Управляющие конструкции

Управляем ходом выполнения программы. Используем процедурный подход в написании программ. Познакомимся с рекурсией.

[Перечисления](#_yc08fgkxw0up)

[goto](#_c713xnuw3xem)

[Условия](#_rdo8o6k59wxq)

[Тернарная операция](#_kdae2flu2io3)

[Выражение1 ? Выражение2 : ВыражениеЗ;](#_q47ac8muwbd9)

[Примеры](#_88ygkqludpmx)

[Нахождение максимального из двух чисел:](#_8dkbjcocd6v2)

[Четность числа](#_uahxzmipyede)

[Оператор выбора](#_51ahcwjohi45)

[Пример использования switch. Месяц года.](#_43d219etsexg)

[Циклы](#_5azp89rm89m4)

[Цикл while](#_ozahbzcgsud9)

[Цикл do while](#_uyfmuoe4etgo)

[Оператор цикла for](#_260yf04bt2j4)

[Пример задачи:](#_fxqt0xwodq1v)

[Оператор цикла forech](#_3vz5h8a0umja)

[continue, break](#_22f29e4vr7b6)

[Вложенные циклы](#_uvhe1bcrurum)

[\*Рекурсия](#_zwim5z73qv0)

[Пример 1. Цикл с помощью рекурсии](#_v87eb7wyi1oz)

[Пример 2. Найти сумму цифр числа A.](#_ejw6m18ftd)

[Способ 1. Нерекурсивный](#_41vzgkgemm46)

[Способ 2. Рекурсивный](#_e09f2huj6dxk)

[Структуры для работы со временем](#_vmgom9716pap)

[Процедурное программирование](#_kq37q623g7m7)

[Практическая часть урока](#_r0go6aajvopk)

[Задача 1. Алгоритм нахождения НОД и организация метода](#_h5kkgbbqtffb)

[Задача 2. Сумма двух последних цифр](#_k5wf6ok3tcr4)

[Задача 3. Сложные условия](#_se6su3oebhhf)

[Задача 4. Задача ОГЭ (9 класс)](#_vd1i25vr37)

[Задача 5. Учимся подсчитывать эффективность программы](#_mw0wmylegs88)

[Задача 6. Дано натуральное число n. Вычислить n!](#_qiyu4h6laea)

[Задача 7. Последовательность Фибоначчи](#_nxk3j0mbkfex)

[Задача 8. “Ханойская башня”](#_90ugb1tpvol1)

[Домашнее задание](#_3tfrjxxltv85)

[Дополнительные материалы](#_q7971z70gyms)

[Используемая литература](#_vbx1glwg3yro)

# 

# Перечисления

Перечисления представляют собой множество именованных целочисленных констант. Перечисления существо облегчают труд программиста, позволяют не запоминать номера, например, цветов символов, а представлять их в виде символьных констант. Перечислимый тип данных объявляется с помощью ключевого слова enum.

|  |
| --- |
| enum Days {Sat, Sun, Mon, Tue, Wed, Thu, Fri}; |

В .NET Framework большое количество встроенных перечислений. Вот некоторые из них:

* System.ConsoleColor - цвета консоли;
* System.ConsoleKeys - коды клавиш;
* System.IO.FileMode - режимы работы с файлами.

# goto

goto - один из самых старых способов повлиять на ход выполнения программы.

|  |
| --- |
| начало:  …  goto начало; |

Хотя он прост в понимании, этот оператор не рекомендуется использовать при написании программ.

# Условия

В языке C# условие реализовано оператором if:

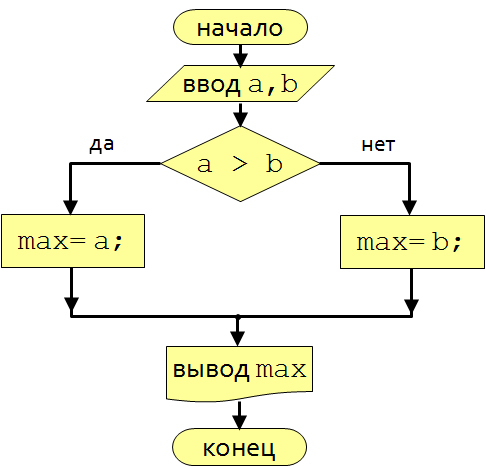
|  |
| --- |
| if (<условие>) {  // что делать, если условие верно  }  else {  // что делать, если условие неверно  }; |

Особенности:

* вторая часть (else …) может отсутствовать (неполная форма);
* если в блоке один оператор, то можно убрать скобки { и }.

