# Алгоритмы

# Литература


1. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ. — М.: МЦНМО. — 960 с.
2. Вирт Н. Алгоритмы + структуры данных = программы. - М.: Мир, 2003.
3. Кнут Д. Э. - Искусство программирования. Том 1. Основные Алгоритмы.
4. Скиена С. Алгоритмы. Руководство по разработке. – 2-е изд.:Пер. с англ. – СПб.: БХВ-Петербург, 2011. – 720 с.

# Задачи алгоритмизации

1. Построение нового или модификация некоторого ранее разработанного или определенного алгоритма.
2. Доказательство правильности алгоритма (верификация, тестирование).
3. Реализация применения разработанного или модифицированного алгоритма.
4. Анализ, оценка алгоритма по некоторым критериям его эффективности.

# Алгоритм

**Алгоритм** – точное предписание, которое определяет процесс, ведущий от исходных данных к требуемому результату и достаточно определенное для того, чтобы ее можно было выполнить при помощи некоторого автоматического устройства.

**Алгоритм** – это формально описанная вычислительная процедура, получающая исходные данные, называемые также входом алгоритма или его аргументом, и выдающая результат вычислений на выход.

- ученый средневекового Востока - Муххамад ибн Муса ал-Хорезми (Магомет, сын Моисея, из Хорезма), в латинских переводах с арабского ал-Хорезми его имя определялось как algorismi.

# Алгоритм


Существует некоторое абстрактное устройство, способное распознать инструкции и выполнить предписываемые ими действия

# Виды алгоритмов

- **Детерминированный** алгоритм задает определенные действия, обозначая их в единственной и достоверной последовательности
- **Стохастический** (вероятностный) алгоритм дает программу решения задачи несколькими путями, приводящими к вероятностному достижению результата
- **Эвристический** алгоритм это такой алгоритм, в котором достижение конечного результата однозначно не определено, так же как не обозначена вся последовательность действий.

# Основные требования предъявляемые к алгоритмам

- Алгоритм должен иметь одну или несколько выходных величин.
- Конечность (или сходимость алгоритма)
- Результативность
- Последовательность шагов алгоритма детерминированна
- Алгоритм предполагает наличие механизма реализации
- Алгоритм должен обладать определенностью
- Массовость алгоритма

# Алгоритм поиска НОД двух целых чисел

*Вычисление НОД чисел m и n при помощи алгоритма Евклида*

- Шаг 1. Если $n = 0$, вернуть $m$ в качестве ответа и закончить работу; иначе перейти к шагу 2.
- Шаг 2. Поделить нацело $m$ на $n$ и присвоить значение остатка переменной $r$.
- Шаг 3. Присвоить значение $n$ переменной $m$, а значение $r$ — переменной $n$. Перейти к шагу 1.

# Алгоритм поиска НОД двух целых чисел

*Вычисление НОД чисел методом последовательного перебора*

- Шаг 1. Присвоить значение функции min {*m, n*} переменной *t*.
- Шаг 2. Разделить *m* на *t*. Если остаток равен нулю, перейти к шагу 3; иначе перейти к шагу 4.
- Шаг 3. Разделить *n* на *t*. Если остаток равен нулю, вернуть *t* в качестве ответа и закончить работу; иначе перейти к шагу 4.
- Шаг 4. Вычесть 1 из *t*. Перейти к шагу 2.

# Алгоритм поиска НОД двух целых чисел

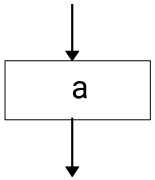
- *Вычисление НОД чисел m и n школьным методом*
- Шаг 1. Разложить на простые множители число *m*.
- Шаг 2. Разложить на простые множители число *n*.
- Шаг 3. Для простых множителей чисел *m и n*, найденных на шаге 1 и 2, выделить их общие делители. Если p является общим делителем чисел тип и встречается в их разложении на простые множители, соответственно, $p_m$ и $p_n$ раз, то при выделении нужно повторить это min $\{p_m,p_n\}$ раз.
- Шаг 4. Вычислить произведение всех выделенных общих делителей и вернуть его в качестве результата поиска НОД двух указанных чисел.

