркивания с точки зрения грамматики языка идеально подходит на роль имени любого объекта, если только она начинающаяся с буквы. Фрагмент программы, содержащий подобную переменную, будет синтаксически безупречен.

И всё же имеет смысл воспользоваться дополнительной возможностью облегчить восприятие и понимание последовательностей операторов. Для этого достаточно закодировать с помощью имён содержательную информацию.

Желательно создавать составные осмысленные имена. При создании подобных имён в одно слово можно "втиснуть" предложение, которое в доступной форме представит информацию о типе объекта, его назначении и особенностях использования.

Вы можете давать программным объектам любые имена, но необходимо, чтобы они отличались от зарезервированных слов, используемых языком, потому что компилятор все равно не примет переменные с "чужими" именами.

Список наиболее часто встречающихся зарезервированных слов:

abstract event new struct as explicit null switch base extern object this bool false operator throw break finally out true byte fixed override try case float params typeof catch for private uint char foreach protected ulong checked goto public unchecked class if readonly unsafe const implicit ref ushort continue in return using decimal int sbyte virtual default interface sealed volatile delegate internal short void do is sizeof while double lock stackalloc else long static enum namespace string

Основная структура программы

Для консольного приложения автоматически создается структура:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication1
{
 class Program
  {
   static void Main(string[] args)
     {
      //здесь писать текст программы
     }
  }
}
```

Имя программы выбирается программистом самостоятельно в соответствии с правилами построения идентификаторов и совпадает с именем файла на диске.

Все объекты, не являющиеся зарезервированными в языке программирования, наличие которых обусловлено инициативой программиста, перед первым использованием в программе должны быть описаны. Это производится для того, чтобы компьютер перед выполнением программы зарезервировал память под соответствующие объекты и поставил в соответствие этим участкам памяти идентификаторы.

Комментарии

Помимо отступов, большие логически замкнутые блоки программы удобно разделять строками-комментариями, содержащими информацию о смысле последующего блока.

```
// Двойная косая черта до конца строки
/*  многострочный
    комментарий */
```

Во время компилирования программы комментарии игнорируются. Следовательно, их можно добавлять в любом месте программы. Можно даже разорвать оператор вставкой комментария.

Оператор присваивания

Самым простым действием над переменной является занесение в нее величины соответствующего типа. Иногда говорят об этом, как о присвоении переменной конкретного значения. Такая команда (оператор) в общем виде выглядит следующим образом:

<Имя переменной> = <Выражение>;

Тип переменной и тип выражения должны быть совместимы. Выражение, указанное справа от знака "=", должно приводить к значению того же типа, какого и сама переменная, или типа, совместимого с переменной относительно команды присваивания. Например, переменной типа double можно присвоить значение типа Int или byte (впрочем, наоборот делать нельзя). Выражение будет сначала вычислено, затем, его результат будет положен в ячейки памяти, отведенные для переменной.

Выражение - это конструкция, возвращающая значение. Например

```
A = X;          // переменная
B =15;          // целая константа
D = Math.Sin(X); // вызов функции
E = X * Y;      // произведение  X на Y
F = Z / (1 - Z); // отношение Z к  (1 - Z)
G =(X == 1.5);   // логическое выражение
H = ! J;        // отрицание логической переменной
```

Также можно делать множественное присваивание (во все переменные заносится самое правое значение):

```
a = b = c = d = 18;
```

Операторы ввода-вывода

Как мы уже говорили, любой алгоритм должен быть результативным. В общем случае это означает, что он должен сообщать результат своей работы потребителю: пользователю-человеку или другой программе (например, программе управления принтером). Мы не будем описывать здесь внутренние автоматические процессы, использующие сигналы непрерывно функционирующих программ, а сосредоточим внимание на взаимодействии программы и человека, то есть на процессах ввода информации с клавиатуры и вывода ее на экран.

В программировании существует специальное понятие консоль, которое обозначает клавиатуру при вводе и монитор при выводе.

Операторы ввода

Для того чтобы получить данные, вводимые пользователем вручную (то есть с консоли), применяются команды

<строковая переменная>=Console.ReadLine();

Для того чтобы преобразовать к нужному типу данных используем объект Convert

<переменная целого типа> = Convert.ToInt32(<строковая переменная>);

<переменная действ типа> = Convert.ToDouble(<строковая переменная>);

<переменная логического типа> = Convert.ToBoolean(<строковая переменная>);

<переменная целого типа> = Convert.ToByte(<строковая переменная>);
<переменная символьного типа> = Convert.ToChar(<строковая переменная>);
<переменная целого типа> = Convert.ToInt64(<строковая переменная>);
Пример

```
Int32 x; Double y; string s;
s = Console.ReadLine();
x = Convert.ToInt32(s);
//без промежуточной строковой переменной
y = Convert.ToDouble(Console.ReadLine());
```

Типы вводимых значений должны совпадать с типами указанных переменных, иначе возникает ошибка. Поэтому нужно внимательно следить за правильностью вводимых данных.

Для того, чтобы обеспечить ожидание ввода любой клавиши в конце программы используют

```
Console.ReadKey();
```

Операторы вывода

Сделаем одно важное замечание: ожидая от человека ввода с клавиатуры, не нужно полагать, что он окажется ясновидящим и просто по мерцанию курсора на черном экране догадается, какого типа переменная нужна ожидающей программе. Старайтесь всегда придерживаться правила: "лысый" ввод недопустим! Перед тем как считывать что-либо с консоли, необходимо сообщить пользователю, что именно он должен ввести: смысл вводимой информации, тип данных, максимальное и минимальное допустимые значения и т.п.

Примером неплохого приглашения служит, скажем, такая строчка: Введите два вещественных числа (0.1<1000000) - длины катетов.

Впрочем, и ее можно улучшить, сообщив пользователю не только допустимый диапазон ввода, но и ожидаемую точность (количество знаков после запятой).

Средства, позволяющие организовать выдачу информации на экран, мы здесь и рассмотрим.

Для того чтобы вывести на экран какое-либо сообщение, воспользуйтесь процедурой Console.Write или Console.WriteLine

Первая из них, напечатав на экране все, о чем ее просили, оставит курсор в конце выведенной строки, а вторая переведет его в начало следующей строчки.

```
Console.WriteLine(s);// переменная
Console.WriteLine(55.3);// константа
Console.WriteLine(y*3+7);// выражение
Console.Write(z);// переменная
Console.Write(-5.3);// константа
Console.Write(i*3+7/j);// выражение
```

Чтобы при выводе данных они не склеивались, нужно позаботиться о пробелах между выводимыми переменными, также можно выводить списком несколько переменных:

Console.WriteLine("Это число A={0} далее B={1} и, наконец их сумма {2}", a, b, a + b);

Для форматирования числовых результатов можно использовать метод String.Format или метод Console.Write, вызывающий метод String.Format. Формат задается с помощью строк формата. Спецификация формата следующая: {N,M:Axx}, где N указывает позицию элемента в списке выводимых переменных (нумерация начинается с 0); M - задаёт ширину области, в которую будет помещено форматированное значение, если M отсутствует или отрицательно, значение будет выровнено влево, в противном случае - вправо; Axx - является необязательной строкой форматирующих кодов, которые используются для управениея форматированием чисел, даты и времени, денежных знаков и т.д.

В следующей таблице приведены поддерживаемые строки стандартных форматов. Строка формата принимает следующую форму: Axx, где A — описатель формата, а xx — описатель точности. Описатель формата управляет типом форматирования, применяемым к числовому значению, а описатель точности управляет количеством значащих цифр или десятичных знаков форматированного результата.

C или c Валюта
Console.Write("{0:C}", 2.5); Console.Write("{0:C}", -2.5); // $2.50 ($2.50)
D или d Десятичный формат
Console.Write("{0:D5}", 25); // 00025
E или e Инженерный формат
Console.Write("{0:E}", 250000); //2.500000E+005
F или f Формат с фиксированной запятой
Console.Write("{0:F2}", 25); Console.Write("{0:F0}", 25); // 25.00 25
G или g Общий формат
Console.Write("{0:G}", 2.5); // 2.5
N или n Числовой формат
Console.Write("{0:N}", 2500000); //2,500,000.00
X или x Шестнадцатеричный формат
Console.Write("{0:X}", 250); Console.Write("{0:X}", 0xffff); // FA FFFF
Пример
```
Int32 x; Double y; string s;
Console.Write("Введите X=");
s = Console.ReadLine();
x = Convert.ToInt32(s);
Console.Write("Введите Y=");
s = Console.ReadLine();
y = Convert.ToDouble(s);
Console.WriteLine("Произведение X*Y= {0,9:g}" ,x*y);
Console.ReadKey();
```
Пример простейшей программы
```
string s1;
Console.Write("Введите ваше имя ");
s1 = Console.ReadLine();
```

```
Console.WriteLine("Мы рады вас приветствовать, {0:g}, в нашей программе",s1);
Console.ReadKey();
```

Арифметические операторы

Для каждого типа данных определены действия, применимые к его значениям. Итак, поговорим теперь об операциях - стандартных действиях, разрешенных для переменных того или иного базового типа данных.

Бинарные

Оператор	Операция	Тип операндов	Тип рез.	Пример
+	сложение	целый, вещ.	целый, вещ.	A =X + Y;
-	вычитание	целый, вещ.	целый, вещ.	A =Result-1;
*	произведение	целый, вещ.	целый, вещ.	A =P * I;

Оператор деления (/) делит первый операнд на второй. Все числовые типы имеют предопределенные операторы деления.

При делении двух целых чисел результат всегда является целочисленным. Например, результат деления 5 на 2 — 2. Чтобы получить частное в виде рационального числа или дроби, присвойте делителю или делимому тип float или double. Чтобы явно присвоить тип, можно поместить десятичный разделитель после числа, как показано в следующем примере.

Пример 1:

```
Console.WriteLine(5 / 2);
Console.WriteLine(5D / 2);
Console.WriteLine(5 / 2.1);
Console.WriteLine(5.1 / 2);
Console.WriteLine(-5 / 2);
/* напечатается
2
2.5
2.38095238095238
2.55
-2  */
```

Оператор % - остаток от целочисленного деления, например A=Y % 6;

Пример 2:

```
Console.WriteLine(24 % 6);
Console.WriteLine(24 % 7);
Console.WriteLine(7 % 7);
Console.WriteLine(8 % 12);
/*напечатается
0
3
0
8  */
```

Унарные

Оператор Операция Тип операндов Тип рез. Пример
+ (унарный) знак плюс целый, вещ. целый, вещ. A =+7;
- (унарный) знак минус целый, вещ. целый, вещ. A =-X;

Оператор увеличения (++) увеличивает свой операнд на 1. Оператор увеличения может находиться как до, так и после операнда:

Первой формой является префиксная операция увеличения. Результатом этой операции является значение операнда после его увеличения. Второй формой является постфиксная операция увеличения. Результатом этой операции является значение операнда до его увеличения. Числовые типы и типы перечисления имеют предопределенные операторы увеличения. Типы, определенные пользователем, могут вызвать перегрузку оператора ++. Операции с целыми типами обычно разрешены в перечислениях.

```
double x;
x = 1.5;
Console.WriteLine(++x);  // напечатается 2.5
x = 1.5;
Console.WriteLine(x++);  // напечатается 1.5
Console.WriteLine(x);    // напечатается 2.5
```

Оператор уменьшения (--) уменьшает свой операнд на 1. Оператор уменьшения может находиться как до, так и после операнда: и --variable и variable--. Первой формой является префиксная операция уменьшения. Результатом этой операции является значение операнда после его уменьшения. Второй формой является постфиксная операция уменьшения. Результатом этой операции является значение операнда до его уменьшения.

Комбинированные операторы присваивания

Также имеется возможность комбинировать оператор присваивания с арифметическими операторами для образования составных операторов присваивания:

```
+=
-=
*=
/=
%=
```

Пример:

```
i += 7 * j;  // Эквивалентно  i = i + 7 * j;
m /= 3 + k;  // Эквивалентно  m = m / (3 + k);
```

Логические операторы

Логические операции (&&, ||, !, ^) применимы только к значениям типа bool. Их результатом также служат величины типа bool.

Оператор Операция Пример
! лог. отрицание F=! C;
&& лог. умножение F=D && T;
|| лог. сложение F=A || B;

^ исключающее или F=A ^ B;

Таблица истинности

	О п е р а ц и и		
Операнды	||	& &	^
true,true	true	true	false
true,false	true	false	true
false,true	true	false	true
false,false	false	false	false

!	
true	false
false	true

Наряду с && и || можно использовать & и | соответственно. Отличие & от && состоит в том, что & может выполнять побитовую операцию И для целых чисел, но также работает и с логическим типом, например 5 & 3 = 1 .

Отличие | от || состоит в том, что | может выполнять побитовую операцию ИЛИ для целых чисел, но также работает и с логическим типом.

Операторы отношения

Операции отношения (==, !=, >, <, <=, >=) применимы ко всем базовым типам. Их результатами также являются значения типа bool.

Оператор	Операция	Тип операндов	Тип рез.	Пример
==	равно	простой	bool	F= C == 7;
!=	не равно	простой	bool	F= X != Y;
<	меньше	простой	bool	F= X < Y;
>	больше	простой	bool	F= L > 0;
<=	меньше или равно	простой	bool	F= C <= I;
>=	больше или равно	простой	bool	F= I >= 1;

Пример: Попадает ли X в интервал от 0 до 1 или в интервал от 3 до 5

```
z = x > 0 && x < 1 || x > 3 && x < 5;
```

Стандартные функции

Math.Abs(x) Модуль числа

Math.Exp(x) Вычисляется e в степени x

Math.Floor(x) Возвращает наибольшее целое число, которое меньше или равно указанному числу.

Math.Log(x) Натуральный логарифм числа x

Math.Log10(x) Десятичный логарифм числа x.

Math.Pow(B,E) Возводит B в любую степень

Math.Round(x) Округляет значение до ближайшего целого или указанного количества десятичных знаков

Math.Truncate(x) Вычисляет целую часть числа

Math.Ceiling(x) Возвращает наименьшее целое число, которое больше или равно заданному числу.

Math.PI Число 3,1415...
Math.E Число 2,7128...
Math.Sin(x) Math.Cos(x) Синус, косинус в радианах
Math.Atan(x) Арктангенс в радианах
Math.Sqrt(x) Квадратный корень числа X.
Пример: Реализовать в виде оператора следующее выражение:

$$r = \frac{(a+b)\sin x^2 + ab}{\sqrt{\cos^2\left(x + \frac{\pi}{2}\right) - b}}$$

```
r =((a+b)*Math.Sin(Math.Pow(x,2))+a*b)/
  Math.Sqrt(Math.Pow(Math.Cos(x+Math.PI/2),2)-b)
```

Лабораторная работа №1

Целью лабораторного занятия является приобретение навыков практического применения знаний для создания простейших программ с неразветвлённым вычислительным процессом.

В **задачи** лабораторного занятия входят:

- закрепление знаний студентов в процессе выполнения анализа алгоритма решаемой задачи;
- приобретение умений и навыков использования современных сред разработки программного обеспечения.

Формой выполнения лабораторной работы является поиск оптимальных алгоритмов решения программных задач с целью изучения основ программирования с формированием выводов и заключений.

Задание на лабораторную: Ознакомиться со средой **Microsoft Visual C#** , решить 3 задачи по вариантам.

Примеры решения задач

Пример №1

Составить программу для вычисления значения y(x) и для некоторых x произвести вычисления

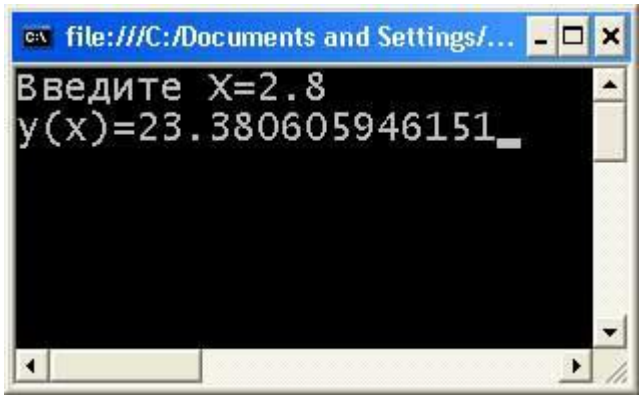
$$y = \frac{\frac{x}{2} \cdot e^x + \ln(1 + e^x)}{\sin^2 x - \sin\sqrt{x}}$$

```
Double x,y; string s;          //Описываем переменные
 Console.Write("Введите x=");  //Выдаём приглашение для ввода x
 s = Console.ReadLine();       //Вводим строку s
 x = Convert.ToDouble(s);      //Преобразовываем строку в x

 y = (x / 2 * Math.Exp(x) + Math.Log(1 + Math.Exp(x))) /
    (Math.Pow(Math.Sin(x), 2)
    + Math.Sin(Math.Sqrt(x)));//Вычисляем y согласно формуле
```

Console.Write("y(x)={0}",y); //Печатаем полученный результат

Console.ReadKey(); //Ожидаем нажатия на любую клавишу в конце программы



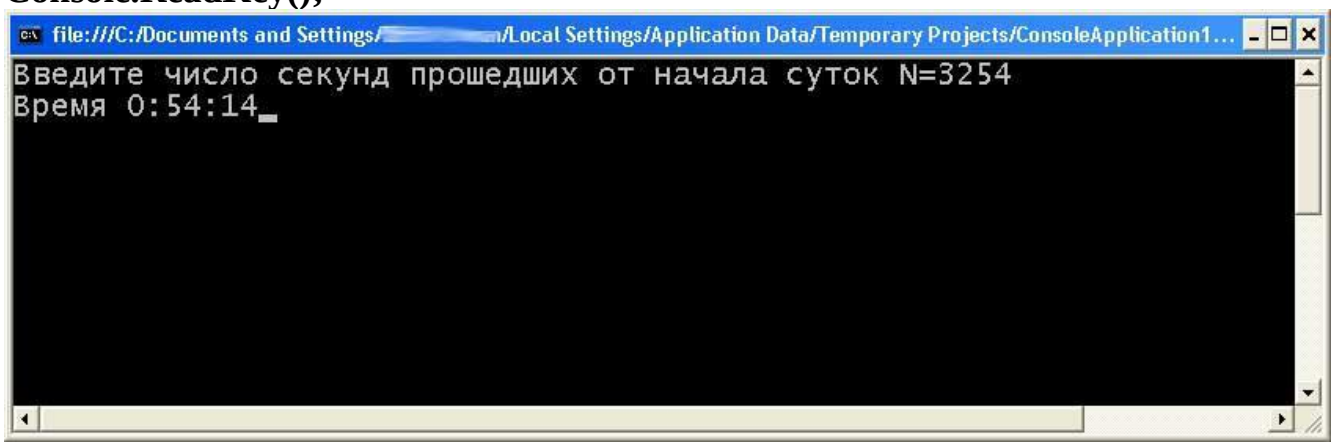
Пример №2

С начала суток прошло N секунд (N — целое). Записать текущее время выраженное в часах, минутах, секундах. Даже если прошло несколько суток, отобразить время последних суток.