Задача. Ввести два числа и вывести наибольшее из них.

Алгоритм решения:

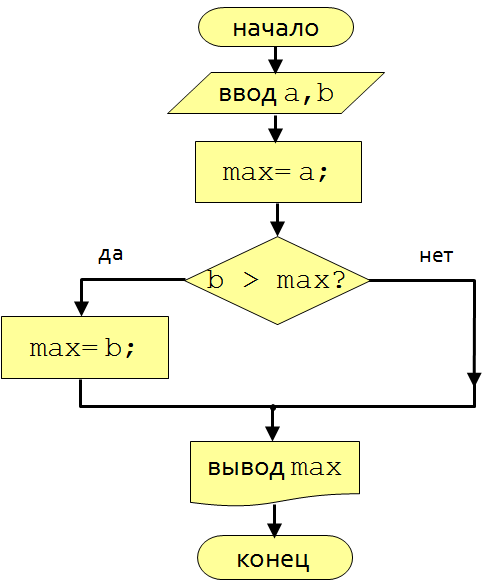


Пример программы, реализующей алгоритм:

|  |
| --- |
| using System;  namespace Lesson2\_002  {  class Program  {  static void Main()  {  int a, b, max;  Console.WriteLine("Введите два целых числа");  a = Convert.ToInt32(Console.ReadLine());  b = Convert.ToInt32(Console.ReadLine());  if (a > b)  {  max = a;  }  else  {  max = b;  };  Console.WriteLine("Наибольшее число {0}", max);  }  }  }  using System;  namespace Lesson2\_002  {  class Program  {  static void Main()  {  int a, b, max;  Console.WriteLine("Введите два целых числа.");  Console.Write("Первое число:");  a = Convert.ToInt32(Console.ReadLine());  Console.Write("Второе число:");  b = Convert.ToInt32(Console.ReadLine());  if (a > b)  {  max = a;  }  else  {  max = b;  };  Console.WriteLine("Наибольшее число {0}", max);  Console.WriteLine("Для выхода из приложение нажмите Enter");  Console.ReadLine();  }  }  } |



Неполная форму условного оператора:



|  |
| --- |
| using System;  namespace Lesson2\_002  {  class Program  {  static void Main()  {  int a, b, max;  Console.WriteLine("Введите два целых числа");  a = Convert.ToInt32(Console.ReadLine());  b = Convert.ToInt32(Console.ReadLine());  max = a;  if (b > max)  max = b;  Console.WriteLine("Наибольшее число {0}", max);  }  }  } |
| using System;  namespace Lesson2\_002  {  class Program  {  static void Main()  {  int a, b, max;  Console.WriteLine("Введите два целых числа.");  Console.Write("Первое число:");  a = Convert.ToInt32(Console.ReadLine());  Console.Write("Второе число:");  b = Convert.ToInt32(Console.ReadLine());  max = a;  if (b > max)  max = b;  Console.WriteLine("Наибольшее число {0}", max);  Console.WriteLine("Для выхода из приложение нажмите Enter");  Console.ReadLine();  }  }  } |

# \*Тернарная операция

Тернарный оператор (?) относится к числу самых примечательных в C#. Он представляет собой условный оператор и часто используется вместо определенных видов конструкций if-then-else. Ниже приведена общая форма этого оператора:

# Выражение1 ? Выражение2 : ВыражениеЗ;

Если Выражение1 - истинно, то берется Выражение2, иначе берется Выражение3.