# Основные способы описания алгоритмов

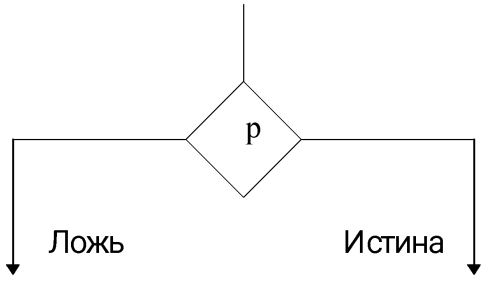
- словесно-формульный
- структурный или блок-схемный
- табличный
- с помощью граф-схем

# Представление алгоритмов в виде блок-схем

***Блок - схема*** представляет собой двухмерный рисунок, построенный из управляющих структур. При рисовании этих структур используются специальные обозначения.



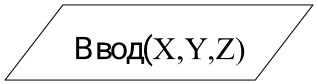

**Обозначение обработки.** Действие, которое необходимо выполнить, обозначается прямоугольником, в который входит и из которого выходит ровно одна линия управления.
Прямоугольник называется **узлом обработки**, или **функциональным узлом**.



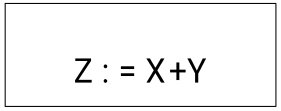

**Обозначение проверки.** Операция проверки обозначается символом, который называется **условным (предикатным) узлом**. Он представляет собой ромб, в который входит одна линия управления, а выходят две. В результате проверки помещенного внутри ромба условия (предиката) **p** выбирается один из выходов (но не оба сразу).

# Управляющие структуры

Безальтернативные вычисления (управляющая структура "следование")




**Ввод данных.** Предписание на ввод данных содержит указание устройства ввода и имя переменной, значение которой надо ввести.




**Изменение данных**. Чаще всего предписание на изменение данных представляется в виде оператора присваивания (последовательности операторов присваивания).

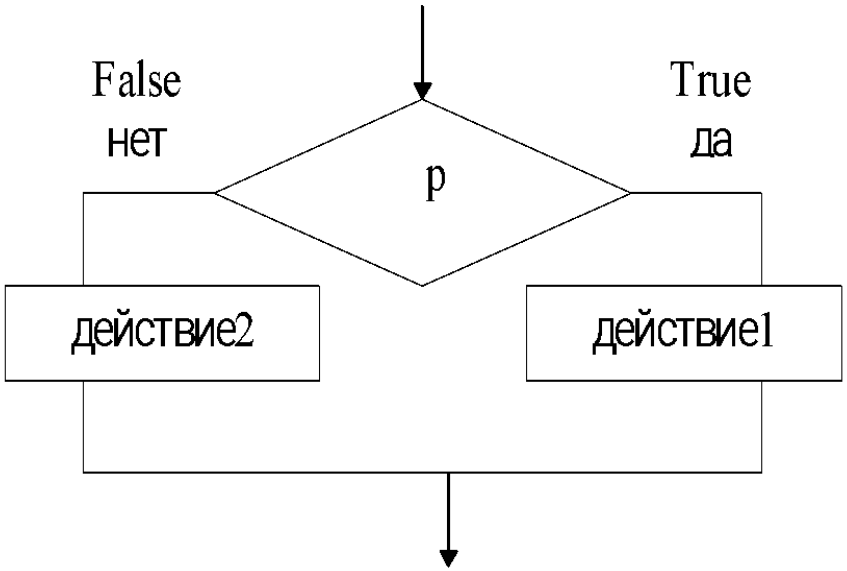

**Вывод данных.** Предписание на вывод данных содержит указание устройства вывода и имя переменной, значение которой следует вывести.

