Технология программирования

Функции

Структуры

Функции

Функции – относительно самостоятельные фрагменты программы, спроектированные для решения конкретных задач и снабженные именем.

В объектно-ориентированных языках функции размещаются внутри классов и называются **методами**.

Статические методы полностью аналогичны функциям из процедурных языков.

Функции

Функции аналогичны программам в миниатюре и имеют общее название **подпрограммы**.

Появление функций было значительным шагом в развитии программирования.

Функции

Использование функций дает следующие преимущества:

- Экономия памяти кода за счет размещения многократно повторяющихся частей программ в функции.
- Позволяет работать группе программистов над одной сложной задачей
- Функции облегчают чтение, внесение изменений и коррекцию ошибок в программе.

Функции

- В функциях могут быть описаны собственные константы, типы, переменные. В этом случае они называются **локальными**. Их **область действия** распространяется только на те функции, в которых они описаны.
- Имена локализованных переменных **могут совпадать** с ранее объявленными внешними именами. В этом случае считается, что локальное имя "закрывает" внешнее и делает его недоступным

Функции

- Память под локальные переменные выделяется в момент вызова подпрограммы и освобождается после завершения её выполнения.
- Доступ к ним возможен только из той подпрограммы, в которой они описаны.

Функции

Общая форма записи функции:

<тип> <имя> (<формальные параметры>) { <тело функции> }

Вызов функции производится при помощи оператора вызова функции:

<имя>(<фактические параметры>)

Функции

Функции могут возвращать результат выполнения.

- Если функция не возвращает результат, то в качестве ее типа указывается ключевое слово **void**.
- В теле функций, возвращающих значение, обязательно должен присутствовать оператор возврата **return**. После ключевого слова **return** записывается выражение, значение которого вставится вместо имени функции в точке вызова.