```
int N,hours,minutes,seconds; string s;
  Console.Write("Введите число секунд прошедших от начала суток N=");
  s = Console.ReadLine(); //вводим произвольное число секунд
  N = Convert.ToInt32(s);

  hours = N / 3600 % 24;  // находим число часов, и оставляем только
                          //часы в последних сутках, если их больше 24
  minutes = N % 3600 / 60; // определяем число минут
  seconds = N % 3600 % 60; // определяем число секунд

  Console.Write("Время {0}:{1}:{2}", hours, minutes, seconds);
Console.ReadKey();
```



Задание №1

Задачи на **ввод и вывод данных , оператор присваивания , арифметические операторы , стандартные функции** . Все входные и выходные данные в заданиях этой группы являются **вещественными числами** . Не требуется выполнять проверку введённых пользователем данных.

Задачи по вариантам

- 1. Даны катеты прямоугольного треугольника a и b. Найти его гипотенузу c и периметр P.
- 2. Даны два круга с общим центром и радиусами R1 и R2 (R1 > R2). Найти площади этих кругов S1 и S2, а также площадь S3 кольца, внешний радиус которого равен R1, а внутренний радиус равен R2.
- 3. Дана длина L окружности. Найти ее радиус R и площадь S круга, ограниченного этой окружностью.
- 4. Даны три точки A, B, C на числовой оси. Точка C расположена между точками A и B. Найти произведение длин отрезков AC и BC.
- 5. Даны координаты двух противоположных вершин прямоугольника: (x1, y1), (x2, y2). Стороны прямоугольника параллельны осям координат. Найти периметр и площадь данного прямоугольника.
- 6. Даны координаты трех вершин треугольника: (x1, y1), (x2, y2), (x3, y3). Найти его периметр и площадь, используя формулу для расстояния между двумя точками на плоскости. Для нахождения площади треугольника со сторонами a, b, c использовать формулу Герона:$S =(p\cdot(p – a)\cdot(p – b)\cdot(p – c))^{1/2}$, где p = (a + b + c)/2 — полупериметр.
- 7. Дано значение температуры T в градусах Фаренгейта. Определить значение этой же температуры в градусах Цельсия. Температура по Цельсию TC и температура по Фаренгейту TF связаны следующим соотношением:TC = (TF – 32)·5/9.
- 8. Известно, что X кг шоколадных конфет стоит A рублей, а Y кг ирисок стоит B рублей. Определить, сколько стоит 1 кг шоколадных конфет, 1 кг ирисок, а также во сколько раз шоколадные конфеты дороже ирисок.
- 9. Скорость лодки в стоячей воде V км/ч, скорость течения реки U км/ч (U < V). Время движения лодки по озеру T1 ч, а по реке (против течения) — T2 ч. Определить путь S, пройденный лодкой (путь = время · скорость). Учесть, что при движении против течения скорость лодки уменьшается на величину скорости течения.
- 10. Скорость первого автомобиля V1 км/ч, второго — V2 км/ч, расстояние между ними S км. Определить расстояние между ними через T часов, если автомобили первоначально движутся навстречу друг другу.

Задание №2

Задачи на **целочисленные операции** . Все входные и выходные данные в заданиях этой группы являются целыми числами. Все числа, для которых указано количество цифр (двузначное число, трехзначное число и т. д.), считаются положительными. Не требуется выполнять проверку введённых пользователем данных.

Задачи по вариантам

- 1. Даны целые положительные числа A и B (A > B). На отрезке длины A размещено максимально возможное количество отрезков длины B (без наложений). Используя операцию деления нацело, найти количество отрезков B, размещенных на отрезке AB.
- 2. Дано трехзначное число. Найти сумму и произведение его цифр.
- 3. Дано трехзначное число. Вывести число, полученное при прочтении исходного числа справа налево.
- 4. Дано трехзначное число. В нем зачеркнули первую слева цифру и приписали ее справа. Вывести полученное число.
- 5. Дано трехзначное число. В нем зачеркнули первую справа цифру и приписали ее слева. Вывести полученное число.
- 6. Дано трехзначное число. Вывести число, полученное при перестановке цифр сотен и десятков исходного числа (например, 123 перейдет в 213).
- 7. Дано целое число, большее 999. Используя только целочисленные операции, найти цифру, соответствующую разряду сотен в записи этого числа.
- 8. Дни недели пронумерованы следующим образом: 1 — понедельник, 2 — вторник, …, 6 — суббота, 7 — воскресенье. Дано целое число K, лежащее в диапазоне 1–365, и целое число N, лежащее в диапазоне 1–7. Определить номер дня недели для K-го дня года, если известно, что в этом году 1 января было днем недели с номером N.
- 9. Даны целые положительные числа A, B, C. На прямоугольнике размера A * B размещено максимально возможное количество квадратов со стороной C (без наложений). Найти количество квадратов, размещенных на прямоугольнике, а также площадь незанятой части прямоугольника.
- 10. Дан номер некоторого года (целое положительное число). Определить соответствующий ему номер столетия, учитывая, что, к примеру, началом 20 столетия был 1901 год.

Задание №3

Задачи на использование **логических операторов , операторов отношения .** Во всех заданиях данной группы требуется вывести логическое значение True, если приведенное высказывание для предложенных исходных данных является истинным, и значение False в противном случае. Все числа, для которых указано количество цифр (двузначное число, трехзначное число и т. д.), считаются целыми положительными. Не требуется выполнять проверку введённых пользователем данных. **Использование IF и оператора "?" недопустимо.**

Задачи по вариантам

- 1. Даны координаты двух различных полей шахматной доски x1, y1, x2, y2 (целые числа, лежащие в диапазоне 1–8). Проверить истинность высказывания: «Данные поля имеют одинаковый цвет». Если пользователь введёт дважды координаты одной и той же клетки считать решение задачи ложью.
- 2. Даны координаты двух различных полей шахматной доски x1, y1, x2, y2 (целые числа, лежащие в диапазоне 1–8). Проверить истинность высказывания: «Ладья за один ход может перейти с одного поля на другое». Если пользователь

введёт дважды координаты одной и той же клетки считать решение задачи ложью.

- 3. Даны координаты двух различных полей шахматной доски x1, y1, x2, y2 (целые числа, лежащие в диапазоне 1–8). Проверить истинность высказывания: «Король за один ход может перейти с одного поля на другое». Если пользователь введёт дважды координаты одной и той же клетки считать решение задачи ложью.
- 4. Даны координаты двух различных полей шахматной доски x1, y1, x2, y2 (целые числа, лежащие в диапазоне 1–8). Проверить истинность высказывания: «Слон за один ход может перейти с одного поля на другое». Если пользователь введёт дважды координаты одной и той же клетки считать решение задачи ложью.
- 5. Даны координаты двух различных полей шахматной доски x1, y1, x2, y2 (целые числа, лежащие в диапазоне 1–8). Проверить истинность высказывания: «Ферзь за один ход может перейти с одного поля на другое». Если пользователь введёт дважды координаты одной и той же клетки считать решение задачи ложью.
- 6. Даны координаты двух различных полей шахматной доски x1, y1, x2, y2 (целые числа, лежащие в диапазоне 1–8). Проверить истинность высказывания: «Конь за один ход может перейти с одного поля на другое». Если пользователь введёт дважды координаты одной и той же клетки считать решение задачи ложью.
- 7. Даны целые числа a, b, c, являющиеся сторонами некоторого треугольника. Проверить истинность высказывания: «Треугольник со сторонами a, b, c является прямоугольным».
- 8. Дано трехзначное число. Проверить истинность высказывания: «Цифры данного числа образуют возрастающую или убывающую последовательность».
- 9. Дано целое положительное число. Проверить истинность высказывания: «Данное число является нечетным трехзначным».
- 10. Даны координаты поля шахматной доски x, y (целые числа, лежащие в диапазоне 1–8). Учитывая, что левое нижнее поле доски (1, 1) является черным, проверить истинность высказывания: «Данное поле является белым».

2. Ветвления

Оператор условия (if)

Условные операторы позволяют осуществить ветвление алгоритма и делают возможность выбрать для выполнения один из операторов.

Синтаксис оператора if можно представить следующим образом:

if (выражение_лог_типа) оператор; //сокращенная форма

if (выражение_лог_типа) оператор1; else оператор2; // полная форма

В выражении должен получаться результат, имеющий логический тип. Если результатом выражения является истинное значение (True), то выполняется оператор, следующий за условием в скобках. Если результатом выражения является значение False и присутствует ключевое слово else, то выполнятся оператор, следующий за ключевым словом else. Если ключевое слово else отсутствует, то никакой оператор не выполняется. В предшествующем else операторе точка с запятой указывается. В общем случае ключевое слово else связывается с ближайшим ключевым словом if, которое еще не связано с ключевым словом else. Если вместо указанных операторов1,2 требуется выполнить несколько операторов используются операторные скобки {}.

Пример 1:

```
if (x < 1.5)
        z = z + y;
else
        z = 3.4;
```

Пример 2: Даны a,b,c - коэффициенты квадратного уравнения. Получить корни уравнения.

```
Double a,b,c,D,x1,x2; string s;          //Описываем переменные
Console.Write("Введите A=");  //Выдаём приглашение для ввода a
s = Console.ReadLine();       //Вводим строку s
a = Convert.ToDouble(s);      //Преобразовываем строку в a
Console.Write("Введите B=");  //Выдаём приглашение для ввода b
s = Console.ReadLine();       //Вводим строку s
b = Convert.ToDouble(s);      //Преобразовываем строку в b
Console.Write("Введите C=");  //Выдаём приглашение для ввода c
s = Console.ReadLine();       //Вводим строку s
c = Convert.ToDouble(s);      //Преобразовываем строку в c
D = Math.Pow(b, 2) - 4 * a * c; //Определяем дискриминант
if (D > 0)   // если дискриминант положительный будет два корня
  {
    x1 = (-b + Math.Sqrt(D)) / 2 / a;
    x2 = (-b - Math.Sqrt(D)) / 2 / a;
    Console.WriteLine("Два действительных корня x1={0}, x2={1}", x1, x2);
  }
else
   if (D==0)  //если дискриминант равен нулю будет всего один корень
```

```
        {
          x1 = -b / 2 / a;
          Console.WriteLine("Единственный действительный корень x={0}", x1);
        }
    else            //все остальные случаи ,
        {          //т.е. дискриминант отрицательный - решений нет
          Console.WriteLine("Действительных корней нет");
        }
Console.ReadKey();
```

Оператор ?

Оператор ? относится к числу самых примечательных в C#. Он представляет собой условный оператор и часто используется вместо определенных видов конструкций if-then-else. Оператор ? иногда еще называют тернарным, поскольку для него требуются три операнда. Ниже приведена общая форма этого оператора. Синтаксис:

Выражение1 ? Выражение2 : Выражение3;

Здесь Выражение1 должно относиться к типу bool, а Выражение2 и Выражение3 — к одному и тому же типу. Обратите внимание на применение двоеточия и его местоположение в операторе ?.

Значение выражения ? определяется следующим образом. Сначала вычисляется Выражение1. Если оно истинно, то вычисляется Выражение2, а полученный результат определяет значение всего выражения ? в целом. Если же Выражение1 оказывается ложным, то вычисляется Выражение3, и его значение становится общим для всего выражения ?. Рассмотрим следующий пример, в котором переменной absval присваивается значение переменной val.

```
absval = val < 0 ? -val : val; // получить абсолютное значение переменной val
```

В данном примере переменной absval присваивается значение переменной val, если оно больше или равно нулю. Если же значение переменной val отрицательно, то переменной absval присваивается результат отрицания этого значения, что в итоге дает положительное значение.

Оператор варианта (switch)

Оператор варианта **switch** состоит из выражения (переключателя) и списка операторов, каждому из которых предшествует одна или более констант (они называются константами выбора) или ключевое слово **default**. Все константы выбора предваряются ключевым словом **case**, должны быть уникальными и иметь тип, совместимый с типом переключателя.

Пример 1: Выдать введённое число в словесной интерпретации

```
int A; string s;
Console.Write("Введите A=");
s = Console.ReadLine();
A = Convert.ToInt32(s);
switch (A)
```

```
{
 case 1:  Console.WriteLine("Один");  break;
 case 2:  Console.WriteLine("Два");   break;
 case 3:  Console.WriteLine("Три"); break;
 case 4:  Console.WriteLine("Четыре"); break;
 default: Console.WriteLine("Остальные числа"); break;
}
```

Управление передается оператору **case**, совпадающему со значением оператора **switch**. Оператор **switch** может включать любое количество экземпляров **case**, но два оператора **case** не могут иметь одинаковое значение. Выполнение текста оператора начинается с выбранного оператора и продолжается до тех пор, пока оператор **break** не передаст управление за пределы текста **case**. Оператор перехода, такой как **break**, требуется после каждого блока **case**, включая последний блок, вне зависимости от того, какой из двух операторов — **case** или **default** — там использован. Язык C# (в отличие от оператора switch в языке C++) не поддерживает неявное "проваливание" от одной подписи оператора **case** к другой, однако есть одно исключение. Исключением является случай, когда оператор case не имеет кода.

3. Циклы

Оператор цикла с параметром (for)

Цикл **for** повторно выполняет оператор или блок операторов, пока определенное выражение не примет значение **false**. Цикл for удобно использовать для итераций в массивах и для последовательной обработки. В следующем примере значение i печатается на экран, и затем увеличивается на 1 за каждое прохождение цикла. Рекомендуется использовать оператор **for** в тех случаях, когда заранее известно число повторений.

```
int i;
for (i = 1; i <= 5; i++)
      Console.WriteLine(i);
```

Инструкция **break** прекращает выполнение цикла и управление передаётся за цикл. Инструкция **continue** выполняет пропускание операторов в теле цикла идущих ниже, осуществляется вычисление следующего параметра цикла и его дальнейшее повторение.

Пример: Дано целое число N > 0. Найти сумму $1^1 + 2^2 + \ldots + N^N$, не используя функций непосредственного возведения в степень.

```
int N, i,j, sum, s1; string s;
Console.Write("Введите N=");
s = Console.ReadLine();
N = Convert.ToInt32(s);
sum=0;
for (i=1;i<=N;i++)
  {
   s1=i;
   for  (j=1;j < i;j++) s1=s1*i;
   sum = sum + s1;
  }
Console.WriteLine("Сумма ряда равна {0} ", sum);
Console.ReadKey();
```

Оператор цикла с постусловием (do - while)

Оператор **do** повторно выполняет оператор или блок операторов, заключенных в {}, пока определенное выражение не примет значение **false**. В следующем примере операторы цикла **do - while** выполняются до тех пор, пока пользователь не введёт значение из допустимого диапазона. Если в теле цикла всего один оператор, фигурные скобки ставить не нужно.

```
int N; string s;
 do
  {
   Console.Write("Введите N в интервале 0..10  N=");
   s = Console.ReadLine();
   N = Convert.ToInt32(s);
```

```
    }
  while (N<0|| N>10);
Console.WriteLine("Введённое число равно {0} ", N);
Console.ReadKey();
```

Цикл **do-while** выполняется **как минимум один раз**, так как вычисление значения условного выражения выполняется после тела цикла. В любой точке блока **do-while** цикл можно разорвать с помощью оператора **break**. Для перехода непосредственно к оператору вычисления выражения **while** используется оператор **continue**; если выражение имеет значение **true**, выполнение продолжается в первом операторе цикла. Если выражение имеет значение false, выполнение продолжается в первом операторе после цикла **do-while**.

Оператор цикла с предусловием (while)

Оператор **while** выполняет оператор или блок операторов, пока определенное выражение не примет значение **false**. Поскольку перед каждым выполнением цикла выражение **while** тестируется, цикл **while**выполняется от нуля до нескольких раз. Это отличает его от цикла **do**, который выполняется от одного до нескольких раз.

Цикл **while** может быть прерван оператором **break**, который передает управление за пределы цикла. Чтобы передать управление на следующую итерацию без выхода из цикла, используйте оператор **continue**.

Пример: Внесённая в банк сумма равна S, процентная ставка N годовых, определить с помощью цикла while на сколько лет нужно сделать размещение чтобы получить сумму F.

```
double S, F, N; string s; int i;
Console.Write("Введите начальную сумму  S=");
s = Console.ReadLine();
S = Convert.ToDouble(s);
Console.Write("Введите процентную ставку годовых (%)  N=");
s = Console.ReadLine();
N = Convert.ToDouble(s);
Console.Write("Введите желаемую сумму  F=");
s = Console.ReadLine();
F = Convert.ToDouble(s);
i = 0; //счётчик годов
while (S < F)
{
   S = S * (1 + N / 100);
   i++;
}
Console.WriteLine("Нужную сумму придётся ждать {0} лет, она составит {1} ",
i,S);
Console.ReadKey();
```

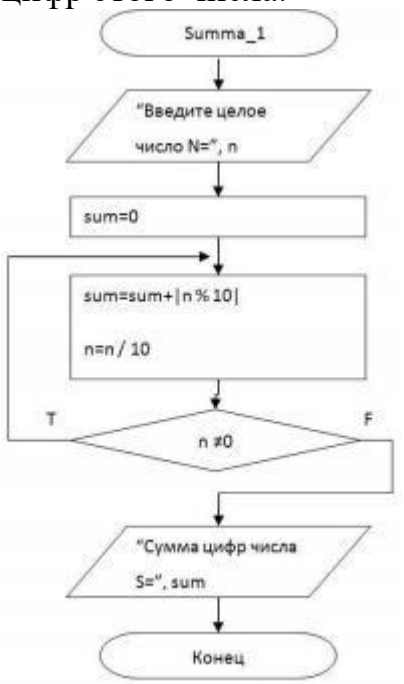
Лабораторная работа №2

Задание на лабораторную: В среде **Microsoft Visual C#** , решить 5 задач по вариантам и оформить отчёт.

Пример решения задач

Пример №1

Написать программу, которая считывает введённое пользователем с клавиатуры целое число (использовать переменную целого типа) и выдает на экран дисплея сумму цифр этого числа.



Блок-схема для примера 1

Решение

Описание алгоритма: поскольку программа должна суммировать цифры числа, предполагается выполнение циклических действий, так как в числе может быть несколько разрядов. Из всех операторов цикла лучше всего подходит цикл с постусловием, который выполняется как минимум один раз: это его свойство очень важно для нас, так как в любом числе хотя бы один разряд всегда есть.

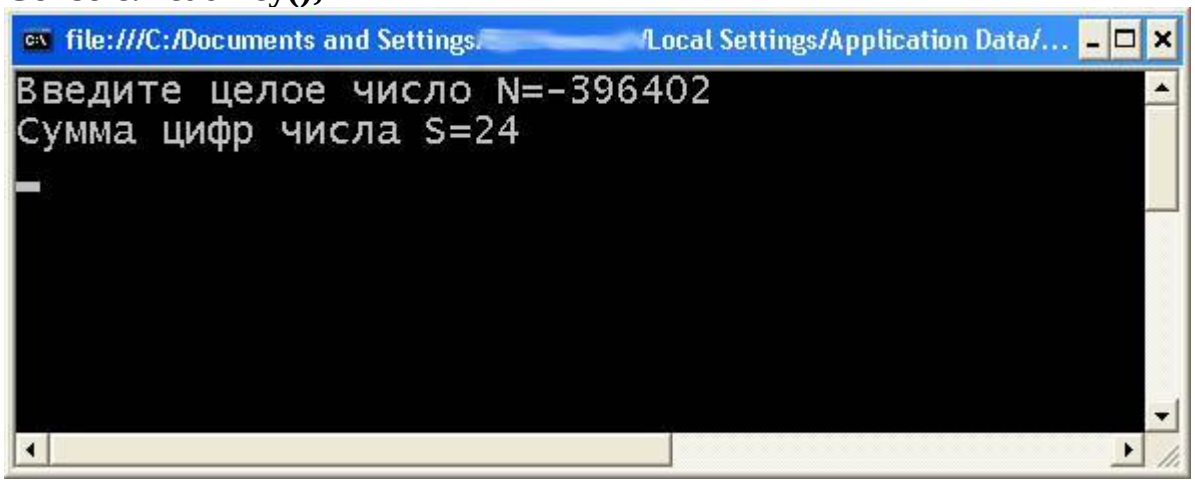
В цикле будем суммировать разряды числа, на каждом шаге выделяя по одному разряду. Выделение самого младшего разряда числа (самой правой цифры числа) будем осуществлять нахождением остатка от деления на 10. Поскольку введённое число может быть отрицательным, то и остаток от деления нужно брать по модулю на каждом шаге. Далее необходимо отбросить самый младший разряд числа, для этого воспользуемся целочисленным делением на 10. Цикл будем повторять до тех пор, пока все разряды числа не закончатся, то есть в результате деления на 10 останется нуль.