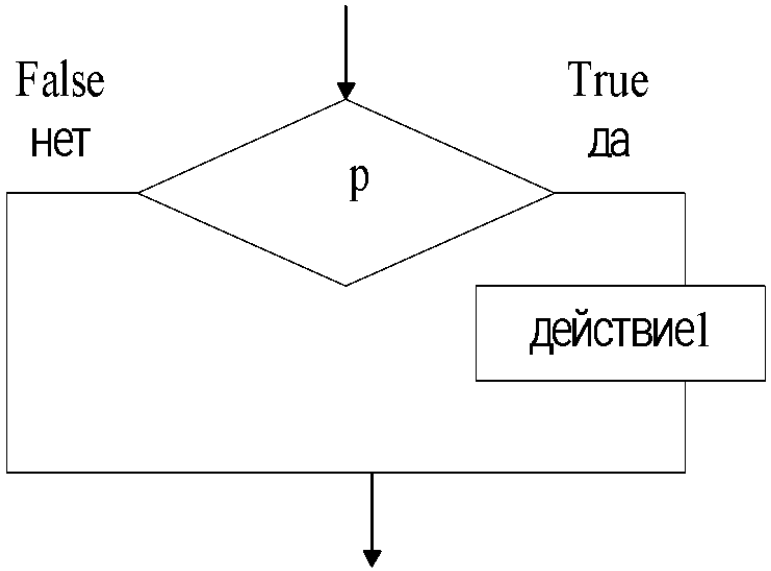
Управляющая структура "следование" используется в случае, когда алгоритм представляется как последовательность, элементами которой служат только действия по преобразованию данных. В этом случае алгоритм называют **линейным алгоритмом**.

# Управляющие структуры

## Альтернативные вычисления (управляющая структура «выбор»)

Для реализации альтернативного двоичного выбора используется управляющая структура **выбор** в двух ее модификациях:



# Пример.

- Дана точка в декартовой системе координат P(x,y).
- Требуется составить условие определения в какой четверти находится данная точка.

# Пример.

Алгоритмическое решение.

- Вариант 1. Последовательный перебор. Последовательно составляются условия на принадлежность к каждой четверти.
- Вариант 2. Метод деления пополам. Сначала сравнивается первая координата на условие принадлежности точки к половине системы координат. Затем уточняется к какой из ее двух частей принадлежит точка

Для решения задач выбора применяются:

- Операции сравнения, которые возвращают True (истина) или False (ложь):
  - меньше, больше, равно, не равно, больше или равно, меньше или равно
- Логические операторы, применяемые для проверки одновременно несколько условий:
  - X and Y (Истина, если оба значения X и Y истинны)
  - X or Y (Истина, если хотя бы одно из значений X или Y истинно)
  - not X (Истина, если X ложно)

# Пример.

Программное решение.

- Вариант 1. Последовательный перебор. Последовательно составляются условия на принадлежность к каждой четверти.

```
int x = Convert.ToInt32(Console.ReadLine());
int y = Convert.ToInt32(Console.ReadLine());
if (x > 0 && y > 0)
    Console.WriteLine("Первая четверть");
else if (x > 0 && y < 0)
Console.WriteLine("Четвертая четверть");
else if (y > 0)
Console.WriteLine("Вторая четверть");
else
    Console.WriteLine("Третья четверть");
```