Вывод на экран текста четное или нечетное число:

|  |
| --- |
| Console.WriteLine(x+" "+(x%2==0)?"Четное":"Нечетное");  Console.WriteLine(x + " - " + ((x % 2 == 0) ? "четное число" : "нечетное число")); |

# Примеры

### Нахождение максимального из двух чисел:

Вариант 1:

|  |
| --- |
| static void Main(string[] args)  {  int a = 100;  int b = 200;  int max=a>b? a:b;  } |

Вариант 2:

|  |
| --- |
| static void Main(string[] args)  {  int a = 100;  int b = 200;  int max;  if (a > b) max=a;else max=b;  } |

### Четность числа

Написать метод, проверяющий четность числа:

|  |
| --- |
| static bool Odd(int a)  {  return a%2==0;  }  static void Main(string[] args)  {  int x;  Console.WriteLine("Введите целое число.");  x = Convert.ToInt32(Console.ReadLine());  Console.WriteLine(x + " - " + ((Odd(x)) ? "четное число" : "нечетное число"));  Console.WriteLine("Для выхода из приложение нажмите Enter");  Console.ReadLine();  }  static bool Odd(int a)  {  return a % 2 == 0;  } |

# 

# 

# Оператор выбора

Оператор switch предназначен для разветвления процесса выполнения программы:

|  |
| --- |
| static void Main(string[] args)  {  int caseSwitch = 1; switch (caseSwitch) {  case 1:  Console.WriteLine("Case 1");  break;  case 2:  Console.WriteLine("Case 2");  break;  default:  Console.WriteLine("Default case");  break; }  } |

### Пример использования switch. Месяц года.

Вводится число, программа печатает, какой это месяц года:

|  |
| --- |
| static void Main()  {  int m = 6;  string s;  switch (m)  {  case 1:  case 2:  case 12: s="Зима";  break;  case 3:  case 4:  case 5: s="Весна";  break;  case 6:  case 7:  case 8: s = "Лето";  break;  case 9:  case 10:  case 11: s="Осень";  break;  default:  s="Ничего";  break;  }  Console.WriteLine(s);  } |

# Перечисления

Перечисления представляют собой множество именованных целочисленных констант. Перечисления существо облегчают труд программиста, позволяют не запоминать номера, например, цветов символов, а представлять их в виде символьных констант. Перечислимый тип данных объявляется с помощью ключевого слова enum.

|  |
| --- |
| enum Days {Sat, Sun, Mon, Tue, Wed, Thu, Fri}; |

В .NET Framework большое количество встроенных перечислений. Вот некоторые из них:

* System.ConsoleColor - цвета консоли;
* System.ConsoleKeys - коды клавиш;
* System.IO.FileMode - режимы работы с файлами.

using System;

namespace Lesson2\_002

{

class Program

{

enum Months { None, January, February, March, April, May, June, July, August, September, October, November, December };

enum Seasons { None, Winter, Spring, Summer, Autumn };

static void Main(string[] args)

{

Months month = Months.January;

Seasons season = Seasons.None;

switch (month)

{

case Months.January:

case Months.February:

case Months.December: season = Seasons.Winter;

break;

case Months.March:

case Months.April:

case Months.May: season = Seasons.Spring;

break;

case Months.June:

case Months.July:

case Months.August: season = Seasons.Summer;

break;

case Months.September:

case Months.October:

case Months.November: season = Seasons.Autumn;

break;

}

Console.WriteLine(season);

Console.WriteLine("Для выхода из приложение нажмите Enter");

Console.ReadLine();

}

}

}

# Циклы

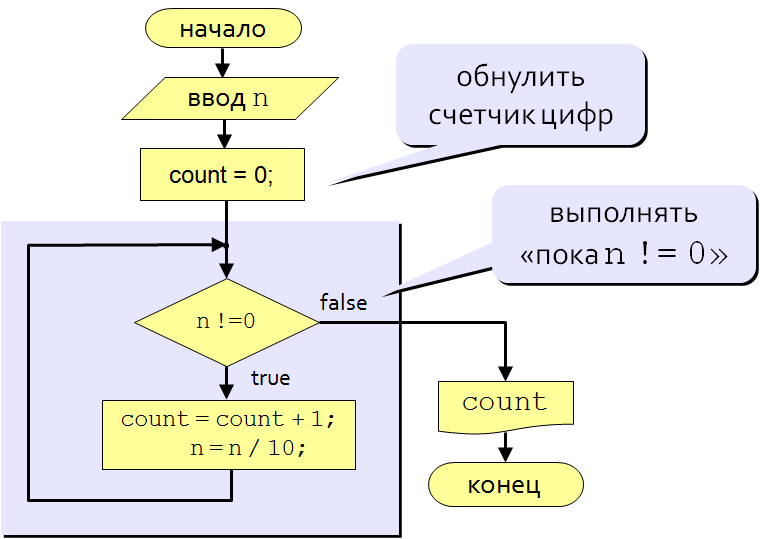
Цикл – это многократное выполнение одинаковой последовательности действий.