Пример

Найти корни уравнения

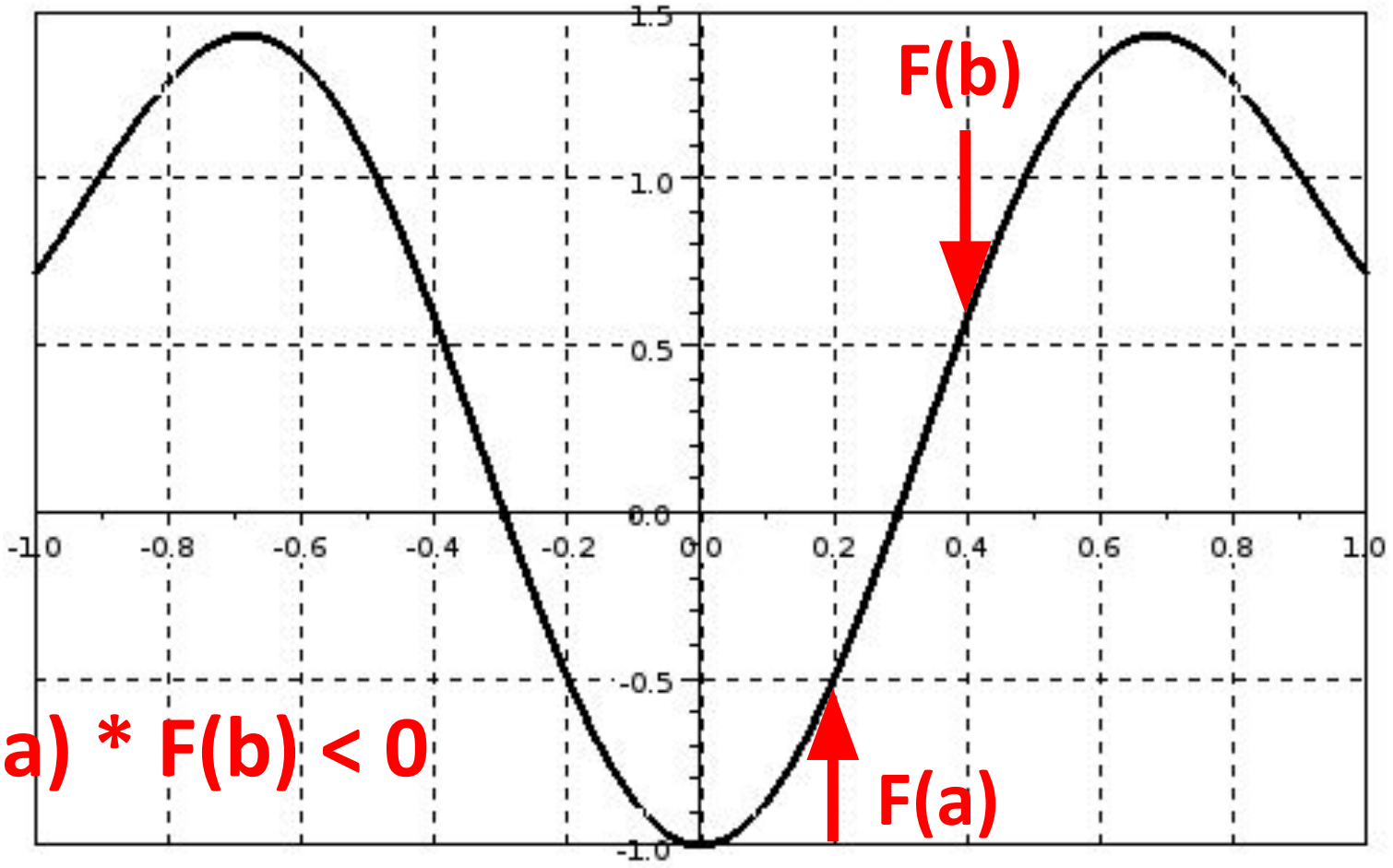
$$x^2 - \cos(5x) = 0$$

при помощи методов:

- половинного деления (дихотомии)
- хорд
- касательных (Ньютона)
- простых итераций