# Пример.

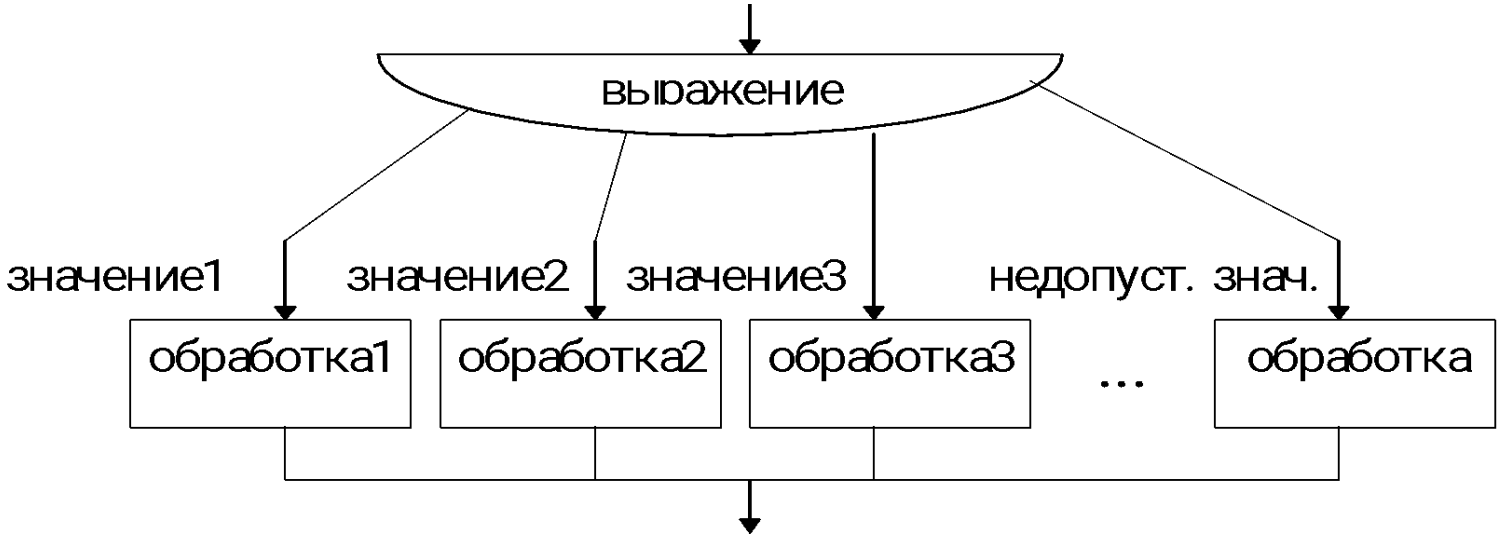
Программное решение.

- Вариант 2. Метод деления пополам. Сначала сравнивается первая координата на условие принадлежности точки к половине системы координат. Затем уточняется к какой из ее двух частей принадлежит точка

```
if (x > 0)
      if (y > 0)
          Console.WriteLine("Первая четверть");
      else
          Console.WriteLine("Четвертая четверть");
else
      if (y > 0)
          Console.WriteLine("Вторая четверть");
      else
          Console.WriteLine("Третья четверть");
```

# Управляющие структуры

Альтернативные вычисления
(управляющая структура «множественный выбор»)