В C# доступны 4 разновидности цикла:

* цикл while;
* цикл do while;
* цикл for;
* цикл foreach.

## Цикл while

Задача: Ввести целое положительное число (<2000000000) и определить количество цифр в нем.



|  |
| --- |
| int n=int.Parse(Console.ReadLine());  int count=0;  while(n!=0)  {  count++;  n=n/10;// так как n-целое, деление целочисленное  }  Console.WriteLine(count); |

Формат оператора while:

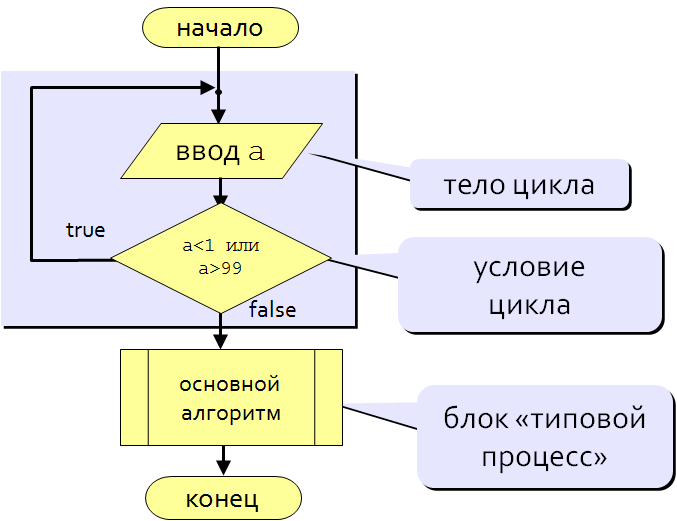
|  |
| --- |
| while (условие)  {  оператор; } |

|  |
| --- |
|  |

* можно использовать сложные условия;
* если в теле цикла только один оператор, скобки { и } можно не писать.

## Цикл do while

Задача. Организовать ввод данных, ограничив значения числами от 1 до 99.



|  |
| --- |
| int a,count=0;  do  {  Console.Write("Введите возраст:");  a=int.Parse(Console.ReadLine());  count++;  }  while(a<1 || a>99);// Повторять пока условие истинно(true)  Console.WriteLine("Вы сделали "+count+" попыток ввода"); |

|  |
| --- |
| do  {   операторы;  }  while (условие); |

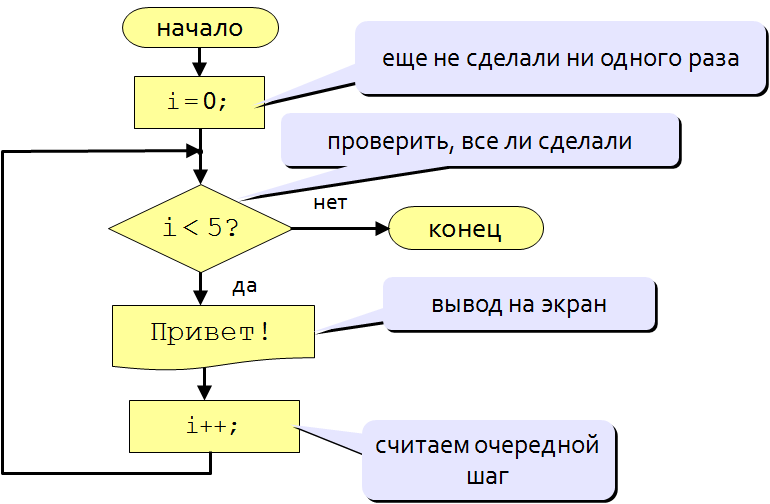
Тело цикла всегда выполняется хотя бы один раз.

## 

## 

## Оператор цикла for

Задача. Вывести на экран 5 раз слово «Привет».



|  |
| --- |
| for(int i=0;i<5;i++)  {  Console.Write(i+" ");  Console.WriteLine("Привет");  } |

|  |
| --- |
| for (инициализация; условие; итерация) {  оператор; } |

* Если тело цикла состоит из одного оператора, то операторные скобки { и } можно не писать:
* Условие каждый раз пересчитывается.