```
int n, sum; string s; //описываем необх. переменные
Console.Write("Введите целое число N=");
```

```
s = Console.ReadLine(); //вводим число
n = Convert.ToInt32(s); //преобразовываем к целому типу

sum = 0; //обнуляем сумму перед циклом
 do {
     sum +=Math.Abs( n % 10); //прибавляем к сумме
                        //самую правую цифру числа n
     n /=10; //челочисленно делим  n на 10,
           //т.е. отбрасываем самую правую цифру
   }while (n!=0); //повторяем пока не закончатся
              //все разряды числа n

Console.WriteLine("Сумма цифр числа S={0}",sum);
Console.ReadKey();
```



Задание №1

Задачи на использование **операторов условия** . Осуществить **ввод** необходимых данных, выполнить реализацию алгоритма с использованием операторов условия, обеспечить вывод полученных результатов. Не допускается использование операторов, прерывающих ход программы (**break**, **goto**).

Задачи по вариантам

- 1.Ввести номер года (положительное целое число). Определить количество дней в этом году, учитывая, что обычный год насчитывает 365 дней, а високосный — 366 дней. Високосным считается год, делящийся на 4, за исключением тех годов, которые делятся на 100 и не делятся на 400 (например, годы 300, 1300 и 1900 не являются високосными, а 1200 и 2000 — являются). См. также:
- 2.Ввести целочисленные координаты трех вершин прямоугольника, стороны которого параллельны координатным осям. Найти координаты его четвертой вершины. Если пользователь введёт координаты точек так, что нельзя получить прямоугольник со сторонами, параллельными координатным осям, вывести соответствующее сообщение.
- 3.На числовой оси расположены три точки: A, B, C. Определить, какая из двух последних точек (B или C) расположена ближе к A, и вывести эту точку и ее рас-

стояние от точки A. Если пользователь введёт координаты B и C так, что они будут равноотстоящими от A, совпадать с A или между собой - выдать соответствующее сообщение.

- 4.Ввести четыре целых числа A, B, C, D, одно из которых отлично от трех других, равных между собой. Определить порядковый номер числа, отличного от остальных. Если пользователь введёт числа так, что они не будут соответствовать условию задачи - выдать сообщение об ошибке.
- 5.Ввести три числа A,B,C. Если среди них имеется хотя бы одно четное вывести максимальное из них, иначе - минимальное. Если пользователь введёт числа так, что среди них нельзя будет определить лишь одно максимальное/минимальное - выдать соответствующее сообщение.
- 6.Ввести три переменные вещественного типа: A, B, C. Если их значения упорядочены по возрастанию или убыванию, то удвоить их; в противном случае заменить знак каждой переменной на противоположный. Вывести новые значения переменных A, B, C.
- 7.Ввести три числа A, B, C. Найти сумму двух наибольших из них. Если пользователь введёт числа так, что среди них нельзя будет определить два наибольших - выдать соответствующее сообщение.
- 8.Ввести три числа A, B, C. Вывести вначале наименьшее, а затем наибольшее из данных чисел. Если пользователь введёт числа так, что среди них нельзя будет определить одно наименьшее/наибольшее - выдать соответствующее сообщение.
- 9.Ввести три числа A, B, C. Вывести среднее по величине из них (то есть число, расположенное между наименьшим и наибольшим). Если пользователь введёт числа так, что среди них нельзя будет определить среднее - выдать соответствующее сообщение.
- 10.Ввести три целых числа A, B, C. Найти количество положительных и количество отрицательных чисел в исходном наборе.

Задание №2

Задачи на использование **операторов варианта** . Осуществить **ввод** необходимых данных, выполнить реализацию алгоритма с использованием операторов варианта, обеспечить вывод полученных результатов. Не допускается использование массивов и операторов **goto**.

Задачи по вариантам

- 1.Даны два целых числа: D (день) и M (месяц), определяющие правильную дату невисокосного года. Вывести значения D и M для даты, предшествующей указанной. Если пользователь вводит D и M несоответствующие календарю - выдать сообщение об ошибке.
- 2.Мастям игральных карт присвоены порядковые номера: 1 — пики, 2 — трефы, 3 — бубны, 4 — червы. Достоинству карт, старших десятки, присвоены номера: 11 — валет, 12 — дама, 13 — король, 14 — туз. Даны два целых числа: N — достоинство (6 <= N <= 14) и M — масть карты (1 <= M <= 4). Вывести название соответствующей карты вида «шестерка бубен», «дама червей», «туз треф» и т. п. Если пользователь введёт данные не соответствующие условию задачи - выдать сообщение об ошибке.

- 3.Дано целое число в диапазоне 20–69, определяющее возраст (в годах). Вывести строку-описание указанного возраста, обеспечив правильное согласование числа со словом «год», например: 20 — «двадцать лет», 32 — «тридцать два года», 41 — «сорок один год». Если пользователь введёт данные не соответствующие условию задачи - выдать сообщение об ошибке.
- 4.Дано целое число в диапазоне 10–40, определяющее количество учебных заданий по некоторой теме. Вывести строку-описание указанного количества заданий, обеспечив правильное согласование числа со словами «учебное задание», например: 18 — «восемнадцать учебных заданий», 23 — «двадцать три учебных задания», 31 — «тридцать одно учебное задание». Если пользователь введёт данные не соответствующие условию задачи - выдать сообщение об ошибке.
- 5.Дано целое число в диапазоне 100–999. Вывести строку-описание данного числа, например: 256 — «двести пятьдесят шесть», 814 — «восемьсот четырнадцать». Если пользователь введёт данные не соответствующие условию задачи - выдать сообщение об ошибке.
- 6.В восточном календаре принят 60-летний цикл, состоящий из 12-летних подциклов, обозначаемых названиями цвета: зеленый, красный, желтый, белый и голубой. При этом каждый цвет следует по два года подряд. В каждом подцикле годы носят названия животных: крысы, коровы, тигра, зайца, дракона, змеи, лошади, овцы, обезьяны, петуха, собаки и свиньи. По номеру года определить его название, если 4 год нашей эры — начало цикла: «год зеленой крысы».
- 7.Составить программу, которая бы присваивала переменной T значение true, если дата d1,m1 предшествует(в рамках года) дате d2,m2 и значение false иначе(d1 и d2-дата,m1 и m2-месяц). Переменную T распечатать. Год считать невисокосным. Если введённые даты не соответствуют календарю - выдать сообщение об ошибке.
- 8.Составить программу, которая бы по введённому значению некоторой длины в метрах выводила бы это значение с использованием наиболее подходящей кратной или дольной приставки (км,м,дм,см,мм,мкм,нм). Подсказка: для нахождения порядка числа использовать десятичный логарифм.
- 9.Для натурального числа K напечатать фразу "мы нашли K грибов в лесу", согласовав окончание слова "гриб" с числом K. Обратите внимание на особое согласование в случае когда 10<K<20.
- 10.Составить программу, которая бы реализовала следующий алгоритм: переменной T присвоить значение true если сочетание D(день) M(месяц) G(год) образует правильную дату, и значение false- иначе(учитывая количество дней в месяце и название месяца). Переменную T распечатать. Примечание: високосным считается год, делящийся на 4, за исключением тех годов, которые делятся на 100 и не делятся на 400 (например, годы 300, 1300 и 1900 не являются високосными, а 1200 и 2000 — являются).

Задание №3

Задачи на использование **операторов цикла for** . Осуществить **ввод** необходимых данных, выполнить реализацию алгоритма с использованием операторов цикла **for**, обеспечить вывод полученных результатов. Не разрешается использовать другие

операторы цикла. Не допускается использование массивов и операторов, прерывающих ход программы (**break**, **goto**).

Задачи по вариантам

- 1.Ввести целое число N > 1 и две вещественные точки на числовой оси: A, B (A < B). Отрезок [A, B] разбит на N равных отрезков. Вывести H — длину каждого отрезка, а также значения функции f(x) = 1 – sin(x) в точках, разбивающих отрезок [A, B]: f(A), f(A + H), f(A + 2H), ..., f(B).
- 2.Ввести целое число N > 0 и вещественное a > 0. Последовательность вещественных чисел определяется следующим образом $x_{n+1}=(x_n+a/x_n)/2$. Считая $x_0=a$ вывести первые N членов последовательности. Такой способ применяли еще в древнем Вавилоне для вычисления квадратного корня числа a. После выдачи последовательности распечатать значение квадратного корня из a, вычисленное стандартной функцией. См. также: **Квадратный корень** , **Итерационная формула Герона**
- 3.Ввести целое число N > 1. Последовательность чисел Фибоначчи F_K (целого типа) определяется следующим образом: F_1 =1, F_2= 1, $F_K=F_{K-2} + F_{K-1}$, K = 3, 4, ... N. Вывести элементы F_1, F_2, …, F_N. См. также: **Числа Фибоначчи** .
- 4.Ввести целое число N > 0. Последовательность вещественных чисел A_K определяется следующим образом: $A_0=1/0!$, $A_K=1/K!$, K=1, 2, ...,N. Вывести сумму последовательности. Примечание K! — это K–факториал — обозначает произведение всех целых чисел от 1 до K. См. также: **Факториал** , **Ряд Тейлора**
- 5.Логистическое отображение (также известное, как квадратичное отображение или отображение Фейгенбаума) даётся формулой $x_{n+1}=r*x_n*(1-x_n)$. Считая $x_0=0.333$ распечатать N первых элементов отображения. Величину r, принадлежащую интервалу (0..4) вводит пользователь. (При r>3.6 должна наблюдаться хаотическая последовательность). В качестве тестового примера построить последовательности при разных значениях r. См. также: **Логистическое отображение**
- 6.Ввести целое число N > 2. Последовательность целых чисел A_K определяется следующим образом: A_1=1, A_2=2, A_3=3, $A_K=A_{K–1} + A_{K–2} – 2*A_{K–3}$, K = 4, 5, ... N. Вывести элементы A_1, A_2, ..., A_N.
- 7.Ввести вещественное число X и целое число N > 0. Найти значение выражения $X – X^3/(3!) + X^5/(5!) – ... + (–1)^N*X^{2*N+1}/((2*N+1)!)$, которое является приближенным значением функции sin в точке X. Отобразить сумму ряда и рассчитанное с помощью функции sin значения. См. также: **Факториал** , **Ряд Тейлора**
- 8.Ввести целое число N > 0. Найти квадрат данного числа, используя для его вычисления следующую формулу: $N^2=1 + 3 + 5 + ... + (2*N – 1)$. После добавления к сумме каждого слагаемого выводить текущее значение суммы (в результате будут выведены квадраты всех целых чисел от 1 до N).
- 9.Ввести целое число N > 0. Найти значение выражения 1.1 – 1.2 + 1.3 – ... (N слагаемых, знаки чередуются). Условный оператор не использовать.
- 10.Ввести целое число N > 0. Среди цифр этого числа выделить только чётные, из которых составить другое число и вывести. Например, при N=3854972 ответом будет число 842.

Задание №4

Задачи на использование **операторов цикла с постусловием**. Осуществить **ввод** необходимых данных, выполнить реализацию алгоритма с использованием операторов цикла **do - while**, обеспечить вывод полученных результатов. Использование других операторов цикла недопустимо. Не допускается использование массивов и операторов, прерывающих ход программы (**break**, **goto**).

Задачи по вариантам

- 1.Ввести два целых числа N1 и N2. Если N1 >N2, найти сумму целых чисел в диапазоне N1 ... N2. Если N2 больше N1, найти сумму целых чисел в диапазоне N2... N1. Если N1 равно N2, вывести на экран соответствующее сообщение.
- 2.Осуществить ввод последовательности целых чисел, определить третье положительное число и подсчитать количество цифр в нем. Последовательность потенциально не ограничена, окончанием последовательности служит третье положительное число.
- 3.Осуществить ввод последовательности целых чисел, определить максимальное четное число, его порядковый номер и подсчитать сумму его цифр. Последовательность потенциально не ограничена, окончанием последовательности служит число 0. Если окажется, что чётных чисел в последовательности не было, вывести соответствующее сообщение.
- 4.Осуществить ввод последовательности целых чисел и сравнить, что больше, сумма положительных или произведение отрицательных. Последовательность потенциально не ограничена, окончанием последовательности служит число 0.
- 5.Осуществить ввод последовательности целых чисел и определить предпоследнее отрицательное число. Последовательность потенциально не ограничена, окончанием последовательности служит число 0. Если окажется, что в последовательности было менее двух отрицательных чисел, вывести соответствующее сообщение.
- 6.Осуществить ввод целого числа M. На промежутке от 1 до M найти все числа Армстронга. Натуральное число из n цифр называется числом Армстронга, если сумма его цифр, возведенных в n-ю степень, равна самому числу. Примеры: $153=1^3 + 5^3 + 3^3$; $1634=1^4+6^4+3^4+4^4$. См. также: **Число Армстронга**
- 7.Ввести действительное число x и натуральное число n. Вычислить $x*(x - n)*(x - 2*n)(x - 3*n)\ldots(x - n^2)$.
- 8.Осуществить ввод последовательности целых чисел. Определить, сколько из них и какие принимают наибольшее значение. Последовательность потенциально не ограничена, окончанием последовательности служит число 0.
- 9.Осуществить ввод последовательности целых чисел в количестве не меньшем двух. Вычислить сумму тех из них, порядковые номера которых - простые числа. Последовательность потенциально не ограничена, окончанием последовательности служит число 0. См. также: **Простые числа**
- 10.Осуществить ввод последовательности целых чисел в количестве не меньшем трёх. Определить, сколько из них больше своих "соседей", т.е. предыдущего и последующего чисел. Последовательность потенциально не ограничена, окончанием последовательности служит число 0.

Задание №5

Задачи на использование **операторов цикла с предусловием**. Осуществить **ввод** необходимых данных, выполнить реализацию алгоритма с использованием операторов цикла **while**, обеспечить вывод полученных результатов. Использование других операторов цикла недопустимо. Не допускается использование массивов и операторов, прерывающих ход программы (**break**, **goto**).

Задачи по вариантам

- 1.Ввести целое число N > 0, являющееся некоторой степенью числа 2: $N=2^K$. Найти целое число K — показатель этой степени. Не разрешается использовать логарифм. Если пользователь введёт число не являющееся степенью числа 2 - вывести соответствующее сообщение.
- 2.Ввести целое число N > 0. Используя операции деления нацело и взятия остатка от деления, найти число, полученное при прочтении числа N справа налево
- 3.Ввести целое число N > 1. Если оно является простым, то есть не имеет положительных делителей, кроме 1 и самого себя, то вывести это число, иначе вывести ближайшее большее простое число. См. также: **Простые числа**
- 4.Ввести целое число N > 1. Последовательность чисел Фибоначчи F_K определяется следующим образом: $F_1=1$, $F_2=1$, $F_K=F_{K-2} + F_{K-1}$, K=3, 4, ... Проверить, является ли число N числом Фибоначчи. Если является, то вывести True, если нет — вывести False. См. также: **Числа Фибоначчи** .
- 5.Ввести вещественное число e > 0. Последовательность вещественных чисел A_K определяется следующим образом: $A_1=1$, $A_2=2$, $A_K=(A_{K-2} + 2\cdot A_{K-1})/3$, K = 3, 4, … . Найти первый из номеров K, для которых выполняется условие $|A_K – A_{K-1}| < e$, и вывести этот номер, а также числа A_{K-1} и A_K.
- 6.Ввести положительные числа A, B, C. На прямоугольнике размера A x B размещено максимально возможное количество квадратов со стороной C (без наложений). Найти количество квадратов, размещенных на прямоугольнике. Операции умножения и деления не использовать.
- 7.Ввести два целых числа a и b. Вычислить НОД (a,b) - наибольший общий делитель a и b. Делителями называются числа, которые делят без остатка заданное число, кроме единицы и самого этого числа. См. также: **Наибольший общий делитель**
- 8.Ввести натуральное (целое неотрицательное) число a и целое положительное число d. Вычислить частное q и остаток r при делении a на d, не используя операций целочисленного деления и нахождения остатка. Разрешается использовать только целые переменные и целочисленные операции.
- 9.Ввести целые положительные числа A и B. Найти их наибольший общий делитель (НОД), используя алгоритм Евклида: НОД(A, B)=НОД(B, A mod B), если $B \neq 0$; НОД(A, 0)=A, где «mod» обозначает операцию взятия остатка от деления. См. также: **Наибольший общий делитель** , **Алгоритм Евклида**
- 10.Ввести целое число N > 1. Найти первое из чисел Фибоначчи, большее чем N. Последовательность чисел Фибоначчи F_K (целого типа) определяется следую-

щим образом: $F_1 = 1$, $F_2 = 1$, $F_K = F_{K-2} + F_{K-1}$, K = 3, 4, ... N. См. также: **Числа Фибоначчи** .

4. Массивы

Массивы одномерные

Хранение группы связанных элементов данных является основным требованием большинства программных приложений. Для этого существуют два основных способа: массивы и коллекции.

Индексный массив — именованный набор однотипных переменных, расположенных в памяти непосредственно друг за другом, доступ к которым осуществляется по индексу.

Коллекция в программировании — программный объект, содержащий в себе, тем или иным образом организованный, набор значений одного или различных типов, и позволяющий обращаться к этим значениям.

Массивы являются коллекциями объектов одного типа. Поскольку длина массивов практически не ограничена, они могут использоваться для хранения тысяч или даже миллионов объектов, но размер массива должен быть указан при его создании. Каждый элемент массива доступен по числовому индексу, указывающему позицию или ячейку, в которой объект хранится в массиве. Массивы могут хранить как ссылочные типы, так и типы значений. Массив является индексированной коллекцией объектов.

Одномерный массив объектов объявляется следующим образом:

тип_элемента[] имя_массива;

int[] myArray1;

Часто элементы в массиве инициализируются в это же время, как показано ниже:

int[] massiv1 = new int[12];

Здесь между разделителями [и] находится константное выражение, значение которого определяет размерность массива.

Значение по умолчанию числовых элементов массива задано равным нулю, а элементы ссылок имеют значение **null**, но значения можно инициализировать при создании массива следующим образом:

int[] myArray2 = new int[] {7, 13, 4, 2};

int[] myArray3 = {5, 9, 3, 17, 9};

Можно создавать массивы из строк:

string[] days = {"Sun", "Mon", "Tue", "Wed", "Thr", "Fri", "Sat"};

Индексация массивов начинается с нуля, поэтому номер первого элемента массива равен 0. Обращение к элементу массива осуществляется через квадратные скобки [и].

myArray2[2]=17;

Пример: Написать программу, осуществляющую ввод целых чисел в массив. Из полученного массива распечатать элементы превышающие среднее арифметическое всего массива.

```
//описываем необходимые переменные
int i,N; string s;double av;
//вводим число элементов
```

```
Console.Write("Введите число элементов массива N=");
s = Console.ReadLine();
N = Convert.ToInt32(s);
//создаём массив необходимой длины
int[] massiv1 = new int[N]; av = 0;
//запускаем цикл по всем элементам массива
for (i = 0; i < N; i++)
  {
    //вводим i-ый элемент
    Console.Write("Введите {0}-й элемент массива ",i);
    s = Console.ReadLine();
    massiv1[i] = Convert.ToInt32(s);
    //суммируем все элементы массива
    av += massiv1[i];
  }
//находим среднее арифметическое
av /=N;
//запускаем цикл по всем элементам массива
for (i = 0; i < N; i++)
  //печатаем только элементы больше среднего
  if (massiv1[i] > av)  Console.Write("{0} ", massiv1[i]);
Console.ReadKey();
```

Оператор foreach

Оператор **foreach** повторяет группу внедренных операторов для каждого элемента в коллекции массива или объекта. Оператор **foreach** используется для итерации коллекции с целью получения необходимой информации, однако его не следует использовать для изменения содержимого коллекции во избежание непредвиденных побочных эффектов.

Внедренные операторы продолжают выполняться для каждого элемента массива или коллекции. После завершения итерации всех элементов коллекции управление переходит к следующему оператору после блока **foreach**.

В любой точке блока **foreach** можно разорвать цикл с помощью ключевого слова **break** или перейти к следующей итерации в цикле с помощью ключевого слова **continue**.

Цикл **foreach** также может быть разорван при помощи операторов goto, return или throw.

Пример: Распечатать содержимое массива в одну колонку

```
int[] myarray5 = new int[] { 0, 1, 2, 3, 5, 8, 13 };
// В операторе foreach требуется указать и тип, и идентификатор
foreach (int i in myarray5)
      Console.WriteLine(i);
```

Массивы многомерные

Массивы могут иметь несколько измерений. Например, матрица - это двухмерный массив. В примере показано объявление двухмерного и трёхмерного массивов:

```
int[,] my_matrix1 = new int[4, 2];
int[, ,] my_cube1 = new int[4, 2, 3];
```

Массив можно инициализировать при объявлении, как показано в следующем примере:

```
int[,] array2D = new int[,] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };
int[, ,] array3D = new int[,,] { { { 1, 2, 3 } }, { { 4, 5, 6 } } };
int[,] array4 = { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };
```

Если нужно создать переменную массива без инициализации, то необходимо использовать оператор **new**, чтобы присвоить массив переменной. Например:

```
int[,] array5;
array5 = new int[,] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };   // OK
//array5 = {{1,2}, {3,4}, {5,6}, {7,8}};   // Будет ошибка!
```

В следующем примере кода переменные массивов инициализируются значениями по умолчанию:

```
int[,] array6 = new int[10, 10];
```

Доступ к каждому элементу массива можно осуществить через квадратные скобки. Индексация многомерных массивов начинается с нуля:

```
array5[2, 1] = 25;
```

Пример 1: осуществить ввод и распечатку содержимого матрицы

```
 //описываем необходимые переменные
int i,j, M,N; string s;
//вводим число элементов
Console.Write("Введите число строк матрицы N=");
s = Console.ReadLine();
N = Convert.ToInt32(s);
Console.Write("Введите число столбцов матрицы M=");
s = Console.ReadLine();
M = Convert.ToInt32(s);
 //создаём двухмерный массив необходимой длины
int[,] my_matrix= new int[N,M];
//запускаем циклы по всем элементам матрицы
for (i = 0; i < N; i++) //цикл по строкам
 for (j = 0; j < M;j++ ) //цикл по столбцам
   {
    //вводим i,j-ый элемент
    Console.Write("Введите элемент матрицы ({0},{1}) ", i+1,j+1);
    s = Console.ReadLine();
    my_matrix[i,j] = Convert.ToInt32(s);
   }
Console.WriteLine();
for (i = 0; i < N; i++)
```

```
{
  for (j = 0; j < M; j++)
    Console.Write("{0,3} ", my_matrix[i, j]);
  Console.WriteLine();
}
```

5. Строки

Средства работы со строками

Строка в C# — это массив знаков, объявленный с помощью ключевого слова **string**. Строковый литерал объявляется с помощью кавычек, как показано в следующем примере.

```
string s = "Hello, World!";
```

Переприсваивание

Строки можно целиком переприсваивать:

```
string s1 = "Hello";
string s2 = s1; //переприсваиваем
```

Объединение строк

Можно объединять строки с помощью оператора +, как показано в следующем примере:

```
string s1 = "orange";
string s2 = "red";
s1 += s2;
Console.WriteLine(s1);  // напечатается "orangered"
```

Escape-знаки

Строки могут содержать escape-знаки, такие как "\n" (новая строка) и "\t" (табуляция). Пример:

```
string hello = "Hello\nWorld!";
Console.WriteLine(hello);
/* Напечатается
Hello
World!
*/
string s = "чтобы вставить \"кавычки\" в строку используем обратный слэш";
Console.WriteLine(s);
```

Если требуется добавить в строку обратную косую черту, перед ней нужно поставить еще одну обратную косую черту. Следующая строка:

```
string filePath = "\\\\My Documents\\";
Console.WriteLine(filePath);
/* Напечатается:
\\My Documents\
*/
```

Точные строки: символ @

Символ @ служит для того, чтобы конструктор строк пропускал escape-знаки и переносы строки. Следующие две строки являются идентичными:

```
string p1 = "\\\\My Documents\\My Files\\";
```

```
string p2 = @"\\My Documents\My Files\";
```

Чтобы поставить в точной строке знак двойных кавычек, нужно использовать по два таких знака, как показано в следующем примере:

```
string s = @"You say ""goodbye"" and I say ""hello""";
```

Доступ к отдельным знакам

Квадратные скобки [] служат для доступа к отдельным знакам в объекте **string**, но при этом возможен доступ только для чтения:

```
string str = "test";
char x = str[2];  // x = 's';
string s5 = "Printing backwards";
for (int i = 0; i < s5.Length; i++)
    Console.Write(s5[s5.Length - i - 1]);
  // напечатается "sdrawkcab gnitnirP"
```

Извлечение подстрок

Для извлечения подстроки из строки используется метод **Substring**.