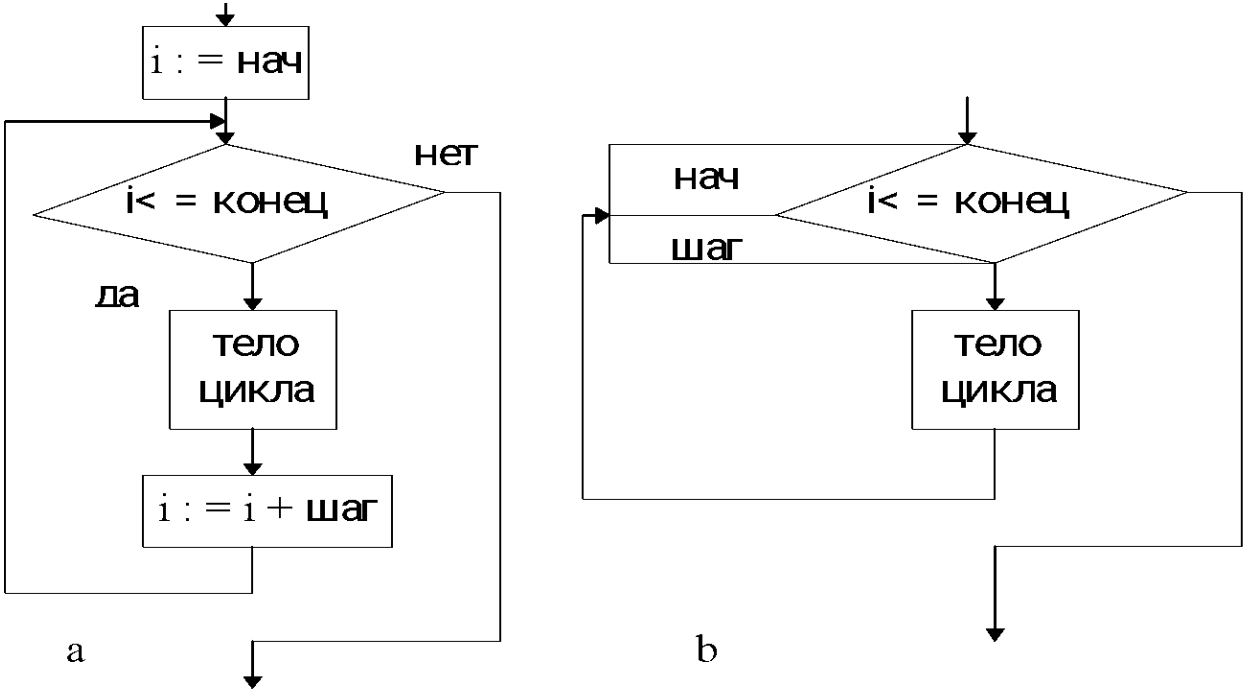


# Множественный выбор. Пример

```
bool ok = true;
Console.Write("A= ");
int a = int.Parse(Console.ReadLine());
Console.Write("OP= ");
char op = char.Parse(Console.ReadLine());
Console.Write("B= ");
int b = int.Parse(Console.ReadLine());
float res = 0;
switch (op)
{
    case '+': res = a + b; break;
    case '-': res = a - b; break;
    case '*': res = a * b; break;
    case '/':
    case ':':
        if (b != 0)
        {
            res = (float)a / b; break;
        }
        else
     ok = false; break;
}
if (ok) Console.WriteLine("{0} {1} {2} = {3}", a, op, b, res);
    else Console.WriteLine("error");
```

# Управляющие структуры

Повторяющиеся вычисления (управляющая структура «цикл для»)