### 

### 

### Пример задачи:

Даны два целых числа A и B (A меньше B). Вывести в порядке убывания все целые числа от B до A , а также количество N и сумму S этих чисел.

|  |
| --- |
| static void Main()  {  int a = int.Parse(Console.ReadLine());  int b = int.Parse(Console.ReadLine());  int k = 0,s=0;  for (int i = b; i >= a; i--)  {  Console.Write(i + " ");  k++;  s=s+i;  }  Console.WriteLine("\nk={0} s={1}",k,s);  } |

## 

## Оператор цикла foreach

Оператор цикла foreach служит для циклического обращения к элементам коллекции, которая представляет собой группу объектов.

**foreach (тип имя\_переменной\_цикла in коллекция) оператор;**

|  |
| --- |
| using System;  namespace @foreach  {  class Program  {  static void Main(string[] args)  {  string s = "Hello, Foreach";  foreach (char c in s)  Console.Write("{0} ", c);  }  }  } |

Подробнее мы с ним познакомимся, когда будет изучать массивы.

## continue, break

* continue и break - операторы для управления циклами;
* continue - выполнение следующей итерации цикла;
* break - прерывание текущей итерации цикла.

|  |
| --- |
| using System;  namespace @foreach  {  class Program  {  static void Main(string[] args)  {  string s = "1. Привет, Foreach. \n2. А так же break и continue! А это не выведется";  foreach (char c in s) {  // Пропускаем цифры  if (c >= '0' && c<='9') continue;  // Если встречаем ! прерываем цикл  if (c == '!') break;  Console.Write("{0} ", c);  }  }  }  } |

# 

# 

# Вложенные циклы

Довольно часто один цикл приходится вкладывать в другой цикл. Давайте рассмотрим вложенные циклы на примере заполнения экрана звездочками.

|  |
| --- |
| using System;  namespace for\_for2  {  class Program  {  static void Main(string[] args)  {  //Внешний цикл  for (int i = 0; i < 80; i++)  //Внутренний цикл  for (int j = 0; j < 24; j++)  {  Console.SetCursorPosition(i, j); // устанавливаем позицию курсора  Console.Write('\*');  System.Threading.Thread.Sleep(20);// делаем паузу  Console.Title = "i=" + i + " j=" + j;  }  Console.ReadKey();  }  }  } |

Вложенные циклы будут продемонстрированы при работе с двумерными массивами.

# \*Рекурсия

Рекурсией называется механизм работы программы, в котором для решения задачи из подпрограммы вызывается та же самая подпрограмма. Этот способ является альтернативой циклам и в некоторых случаях позволяет написать весьма красивые алгоритмы решения задачи.  
Следует понимать, что любой рекурсивный метод можно преобразовать в обычный метод. И практически любой метод можно преобразовать в рекурсивный, если выявить рекуррентное соотношение между вычисляемыми в методе значениями.

### Пример 1. Цикл с помощью рекурсии

|  |
| --- |
| // Пример вывода чисел от a до b с использованием рекурсивного алгоритма  static void Loop(int a, int b)  {  Console.Write("{0,4} ",a);  if (a<b) Loop(a+1,b);  } |

## 

## Пример 2. Найти сумму цифр числа A.

Получить последнюю цифру можно, если найти остаток от деления числа на 10. В связи с этим для разложения числа на составляющие его цифры можно использовать следующий алгоритм:

1. Находим остаток при делении числа А на 10, т.е. получаем крайнюю правую цифру числа;
2. Находим целую часть числа при делении A на 10, т.е. отбрасываем от числа A крайнюю правую цифру;
3. Если преобразованное A > 0, то переходим на пункт 1. Иначе число равно нулю и отделять от него больше нечего.