```
string s3 = "Visual C# Express";
string s4 = s3.Substring(7, 2);
Console.WriteLine(s4);
// напечатается "C#"
```

Замена по образцу

Для замены подстроки в строке по образцу используется метод **Replace**.

```
string s3 = "Visual C# Express";
string s5 = s3.Replace("C#", "Basic");
Console.WriteLine(s5);
// напечатается "Visual Basic Express"
```

Смена регистра

Чтобы изменить регистр букв в строке (сделать их заглавными или строчными) следует использовать **ToUpper()** или **ToLower()**, как показано в следующем примере:

```
string s6 = "АгПА";
Console.WriteLine(s6.ToUpper());
// Напечатается АГПА
Console.WriteLine(s6.ToLower());
// Напечатается агпа
Console.WriteLine(s6);
// Напечатается АГПА
```

Сравнения

Самый простой способ сравнения двух строк — использовать операторы == и !=, осуществляющие сравнение с учетом регистра:

```
string color1 = "red";
string color2 = "green";
string color3 = "red";
if (color1 == color3)
  Console.WriteLine("Строки равны");
if (color1 != color2)
  Console.WriteLine("Строки не равны");
```

Не допускается использование >,<,>=,<= для сравнения строк. Для строковых объектов существует метод CompareTo(), возвращающий целочисленное значение, зависящее от того, что одна строка может быть меньше (<), равна (==) или больше другой (>). При сравнении строк используется значение Юникода, при этом значение строчных букв меньше, чем значение заглавных. Дополнительные сведения о правилах сравнения строк см. в разделах CompareTo().

```
string string1 = "ИИТ";
string string2 = "Иит";
int result = string1.CompareTo(string2);
if (result > 0)
    Console.WriteLine("{0} больше чем {1}", string1, string2);
 else if (result == 0)
    Console.WriteLine("{0} равно {1}", string1, string2);
 else if (result < 0)
     Console.WriteLine("{0} меньше чем {1}", string1, string2);
// Напечатается ИИТ больше чем Иит
```

Удаление фрагментов и вставка строк в строки

Может быть выполнена с помощью методов Remove и Insert:

```
string x = "ZX Spectrum";
x=x.Remove(2, 1);
x = x.Insert(2,"-");
Console.WriteLine(x);
Console.ReadKey();
```

Пустые строки

Пустая строка — это экземпляр объекта String, содержащий 0 знаков. Пустые строки часто используются в различных сценариях программирования, представляя пустое текстовое поле. Для пустых строк можно вызывать методы, потому что такие строки являются допустимыми объектами String. Пустые строки инициализируются следующим образом:

```
string s = "";
```

Массивы из строк

Могут быть инициализированы начальными значениями:

```
string[] x =
        {
```

```
            "Краснодарский край",
            "Армавир",
            "Розы Люксембург",
            "156"
           };
Console.WriteLine(x[0]);
//напечатается Краснодарский край
Console.WriteLine(x[1][0]);
//напечатается А
```

Разбиение строк

Следующий пример кода демонстрирует возможность разбора строки при помощи метода String.Split. Работа метода заключается в возврате массива строк, в котором каждый элемент представляет слово. В качестве ввода **Split** принимает массив символов, определяющих какие символы должны использоваться в качестве разделителей. В этом примере используются пробелы, запятые, точки, двоеточия и табуляция. Массив, содержащий эти разделители, передается в **Split**, и каждое слово в предложении выводится отдельно при помощи результирующего массива строк.

Пример: разбить предложение на слова

```
char razdelitel = ' ';
string text = "Шла Саша по шоссе и сосала сушку";
Console.WriteLine("Исходный текст: '{0}'", text);
string[] words = text.Split(razdelitel);
Console.WriteLine("{0} слов в тексте:", words.Length);
foreach (string s in words)
   Console.WriteLin
```

В качестве разделителя может выступать массив символов.

```
char[] delimiterChars = { ' ', ',', '.', ':', '\t' };
string text = "one\ttwo three:four,five six seven";
Console.WriteLine("Original text: '{0}'", text);
string[] words = text.Split(delimiterChars);
Console.WriteLine("{0} words in text:", words.Length);
foreach (string s in words)
      Console.WriteLine(s);
```

Пример: Ввести текстовую строку. Напечатать слова, в которых первая буква встречается еще хотя бы раз.

```
Console.Write("Введите предложение: ");
string s=Console.ReadLine();
string[] words= s.Split(' ','.',',');
foreach (string word in words)
  {
    char c=word[0];
```

```
    bool flag = false;
    int i=1;
    while (i < word.Length & !flag)
      {
       if (word[i] == c) flag = true;
       i++;
      }
    if (flag) Console.WriteLine(word);
  }
Console.ReadKey();
```

Преобразование строк в другие типы

С помощью объекта Convert:

```
N = Convert.ToInt32(s1);
M = Convert.ToDouble(s2);
F = Convert.ToBoolean(s3);
B = Convert.ToByte(s4);
C = Convert.ToChar(k);
s5= Convert.ToString(x);
```

С помощью метода Parse:

```
N = int.Parse(s1);
N = int.Parse(Console.ReadLine());
M = Double.Parse(s2);
F = bool.Parse(s3);
```

Лабораторная работа №3

Задание на лабораторную: В среде **Microsoft Visual C#** , решить 5 задач по вариантам и оформить отчёт.

Примеры решения задач

№1. Написать программу, осуществляющую ввод целых чисел в массив. Из полученного массива распечатать элементы превышающие среднее арифметическое всего массива.

```
//описываем необходимые переменные
int i,N; string s;double av;
//вводим число элементов
Console.Write("Введите число элементов массива N=");
s = Console.ReadLine();
N = Convert.ToInt32(s);

//создаём массив необходимой длины
```

```
int[] massiv1 = new int[N]; av = 0;

//запускаем цикл по всем элементам массива
for (i = 0; i < N; i++)
 {
  //вводим i-ый элемент
  Console.Write("Введите {0}-й элемент массива ",i);
  s = Console.ReadLine();
  massiv1[i] = Convert.ToInt32(s);
  //суммируем все элементы массива
  av += massiv1[i];
 }
//находим среднее арифметическое
av /=N;

//запускаем цикл по всем элементам массива
for (i = 0; i < N; i++)
 //печатаем только элементы больше среднего
 if (massiv1[i] > av)  Console.Write("{0} ", massiv1[i]);

Console.ReadKey();
```

№2. Ввести текстовую строку. Напечатать слова, в которых первая буква встречается еще хотя бы раз.

```
Console.Write("Введите предложение: ");
string s=Console.ReadLine();
string[] words= s.Split(' ','.',',');
foreach (string word in words)
 {
   char c=word[0];
   bool flag = false;
   int i=1;
   while (i < word.Length & !flag)
    {
      if (word[i] == c) flag = true;
      i++;
    }
   if (flag) Console.WriteLine(word);
 }
Console.ReadKey();
```

№3. Преобразовать содержимое матрицы так, чтобы в каждой строке была возрастающая последовательность

```
int i,j,k,b,M,N;
int[,] my_matrix = new int[,] { { 1, 3 , 4}, { 1, 4, 8 }, { 5, 8 ,6 }, { 7, 8, 2 } };
N = 4; M = 3;
 for (i = 0; i < N; i++) //цикл по строкам
  for (j = 0; j < M-1; j++) //цикл по столбцам
   for (k = 0; k < M - j-1; k++)
    if (my_matrix[i, k] > my_matrix[i, k+1])
      {
       b = my_matrix[i, k];
       my_matrix[i, k] = my_matrix[i, k + 1];
       my_matrix[i, k + 1] = b;
      }
```

Задание №1

Задачи на использование **одномерных целочисленных массивов** . Условие вида "дан массив" означает, что пользователем вводится величина размерности и все элементы массива с клавиатуры. Осуществить ввод необходимых данных, выполнить реализацию алгоритма, обеспечить вывод полученных результатов. Если по ходу решения задачи требуется создание дополнительных массивов, размерность которых изначально неизвестна, необходимо выполнить предварительную обработку исходного массива, для выяснения размерности вновь создаваемого. Не допускается использование операторов, прерывающих ход программы (**break**, **goto**). Ввод массивов, обработка и вывод результатов реализуется отдельными **методами** .

Задачи по вариантам

- 1 Даны два массива A и B одинакового размера N. Сформировать новый массив C того же размера, каждый элемент которого равен максимальному из элементов массивов A и B с тем же индексом.
- 2 Дан целочисленный массив A размера N. Переписать в новый целочисленный массив B все четные числа из исходного массива (в том же порядке) и вывести размер полученного массива B и его содержимое.
- 3 Дан массив A размера N. Сформировать новый массив B того же размера по следующему правилу: элемент B_K равен сумме элементов массива A с номерами от 0 до K.
- 4 Дан массив A размера N. Сформировать новый массив B того же размера по следующему правилу: элемент B_K равен среднему арифметическому элементов массива A с номерами от 0 до K.
- 5 Дан массив A размера N. Сформировать два новых массива B и C: в массив B записать все положительные элементы массива A, в массив C — все отрицательные (сохраняя исходный порядок следования элементов). Вывести вначале размер и содержимое массива B, а затем — размер и содержимое массива C.
- 6 Даны два массива A и B, элементы которых упорядочены по возрастанию. Объединить эти массивы так, чтобы результирующий массив C остался упорядоченным по возрастанию

- 7 Даны два массива A и B. Распечатать те элементы, которые присутствуют в обоих массивах.
- 8 Даны два массива A и B. Распечатать те элементы массива A, которых нет в массиве B. Распечатать те элементы массива B, которых нет в массиве A.
- 9 Даны два массива A и B. Определить которых из них имеет больший диапазон, т.е. разницу между самым большим и самым меньшим значением.
- 10 Дан целочисленный массив A размера N. Переписать в новый целочисленный массив B все элементы с порядковыми номерами, кратными трем (3, 6, …), и вывести размер полученного массива B и его содержимое. Условный оператор не использовать.

Задание №2

Задачи на исследование серий в **одномерных целочисленных массивах** . Условие вида "дан массив" означает, что пользователем вводится величина размерности и все элементы массива с клавиатуры. Осуществить ввод необходимых данных, выполнить реализацию алгоритма, обеспечить вывод полученных результатов. Если по ходу решения задачи требуется создание дополнительных массивов, размерность которых изначально неизвестна, необходимо выполнить предварительную обработку исходного массива, для выяснения размерности вновь создаваемого. Не допускается использование операторов, прерывающих ход программы (**break**, **goto**). Ввод массивов, обработка и вывод результатов реализуется отдельными **методами** .

Задачи по вариантам

- 1 Дан целочисленный массив A размера N. Назовем серией группу подряд идущих одинаковых элементов, а длиной серии — количество этих элементов (длина серии может быть равна 1). Сформировать два новых целочисленных массива B и C одинакового размера, записав в массив B длины всех серий исходного массива, а в массив C — значения элементов, образующих эти серии.
- 2 Дан целочисленный массив размера N. Вставить перед каждой его серией элемент с нулевым значением. Серия - это группа подряд идущих одинаковых элементов, длина серии — количество этих элементов (длина серии может быть равна 1).
- 3 Дан целочисленный массив размера N. Вставить после каждой его серии элемент с нулевым значением. Серия - это группа подряд идущих одинаковых элементов, длина серии — количество этих элементов (длина серии может быть равна 1).
- 4 Дан целочисленный массив размера N. Преобразовать массив, увеличив каждую его серию на один элемент. Серия - это группа подряд идущих одинаковых элементов, длина серии — количество этих элементов (длина серии может быть равна 1).
- 5 Дан целочисленный массив размера N. Преобразовать массив, уменьшив каждую его серию на один элемент. Серия - это группа подряд идущих одинаковых элементов, длина серии — количество этих элементов (длина серии может быть равна 1).
- 6 Дано целое число K и целочисленный массив размера N. Удалить из массива серию с номером K. Если серий в массиве меньше K, то вывести массив без измене-

ний. Серия - это группа подряд идущих одинаковых элементов, длина серии — количество этих элементов (длина серии может быть равна 1).

- 7 Дано целое число K и целочисленный массив размера N. Поменять местами первую серию массива и его серию с номером K. Если серий в массиве меньше K, то вывести массив без изменений. Серия - это группа подряд идущих одинаковых элементов, длина серии — количество этих элементов (длина серии может быть равна 1).
- 8 Дано целое число L и целочисленный массив размера N. Заменить каждую серию массива, длина которой меньше L, на один элемент с нулевым значением. Серия - это группа подряд идущих одинаковых элементов, длина серии — количество этих элементов (длина серии может быть равна 1).
- 9 Дан целочисленный массив размера N. Преобразовать массив, увеличив его первую серию наибольшей длины на один элемент. Серия - это группа подряд идущих одинаковых элементов, длина серии — количество этих элементов (длина серии может быть равна 1).
- 10 Дан целочисленный массив размера N. Преобразовать массив, увеличив все его серии наибольшей длины на один элемент. Серия - это группа подряд идущих одинаковых элементов, длина серии — количество этих элементов (длина серии может быть равна 1).

Задание №3

Задачи на обработку **одномерных целочисленных массивов** . Условие вида "дан массив" означает, что пользователем вводится величина размерности и все элементы массива с клавиатуры. Осуществить ввод необходимых данных, выполнить реализацию алгоритма, обеспечить вывод полученных результатов. Если по ходу решения задачи требуется создание дополнительных массивов, размерность которых изначально неизвестна, необходимо выполнить предварительную обработку исходного массива, для выяснения размерности вновь создаваемого. Не допускается использование операторов, прерывающих ход программы (**break**, **goto**). Ввод массивов, обработка и вывод результатов реализуется отдельными **методами** .

- 1 Дан одномерный целочисленный массив из n элементов. Найти количество различных чисел среди элементов этого массива. Например, если задан массив, состоящий из чисел 10,13,10,18,5,10,5, то ответ будет 4, поскольку различные числа это 10,13,18,5. Рекомендуется использовать ещё один массив для хранения различных чисел.
- 2 Расставить по возрастанию одномерный целочисленный массив из n элементов. При упорядочивании разрешается менять местами только два соседних элемента. Результат распечатать.
- 3 Дан одномерный целочисленный массив из n элементов. Определить и распечатать все локальные экстремумы в нём. Экстремумами не могут быть крайние элементы.
- 4 Дан массив A размера N. Упорядочить его по возрастанию методом сортировки простым выбором: найти максимальный элемент массива и поменять его местами с последним элементом; выполнить описанные действия N – 1 раз, каждый

раз уменьшая на 1 количество анализируемых элементов и выводя содержимое массива.

- 5 Дан целочисленный массив размера N. Удалить из массива все одинаковые элементы, оставив их первые вхождения.
- 6 Дан целочисленный массив размера N. Удалить из массива все элементы, встречающиеся менее трех раз, и вывести размер полученного массива и его содержимое.
- 7 Дан целочисленный массив размера N. Удалить из массива все элементы, встречающиеся ровно два раза, и вывести размер полученного массива и его содержимое.
- 8 Дан целочисленный массив размера N. Удалить из массива все соседние одинаковые элементы, оставив их первые вхождения
- 9 Дан массив A размера N. Не изменяя данный массив, вывести номера его элементов в том порядке, в котором соответствующие им элементы образуют возрастающую последовательность.
- 10 Дан целочисленный массив размера N. Удалить из массива все одинаковые элементы, оставив их последние вхождения.

Задание №4

Задачи на обработку **строк** . Условие вида "дана строка" означает, что пользователем вводится строка с клавиатуры. Осуществить ввод необходимых данных, выполнить реализацию алгоритма, обеспечить вывод полученных результатов. Не допускается использование операторов, прерывающих ход программы (**break**, **goto**). Ввод строк, обработка и вывод результатов реализуется отдельными**методами** .

- 1 Даны строки S и S0. Проверить, содержится ли строка S0 в строке S. Не использовать стандартные средства для поиска подстрок.
- 2 Даны строки S и S0. Найти количество вхождений строки S0 в строку S. Не использовать стандартные средства для поиска подстрок.
- 3 Дана строка S. Разделить строку на отдельные слова не используя стандартные средства для разбиения строк.
- 4 Дана строка S. Из строки требуется удалить текст, заключенный в фигурные скобки. В строке может быть несколько фрагментов, заключённых в фигурные скобки. Возможно использование вложенных фигурных скобок и, следовательно необходимо, чтобы программа это учитывала.
- 5 Дана строка S. Найти количество различных букв в ней. Программа должна работать без учёта регистра букв.
- 6 Дана строка S. Найти количество различных слов в ней. Программа должна работать без учёта регистра букв.
- 7 Дана строка S. Определить есть ли в строке удвоенные буквы (пара соседствующих одинаковых букв), напечатать слова, содержащие их.
- 8 Дана строка S (предложение). Найти самое длинное слово в строке не используя стандартные средства для разбиения строк.
- 9 Дана строка S (предложение). Составить программу, определяющую является ли текст перевёртышем без учёта пробелов.

- 10 Дана строка. Вывести все слова, у которых первая и последняя буквы одинаковые не используя стандартные средства для разбиения строк.

Задание №5

Задачи на **двухмерные массивы** . Условие вида "дана матрица" означает, что пользователем вводится с клавиатуры размерность и все элементы. Осуществить ввод необходимых данных, выполнить реализацию алгоритма, обеспечить вывод полученных результатов. Не допускается использование операторов, прерывающих ход программы (**break**, **goto**). Ввод многомерных массивов, обработка и вывод результатов реализуется отдельными **методами** .

- 1 Дана целочисленная матрица размера M x N. Найти номер первого из ее столбцов, содержащих максимальное количество одинаковых элементов.
- 2 Дана матрица размера M x N. Найти ее строки, элементы которых упорядочены по возрастанию.
- 3 Дана целочисленная матрица размера M x N. Различные строки матрицы назовем похожими, если совпадают множества чисел, встречающихся в этих строках. Найти строки, похожие на первую строку данной матрицы.
- 4 Дана целочисленная матрица размера M x N. Найти ее строки, все элементы которых различны.
- 5 Дана квадратная целочисленная матрица размера M. Написать программу, которая проверяет, является ли введенная с клавиатуры матрица магическим квадратом. Магическим квадратом называется матрица, сумма элементов которой в каждой строке, в каждом столбце и по каждой диагонали одинакова.
- 6 Дана матрица размера M x N. Упорядочить ее столбцы так, чтобы их последние элементы образовывали убывающую последовательность.
- 7 Дана матрица размера M x N. Элемент матрицы называется ее локальным максимумом, если он больше всех окружающих его элементов. Поменять знак всех локальных максимумов данной матрицы на противоположный. При решении допускается использовать вспомогательную матрицу.
- 8 Дана матрица размера M x N. Упорядочить ее строки так, чтобы их минимальные элементы образовывали убывающую последовательность.
- 9 Дана целочисленная матрица размера M x N. Найти ее строки, содержащие равное количество положительных и отрицательных элементов (нулевые элементы матрицы не учитываются). Если таких строк нет, то вывести соответствующее сообщение.
- 10 Дана матрица размером m*n, определить количество и координаты особых элементов матрицы. Элемент считается особым, если он больше суммы остальных элементов своего столбца и при этом в его строке слева от него находятся элементы меньше него, а справа больше него. (Особый элемент может быть крайним в строке)

6. Методы

Метод представляет собой блок кода, содержащий набор инструкций. В C# все инструкции выполняются в контексте метода. В этом разделе описываются именованные методы. Другой тип методов, называемых анонимными функциями, описан в других разделах документации.

Методы объявляются в классе или в структуре путем указания уровня доступа, возвращаемого значения, имени метода и списка параметров этого метода. Все вместе эти элементы образуют подпись метода. Параметры заключаются в круглые скобки и разделяются запятыми. Пустые скобки указывают на то, что у метода нет параметров.

Методы имеют смысл подпрограмм. В случае, если метод возвращает какое-либо значение, его использование схоже с использованием функции. Если метод не возвращает никаких значений - его применение аналогично процедуре.

Методы могут возвращать значения вызывающим их объектам. Если тип возвращаемого значения, указываемый перед именем метода, не равен **void**, для возвращения значения используется ключевое слово **return**. В результате выполнения инструкции с ключевым словом **return**, после которого указано значение нужного типа, вызвавшему метод объекту будет возвращено это значение. Кроме того, ключевое слово return останавливает выполнение метода. Если тип возвращаемого значения **void**, инструкцию **return** без значения все равно можно использовать для завершения выполнения метода. Если ключевое слово **return** отсутствует, выполнение метода завершится, когда будет достигнут конец его блока кода. Для возврата значений методами с типом возвращаемого значения отличным от **void** необходимо обязательно использовать ключевое слово **return**.

Чтобы использовать возвращаемое методом значение в вызываемом методе, вызов метода можно поместить в любое место кода, где требуется значение соответствующего типа.

Пример и использованием возвращаемого значения:

```
static string my_func()
 { //данный метод возвращает строку
   return "Hello world!";
 }
static void Main(string[] args)
 { //"Главный" метод
   string s = "*** "+my_func()+" ***";
   Console.WriteLine(s);
 }
```

Пример с использованием **void**:

```
static void my_proc()
 {
   Console.WriteLine( "Hello world!");
 }
```

```
static void Main(string[] args)
 {
   my_proc();
 }
```

Передача параметров типа значения

Переменная типа значения содержит данные непосредственно, в противоположность переменной ссылочного типа, которая содержит ссылку на данные. Поэтому передача переменной типа значения методу означает передачу методу копии переменной. Любые изменения параметра, выполняемые внутри метода, не влияют на исходные данные, хранимые в переменной. Если требуется, чтобы вызываемый метод изменял значение параметра, его следует передавать ссылкой с помощью ключевого слова **ref** или **out**.

Передача типов значений с помощью значения

Пример: написать метод вычисляющий факториал числа, и возвращающий это значение.