Пример

$$x^2 - \cos(5x) = 0$$

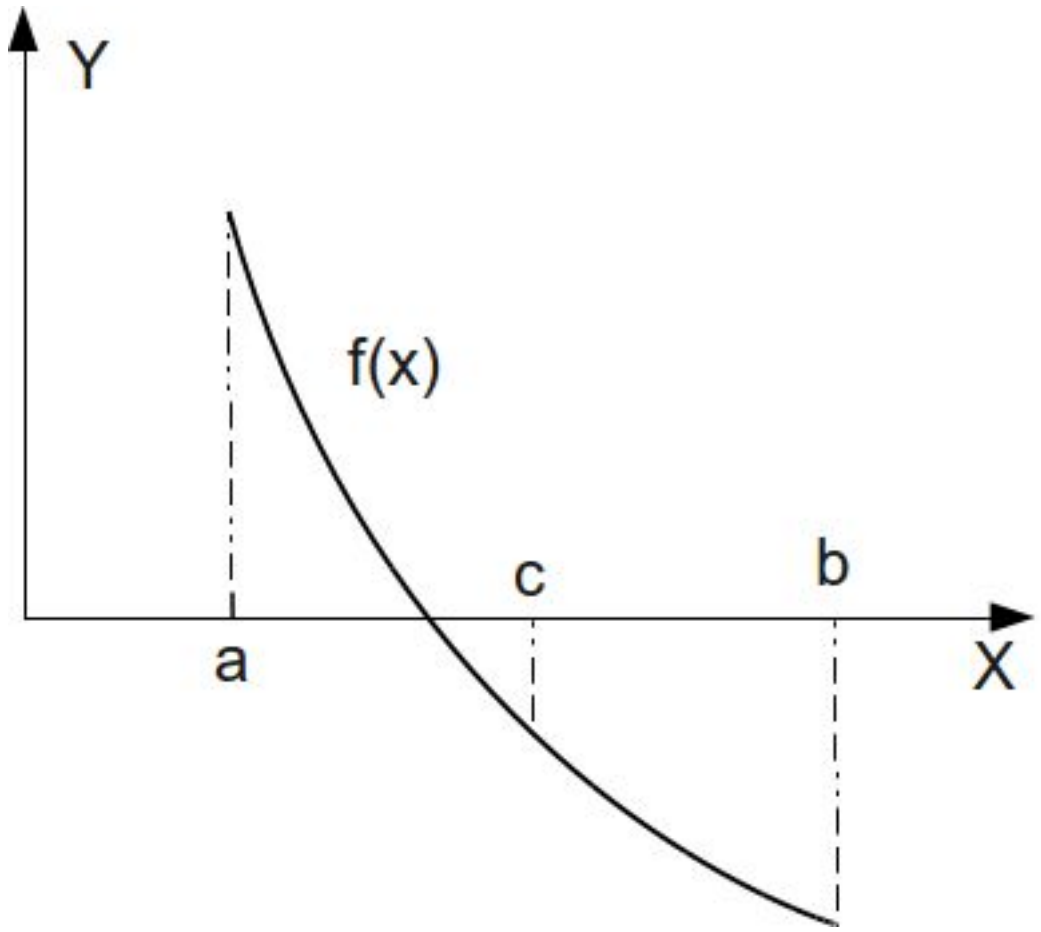


Пример

$$x^2 - \cos(5x) = 0$$

```
static double F(double x) {
    return x * x - Cos(5 * x);
}
```

Метод половинного деления



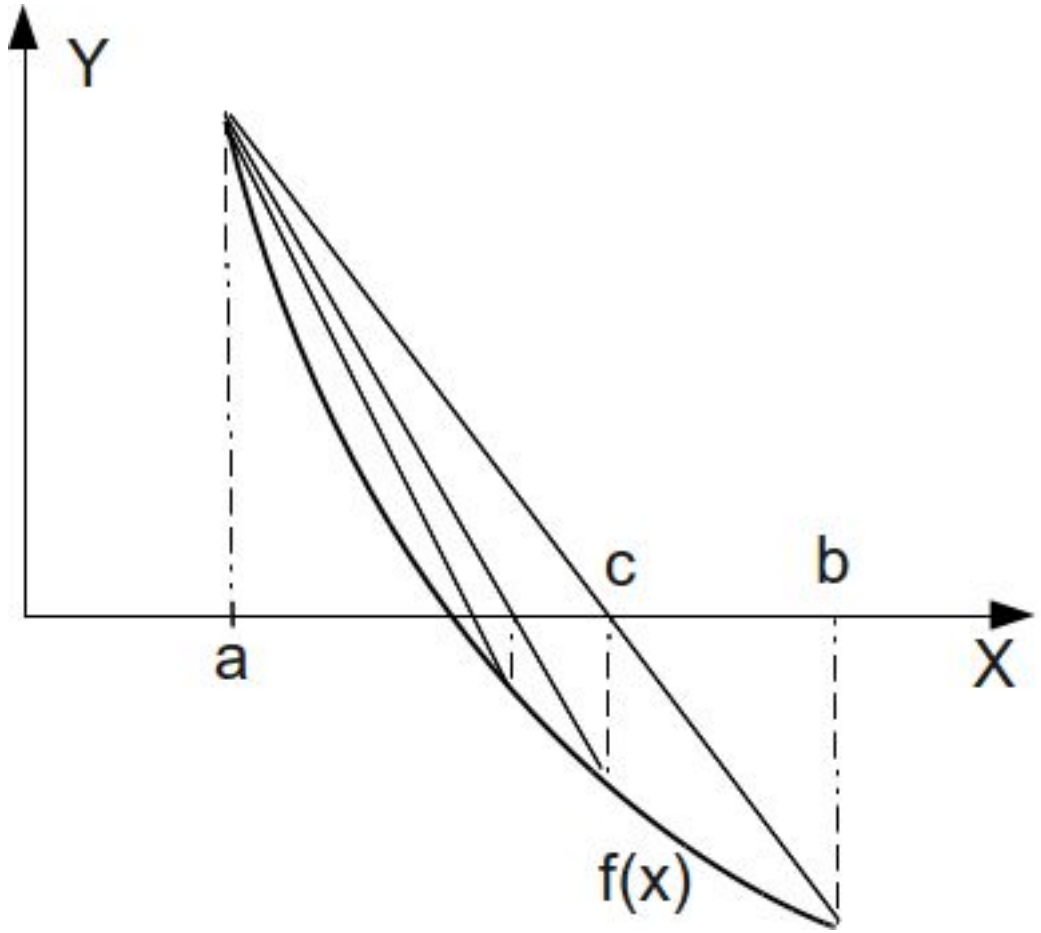
Метод половинного деления

1. Находим точку **c**
2. Находим значение **F(c)**
3. Если **F(a)*F(c)<0**, то корень лежит на интервале **[a, c]**, иначе корень лежит на интервале **[a, b]**
4. Если величина интервала меньше либо равна **e**, то корень найден, иначе возвращаемся к 1.