### Способ 1. Нерекурсивный

|  |
| --- |
| static long Sum(long a )// нерекурсивный метод  {  long s=0;  while (a>0) // пока a больше нуля  {  s=s+a%10; // добавляем к сумме последнюю цифру числа а  a=a/10; // отбрасываем от числа а последнюю цифру  }  return s; // возвращаем в качестве результата сумму цифр числа a  } |

### Способ 2. Рекурсивный

|  |
| --- |
| static long RecursiveSum  (long a) // рекурсивный метод  {  if (a==0) // если a =0, то  return 0; // возвращаем 0  else return RecursiveSum (a/10)+ a%10; // иначе, вызываем рекурсивно сами себя  } |

# Структуры для работы со временем

В .NET Framework огромное количество уже готовых структур. Как создавать собственные структуры познакомимся на следующем уроке, а сейчас познакомимся с одной из них:

**DateTime** - хранит дату и время.

Особенно полезно знать, что при вычитании двух структур DateTime мы получаем промежуток времени. Это позволяет измерять время выполнения программы засекая время в начале и в конце программы.

|  |
| --- |
| DateTime start=DateTime.Now;  System.Threading.Thread.Sleep(20);// делаем паузу  DateTime finish=DateTime.Now; Console.WriteLine(finish-start); |

# Процедурное программирование

Процедурное программирование - это программирование, при котором последовательно выполняемые операторы собираются в подпрограммы, то есть в более крупные единицы кода. ООП это дальнейшее развитие идеи процедурного программирования, поэтому нужно понять, что такое процедурное программирование. Давайте рассмотрим, как применяется процедурное программирование на простом примере.

**Задача**. Вывести все числа от 1 до 100 сумма цифр которых чётна.

Здесь можно выделить следующие подпрограммы: подсчет суммы цифр числа (NumberSumm), определение четности числа (isOdd), организация цикла для перебора от 1 до 100 (Main).

|  |
| --- |
| static int NumberSumm(int n)  {  int s = 0;  while (n != 0)  {  s = s + n % 10;  n = n / 10;  }  return s;  }  static bool isOdd(int n)  {  return n % 2 == 0;  }  static void Main()  {  for (int i = 1; i <= 100; i++)  {  int sc = NumberSumm(i);  if (isOdd(sc)) Console.WriteLine("{0} {1}",i,sc);  }  } |

Умение представить программу в виде подпрограмм является очень важным умением программиста.

# 

# 

# Практическая часть урока

### Задача 1. Алгоритм нахождения НОД и организация метода

Реализовать метод нахождения NOD, используя алгоритм Евклида:

|  |
| --- |
| using System;  namespace Euclid  {  class Program  {  static int NOD(int a,int b)  {  while (a != b)  if (a > b) a = a - b; else b = b - a;  return a;  }  static void Main()  {  int a = 125;  int b = 200;  Console.WriteLine(NOD(a, b));  }  }  } |

### Задача 2. Сумма двух последних цифр

Вывести в диапазоне от 10 до 100 все числа, сумма двух последних цифр которых равна 10:

|  |
| --- |
| static bool Check(int a)  {  if ((a % 10 + a / 10 % 10 == 10)) return true; else return false;  }  static void Main(string[] args)  {  for (int i = 10; i <= 100; i++)  if (Check(i)) Console.WriteLine(i);  }  using System;  namespace Summ10  {  class Program  {  static bool Check(int a)  {  if ((a % 10 + a / 10 % 10 == 10)) return true; else return false;  }  static void Main(string[] args)  {  for (int i = 10; i <= 100; i++)  if (Check(i)) Console.WriteLine(i);  }  }  } |

### 

### 

### Задача 3. Сложные условия

С клавиатуры вводится возраст от 1 до 50. Требуется написать программу, которая правильно определит, какое слово нужно написать после возраста.

Вам 11 лет.  
 Вам 21 год.  
 Вам 33 года.

|  |
| --- |
| static void Main(string[] args)  {  int x;  Console.WriteLine("Введите возраст:");  x = int.Parse(Console.ReadLine());  string s = "Вам "+x;  // Год - когда заканчивается на один, кроме 11.  if (x % 10 == 1 && x!=11) s += " год";  // Года  if (x==11 || x >= 2 && x<=4 || (x >= 22 && x <= 24) || (x >= 32 && x <= 34) || (x > 41 && x < 45)) s += " года";  // Лет  if ((x >= 5 && x <= 20) || (x >= 25 && x <= 30) || (x >= 35 && x < 41) || (x > 44 && x < 51)) s += " лет";  Console.WriteLine(s);  }  using System;  namespace Age  {  class Program  {  static void Main(string[] args)  {  int x;  Console.WriteLine("Введите возраст, до 50 лет:");  x = int.Parse(Console.ReadLine());  string s = "Вам " + x;  // Год - когда заканчивается на один, кроме 11.  if (x % 10 == 1 && x != 11) s += " год";  else  // Года  if ( (x >= 2 && x <= 4) || (x >= 22 && x <= 24) || (x >= 32 && x <= 34) || (x > 41 && x < 45)) s += " года";  else  // Лет  if ((x == 11) || (x >= 5 && x <= 20) || (x >= 25 && x <= 30) || (x >= 35 && x < 41) || (x > 44 && x < 51)) s += " лет";  Console.WriteLine(s);  }  }  } |