```
static int factorial(int n)
{
 int i, res; res = 1;
 for (i = 1; i <= n; i++)
    res = res * i;
 return res;
}
static void Main(string[] args)
{
 int i;//независимая переменная
 for (i = 0; i <= 13;i++)
   Console.WriteLine("{0}! = {1}",i,factorial(i));
 Console.ReadKey();
}
```

Передача типов значений с помощью ссылки

Ключевое слово **out** используется для передачи аргументов по ссылке. Оно похоже на ключевое слово **ref**, за исключением того, что **ref** требует инициализации переменной перед ее передачей. Для работы с параметром **out** определение метода и вызывающий метод должны явно использовать ключевое слово **out**. Таким образом, если значение параметра до вызова метода не определено, используют **out**, если определено используют **ref**.

Рассмотрим пример подпрограммы ввода, обработки и вывода массива. "Ввести массив, заменить его элементы факториалами, вывести массив".

```
static void enter(out int[] massiv1)
 {
   int i, N; string s;
```

```
  //вводим число элементов
  Console.Write("Введите число элементов массива N=");
  s = Console.ReadLine();
  N = Convert.ToInt32(s);
  //создаём массив необходимой длины
  massiv1 = new int[N];
  //запускаем цикл по всем элементам массива
  for (i = 0; i < N; i++)
    {
     //вводим i-ый элемент
     Console.Write("Введите {0}-й элемент массива ", i);
     s = Console.ReadLine();
     massiv1[i] = Convert.ToInt32(s);
    }
 }
static void process(ref int[] massiv1)
 {
  for(int j=0;j< massiv1.Length;j++)
    {
     int f = 1;
     for (int i=2;i<=massiv1[j];i++)
     f=f*i;
     massiv1[j] = f;
    }
 }
static void output(int[] mas)
 {
  Console.WriteLine("Список элементов массива");
  foreach (int i in mas)
  Console.Write("{0} ", i);
 }
static void Main(string[] args)
 {
  int[] my_massiv;
  enter(out my_massiv);
  process(ref my_massiv);
  output(my_massiv);
  Console.ReadKey();
 }
```

7. Текстовые файлы

Установочные и завершающие операции

Используя классы **StreamWriter** и **StreamReader** можно создать свои файловые объекты для выполнения различных операций с текстовыми файлами.

```
//создание нового файла или перезапись существующего
StreamWriter outStream =
  new StreamWriter(@"c:\a1.txt");
//outStream - созданный нами объект
//открытие существующего файла на чтение
StreamReader inStream =
  new StreamReader(@"c:\a1.txt");
//inStream - созданный нами объект
```

Для завершения работы с файлом используют метод Close.

```
outStream.Close();
inStream.Close();
```

Операции ввода-вывода

Для записи в открытый файл используют методы Write и Writeline.

```
outStream.WriteLine(x);
outStream.Write("Z={0}",z);
outStream.Write();
```

Для чтения строки из открытого файла используют метод ReadLine.

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

Для проверки, достигнут ли конец файла при чтении используют метод **Peek** или свойство **EndOfStream**.

```
while (inStream.Peek()>-1)
 {
  string s = inStream.ReadLine();
  Console.Writeline(s);
 }
//равнозначный вариант
while (!inStream.EndOfStream)
 {
  string s = inStream.ReadLine();
  Console.Writeline(s);
 }
```

Для того, чтобы считывать отдельные элементы из строки, её считывают целиком, а затем разбивают на массив элементов с помощью метода **Split**.

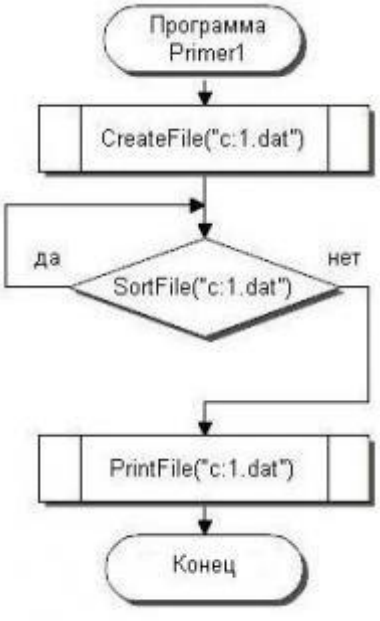
```
while (inStream.Peek()>-1)
 {
```

```
string s = inStream.ReadLine();
string[] elements = s.Split( ' ', ',', '.', ':', '\t' );
foreach (string e in elements)
 {
  if (e != "")
   {
    int d = Convert.ToInt32(e);
    Console.Write("{0}  ", d);
   }
 }
Console.WriteLine();
}
```

Лабораторная работа №4

Задание на лабораторную: В среде Microsoft Visual C# , решить 2 задачи по вариантам и оформить отчёт.
Примеры решённых задач

Пример №1
Написать программу, которая вводит с клавиатуры файл целых чисел, упорядочивает его по возрастанию и выдает на экран дисплея содержимое полученного файла. Массивы использовать нельзя.



Блок-схема главной программы

Блок-схемы подпрограмм

Решение

Описание алгоритма: Ввод данных в файл осуществим используя реализацию последовательного считывания данных с клавиатуры используя цикл **for**. Упорядочивание файла будем осуществлять следующим образом: считываем из исходного файла первые два элемента и сравниваем их, в новый файл заносим меньший из них. Считываем новый элемент и повторяем сравнение. В результате будет создан новый файл, в котором элементы будут лишь частично упорядочены. Если в течении прохода по файлу будет хотя бы одна перестановка, установим специальную логическую переменную в значение "Истина" т.е. будем считать, что файл еще упорядочен не до конца. Удалим исходный файл, а полученный переименуем в исходный. В зависимости от состояния логической переменной, возможно потребуется выполнить ещё раз упорядочивание файла - для этого воспользуемся циклом с постусловием т.к. один проход по файлу нужен обязательно.

```
using System.IO;
...
static void CreateFile(string filename)
 {
  BinaryWriter outBin = new BinaryWriter
    (File.Open(filename, FileMode.Create));
  Console.Write("Введите количество элементов для записи в файл ");
  string s = Console.ReadLine();
  int n = Convert.ToInt32(s);
  for (int i = 1; i <= n; i++)
   {
    Console.Write("Введите {0}-й элемент файла ", i);
    s = Console.ReadLine();
    int m = Convert.ToInt32(s);
    outBin.Write(m);
   }
  outBin.Close();
 }
```

```
static bool SortFile(string filename1)
 {
  string filename2 = filename1 + ".bak";
  BinaryReader inBin = new BinaryReader
     (File.Open(filename1, FileMode.Open));
  BinaryWriter outBin = new BinaryWriter
     (File.Open(filename2, FileMode.Create));
  bool first = true;
  bool perest = false;
  int pre = 0;
  while ((inBin.PeekChar()) >= 0)
   {
    int e = inBin.ReadInt32();
    if (first) { pre = e; first = false; }
    else
     {
      if (e >= pre) { outBin.Write(pre); pre = e; }
      else { outBin.Write(e); perest = true; }
     }
   }
  outBin.Write(pre);
  inBin.Close();
  outBin.Close();
  File.Delete(filename1);
  File.Move(filename2, filename1);
  return perest;
}
static void PrintFile(string filename)
{
  BinaryReader inBin = new BinaryReader
     (File.Open(filename, FileMode.Open));
  while ((inBin.PeekChar()) >= 0)
    {
     int e = inBin.ReadInt32();
     Console.WriteLine(e);
    }
  inBin.Close();
}
static void Main(string[] args)
{
 CreateFile("c:\\s1.dat");
 do
   {
    } while (SortFile("c:\\s1.dat"));
```

```
 Console.WriteLine("Упорядоченный файл");
 PrintFile("c:\\s1.dat");
 Console.ReadKey();
}
```

Задание №1

Задачи на использование **двоичных файлов** . Условие вида "дан файл" означает, что пользователем вводится количество элементов и все элементы файла с клавиатуры. Осуществить ввод данных для файла, выполнить реализацию алгоритма обработки и создания нового файла, обеспечить вывод полученных результатов используя отдельные **методы** .

При обработке исходного файла считать число элементов в нём неизвестным. Не допускается использование массивов. **Для решения задачи предварительно составляется блок-схема** .

- 1 Дан файл целых чисел. Создать новый файл целых чисел, содержащий длины всех серий исходного файла (серией называется набор последовательно расположенных одинаковых элементов, а длиной серии — количество этих элементов). Например, для исходного файла с элементами 1, 5, 5, 5, 4, 4, 5 содержимое результирующего файла должно быть следующим: 1, 3, 2, 1.
- 2 Дан файл вещественных чисел. Создать файл целых чисел, содержащий длины всех убывающих последовательностей элементов исходного файла. Например, для исходного файла с элементами 1.7, 4.5, 3.4, 2.2, 8.5, 1.2 содержимое результирующего файла должно быть следующим: 3, 2. Последовательность не может иметь длину меньше, чем 2.
- 3 Дан файл целых чисел. Создать два новых файла, первый из которых содержит четные числа из исходного файла, а второй — нечетные (в том же порядке). Если четные или нечетные числа в исходном файле отсутствуют, то соответствующий результирующий файл оставить пустым.
- 4 Дан файл вещественных чисел. Создать файл целых чисел, содержащий номера всех локальных максимумов исходного файла в порядке возрастания (локальным максимумом называется элемент, который больше своих соседей, самый первый и самый последний элементы не могут считаться локальными максимумами).
- 5 Дан файл вещественных чисел. Создать на его основе новый файл в котором заменить каждый элемент исходного файла, кроме начального и конечного, на его среднее арифметическое с предыдущим и последующим элементом.
- 6 Дан файл целых чисел. Создать на его основе новый файл, размером в 15 элементов. Если исходный файл имеет размер больший или равный 15 - выбрать из него только первые 15 элементов. Если исходный файл имеет размер меньше, чем 15 - продублировать его элементы до необходимого количества.
- 7 Дан файл целых чисел. Создать на его основе новый файл в котором выполнить дублирование всех положительных элементов исходного файла.
- 8 Дан файл целых чисел. Создать на его основе новый файл в котором заменить каждое положительное число на три нулевых элемента.

- 9 Дан файл целых чисел. Создать на его основе новый файл в котором после каждого из первых трёх наибольших по абсолютной величине чисел вставить элемент со значением ноль.
- 10 Дан файл целых чисел. Создать на его основе новый файл в которой занести элементы исходного файла, образующие в сумме с двумя соседними чётное число.

8. Классы и объекты

Классы — это, по сути, шаблоны, по которым можно создавать объекты. Каждый объект содержит данные и методы, манипулирующие этими данными. Класс определяет, какие данные и функциональность может иметь каждый конкретный объект (называемый экземпляром) этого класса. Например, если имеется класс, представляющий заказчика, он может определять такие поля, как CustomerlD, FirstName, LastName и Address, которые нужны ему для хранения информации о конкретном заказчике. Он также может определять функциональность, которая работает с данными, хранящимися в этих полях. Вы создаете экземпляр этого класса для представления конкретного заказчика, устанавливаете значения полей экземпляра и используете его функциональность.

```
class PhoneCustomer
 {
  public const string DayOfSendingBill = "Monday";
  public int CustomerlD;
  public string FirstName;
  public string LastName;
 }
```

Структуры отличаются от классов тем, как они сохраняются в памяти и как к ним осуществляется доступ (классы — это ссылочные типы, размещаемые в куче, структуры — типы значений, размещаемые в стеке), а также некоторыми свойствами (например, структуры не поддерживают наследование). Из соображений производительности вы будете использовать структуры для небольших типов данных. Однако в отношении синтаксиса структуры очень похожи на классы. Главное отличие состоит в том, что при их объявлении используется ключевое слово struct вместо class. Например, если вы хотите, чтобы все экземпляры PhoneCustomer размещались в стеке, а не в управляемой куче, то можете написать следующий код:

```
struct PhoneCustomerStruct
{
public const string DayOfSendingBill = "Monday";
public int CustomerlD;
public string FirstName;
public string LastName;
}
```

При создании как классов, так и структур используется ключевое слово new для объявления экземпляра. В результате объект создается и инициализируется. В следующем примере по умолчанию все поля обнуляются:

```
PhoneCustomer myCustomer = new PhoneCustomer(); // работает с классом
PhoneCustomerStruct myCustomer2 = new PhoneCustomerStruct(); // работает
//со структурой
```

В большинстве случаев классы используются чаще структур. По этой причине в главе сначала рассматриваются классы, а затем обсуждается разница между класса-

ми и структурами и специфические причины выбора структур вместо классов. Если только не указано обратное, вы можете рассчитывать на то, что код, работающий с классом, будет также работать со структурой.

Классы

Данные и функции, объявленные внутри класса, известны как члены класса. В официальной терминологии Microsoft делается различие между данными-членами и функциями-членами. В дополнение к членам, классы могут содержать в себе вложенные типы (такие как другие классы). Доступность членов класса может быть описана как **public, protected, internal protected, private** или **internal**.

Данные-члены

Данные-члены — это те члены, которые содержат данные класса — поля, константы, события. Данные-члены могут быть статическими (static). Член класса является членом экземпляра, если только он не объявлен явно как static.

Поля (field) — это любые переменные, ассоциированные с классом. Вы уже видели поля в классе PhoneCustomer в предыдущем примере. После создания экземпляра объекта PhoneCustomer к его полям можно обращаться с использованием синтаксиса **Объект.ИмяПоля**, как показано в следующем примере:

```
PhoneCustomer Customerl = new PhoneCustomer();
Customerl.FirstName = "Simon";
```

Константы могут быть ассоциированы с классом тем же способом, что и переменные. Константа объявляется с помощью ключевого слова const. Если она объявлена как public, то в этом случае становится доступной извне класса.

```
class PhoneCustomer
{
public const string DayOfSendingBill = "Monday";
public int CustomerlD;
public string FirstName;
public string LastName;
}
```

Для того чтобы продемонстрировать классы на конкретных примерах, разработаем постепенно класс, инкапсулирующий информацию о зданиях, в том числе о домах, складских помещениях, учреждениях и т.д. В этом классе будут храниться три элемента информации о зданиях: количество этажей, общая площадь и количество жильцов. Ниже приведен первый вариант. В нем определены три переменные экземпляра: **Floors**, **Area** и **Occupants**. Как видим, в нём вообще отсутствуют методы. Это означает, что в настоящий момент этот класс состоит только из данных.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
namespace BuildingProbe
{
   class Building
   {
      public int Floors; // количество этажей
      public int Area; // общая площадь здания
      public int Occupants; // количество жильцов
   }

   //В этом классе объявляется объект типа Building,
   class Program
   {
      static void Main(string[] args)
      {
         Building house = new Building(); // создать объект типа Building
         int areaPP; // площадь на одного человека
         // Присвоить значения полям в объекте house,
         house.Occupants = 4;
         house.Area = 250;
         house.Floors = 2;
         // Вычислить площадь на одного человека.
         areaPP = house.Area / house.Occupants;

         Console.WriteLine("Дом имеет:\n " +
         house.Floors + " этажа\n " +
         house.Occupants + " жильца\n " +
         house.Area +
         " кв. метров общей площади, из них\n " +
         areaPP + " приходится на одного человека");
         Console.ReadLine();
      }
   }
}
```

Эта программа состоит из двух классов: **Building** и **Program**. В классе **Program** сначала создается экземпляр **house** класса **Building** с помощью метода **Main ()**, а затем в коде метода **Main ()** осуществляется доступ к переменным экземпляра **house** для присваивания им значений и последующего использования этих значений. Следует особо подчеркнуть, что **Building** и **BuildingDemo** — это два совершенно отдельных класса. Единственная взаимосвязь между ними состоит в том, что в одном из них создается экземпляр другого. Но, несмотря на то, что это раздельные классы, у кода из класса **Program** имеется доступ к членам класса **Building**, поскольку они объявлены как открытые (public). Если бы при их объявлении не был

указан спецификатор доступа public, то доступ к ним ограничивался бы пределами **Building**, а следовательно, их нельзя было бы использовать в классе **Program**.

События — это члены класса, позволяющие объекту уведомлять вызывающий код о том, что случилось нечто достойное упоминания, например, изменение свойства класса либо некоторое взаимодействие с пользователем. Клиент может иметь код, известный как обработчик событий, реагирующий на них.

Функции-члены

Функции-члены — это члены, которые обеспечивают некоторую функциональность для манипулирования данными класса. Они включают методы, свойства, конструкторы, финализаторы, операции и индексаторы.

- Методы (method) — это функции, ассоциированные с определенным классом. Как и данные-члены, по умолчанию они являются членами экземпляра. Они могут быть объявлены статическими с помощью модификатора static.

- Свойства (property) — это наборы функций, которые могут быть доступны клиенту таким же способом, как общедоступные поля класса. В C# предусмотрен специальный синтаксис для реализации чтения и записи свойств для классов, поэтому писать собственные методы с именами, начинающимися на Set и Get, не понадобится. Поскольку не существует какого-то отдельного синтаксиса для свойств, который отличал бы их от нормальных функций, создается иллюзия объектов как реальных сущностей, предоставляемых клиентскому коду.

- Конструкторы (constructor) — это специальные функции, вызываемые автоматически при инициализации объекта. Их имена совпадают с именами классов, которым они принадлежат, и они не имеют типа возврата. Конструкторы полезны для инициализации полей класса.

- Финализаторы (finalizer) похожи на конструкторы, но вызываются, когда среда CLR определяет, что объект больше не нужен. Они имеют то же имя, что и класс, но с предшествующим символом тильды (~). Предсказать точно, когда будет вызван финализатор, невозможно.

- Операции (operator) — это простейшие действия вроде + или -. Когда вы складываете два целых числа, то, строго говоря, применяете операцию + к целым. Однако C# позволяет указать, как существующие операции будут работать с пользовательскими классами (так называемая перегрузка операций).

- Индексаторы (indexer) позволяют индексировать объекты таким же способом, как массив или коллекцию.

Создание объектов

В предыдущем примере для объявления объекта типа **Building** использовалась следующая строка кода:

Building house = new Building();

Эта строка объявления выполняет три функции. Во-первых, объявляется переменная **house**, относящаяся к типу класса **Building**. Сама эта переменная не является объектом, а лишь переменной, которая может ссылаться на объект. Во-вторых, создается конкретная, физическая, копия объекта. Это делается с помощью оператора **new**. И наконец, переменной **house** присваивается ссылка на данный объект. Таким образом, после выполнения анализируемой строки объявленная переменная **house** ссылается на объект типа Building. Оператор **new** динамически (т.е. во время выполнения) распределяет память для объекта и возвращает ссылку на него, которая затем сохраняется в переменной.

Следовательно, в C# для объектов всех классов должна быть динамически распределена память. Как и следовало ожидать, объявление переменной **house**можно отделить от создания объекта, на который она ссылается, следующим образом:

Building house; // объявить ссылку на объект
house = new Building(); // распределить память для объекта типа Building

В первой строке объявляется переменная **house** в виде ссылки на объект типа **Building**. Следовательно, **house** — это переменная, которая может ссылаться на объект, хотя сама она не является объектом. А во второй строке создается новый объект типа **Building**, и ссылка на него присваивается переменной house. В итоге переменная **house** оказывается связанной с данным объектом. То обстоятельство, что объекты классов доступны по ссылке, объясняет, почему классы называются ссылочными типами. Главное отличие типов значений от ссылочных типов заключается в том, что именно содержит переменная каждого из этих типов. Так, переменная типа значения содержит конкретное значение. Например, во фрагменте кода

int x;
x = 10;
переменная 'x содержит значение 10, поскольку она относится к типу int, который является типом значения. Но в строке
Building house = new Building();
переменная **house** содержит не сам объект, а лишь ссылку на него.