Метод половинного деления

```
static double Dichotomy(double a, double b,
            double e) {
    double c;
    do {
        c = (a + b) / 2;
        if (F(c) * F(a) < 0) {
            b = c;
        } else {
            a = c;
        }
    } while (Abs(a - b) >= e);
    return c;
}
```

Метод хорд



Метод хорд

Отличается от метода дихотомии тем, что очередное приближение берём не в середине отрезка, а в точке пересечения с осью **X** прямой, соединяющей точки **(a, F(a))** и **(b, F(b))**.

Метод хорд

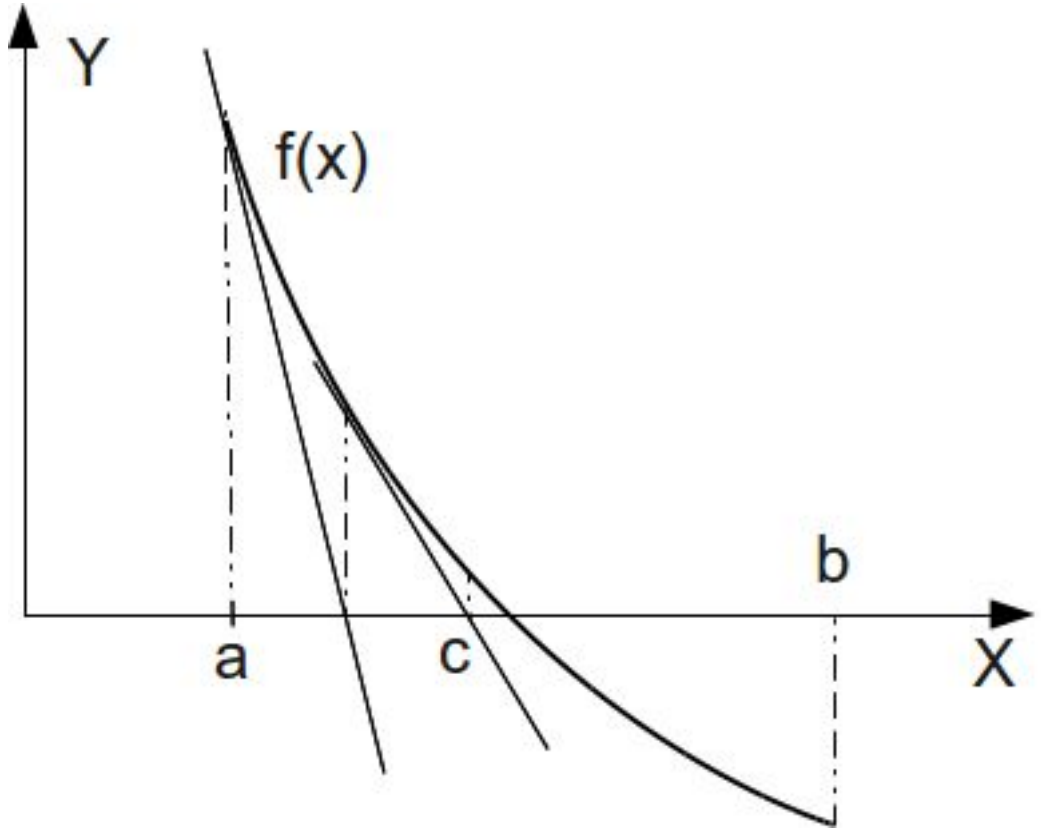
$$\frac{y - F(a)}{F(b) - F(a)} = \frac{x - a}{b - a} \equiv y = \frac{F(b) - F(a)}{b - a} \cdot (x - a) + F(a)$$

$$y = 0: \qquad x = a - \frac{F(a) \cdot (b - a)}{F(b) - F(a)}$$

Метод хорд

```
static double Chords(double a, double b,
          double e) {
    double c;
    do {
        c = a - F(a) / (F(b)-F(a)) * (b - a);
        if (F(c) * F(a) > 0) {
            a = c;
        } else {
            b = c;
        }
    } while (Abs(F(c)) >= e);
    return c;
}
```