Переделайте программу в метод. В качестве параметра методу передается возраст, а метод возвращает строку.

### Задача 4. Задача ОГЭ (9 класс)

Напишите программу, которая в последовательности целых чисел определяет среднее арифметическое положительных чисел, кратных 8. Программа получает на вход целые числа, среди них есть хотя бы одно положительное число, кратное 8, количество введённых чисел неизвестно, последовательность чисел заканчивается числом 0 (0 - признак окончания ввода, не входит в последовательность).

Количество чисел не превышает 1000. Введённые числа по модулю не превышают 30 000.

Программа должна вывести одно число: среднее арифметическое положительных чисел, кратных 8.

|  |
| --- |
| static void Main()  {  int k=0, s=0;  int a = int.Parse(Console.ReadLine());  while (a!=0)  {  if (a>0 && a%8==0) { k++;s = s + a; }  a = int.Parse(Console.ReadLine());  }  Console.WriteLine((double)s / k);  }  using System;  namespace EightMultiple  {  class Program  {  static void Main()  {  int k = 0, s = 0;  int a = int.Parse(Console.ReadLine());  while (a != 0)  {  if (a > 0 && a % 8 == 0) { k++; s = s + a; }  a = int.Parse(Console.ReadLine());  }  Console.WriteLine((double)s / k);  }  }  } |

### Задача 5. Учимся подсчитывать эффективность программы

Для этого давайте научимся подсчитывать время выполнения программы. Решим задачу нахождения простых чисел в диапазоне от 1 до 1000000. Напишем метод проверки, является ли число простым, и используем его для подсчета количества чисел. В начале цикла сохраним текущее время, по выходу из цикла вычтем текущее время из сохраненного и выведем результат на экран.

Текст программы:

|  |
| --- |
| using System;  namespace Prime  {  class Program  {  static bool IsSimple(int n)  {  for (int i = 2; i <= n / 2; i++)  if (n % i == 0) return false;  return true;  }  static void Main(string[] args)  {  DateTime start = DateTime.Now;  int k=0;  for (int i = 2; i < 1000000; i++)  if (IsSimple(i))  {  k++;  Console.WriteLine("{0} {1}",k,i);  }  Console.WriteLine(k);  Console.WriteLine(DateTime.Now - start);  }  }  } |

На самом деле в методе определения простоты числа можно заменить условие на i<=Math.Sqrt(n). Создайте новый метод с новым условием. Подсчитайте время выполнения программы с использованием двух различных методов.

### 

### 

### Задача 6. Дано натуральное число n. Вычислить n!

Без использования рекурсии:

|  |
| --- |
| static uint Factorial(uint n)  {  uint res = 1;  for (uint i = 0; i <= n; i++)  res \*= i;//res=res\*i  return res;  }  static uint Factorial(uint n)  {  uint res = 1;  for (uint i = 0; i <= n; i++)  if (i == 0)  res = 1;  else  res \*= i;  return res;  } |

С использованием рекурсии. (0!=1, n!=n\*(n-1)!):

|  |
| --- |
| static uint Factorial(uint n)  {  if (n == 0) return 1;  else return Factorial(n - 1) \* n;  } |

### Задача 7. Последовательность Фибоначчи

Последовательность Фибоначчи определяется так: a0=0, a1=1, ak=ak-1+ak-2 при k>=2. Дано n, вычислить an

Вариант 1. Использования цикла for:

|  |
| --- |
| static uint Fib(uint n)  {  uint a0 = 0;  uint a1 = 1;  uint a=a1;  for(int i=2;i<= n;i++)  {  a = a0 + a1;  a0 = a1;  a1 = a;  }  return a1;  } |