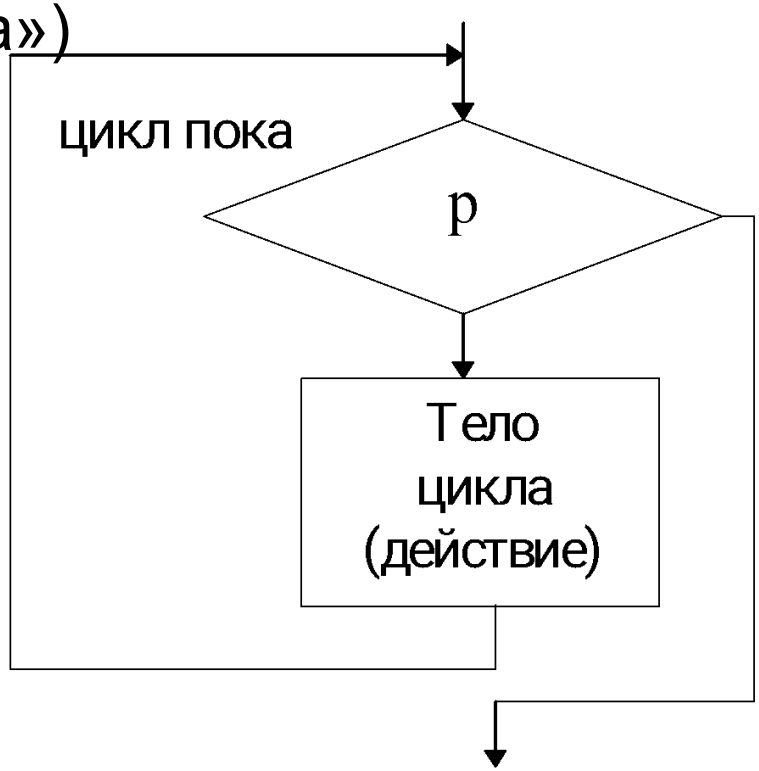

Приращение параметра цикла называется "шаг цикла".
Имеется редко используемое, но удобное обозначение "цикла для" на языке блок - схем (Рис. b).

# Повторяющиеся вычисления (управляющая структура «цикл для»). Пример.

```
Console.WriteLine("n=");
int n = int.Parse(Console.ReadLine());
for (int i = 1; i <= n; i++)
{
   Console.WriteLine(i);
}
```

# Управляющие структуры

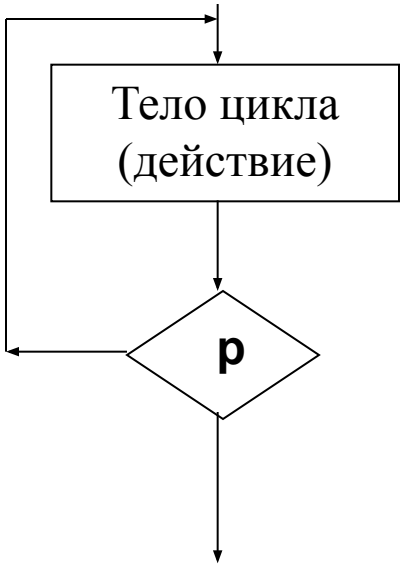
Повторяющиеся вычисления (управляющая структура «цикл пока»)




Предписывает выполнять тело цикла до тех пор **ПОКА** истинно условие **p.**
Тело "цикла пока" может не выполняться ни одного раза

# Управляющие структуры

Повторяющиеся вычисления (управляющая структура «цикл до»)




Предписывает выполнять тело цикла до выполнения условия **p**.
Тело "цикла до" выполняется хотя бы один раз