Метод касательных



Метод касательных

$$y = k \cdot x + m$$
$$k = F''(c)$$
$$y = F''(x) \cdot x + m$$
$$F(c) = F''(c) \cdot c + m \;\rightarrow\; m = F(c) - c \cdot F''(c)$$
$$y = F''(c) \cdot x + F(c) - c \cdot F''(c)$$
$$y = F''(c) \cdot (x - c) + F(c)$$
$$F''(c) \cdot (x - c) + F(c) = 0$$

$$x = c - \frac{F(c)}{F''(c)}$$

Метод касательных

```
static double dF(double x) {
    return 2 * x + 5 * Sin(5 * x);
}

static double ddF(double x) {
    return 2 + 25 * Cos(5 * x);
}
```

Метод касательных

```
static double Tangent(double a, double b,
                      double e) {
    double c;
    if (F(a) * ddF(a) > 0) {
        c = a;
    } else {
        c = b;
    }
    do {
        c = c - F(c) / dF(c);
    } while (Abs(F(c)) >= e);
    return c;
}
```

Метод простой итерации

Необходимо записать уравнение в виде

$$x = \varphi(x),\ |\varphi''(x)| < 1 \text{ на } [a, b]$$

Затем задать начальное приближение x_0 и организовать следующую итерацию:

$$x_{k+1} = \varphi(x_k),\ k = 0, 1, 2, \ldots$$

Вычисление прекратить, если

$$|x_{k+1} - x_k| < e$$

Метод простой итерации

```
static double Fi(double x) {
    return Acos(x * x) / 5.0;
}

static double Iteration(double x, double e) {
    double x0;
    do {
        x0 = x;
        x = Fi(x0);
    } while (Abs(x0 - x) >= e);
    return x;
}
```

Результаты

```
static void Main() {
    var a = 0.2;
    var b = 0.4;

    Write("Введите точность: ");
    var e = double.Parse(ReadLine());

    WriteLine("Метод половинного деления: "

        Dichotomy(a, b, e));
```

Результаты

```
    WriteLine("Метод хорд: " +
        Chords(a, b, e));
    WriteLine("Метод касательных: " +
        Tangent(a, b, e));
    WriteLine("Метод простой итерации:"

        Iteration(a, e));

    ReadKey();
}
```

Результаты

```
Введите точность: 0,1
Метод половинного деления: 0,25
Метод хорд: 0,292954192846601
Метод касательных: 0,292238245020846
Метод простой итерации:0,29538526838326
```