Вариант 2. Использование рекурсии:

|  |
| --- |
| static uint Fib(uint n)  {  if (n == 0) return 0;  if (n == 1) return 1;  return Fib(n - 1) + Fib(n - 2);  } |

### Задача 8. “Ханойская башня”

Реализовать на C# рекурсивный алгоритм игры «Ханойская башня»:

|  |
| --- |
| using System;  namespace Hanoi  {  class Program  {  static void Move(int n,int a,int b,int c)  {  if (n > 0)  {  Move(n - 1, a, c, b);  Console.WriteLine("{0} => {1}", a, b);  Move(n - 1, c, b, a);  }  }  static void Main(string[] args)  {  Move(4, 1, 2, 3);  }  }  }  using System;  namespace Hanoi  {  class Program  {  static void Move(int number, int from, int to, int free)  {  if (number > 0)  {  Move(number - 1, from, free, to);  Console.WriteLine("{0} => {1}", from, to);  Move(number - 1, free, to, from);  }  }  static void Main(string[] args)  {  Move(4, 1, 2, 3);  }  }  } |

Рекурсия способствует лучшему пониманию некоторых проблем, например, для головоломки «Ханойская башня» она дает простое и изящное решение. К сожалению, у рекурсии есть и свои недостатки: иногда ее использование очевидно, но неэффективно.

Так, рекурсивный алгоритм генерирования чисел Фибоначчи требует от программы многократного расчета одних и тех же величин. Это замедляет работу настолько, что расчет более 50 значений становится непрактичным. Таким образом, при разработке рекурсивного метода следует задуматься об его эффективности

# Домашнее задание

1. Написать метод возвращающий минимальное из трех чисел;
2. Написать метод подсчета количества цифр числа;
3. С клавиатуры вводятся числа, пока не будет введен 0. Подсчитать сумму всех нечетных положительных чисел;
4. Реализовать **метод** проверки логина и пароля. На вход подается логин и пароль. На выходе истина, если прошел авторизацию, и ложь, если не прошел(Логин:root, Password:GeekBrains). Используя метод проверки логина и пароля, написать программу: пользователь вводит логин и пароль, программа пропускает его дальше или не пропускает. С помощью цикла do while ограничить ввод пароля тремя попытками;
5. а) Написать программу, которая запрашивает массу и рост человека, вычисляет его индекс массы и сообщает, нужно ли человеку похудеть, набрать вес или все в норме;  
   б) \*Рассчитать, на сколько кг похудеть или сколько кг набрать для нормализации веса.
6. \*Написать программу подсчета количества “Хороших” чисел в диапазоне от 1 до 1 000 000 000. Хорошим называется число, которое делится на сумму своих цифр. Реализовать подсчет времени выполнения программы, используя структуру DateTime.
7. a) Разработать рекурсивный метод, который выводит на экран числа от a до b(a<b);  
   б) \*Разработать рекурсивный метод, который считает сумму чисел от a до b.

Достаточно решить 4 задачи. Разбивайте программы на подпрограммы. Переписывайте в начало программы условие и свою фамилию. Все программы делайте в одном решении.

# Дополнительные материалы

1. [Демоверсии ГИА по информатике](http://gia.edu.ru/ru/graduates_classes/demonstration/);
2. [Индекс массы тела на Википедии](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B4%D0%B5%D0%BA%D1%81_%D0%BC%D0%B0%D1%81%D1%81%D1%8B_%D1%82%D0%B5%D0%BB%D0%B0).
3. Алгоритмы. Теория и практическое применение. Род Стивенс. Глава “Рекурсии”.
4. “[Ханойская башня” на Википедия](https://ru.wikipedia.org/wiki/%D0%A5%D0%B0%D0%BD%D0%BE%D0%B9%D1%81%D0%BA%D0%B0%D1%8F_%D0%B1%D0%B0%D1%88%D0%BD%D1%8F#.D0.9B.D0.B5.D0.B3.D0.B5.D0.BD.D0.B4.D1.8B).

# Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. Т.А. Павловская. “Программирование на языке высокого уровня”, 2009 г. ;
2. Г. Шилдт. “C# 4.0. Полное руководство”;
3. Алгоритмы. Теория и практическое применение. Род Стивенс, Москва 2016 г.
4. [MSDN](https://msdn.microsoft.com/ru-ru/default.aspx).