Методы

Следует отметить, что официальная терминология C# делает различие между функциями и методами. Согласно этой терминологии, понятие "функция-член" включает не только методы, но также другие члены, не являющиеся данными, класса или структуры. Сюда входят индексаторы, операции, конструкторы, деструкторы, а также — возможно, несколько неожиданно — свойства. Они контрастируют с данными-членами: полями, константами и событиями.

Объявление методов

В C# определение метода состоит из любых модификаторов (таких как спецификация доступности), типа возвращаемого значения, за которым следует имя мето-

да, затем списка аргументов в круглых скобках и далее — тела метода в фигурных скобках:

```
[модификаторы] тип_возврата ИмяМетода([параметры])
{
// Тело метода
}
```

Каждый параметр состоит из имени типа параметра и имени, по которому к нему можно обратиться в теле метода. Вдобавок, если метод возвращает значение, то для указания точки выхода должен использоваться оператор возврата вместе с возвращаемым значением. Например:

```
public bool IsSquare(Rectangle rect)
{
return (rect.Height == rect.Width);
}
```

В этом коде применяется один из базовых классов .NET, System.Drawing.Rectangle, представляющий прямоугольники. Если метод не возвращает ничего, то в качестве типа возврата указывается void, поскольку вообще опустить тип возврата невозможно. Если же он не принимает аргументов, то все равно после имени метода должны присутствовать пустые круглые скобки. При этом включать в тело метода оператор возврата не обязательно — метод возвращает управление автоматически по достижении закрывающей фигурной скобки. Следует отметить, что метод может содержать любое необходимое количество операторов возврата:

```
public bool IsPositive(int value)
{
if (value < 0)
return false;
return true;
}
```

Используя возвращаемое значение, можно усовершенствовать рассматривавшийся ранее пример с классом **Building**. Вместо того чтобы вычислять величину площади на одного человека в методе **Main()** , лучше возвратить ее из этого метода **AreaPerPerson()** класса **Building**. Среди прочих преимуществ такого подхода следует особо отметить возможность использовать возвращаемое значение для выполнения других вычислений. Приведенный ниже пример представляет собой улучшенный вариант рассматривавшейся ранее программы с методом **AreaPerPerson ()**, возвращающим величину площади на одного человека.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace BuildingProbe
```

```
{
   class Building
   {
      public int Floors; // количество этажей
      public int Area; // общая площадь здания
      public int Occupants; // количество жильцов
      // Возвратить величину площади на одного человека,
      public int AreaPerPerson()
      {
         return Area / Occupants;
      }

   }

   //В этом классе объявляется объект типа Building,
   class Program
   {
      static void Main(string[] args)
      {
         Building house = new Building(); // создать объект типа Building
         int areaPP; // площадь на одного человека
         // Присвоить значения полям в объекте house,
         house.Occupants = 4;
         house.Area = 250;
         house.Floors = 2;
         // Вычислить площадь на одного человека.
         areaPP = house.AreaPerPerson();

         Console.WriteLine("Дом имеет:\n " +
         house.Floors + " этажа\n " +
         house.Occupants + " жильца\n " +
         house.Area +
         " кв. метров общей площади, из них\n " +
         areaPP + " приходится на одного человека");
         Console.ReadLine();

      }
   }
}
```

Вызов методов

В следующем примере, **MathTest**, демонстрируется определение и создание экземпляров классов, а также определение и вызов методов. Помимо класса, содержа-

щего метод **Main ()**, в нем определен класс по имени **MathTest**, содержащий набор методов и полей.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace MathProbe
{
   class Program
   {
      static void Main(string[] args)
      {
       // Попытка вызова некоторых статических функций
       Console.WriteLine ("Pi равно " + MathTest.GetPi ());
       int x = MathTest.GetSquareOf(5);
       Console.WriteLine("5 в квадрате равно " + x) ;
       // Создание объекта MathTest
       MathTest math = new MathTest (); // это способ, которым в C# создаются
       // экземпляры ссылочного типа
       // Вызов нестатических методов
       math.value = 30;
       Console.WriteLine("Поле value переменной math содержит " + math.value);
       Console.WriteLine("30 в квадрате равно " + math.GetSquare());
       Console.ReadLine();
      }
   }
// Определение класса MathTest, методы которого мы вызываем
   class MathTest
   {
      public int value;
      public int GetSquare()
      {
         return value * value;
      }
      public static int GetSquareOf(int x)
      {
         return x * x;
      }
      public static double GetPi()
      {
         return 3.14159;
      }
```

```
    }
}
```

Запуск на выполнение примера **MathTest** даст следующий результат:

```
Pi равно 3.14159
5 в квадрате равно 25
Поле value переменной math содержит 30
30 в квадрате равно 900
```

Как можно видеть в коде, класс **MathTest** содержит поле, которое хранит число, а также метод для вычисления квадрата этого числа. Кроме того, он включает два статических метода — один для возврата значения числа "пи" и другой — для вычисления квадрата числа, переданного в параметре.

Некоторые свойства этого класса на самом деле не могут служить примером правильного дизайна программы C#. Например, GetPi () следовало бы реализовать в виде константного поля, но остальная часть демонстрирует концепции, которые пока не рассматривались.

Перегрузка методов

В C# допускается совместное использование одного и того же имени двумя или более методами одного и того же класса, при условии, что их параметры объявляются по-разному. В этом случае говорят, что методы перегружаются, а сам процесс называется перегрузкой методов. Перегрузка методов относится к одному из способов реализации полиморфизма в C#.

В общем, для перегрузки метода достаточно объявить разные его варианты, а об остальном позаботится компилятор. Но при этом необходимо соблюсти следующее важное условие: тип или число параметров у каждого метода должны быть разными. Совершенно недостаточно, чтобы два метода отличались только типами возвращаемых значений. Они должны также отличаться типами или числом своих параметров. (Во всяком случае, типы возвращаемых значений дают недостаточно сведений компилятору C#, чтобы решить, какой именно метод следует использовать.) Разумеется, перегружаемые методы могут отличаться и типами возвращаемых значений. Когда вызывается перегружаемый метод, то выполняется тот его вариант, параметры которого соответствуют (по типу и числу) передаваемым аргументам.

Ниже приведен простой пример, демонстрирующий перегрузку методов.

```
namespace ConsoleApplication1
{
  class Overload
    {
    public void OvlDemo()
     {
      Console.WriteLine("Без параметров");
     }
```

```
    // Перегрузка метода OvlDemo с одним целочисленным параметром,
    public void OvlDemo(int a)
      {
       Console.WriteLine("Один параметр: " + a);
      }
    // Перегрузка метода OvlDemo с двумя целочисленными параметрами,
    public int OvlDemo(int a, int b)
      {
       Console.WriteLine("Два параметра: " + a + " " + b);
       return a + b;
      }
    // Перегрузка метода OvlDemo с двумя параметрами типа double,
    public double OvlDemo(double a, double b)
      {
       Console.WriteLine("Два параметра типа double: " +  a + " "+ b);
       return a + b;
      }
    }

  class Program
  {
     static void Main(string[] args)
      {
        Overload ob = new Overload();
        int resI;
        double resD;
        // Вызвать все варианты метода OvlDemoO .
        ob.OvlDemo() ;
        Console.WriteLine ();
        ob.OvlDemo (2);
        Console.WriteLine ();
        resI = ob.OvlDemo(4, 6);
        Console.WriteLine("Результат вызова метода ob.OvlDemo(4, 6): " + resI);
        Console.WriteLine ();
        resD = ob.OvlDemo(1.1, 2.32);
        Console.WriteLine("Результат вызова метода ob.OvlDemo(1.1, 2.32): " + resD);
        Console.ReadLine();
      }
  }
}
```

Вот к какому результату приводит выполнение приведенного выше кода.

```
Без параметров
Один параметр: 2
```

```
Два параметра: 4 6
Результат вызова метода ob.OvlDemo (4, 6): 10
Два параметра типа double: 1.1 2.32
Результат вызова метода ob.OvlDemo (1.1, 2.32): 3.42
```

Конструкторы

В приведенных выше примерах программ переменные экземпляра каждого объекта типа **Building** приходилось инициализировать вручную. Такой прием обычно не применяется в профессионально написанном коде C#. Кроме того, он чреват ошибками. Впрочем, существует лучший способ решить подобную задачу: воспользоваться конструктором. Конструктор инициализирует объект при его создании. У конструктора такое же имя, как и у его класса, а с точки зрения синтаксиса он подобен методу. Но у конструкторов нет возвращаемого типа, указываемого явно. Ниже приведена общая форма конструктора.

```
доступ имя_класса(список_параметров) {
// тело конструктора
}
```

Как правило, конструктор используется для задания первоначальных значений переменных экземпляра, определенных в классе, или же для выполнения любых других установочных процедур, которые требуются для создания полностью сформированного объекта. Кроме того, доступ обычно представляет собой модификатор доступа типа public, поскольку конструкторы зачастую вызываются в классе. А список_параметров может быть как пустым, так и состоящим из одного или более указываемых параметров.

У всех классов имеются конструкторы, независимо от того, определите вы их или нет, поскольку в C# автоматически предоставляется конструктор, используемый по умолчанию и инициализирующий все переменные экземпляра их значениями по умолчанию. Для большинства типов данных значением по умолчанию является нулевое, для типа bool — значение false, а для ссылочных типов — пустое значение.

Но как только вы определите свой собственный конструктор, то конструктор по умолчанию больше не используется. Ниже приведен простой пример применения конструктора.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConstructorProbe
{
   class MyClass
   {
      public int x;
      public MyClass() //конструктор
```

```
        {
            x = 10;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            MyClass tl = new MyClass();
            MyClass t2 = new MyClass();
            Console.WriteLine(tl.x + " " + t2.x);
            Console.ReadLine();
        }
    }
}
```

Деструкторы

Как было показано выше, при использовании оператора new свободная память для создаваемых объектов динамически распределяется из доступной буферной области оперативной памяти. Разумеется, оперативная память не бесконечна, и поэтому свободно доступная память рано или поздно исчерпывается. Это может привести к неудачному выполнению оператора new из-за нехватки свободной памяти для создания требуемого объекта. Именно по этой причине одной из главных функций любой схемы динамического распределения памяти является освобождение свободной памяти от неиспользуемых объектов, чтобы сделать ее доступной для последующего перераспределения. Во многих языках программирования освобождение распределенной ранее памяти осуществляется вручную. Например, в C++ для этой цели служит оператор delete. Но в C# применяется другой, более надежный подход: "сборка мусора".

Система "сборки мусора" в C# освобождает память от лишних объектов автоматически, действуя незаметно и без всякого вмешательства со стороны программиста. "Сборка мусора" происходит следующим образом. Если ссылки на объект отсутствуют, то такой объект считается ненужным, и занимаемая им память в итоге освобождается и накапливается. Эта утилизированная память может быть затем распределена для других объектов.

"Сборка мусора" происходит лишь время от времени по ходу выполнения программы. Она не состоится только потому, что существует один или более объектов, которые больше не используются. Следовательно, нельзя заранее знать или предположить, когда именно произойдет "сборка мусора".

В языке C# имеется возможность определить метод, который будет вызываться непосредственно перед окончательным уничтожением объекта системой "сборки мусора". Такой метод называется деструктором и может использоваться в ряде осо-

бых случаев, чтобы гарантировать четкое окончание срока действия объекта. Например, деструктор может быть использован для гарантированного освобождения системного ресурса, задействованного освобождаемым объектом. Следует, однако, сразу же подчеркнуть, что деструкторы — весьма специфические средства, применяемые только в редких, особых случаях. И, как правило, они не нужны. Но здесь они рассматриваются вкратце ради полноты представления о возможностях языка C#. Ниже приведена общая форма деструктора:

```
~имя_класса() {
// код деструктора
}
```

где имя_класса означает имя конкретного класса.

Следовательно, деструктор объявляется аналогично конструктору, за исключением того, что перед его именем указывается знак "тильда" (~). Обратите внимание на то, что у деструктора отсутствуют возвращаемый тип и передаваемые ему аргументы.

Для того чтобы добавить деструктор в класс, достаточно включить его в класс в качестве члена. Он вызывается всякий раз, когда предполагается утилизировать объект его класса. В деструкторе можно указать те действия, которые следует выполнить перед тем, как уничтожать объект. Следует, однако, иметь в виду, что деструктор вызывается непосредственно перед "сборкой мусора". Он не вызывается, например, в тот момент, когда переменная, содержащая ссылку на объект, оказывается за пределами области действия этого объекта. (В этом отношении деструкторы в C# отличаются от деструкторов в C++, где они вызываются в тот момент, когда объект оказывается за пределами области своего действия.) Это означает, что заранее нельзя знать, когда именно следует вызывать деструктор. Кроме того, программа может завершиться до того, как произойдет "сборка мусора", а следовательно, деструктор может быть вообще не вызван.

Ниже приведен пример программы, демонстрирующий применение деструктора. В этой программе создается и уничтожается большое число объектов. В какой-то момент по ходу данного процесса активизируется "сборка мусора" и вызываются деструкторы для уничтожения ненужных объектов.

```
namespace ConsoleApplication1
{
  class Destruct {
  public int x;
  public Destruct(int i) {
     x = i;
  }
// Вызывается при утилизации объекта.
  ~Destruct() {
     Console.WriteLine("Уничтожить " + x);
   }
// Создает объект и тут же уничтожает его.
```

```
public void Generator(int i) {
      Destruct o = new Destruct (i);
      }
}
 class Program
   {
      static void Main(string[] args)
      {
         int count;
         Destruct ob = new Destruct (0);
         /* А теперь создать большое число объектов.
          В какой-то момент произойдет "сборка мусора".
          Примечание: для того чтобы активизировать
          "сборку мусора", возможно, придется увеличить
          число создаваемых объектов. */
         for (count=1; count < 100000; count++)
         ob.Generator(count);
         Console.WriteLine("Готово!");
         Console.ReadLine();
      }
   }
}
```

Эта программа работает следующим образом. Конструктор инициализирует переменную x известным значением. В данном примере переменная x служит в качестве идентификатора объекта. А деструктор выводит значение переменной x, когда объект утилизируется. Особый интерес вызывает метод Generator (), который создает и тут же уничтожает объект типа Destruct. Сначала в создается исходный объект ob типа Destruct, а затем осуществляется поочередное создание и уничтожение 100 тыс. объектов. В разные моменты этого процесса происходит "сборка мусора". Насколько часто она происходит — зависит от нескольких факторов, в том числе от первоначального объема свободной памяти, типа используемой операционной системы и т.д. Тем не менее в какой-то момент начинают появляться сообщения, формируемые деструктором. Если же они не появятся до окончания программы, т.е. до того момента, когда будет выдано сообщение "Готово!", попробуйте увеличить число создаваемых объектов, повысив предельное количество подсчитываемых шагов в цикле for. И еще одно важное замечание: метод WriteLine () вызывается в деструкторе -Destruct () исключительно ради наглядности данного примера его использования.

Как правило, деструктор должен воздействовать только на переменные экземпляра, определенные в его классе. В силу того что порядок вызова деструкторов не определен точно, их не следует применять для выполнения действий, которые должны происходить в определенный момент выполнения программы.

Свойства

Еще одной разновидностью члена класса является свойство. Как правило, свойство сочетает в себе поле с методами доступа к нему. Как было показано в приведенных ранее примерах программ, поле зачастую создается, чтобы стать доступным для пользователей объекта, но при этом желательно сохранить управление над операциями, разрешенными для этого поля, например, ограничить диапазон значений, присваиваемых данному полю. Этой цели можно, конечно, добиться и с помощью закрытой переменной, а также методов доступа к ее значению, но свойство предоставляет более совершенный и рациональный путь для достижения той же самой цели. Свойства очень похожи на индексаторы. В частности, свойство состоит из имени и аксессоров **get** и **set**. Аксессоры служат для получения и установки значения переменной. Главное преимущество свойства заключается в том, что его имя может быть использовано в выражениях и операторах присваивания аналогично имени обычной переменной, но в действительности при обращении к свойству по имени автоматически вызываются его аксессоры **get** и **set**. Аналогичным образом используются аксессоры **get** и **set** индексатора.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace PropertiesProbe
{
   class SimpProp
   {
      int prop; // поле, управляемое свойством MyProp

      public SimpProp() { prop = 0; } //Конструктор

      /* Это свойство обеспечивает доступ к закрытой переменной экземпляра prop.
      Оно допускает присваивание только положительных значений. */
      public int MyProp
      {
         get
         {
            return prop;
         }
         set
         {
            if (value >= 0) prop = value;
         }
      }
   }
```

```
   class Program
   {
      static void Main(string[] args)
      {
         SimpProp ob = new SimpProp();
         Console.WriteLine("Первоначальное значение ob.MyProp: " + ob.MyProp );
         ob.MyProp = 100; // присвоить значение
         Console.WriteLine("Текущее значение ob.MyProp: " + ob.MyProp);
         // Переменной prop нельзя присвоить отрицательное значение.
         Console.WriteLine("Попытка присвоить значение " +
         "-10 свойству ob.MyProp");
         ob.MyProp = -10;
         Console.WriteLine("Текущее значение ob.MyProp: " + ob.MyProp);

         Console.ReadLine();

      }
   }
}
```

Статические классы

Если класс не содержит ничего кроме статических методов и свойств, этот класс сам по себе может стать статическим. Статический класс функционально представляет собой то же самое, что и класс с приватным статическим конструктором. Создать экземпляр такого класса невозможно. Если указать ключевое слово static в объявлении класса, компилятор будет гарантировать, что к этому классу никогда не будут добавлены нестатические члены. В противном случае будет выдана ошибка компиляции. Это гарантирует также, что экземпляры этого класса никогда не будут созданы. Синтаксис объявления статического класса выглядит следующим образом:

```
static class StaticUtilities
  {
   public static void HelperMethod ()
    {
    }
  }
```

Объект типа StaticUtilities не нужен для вызова HelperMethod(). При вызове указывается имя типа: StaticUtilities.HelperMethod(); .

Модификаторы

Модификаторы могут указывать видимость метода, как, например public или private, или же их природу, например, virtual или abstract. В языке C# определено множество модификаторов, и

сейчас стоит потратить некоторое время на ознакомление с их полным списком.

Модификаторы видимости

Модификаторы видимости указывают, какие другие единицы кода могут видеть элемент

Модификатор
К чему относится
Описание
public
К любым типам или членам
Элемент видим в любом другом коде
protected
к любому члену типа, а также к любому вложенному типу
Элемент видим только любому производному типу
internal
к любым типам или членам
Элемент видим только в пределах включающей его сборки
private
К любому члену типа, а также к любому вложенному типу
Элемент видим только в пределах типа, которому он принадлежит
protected internal
к любому члену типа, а также к любому вложенному типу
Элемент видим только в пределах включающей его сборки, а также в любом коде внутри производного типа

Определения типа могут быть общедоступными или приватными в зависимости от того, хотите ли вы обеспечить его видимость извне сборки. Указывать модификаторы protected, private или protected internal для типов нельзя, поскольку эти уровни видимости не имеют смысла для типа, находящегося в пространстве имен. Это значит, что они могут относиться только к членам. Однако возможно создавать вложенные типы (т.е. типы, содержащиеся внутри других типов) с такой видимостью, поскольку в этом случае типы имеют статус члена. Таким образом, приведенный ниже код вполне корректен:

```
public class OuterClass
 {
  protected class InnerClass
     {
       // и т.д.
     }
 // и т.д.
 }
```

Если есть вложенный тип, он всегда может иметь доступ ко всем членам внешнего типа. Таким образом, в последнем примере любой код внутри InnerClass всегда имеет доступ ко всем членам OuterClass, даже если они объявлены как private.

Другие модификаторы

Модификаторы, перечисленные ранее, могут быть применены к членам типов и характеризуются различным использованием. Некоторые из них также имеет смысл использовать для типов.

Модификатор
К чему относится
Описание
new
К функциям-членам
Член скрывает унаследованный член с той же сигнатурой
static
Ко всем членам
Член не связан с конкретным экземпляром класса
virtual
Только к классам и функциям-членам
Член может быть переопределен в классах - наследниках
abstract
Только к функциям-членам
Виртуальный член, определяющий сигнатуру, но не предоставляющий реализации
override
Только к функциям-членам
Член переопределяет унаследованный виртуальный или абстрактный член базового класса
sealed
К классам, методам и свойствам
Для классов означает, что от таких классов нельзя наследовать. Для свойств и методов — член переопределяет унаследованный виртуальный член, но не может быть переопределен ни одним членом производных классов. Должен применяться в сочетании с override
extern
Только к статическим методам [Dllimport]
Член реализован внешне, на другом языке

Наследование

В объектно-ориентированном программировании (ООП) существуют два различных типа наследования: наследование реализации и наследование интерфейса.

1. Наследование реализации (implementation inheritance) означает, что тип происходит от базового типа, получая от него все поля-члены и функции-члены. При наследовании реализации производный тип адаптирует реализацию каждой функции базового типа, если только в его определении не указано, что реализация функции должна быть переопределена. Такой тип наследования более полезен, когда

нужно добавить функциональность к существующему типу или же когда несколько связанных типов разделяют существенный объем общей функциональности.