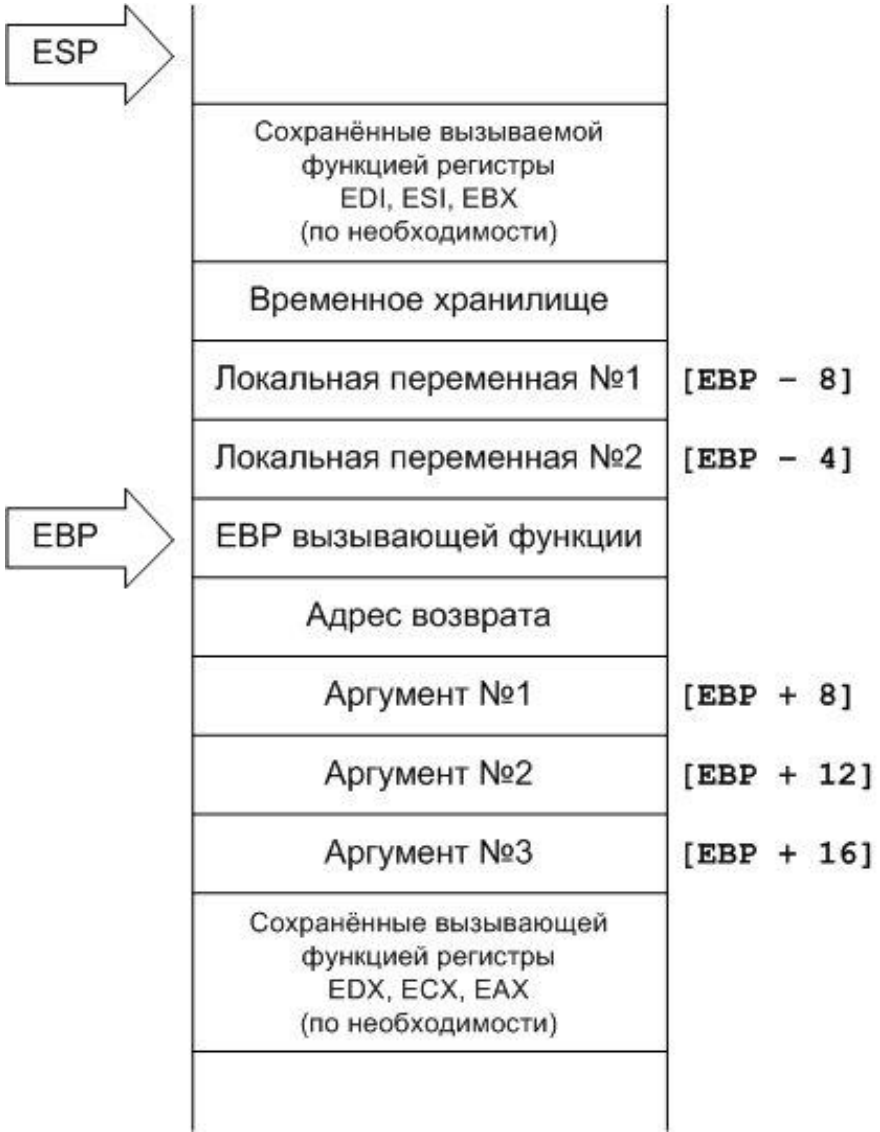
```
Введите точность: 0,01
Метод половинного деления: 0,29375
Метод хорд: 0,296546259648606
Метод касательных: 0,296555952091684
Метод простой итерации:0,296686556064l3
```

Результаты

```
Введите точность: 0,001
Метод половинного деления: 0,29609375
Метод хорд: 0,296546259648606
Метод касательных: 0,296555952091684
Метод простой итерации:0,29653186973732
```

```
Введите точность: 0,0000001
Метод половинного деления: 0,296548366546631
Метод хорд: 0,296548334965078
Метод касательных: 0,296548333317434
Метод простой итерации:0,296548329985194
```

Вызов функции



Системный стек – специальная область памяти, выделяемая для каждой программы в системе.

Windows – 1 MB
Unix – 300 KB

Рекурсия

- **Рекурсия в широком смысле** – это определение объекта посредством ссылки на себя.
- **Рекурсия в программировании** – это пошаговое разбиение задачи на подзадачи, подобные исходной.
- **Рекурсивный алгоритм** – это алгоритм, в определении которого содержится прямой или косвенный вызов этого же алгоритма.