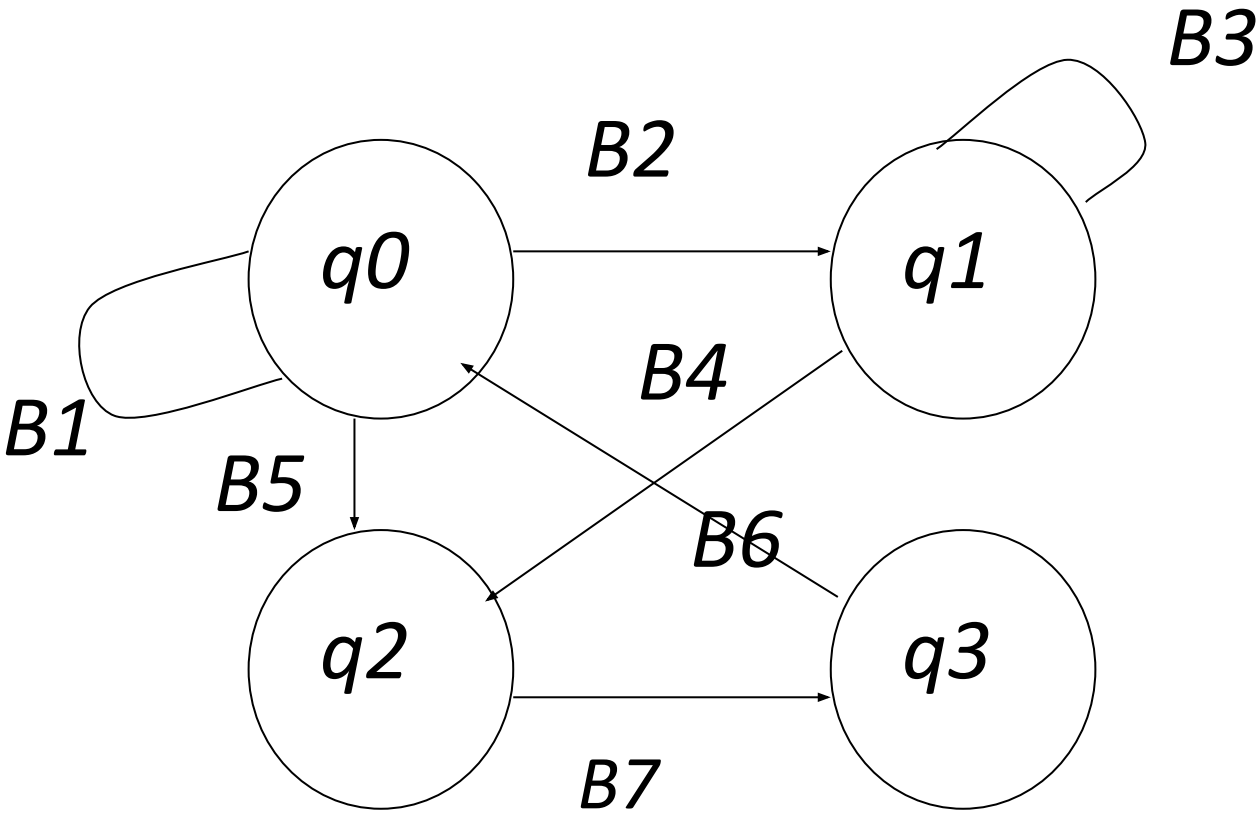
# Пример. Циклы с условием «до» и «после»

```
string answer, text;
  do
  {
     Console.WriteLine("Введите слово");
     text = Console.ReadLine();
     int i = 0, j = text.Length-1;

     while ((i<j) && (text[i] == text[j]))
        {i++; j--;}

     if (text[i] == text[j])
        Console.WriteLine(text +" - это палиндром!");
     else
        Console.WriteLine(text +" - это не палиндром!");
     Console.WriteLine("Продолжим? (y/n)");
     answer = Console.ReadLine();
  }
  while(answer =="y");
```

# Управление процессом цикла

```
Console.WriteLine("n=");
int n = int.Parse(Console.ReadLine());
for (int i = 1; i <= n; i++)
{
    if (i % 2 == 0) continue;
    Console.WriteLine(i);
}
```

# Автоматные графы



# Рекурсия

- ***Рекурсия*** — это такой способ организации обработки данных, при котором программа вызывает сама себя непосредственно, либо с помощью других программ.
- ***рекурсивная функция*** это функция, для которой существует алгоритм вычисления ее значений по произвольному значению аргумента
- **Пример**.

$$n! = \begin{cases} 1, & n = 0 \\ n(n-1)!, & n > 0 \end{cases}$$

```
static long F(int n)  //рекурсивный метод
{
   if (n==0 || n==1)
    return 1;              //не рекурсивная ветвь
   else return n*F(n-1);  //рекурсия - повторный вызов
                          // метода с другим параметром
}
```

# Применение рекурсии

- Алгоритмы, определяющие решение методом проб и ошибок.
  - Процесс проб и ошибок разделяется на отдельные подзадачи, часто эти подзадачи естественно описываются с помощью рекурсии.
- Задача оптимального выбора.
  - Например, найти оптимальную выборку из заданного множества объектов, подчиненную некоторым ограничениям. Процедура, описывающая процесс исследования пригодности объекта вызывается рекурсивно при переходе к следующему объекту, пока все объекты не будут рассмотрены.

# Итерация. Вычисление факториала

- ***Итерация*** — способ организации обработки данных, при котором определенные действия повторяются многократно, не приводя при этом к рекурсивным вызовам программ.