2. Наследование интерфейса (interface inheritance) означает, что тип наследует только сигнатуру функций, но не наследует никакой реализации. Этот тип наследования наиболее полезен, когда нужно специфицировать, что тип обеспечивает доступ к определенным средствам.

В C# поддерживается как наследование реализации, так и наследование интерфейса. Оба типа наследования полностью встроены в язык с самого начала, позволяя принимать решение о том, какой из них использовать, на основе архитектуры приложения.

Некоторые языки, такие как C++, поддерживают то, что известно под названием множественного наследования, когда класс происходит более чем от одного базового класса. Преимущества множественного наследования спорны. С одной стороны, нет сомнений, что можно применять множественное наследование для написания чрезвычайно сложного, но при этом компактного кода, что демонстрирует библиотека C++ ATL. С другой стороны, код, использующий множественное наследование, часто трудно понять и сложно отлаживать (все это также демонстрирует библиотека C++ ATL). Как уже упоминалось, облегчение написания устойчивого кода было одной из ключевых целей проектирования C#. Соответственно, поэтому в C# множественное наследование не поддерживается. Однако C# позволяет типу наследовать множество интерфейсов. Это значит, что класс C# может наследоваться от другого класса и любого количества интерфейсов. На самом деле можно сказать точнее: благодаря наличию System.Object как всеобщего базового типа, каждый класс C# (за исключением Object) имеет строго один базовый класс и дополнительно может иметь любое количество базовых интерфейсов.

Наследование реализации

Для объявления, что класс наследуется от другого класса, применяется следующий синтаксис:

```
class УнаследованныйКласс: БазовыйКласс
{
 //Данные-члены и функции-члены
}
```

Этот синтаксис очень похож на синтаксис C++ и Java. Однако программисты на C++, знакомые с концепцией общедоступного и приватного наследования, должны обратить внимание, что C# не поддерживает приватного наследования; этим объясняется отсутствие квалификатора public или private перед именем базового класса. Поддержка приватного наследования значительно усложняет язык, при этом приносится весьма небольшую выгоду. На практике приватное наследование в C++ все равно используется чрезвычайно редко. Если класс также наследует интерфейсы, то список базового класса и интерфейсов

разделяется запятыми:

```
public class MyDerivedClass: MyBaseClass, IInterface1, IInterface2
{
  // и т.д.
}
```

Для структур синтаксис выглядит так:

```
public struct MyDerivedStruct: IInterface1, IInterface2
 {
   // и т.д.
 }
```

Если при определении класса базовый класс не указан, то компилятор C# предполагает, что базовым классом является System.Object. Поэтому следующие два фрагмента кода эквивалентны:

```
class MyClass: Object // наследуется от System.Object
 {
   // и т.д.
 }
```

и

```
class MyClass // наследуется от System.Object
{
// и т.д.
}
```

Для простоты чаще применяется вторая форма.

Поскольку в C# поддерживается ключевое слово object, служащее псевдонимом класса System.Object, можно записать и так:

```
class MyClass: object // наследуется от System.Object
{
// и т.д.
}
```

Чтобы сослаться на класс Object, используйте ключевое слово object, которое распознается интеллектуальными редакторами вроде Visual Studio .NET. Это облегчит редактирование кода.

Виртуальные методы

Объявляя функцию базового класса как virtual, вы тем самым позволяете ее переопределять в классах-наследниках:

```
class MyBaseClass
 {
  public virtual string VirtualMethod ()
   {
    return "Это - виртуальный метод, определенный в MyBaseClass";
   }
 }
```

Также допускается объявление свойства как virtual. Для виртуального или переопределенного свойства используется такой же синтаксис, что и для невиртуального свойства, за исключением ключевого слова virtual, добавляемого к определению. Синтаксис выглядит следующим образом:

```
public virtual string ForeName
{
  get { return foreName; }
  set { foreName = value; }
}
private string foreName;
```

Для простоты далее речь пойдет в основном о методах, хотя все сведения касаются также и свойств. Концепция, лежащая в основе виртуальных функций C#, идентична стандартной концепции ООП. Виртуальную функцию можно переопределить в классе-наследнике, и когда этот метод будет вызван, запустится его версия, относящаяся к соответствующему типу объекта. В C# по умолчанию функции не являются виртуальными, но (в отличие от конструкторов) могут быть явно объявлены как virtual. Это следует методологии C++: по причинам, связанным с производительностью, функции не виртуальные, если это не указано явно. В отличие от этого, в Java все функции виртуальные. C# имеет отличающийся от C++ синтаксис, поскольку требует явного объявления, когда функция класса-наследника переопределяет другую функцию, с помощью ключевого слова override:

```
class MyDerivedClass: MyBaseClass
 {
  public override string VirtualMethod()
 {
  return "Этот переопределенный метод объявлен в MyDerivedClass";
 }
 }

class Program
  {
     static void Main(string[] args)
     {
        MyBaseClass x = new MyBaseClass();
        Console.WriteLine(x.VirtualMethod());

        MyDerivedClass y = new MyDerivedClass();
        Console.WriteLine(y.VirtualMethod());

        Console.ReadLine();

     }
  }
```

Этот синтаксис переопределения метода исключает потенциальные ошибки времени выполнения, которые могут легко возникать в C++, когда сигнатура метода в классе-наследнике непреднамеренно оказывается отличной от базовой версии, в результате чего метод наследника не может переопределить базовый метод. В C# это всплывает в виде ошибки компиляции, поскольку компилятор легко обнаруживает метод, для которого указан модификатор override, но при этом не имеющий базового метода, который он переопределяет.

Ни поля-члены, ни статические функции не могут быть объявлены виртуальными. Эта концепция просто не имеет смысла ни для каких членов класса, за исключением функций-членов уровня экземпляра.

Вызов базовых версий функций

В C# предусмотрен специальный синтаксис вызова базовых версий метода из производного класса: base. <ИмяМетода> (). Например, если необходимо, чтобы метод производного класса возвращал 90% значения, возвращенного методом базового класса, можете воспользоваться следующим синтаксисом:

```
class CustomerAccount
 {
  public virtual decimal CalculatePrice ()
    {
     // реализация
     return 30.0M;
    }
 }
class GoldAccount: CustomerAccount
 {
  public override decimal CalculatePrice ()
    {
     return base.CalculatePrice () * 0.9M;
    }
 }

class Program
  {
    static void Main(string[] args)
    {
       GoldAccount x = new GoldAccount();
       Console.WriteLine(x.CalculatePrice());

       Console.ReadLine();
    }
  }
```

Отметим, что синтаксис base.<ИмяМетода>() можно использовать для вызова любого метода базового класса — вы не обязаны вызывать его только из переопределенной версии того же метода.

Абстрактные классы и функции

Язык C# позволяет объявлять абстрактными и классы, и функции. Создавать экземпляры абстрактных классов нельзя, поскольку абстрактные функции не имеют реализации и должны быть переопределены в любом неабстрактном классе-наследнике. Очевидно, что абстрактные функции автоматически являются виртуальными (хотя вы не должны применять ключевое слово virtual, так как это приведет к синтаксической ошибке). Если любой класс содержит любую абстрактную функцию, этот класс сам является абстрактным и должен быть объявлен таковым:

```
abstract class Building
  {
    public abstract decimal CalculateHeatingCost (); // абстрактный метод
  }
class MyBuilding : Building
  {
    public override decimal CalculateHeatingCost()
    {
      return 35.2M;
    }
  }

class Program
  {
    static void Main(string[] args)
    {
      MyBuilding x = new MyBuilding();
      Console.WriteLine(x.CalculateHeatingCost());

      Console.ReadLine();
    }
  }
```

Разработчики на C++ отметят также отличие в терминологии: в C++ абстрактные функции часто описываются как чистые виртуальные; в мире же C# единственный корректный термин для них — абстрактные.

Запечатанные классы и методы

C# позволяет объявлять классы и методы как sealed (запечатанные). В случае класса это значит, что наследовать от него нельзя. В случае метода это означает невозможность его переопределения.

```
sealed class FinalClass
```

```
 {
  // и т.д.
 }
class DerivedClass: FinalClass // Неверно. Ошибка компиляции.
 {
  // и т.д.
 }
```

Наиболее вероятная ситуация, когда может понадобиться пометить класс или метод как sealed — это когда класс или метод обеспечивает внутренние действия библиотеки, класса или других разрабатываемых классов, поэтому вы уверены, что любая попытка переопределить некоторую его функциональность приведет к нестабильности кода. Также можно помечать класс или метод как sealed из коммерческих соображений, чтобы предотвратить использование классов способом, противоречащим лицензионным соглашениям. Вообще говоря, нужно быть осторожным с объявлением классов как sealed, потому что, поступая так, вы в некоторой степени ограничиваете возможности их использования. Даже если вы не думаете, что понадобится наследовать от класса или переопределять его члены, все же существует вероятность, что в какой-то момент в будущем кто-то столкнется с ситуацией, которую вы не смогли предвидеть. В библиотеке базовых классов .NET часто встречаются запечатанные классы. Это защищает их от независимых разработчиков, которые могут пожелать унаследовать от них собственные классы. Так, например, string — запечатанный класс. Объявление метода sealed служит той же цели, что и для класса:

```
class MyClass: MyClassBase
 {
  public sealed override void FinalMethod()
   {
    // и т.д.
   }
 }
class DerivedClass: MyClass
 {
  public override void FinalMethod () // Неверно. Ошибка компиляции.
   {
   }
 }
```

Для того чтобы можно было применить ключевое слово sealed к методу или свойству, они должны быть сначала переопределены по отношению к базовому классу. Если вы не хотите, чтобы метод или свойство базового класса переопределялось, то просто не помечайте их как virtual.

Интерфейсы

Как упоминалось ранее, наследуя интерфейс, класс тем самым декларирует, что он реализует определенные функции. Поскольку не все объектно-ориентированные языки поддерживают интерфейсы, в этом разделе подробно описана реализация интерфейсов C#. В этом разделе интерфейсы рассматриваются путем представления полного определения одного из интерфейсов от Microsoft — System. IDisposable. Интерфейс IDisposable содержит один метод Dispose (), предназначенный для реализации классами, которые осуществляют очистку кода:

```
public interface IDisposable
{
  void Dispose();
}
```

Этот фрагмент показывает, что объявление интерфейса синтаксически очень похоже на объявление абстрактного класса. Однако вы должны знать, что ни для одного из членов интерфейса не допускается какой-либо реализации. В общем случае, интерфейс может содержать только объявления методов, свойств, индексов и событий. Создавать экземпляр интерфейса нельзя — он содержит только сигнатуры своих членов. Интерфейс не имеет конструкторов (как можно сконструировать нечто, экземпляр чего не создается?), равно как и полей (поскольку это подразумевает некоторую внутреннюю реализацию). Определению интерфейса также не разрешено содержать перегрузки операций, причем не потому, что с этим связаны какие-то принципиальные проблемы. Причина в том, что назначение интерфейсов состоит в том, чтобы служить общедоступными контрактами, для которых перегрузка операций вызывала бы определенные

проблемы совместимости с другими языками .NET, такими как Visual Basic и .NET, которые не поддерживают перегрузку операций.

Также при определении членов интерфейса не разрешены модификаторы. Члены интерфейса всегда неявно являются public и не могут быть virtual или static. Это оставлено на усмотрение реализаций классов. Таким образом, вполне нормально указывать модификаторы доступа к членам интерфейса в реализующих их классах, что и делается в примерах настоящего раздела.

Рассмотрим, например, интерфейс IDisposable. Если класс пожелает объявить, что он реализует метод Dispose (), то он должен будет реализовать интерфейс IDisposable, что в терминах C# означает, что он наследуется от IDisposable.

```
class SomeClass: IDisposable
 {
  // Этот класс ДОЛЖЕН содержать реализацию
  // метода IDisposable.Dispose (), иначе
  // возникнет ошибка компиляции.
  public void Dispose ()
    {
     // реализация метода Dispose ()
    }
  // остальная часть класса
```

```
}
```

В этом примере, если SomeClass будет наследовать IDisposable, но не будет содержать реализации Dispose (), в точности совпадающей с сигнатурой, определенной в IDisposable, будет выдана ошибка компиляции, поскольку в этом случае класс нарушит контракт реализации интерфейса IDisposable. Разумеется, для компилятора не будет никакой проблемы, если встретится класс, включающий метод Dispose (), но не унаследованный от IDisposable. Проблема будет в том, что другой код не будет иметь возможности распознать, что SomeClass согласен поддерживать средства IDisposable. IDisposable — сравнительно простой интерфейс, потому что в нем определен только один метод. Большинство интерфейсов содержат гораздо большее количество методов.

Определение и реализация интерфейсов

В этом разделе показано, как определять и использовать интерфейсы при разработке короткой программы, реализующей парадигму наследования интерфейсов. Пример описывает банковский счет. Предположим, что вы пишете код, который в конечном итоге обеспечит компьютеризованный перевод денег между банковскими счетами. Пусть существует множество компаний, которые могут реализовывать банковские счета, но все они согласились с тем, что любые классы, представляющие банковские счета, должны реализовывать интерфейс IBankAccount, предусматривающий методы для внесения и съема денежных сумм, а также свойство, возвращающее баланс. Это тот интерфейс, который позволит внешнему коду распознавать различные классы банковских счетов, реализующие различные формы таких счетов. Хотя целью этого является обеспечение взаимодействия банковских счетов между собой для перевода денег, пока мы не будем представлять эту возможность.

В целях упрощения поместим весь код примера в единственный исходный файл. Конечно, если что-то вроде этого примера придется делать в реальной жизни, то можно догадаться, что различные классы банковских счетов не только будут компилироваться в различные сборки, но также будут развернуты на разных компьютерах, принадлежащих разным банкам. Пока все это чересчур сложно для немедленного рассмотрения. Однако чтобы внести определенную долю реализма, определим разные пространства имен для разных компаний.

Для начала потребуется определить интерфейс IBankAccount:

```
namespace Wrox.ProCSharp
 {
  public interface IBankAccount
   {
    void Payln(decimal amount);
    bool Withdraw(decimal amount);
    decimal Balance
     {
      get;
     }
```

```
    }
  }
```

Обратите внимание на имя интерфейса — IBankAccount. Существует соглашение, что имя интерфейса традиционно начинается с буквы I, чтобы сразу было понятно, что это интерфейс.

В большинстве случаев руководство по .NET отвергает так называемую венгерскую нотацию, согласно которой имена предваряются буквой, указывающей на тип определяемого объекта. Интерфейсы — одно из исключений из этого правила, в котором венгерская нотация как раз рекомендуется. Идея заключается в том, что теперь имеется возможность писать классы, представляющие банковские счета. Эти классы не должны быть как-то связанными друг с другом; они могут быть полностью различными. Однако все они декларируют свое представление банковских счетов тем, что реализуют интерфейс IBankAccount.

Начнем с первого класса, описывающего сберегательный счет в Королевском Банке Венеры:

```
namespace Wrox.ProCSharp.VenusBank
  {
    public class SaverAccount: IBankAccount
     {
       private decimal balance;
       public void Payln(decimal amount)
         {
           balance += amount;
         }
       public bool Withdraw(decimal amount)
         {
           if (balance >= amount)
            {
              balance -= amount;
              return true;
            }
           Console. WriteLine ("Попытка перевода денег не удалась.");
           return false;
         }
       public decimal Balance
        {
          get
           {
             return balance;
           }
        }
   public override string ToString()
    {
```

```
        return String.Format("Сберегательный Банк Венеры: Баланс = {0,6:C}", bal-
ance);
      }
   }
}

class Program
   {
      static void Main(string[] args)
      {
         IBankAccount venusAccount = new SaverAccount();
         venusAccount.Payln(200);
         venusAccount.Withdraw(100);
         Console.WriteLine(venusAccount.ToString());

         Console.ReadLine();

      }
   }
```

Достаточно очевидно, что делает реализация этого класса. Вы создаете приватное поле balance и изменяете сумму остатка, указанную в нем, при съеме и зачислении денег на счет. Если предпринимается попытка снять больше денег, чем осталось на счету, выдается сообщение об ошибке. Обратите внимание, что для простоты кода здесь не реализуются дополнительные свойства, такие как имя владельца счета. В реальной жизни эта информация совершенно необходима, но в данный пример это внесло бы излишнюю сложность. Единственной интересной строкой в этом коде является объявление класса: public class SaverAccount: IBankAccount . Она объявляет, что SaverAccount наследуется от одного интерфейса, IBankAccount, и никакого другого базового класса не указано (что, конечно же, означает, что SaverAccount наследуется от System.Object). Кстати говоря, наследование интерфейсов совершенно независимо от наследования классов.

То, что SaverAccount наследуется от IBankAccount, означает, что у SaverAccount есть все члены IBankAccount. Но поскольку сам интерфейс не реализует ни одного из своих методов, SaverAccount должен предоставить для них собственную реализацию. Если реализация любого из них опущена, компилятор это заметит и выдаст соответствующее уведомление. Вспомните также о том, что интерфейс просто указывает на присутствие своих членов. Решение о том, объявлять их virtual или abstract, возложено на класс (хотя абстрактные функции, конечно же, допускаются только в абстрактных классах). Что касается конкретного примера, то здесь нет никаких причин объявлять любую из функций интерфейса виртуальной. Чтобы проиллюстрировать, как различные классы могут реализовать один и тот же интерфейс, предполо-

жим, что Планетарный Банк Юпитера также реализует собственный класс, представляющий банковские счета — GoldAccount:

```
namespace Wrox.ProCSharp.JupiterBank
 {
  public class GoldAccount: IBankAccount
   {
     // и т.д.
   }
 }
```

Мы не будем здесь рассматривать детали класса GoldAccount; в коде примера он будет практически идентичен реализации SaverAccount. Подчеркнем, что GoldAccount не имеет никакой связи с SaverAccount кроме того, что оба они реализуют один и тот же интерфейс.

Теперь, имея готовые классы, их можно протестировать. Первым делом, понадобится несколько операторов using:

```
using System;
using Wrox.ProCSharp;
using Wrox.ProCSharp.VenusBank;
using Wrox.ProCSharp.JupiterBank;

namespace Wrox.ProCSharp
 {
  class MainEntryPoint
 {
   static void Main()
    {
     IBankAccount venusAccount = new SaverAccount ();
     IBankAccount jupiterAccount = new GoldAccount ();
     venusAccount.Payln(200);
     venusAccount.Withdraw(100) ;
     Console.WriteLine(venusAccount.ToString());
     jupiterAccount.Payln(500) ;
     jupiterAccount.Withdraw(600);
     jupiterAccount.Withdraw(100);
     Console.WriteLine(jupiterAccount.ToString());
    }
  }
}
```

Этот код сгенерирует следующий вывод:

```
Сберегательный Банк Венеры: Баланс = £100.00
Попытка перевода денег не удалась.
Планетарный Банк Юпитера: Баланс = £400.00
```

Главный момент, который следует здесь отметить — способ объявления обеих переменных как ссылок на IBankAccount. Это значит, что они могут указывать на любой экземпляр любого класса, реализующего интерфейс. Это также означает, что через эти ссылки можно вызывать только те методы, которые являются частью интерфейса. Если понадобится вызвать любые методы, реализованные классом, но не являющиеся частью интерфейса, то придется выполнить приведение ссылки к соответствующему типу. В примере кода вызывается метод ToString () (не объявленный в IBankAccount) без какого-либо явного приведения, просто потому, что ToString () — это метод System.Object, поэтому компилятор C# знает о том, что он поддерживается любым классом (иначе говоря, приведение любого интерфейса к System.Object осуществляется неявно).

Ссылки на интерфейсы во всех отношениях могут трактоваться как ссылки на классы — однако мощь интерфейсных ссылок в том, что они могут указывать на любые классы, реализующие данный интерфейс. Например, это позволяет формировать массивы интерфейсов, элементы которых являются объектами разных классов:

```
IBankAccount [ ] accounts = new IBankAccount [2] ;
accounts [0] = new SaverAccount ();
accounts [1] = new GoldAccount () ;
```

Однако если попытаться сделать что-то вроде такого:

```
accounts [1] = new SomeOtherClass (); // SomeOtherClass не реализует
                                      // IBankAccount: НЕВЕРНО!!
```

будет получена следующая ошибка компиляции: Cannot implicitly convert type 'Wrox.ProCSharp.SomeOtherClass' to 'Wrox. ProCSharp.IBankAccount' Неявное преобразование типа 'Wrox.ProCSharp.SomeOtherClass' в 'Wrox.ProCSharp. IBankAccount' невозможно

Лабораторная работа №5

Задание на лабораторную: В среде **Microsoft Visual C#** , решить 5 задач по вариантам и оформить отчёт.