Рекурсия

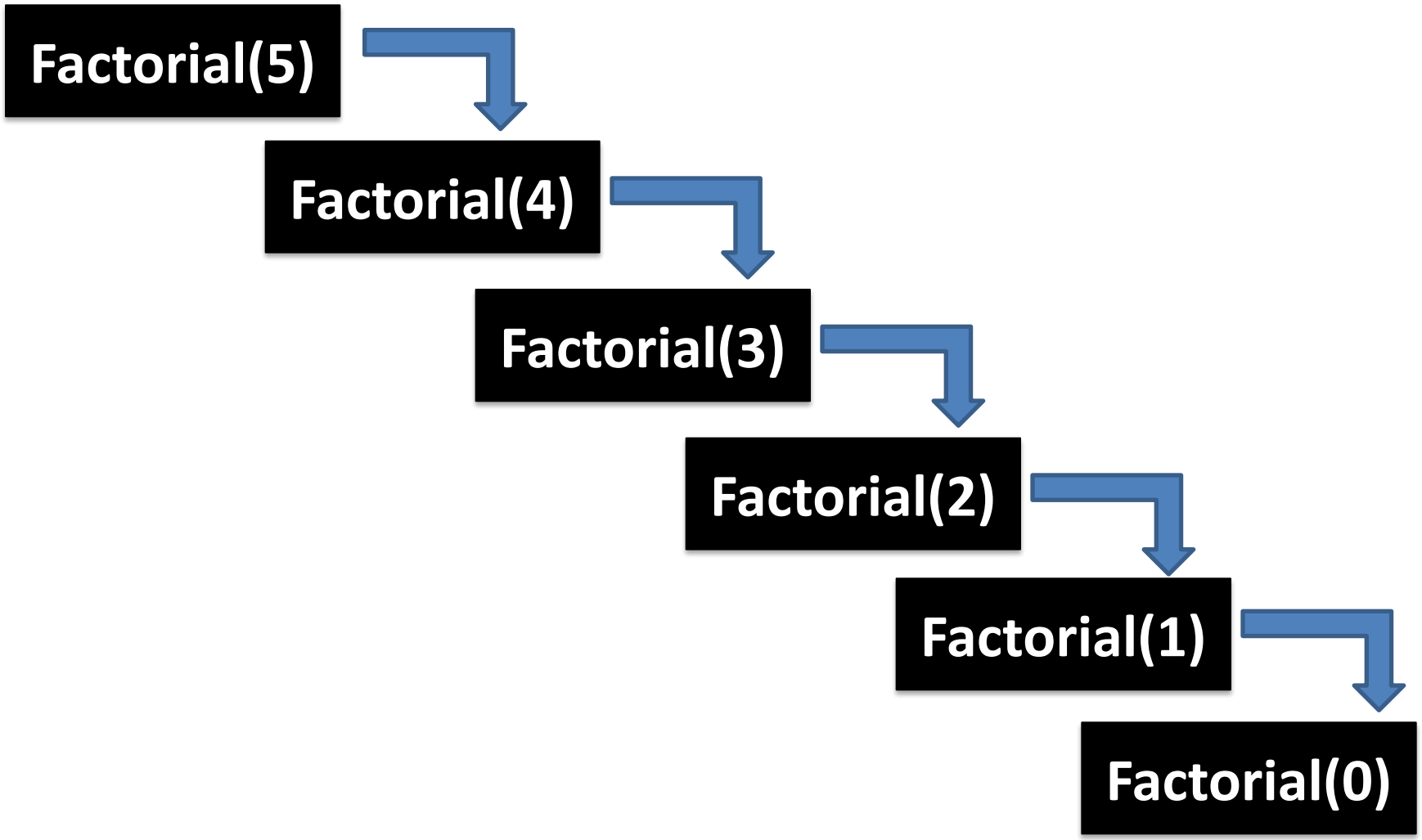
Функция называется **рекурсивной**, если в своем теле она содержит обращение к самой себе с измененным набором параметров. При этом **количество обращений конечно**, так как в итоге решение сводится к базовому случаю, когда ответ очевиден.

Рекурсивное вычисление факториала

```
static int Factorial(int n) {
  if(n <= 1)
    return 1;
  return n*Factorial(n - 1);
}

static void Main() {
  Console.WriteLine(Factorial(5));
}
```