Пример №1. Класс, инкапсулирующий информацию о зданиях, в том числе о домах, складских помещениях, учреждениях и т.д. В этом классе будут храниться три элемента информации о зданиях: количество этажей, общая площадь и количество жильцов.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace BuildingProbe
{
   class Building
   {
```

```
    public int Floors; // количество этажей
    public int Area; // общая площадь здания
    public int Occupants; // количество жильцов
  }

  //В этом классе объявляется объект типа Building,
  class Program
  {
    static void Main(string[] args)
    {
      Building house = new Building(); // создать объект типа Building
      int areaPP; // площадь на одного человека
      // Присвоить значения полям в объекте house,
      house.Occupants = 4;
      house.Area = 250;
      house.Floors = 2;
      // Вычислить площадь на одного человека.
      areaPP = house.Area / house.Occupants;

      Console.WriteLine("Дом имеет:\n " +
      house.Floors + " этажа\n " +
      house.Occupants + " жильца\n " +
      house.Area +
      " кв. метров общей площади, из них\n " +
      areaPP + " приходится на одного человека");
      Console.ReadLine();
    }
  }
}
```

Эта программа состоит из двух классов: **Building** и **Program**. В классе **Program** сначала создается экземпляр **house** класса **Building** с помощью метода **Main ()**, а затем в коде метода **Main ()** осуществляется доступ к переменным экземпляра **house** для присваивания им значений и последующего использования этих значений. Следует особо подчеркнуть, что **Building** и **BuildingDemo** — это два совершенно отдельных класса. Единственная взаимосвязь между ними состоит в том, что в одном из них создается экземпляр другого. Но, несмотря на то, что это раздельные классы, у кода из класса **Program** имеется доступ к членам класса **Building**, поскольку они объявлены как открытые *(public)*. Если бы при их объявлении не был указан спецификатор доступа *public*, то доступ к ним ограничивался бы пределами **Building**, а следовательно, их нельзя было бы использовать в классе **Program**.

События — это члены класса, позволяющие объекту уведомлять вызывающий код о том, что случилось нечто достойное упоминания, например, изменение свой-

ства класса либо некоторое взаимодействие с пользователем. Клиент может иметь код, известный как обработчик событий, реагирующий на них.

Используя возвращаемое значение, можно усовершенствовать рассматривавшийся ранее пример с классом Building. Вместо того чтобы вычислять величину площади на одного человека в методе Main() , лучше возвратить ее из этого метода AreaPerPerson() класса Building. Среди прочих преимуществ такого подхода следует особо отметить возможность использовать возвращаемое значение для выполнения других вычислений. Приведенный ниже пример представляет собой улучшенный вариант рассматривавшейся ранее программы с методом AreaPerPerson (), возвращающим величину площади на одного человека.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace BuildingProbe
{
   class Building
   {
      public int Floors; // количество этажей
      public int Area; // общая площадь здания
      public int Occupants; // количество жильцов
      // Возвратить величину площади на одного человека,
      public int AreaPerPerson()
      {
         return Area / Occupants;
      }

   }

   //В этом классе объявляется объект типа Building,
   class Program
   {
      static void Main(string[] args)
      {
         Building house = new Building(); // создать объект типа Building
         int areaPP; // площадь на одного человека
         // Присвоить значения полям в объекте house,
         house.Occupants = 4;
         house.Area = 250;
         house.Floors = 2;
         // Вычислить площадь на одного человека.
         areaPP = house.AreaPerPerson();
```

```
            Console.WriteLine("Дом имеет:\n " +
            house.Floors + " этажа\n " +
            house.Occupants + " жильца\n " +
            house.Area +
            " кв. метров общей площади, из них\n " +
            areaPP + " приходится на одного человека");
            Console.ReadLine();

        }
    }
}
```

Во 2-ом примере, **MathTest**, демонстрируется определение и создание экземпляров классов, а также определение и вызов методов. Помимо класса, содержащего метод **Main ()**, в нем определен класс по имени **MathTest**, содержащий набор методов и полей.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace MathProbe
{
    class Program
    {
        static void Main(string[] args)
        {
         // Попытка вызова некоторых статических функций
         Console.WriteLine ("Pi равно " + MathTest.GetPi ());
         int x = MathTest.GetSquareOf(5);
         Console.WriteLine("5 в квадрате равно " + x) ;
         // Создание объекта MathTest
         MathTest math = new MathTest (); // это способ, которым в C# создаются
         // экземпляры ссылочного типа
         // Вызов нестатических методов
         math.value = 30;
         Console.WriteLine("Поле value переменной math содержит " + math.value);
         Console.WriteLine("30 в квадрате равно " + math.GetSquare());
         Console.ReadLine();
        }
    }
// Определение класса MathTest, методы которого мы вызываем
    class MathTest
    {
```

```
    public int value;
    public int GetSquare()
    {
      return value * value;
    }
    public static int GetSquareOf(int x)
    {
      return x * x;
    }
    public static double GetPi()
    {
      return 3.14159;
    }
  }
}
```

Запуск на выполнение примера **MathTest** даст следующий результат:

Pi равно 3.14159
5 в квадрате равно 25
Поле value переменной math содержит 30
30 в квадрате равно 900

Задание №1

Задачи на использование классов и объектов, в которых данные описаны в качестве **полей**. Реализовать класс заданной структуры. В нём предусмотреть конструктор для установки начальных значений полей. Создать объект на основе созданного класса. Осуществить использование объекта в программе.

- 1. Класс для решения линейного уравнения $y=kx+b$. Коэффициенты уравнения k, b реализовать с помощью полей вещественного типа. Для решения уравнения предусмотреть метод *Root*.
- 2. Элемент a_i геометрической прогрессии вычисляется по формуле: $a_i=a_0q^i$. Реализовать поля a_0 и q - вещественного типа. Определить метод Elementi() для вычисления заданного элемента прогрессии.
- 3. Угол задан с помощью целочисленных полей *gradus* - градусов, *min* - угловых минут, *sec* - угловых секунд. Реализовать класс, в котором предусмотреть метод*ToRadians* для перевода в радианы.
- 4. Дата задана с помощью целочисленных полей *day* , *month, year*. Предусмотреть метод *IsValid*, проверяющий возможна ли заданная дата.
- 5. В классе создать два целочисленных поля *a* и *b*. Реализовать метод *NOD* для нахождения наибольшего общего делителя для *a* и *b*.
- 6.Целочисленные поля a, b, c, являются сторонами некоторого треугольника. Реализовать метод, проверяющий истинность высказывания: «Треугольник со сторонами a, b, c является прямоугольным»

- 7. Целочисленные поля x и y представляют собой координаты клетки шахматной доски. Учитывая, что левое нижнее поле доски (1, 1) является черным, реализовать метод, проверяющий истинность высказывания: «Данное поле является белым».
- 8. Поле *left* - вещественное число, левая граница диапазона. Поле *right* - вещественное число, правая граница диапазона. Пара этих числе представляет полуоткрытый интервал [left,right). Реализовать класс, в котором предусмотреть метод *rangecheck()* - проверку заданного числа на принадлежность диапазону.
- 9. Комплексное число (*a+jb*) в алгебраической форме задано полями *a* и *b* с помощью метода *Polar* получить запись комплексного числа в показательной форме.
- 10. Время задано тремя целочисленными полями hour, min, sec. Реализовать метод увеличения времени на 1 секунду.

Задание №2

Задачи на использование классов и объектов, в которых данные описаны в качестве **свойств**. Реализовать класс заданной структуры. В нём предусмотреть конструктор для установки начальных значений полей. Создать объект на основе созданного класса. Осуществить использование объекта в программе.

- 1. Реализовать класс для нахождения площади треугольника. Вещественные свойства a,b,c - стороны треугольника. Метод Square находит площадь.
- 2. Реализовать класс для проверки исходных данных. Вещественные свойства a,b,c - стороны треугольника. Метод IsValid проверяет корректность введённых данных.
- 3. Круг на плоскости имеет координаты центра x_0,y_0 - вещественные свойства. Радиус круга r_0 - также задан вещественным свойством. Реализовать метод проверяющий принадлежность точки с координатами (x,y) данному кругу.
- 4. Вещественное свойство *sum* представляет собой сумму вложения под процент. Свойство *proc* - процентную ставку годовых. Реализовать метод, определяющий сумму через *n* лет.
- 5. Шахматный король находится на клетке с координатами (x,y) - целочисленные свойства, которые могут принимать значения от 0 до 8. Реализовать метод, возвращающий все возможные ходы.
- 6. Шахматный ферзь находится на клетке с координатами (x,y) - целочисленные свойства, которые могут принимать значения от 0 до 8. Реализовать метод, возвращающий число возможных ходов.
- 7. Реализовать класс для нахождения углов треугольника. Вещественные свойства a,b,c - стороны треугольника. Метод Angles находит углы.

Задание №3

Задачи на *наследование* классов, в которых данные описаны в качестве **свойств**. Реализовать базовый класс заданной структуры, на основе него создать наследующий класс. В нём предусмотреть конструктор для установки начальных значений полей. Создать объект на основе созданного класса. Осуществить использование объекта в программе.

- 1. Создать класс *Angle* для работы с углами на плоскости. Предусмотреть перевод из градусной меры в радианную, сложение и вычитание углов с учётом приведения к диапазону 0-360. На основе класса *Angle* создать класс *Triangle* для работы с прямоугольным треугольником. Предусмотреть нахождение его площади.
- 2. Создать класс *Money* представляющий количество банкнот достоинством 10, 50, 100, 500, 1000, 5000. Предусмотреть метод *summa*, для вычисления общей суммы. На основе класса *Money* создать класс *Bankomat* предусматривающий снятие любой возможной суммы, пополнение запаса.
- 3. Создать класс *Money* для работы с денежными суммами в котором для рублей и копеек предусмотрены независимые целочисленные данные. Реализовать метод вывода суммы на экран. На основе класса *Money* создать класс *Good* для работы с товаром. Предусмотреть метод, осуществляющий уменьшение цены на заданное число процентов.
- 4. Создать класс *Board* для описания шахматной доски. В нём предусмотреть массив 8x8 элементов и метод для перевода цифр 1-8 в буквы A-H и обратно. На основе класса *Board* создать класс *Composition* для составления шахматной композиции. В нём предусмотреть возможность добавления/удаления фигур на доску, распечатку композиции.
- 5. Создать класс *Points* для хранения координат четырёх точек A, B, C и D на плоскости. В классе предусмотреть возможность распечатки координат каждой точки по отдельности и всех разом. На основе класса *Points* создать класс *Quadrilateral* для работы с четырёхугольником. Предусмотреть методы для проверки существования четырёхугольника, нахождения площади и диагоналей.

Задание №4

Задачи на использование блочных лямбда-выражений. Описать делегат с требуемой сигнатурой. Используя блочное лямбда-выражение реализовать основной алгоритм задачи. Осуществить использование делегата в программе с применением введённых пользователем исходных данных.

- 1.Введены целые положительные числа A и B, такие, что A > B. На отрезке длины A размещено максимально возможное количество отрезков длины B (без наложений). Используя операцию деления нацело, найти количество отрезков B, размещенных на отрезке AB.
- 2.Введено трехзначное число. Найти сумму и произведение его цифр.
- 3.Введено трехзначное число. В нем зачеркнули первую слева цифру и приписали ее справа. Вывести полученное число.
- 4.Дни недели пронумерованы следующим образом: 1 — понедельник, 2 — вторник, …, 6 — суббота, 7 — воскресенье. Дано целое число K, лежащее в диапазоне 1–365, и целое число N, лежащее в диапазоне 1–7. Определить номер дня недели для K-го дня года, если известно, что в этом году 1 января было днем недели с номером N.
- 5.Введены целые положительные числа A, B, C. На прямоугольнике размера A * B размещено максимально возможное количество квадратов со стороной C (без

наложений). Найти количество квадратов, размещенных на прямоугольнике, а также площадь незанятой части прямоугольника.

- 6.Введен некоторый год (целое положительное число). Определить соответствующий ему номер столетия, учитывая, что, к примеру, началом 20 столетия был 1901 год.

Задание №5

Задачи на использование событий. Осуществить использование событий в программе с применением синтаксиса обработчика, рекомендованного для среды .NET Framework.

- 1. Распечатать номера всех шестизначных "счастливых" билетов. Осуществить перебор всех шестизначных чисел и при нахождении "счастливого" номера активировать событие OnTrivial, в обработчике которого вывести этот номер. После распечатки 10 "счастливых" номеров активировать событие OnFullScreen с ожиданием ввода пользователем выбора на продолжение работы программы с очисткой экрана или завершение работы программы. Номер считается "счастливым", если сумма первых трёх цифр равна сумме последних трёх.
- 2. Программа для поиска простых чисел в интервале 0..2млрд методом перебора с проверкой. При нахождении очередного простого числа активировать событие OnPrimeNumber, в обработчике которого вывести это число. После распечатки 10 простых чисел активировать событие OnFullScreen с ожиданием ввода пользователем выбора на продолжение работы программы с очисткой экрана или завершение работы программы.
- 3. Программа для поиска простых чисел среди чисел Фибоначчи. При нахождении очередного простого числа активировать событие OnPrimeNumber, в обработчике которого вывести это число, а также номер этого числа в последовательности Фибоначчи. После распечатки 10 простых чисел активировать событие OnFullScreen с ожиданием ввода пользователем выбора на продолжение работы программы с очисткой экрана или завершение работы программы.
- 4. Программа для поиска чисел Цукермана. При нахождении очередного числа Цукермана активировать событие OnZuckermanNumber, в обработчике которого вывести это число. После распечатки 10 чисел Цукермана активировать событие OnFullScreen с ожиданием ввода пользователем выбора на продолжение работы программы с очисткой экрана или завершение работы программы. См. **Числа Цукермана**
- 5. Программа для поиска натуральных чисел раскладываемых ровно на N простых множителей в интервале 0..2млрд методом перебора с проверкой. При нахождении очередного числа раскладываемого на N простых множителей, активировать событие OnNMultiplier, в обработчике которого вывести это число. После распечатки 10 чисел активировать событие OnFullScreen с ожиданием ввода пользователем выбора на продолжение работы программы с очисткой экрана или завершение работы программы.

Лабораторная работа №6

Задание на лабораторную: В среде Microsoft Visual C#, решить 5 задач по вариантам и оформить отчёт.

Дополнительную информацию см. в приложении.

Задание №1

Задачи на использование базовых компонентов Windows Forms. С помощью визуального конструктора создать обычную форму в которую включить необходимые элементы управления (*Label, TextBox, Button, CheckBox, RadioButton, ListBox, ComboBox*). Требуется предусмотреть обработку введённых данных с проверкой их корректности и выдачу результата или сообщения об ошибке.

- 1. Программа для перевода градусов температуры из шкалы Цельсия в шкалу Фаренгейта и наоборот.
- 2. Программа для расчёта ежемесячного платежа по кредиту, если вводится ставка % годовых, сумма кредита и срок кредита в месяцах.
- 3. Программа отображающая список простых чисел, поиск которых начинается от введённого пользователем числа. Количество чисел также задаётся пользователем.
- 4. Программа, раскладывающая введённое натуральное число на простые сомножители.
- 5. Программа для нахождения наибольшего общего делителя двух натуральных чисел.
- 6. Программа отображающая текстовое представление введённого числа.

Задание №2

Задачи на использование стандартных диалоговых окон Windows Forms. С помощью визуального конструктора создать обычную форму в которую включить необходимые элементы управления (*Label, TextBox, Button, CheckBox, RadioButton, ListBox, ComboBox*). Требуется предусмотреть действия с помощью стандартных диалоговых окон (*ColorDialog, FontDialog, FolderBrowserDialog, OpenFileDialog, SaveFileDialog*) и выдачу данных с учётом выбора.

- 1. Программа выводящая пример набранного текста с заданной стандартными диалогами гарнитурой шрифта, размером и цветом.
- 2. Программа, составляющая список файлов по заданному шаблону из указанной папки.
- 3. Программа составляющая градиентное изображение в графическом поле. Параметры цвета задаются стандартным диалоговым окном.
- 4. Программа для отображения графических файлов, выбираемых стандартным диалоговым окном.
- 5. Программа для добавления, редактирования, удаления текстового списка, хранящегося в ListBox с возможностью сохранения и загрузки текста в файле.

- 6. Программа для создания цветовой настройки приложения (цвет фона, текста, кнопок и т.д.) с возможностью сохранения в выбранном файле и загрузки из него.

Задание №3

Задачи на создание многооконных приложений Windows. С помощью визуального конструктора создать главную форму в которую включить главное меню. Требуется предусмотреть действия по созданию нового документа, загрузке сохранённого документа, сохранению документа с помощью стандартных диалоговых окон (*OpenFileDialog*, *SaveFileDialog*) обработку и отображение данных.

- 1. Программа для транспонирования матриц
- 2. Программа - многодокументный текстовый редактор на основе объекта *RichTextBox*.
- 3. Программа для просмотра, поворота и отражения растровых изображений в многодокументном интерфейсе.
- 4. Программа для выполнения графиков данных из строк исходного изображения.
- 5. Программа для просмотра и масштабирования растровых изображений.
- 6. Программа - многооконный web-браузер.
- 7. Справочник функции или команд, открывающий примеры в новом дочернем окне.

Задание №4

Задачи на использование буфера обмена и технологий перетаскивания данных в Windows. С помощью визуального конструктора создать форму в которую включить основной рабочий элемент и необходимые к нему элементы управления. Требуется предусмотреть действия по копированию данных в буфер обмена и вставке из него, перетаскивание данных с помощью Drag and Drop.

- 1. Программа для транспонирования матриц с возможностью копирования матрицы из одного окна в другое через буфер обмена
- 2. Текстовый редактор с возможностью открытия файлов перетаскиванием из Проводника.
- 4. Многодокументный текстовый редактор на основе объекта TextBox с возможностью копирования/вставки фрагмента текста через буфер обмена

Задание №5

Задачи на создание приложений WPF. С помощью визуального конструктора создать форму приложения WPF в которую включить необходимые элементы управления (*Label, Button, CheckBox, RadioButton, ListBox, ComboBox*). Дизайн внешнего вида формы должен использовать оформление, существенно отличающее его вид от приложений Windows Forms, рекомендовано применение графических и мультимедийных элементов для оформления формы. Требуется предусмотреть обработку введённых данных с проверкой их корректности и выдачу результата или сообщения об ошибке.

- 1. Программа для нахождения простых чисел в заданном пользователем диапазоне

- 2. Программа для деления двух чисел с заданной пользователем точностью, которая может существенно превышать возможности встроенных вещественных типов данных.
- 3. Программа генерирующая **последовательность Морса-Туэ**
- 4. Программа для получения последовательности **чисел трибоначчи**
- 5. Программа генерирующая изображение **кривой Леви**

Задание №6

Задачи на создание приложений XBAP - разновидности приложений WPF, ещё называемой браузерными приложениями XAML. С помощью визуального конструктора создать форму приложения XBAP в которую включить необходимые элементы управления (*Label, Button, CheckBox, RadioButton, ListBox, ComboBox*). Требуется предусмотреть обработку введённых данных с проверкой их корректности и выдачу результата или сообщения об ошибке.

- 1. Программа для получения **последовательности Падована**. Число элементов задаётся пользователем.
- 2. Программа для получения **числовой последовательности Каталана**. Число элементов задаётся пользователем.
- 3. Программа для получения последовательности Дийкстры
- 4. Программа для получения последовательности **чисел Шрёдера**

Задание №7

Задачи на создание многопоточных программ с интерфейсом Windows Forms. С помощью визуального конструктора создать обычную форму в которую включить необходимые элементы управления (*Label, TextBox, Button, CheckBox, RadioButton, ListBox, ComboBox, ProgressBar*). Основной поток программы осуществляет выполнение главной задачи в программе, фоновый поток должен обеспечивать выведение прогресса исполнения главной задачи с помощью ProgressBar. Указанный функционал реализуется с помощью пространства имён *System.Threading*. Требуется предусмотреть обработку введённых данных с проверкой их корректности и выдачу результата или сообщения об ошибке.

- 1. Программа для получения списка простых чисел в указанном пользователем интервале.
- 2. Программа, упорядочивающая содержимое файла по возрастанию.
- 4. Программа для получения разложения на простые сомножители чисел в указанном пользователем интервале.

ЛИТЕРАТУРА

1. Кауфман В.Ш. Языки программирования. Концепции и принципы [Электронный ресурс]/ Кауфман В.Ш.— Электрон. текстовые данные.— М.: ДМК Пресс, 2010.— 464 с.— Режим доступа: http://www.iprbookshop.ru/6932.— ЭБС «IPRbooks»
2. Нейл Дейл Программирование на C++ [Электронный ресурс]/ Нейл Дейл, Чип Уимз, Марк Хедингтон— Электрон. текстовые данные.— М.: ДМК Пресс, 2006.— 672 с.— Режим до-ступа: http://www.iprbookshop.ru/6904.— ЭБС «IPRbooks»
3. Биллиг В.А. Основы объектного программирования на C# (C# 3.0, Visual Studio 2008) [Электронный ресурс]/ Биллиг В.А.— Электрон. текстовые данные.— М.: БИНОМ. Лабо-ратория знаний, Интернет-Университет Информационных Технологий (ИНТУИТ), 2010.— 582 с.— Режим доступа: http://www.iprbookshop.ru/16092.— ЭБС «IPRbooks»
4. Кулямин В.В. Технологии программирования. Компонентный подход: учебное пособие. -М.: Интернет-Университет Информационных Технологий: БИНОМ. Лаборатория знаний, 2010. - 463 с.
5. Богачев К.Ю. Основы параллельного программирования. -М.: БИНОМ. Лаборатория знаний, 2010. - 342 с.
6. Кариев Ч.А. Разработка Windows-приложений на основе Visual C#: учебное пособие. -М.: БИНОМ. Лаборатория знаний: Интернет-Университет Информационных Техноло-гий, 2007. - 767 с.