Рекурсивное вычисление факториала



Рекурсивное вычисление факториала

Factorial(5) `return 5*Factorial(4);`

Factorial(4) `return 4*Factorial(3);`

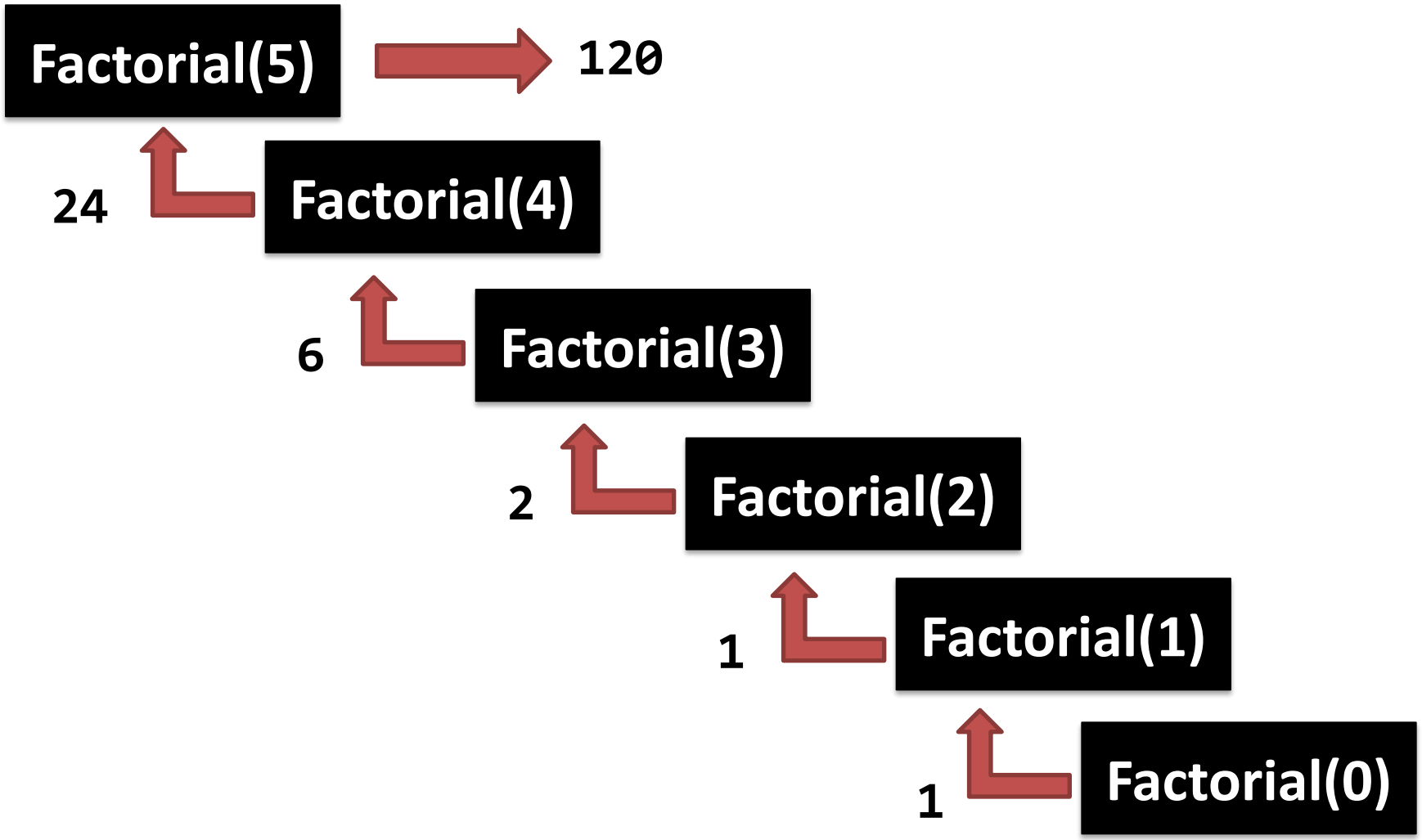
Factorial(3) `return 3*Factorial(2);`

`return 2*Factorial(1);` **Factorial(2)**

`return 1*Factorial(0);` **Factorial(1)**

`if(n <= 1) return 1;` **Factorial(0)**

Рекурсивное вычисление факториала



Ханойская башня

Даны три стержня, на один из которых нанизаны восемь колец, причем кольца отличаются размером и лежат меньшее на большем. Задача состоит в том, чтобы перенести пирамиду из восьми колец за наименьшее число ходов на другой стержень. За один раз разрешается переносить только одно кольцо, причём нельзя класть большее кольцо на меньшее.

Ханойская башня

Рекурсивное решение

1. Решаем задачу «перенести башню из $n-1$ диска на 2-й стержень»
2. Переносим самый большой диск на 3-й стержень
3. Решаем задачу «перенеси башню из $n-1$ диска на 3-й стержень».

Ханойская башня

```
static void FindSolution(int amountOfDisks) {
    MoveDisks(amountOfDisks, 1, 3, 2);
}
```

Ханойская башня

```
static void MoveDisks(int n, int fromPole,
          int toPole, int intermediatePole) {
  if (n > 0) {
    MoveDisks(n - 1, fromPole,
              intermediatePole, toPole);

    WriteLine("Перенос диска {0} со столба {1}
              на столб {2}“, n, fromPole, toPole);

    MoveDisks(n - 1, intermediatePole,
              toPole, fromPole);
  }
}
```

Ханойская башня

```
class Program {
  static void Main(string[] args) {
    FindSolution(2);
    ReadKey();
  }
}
```

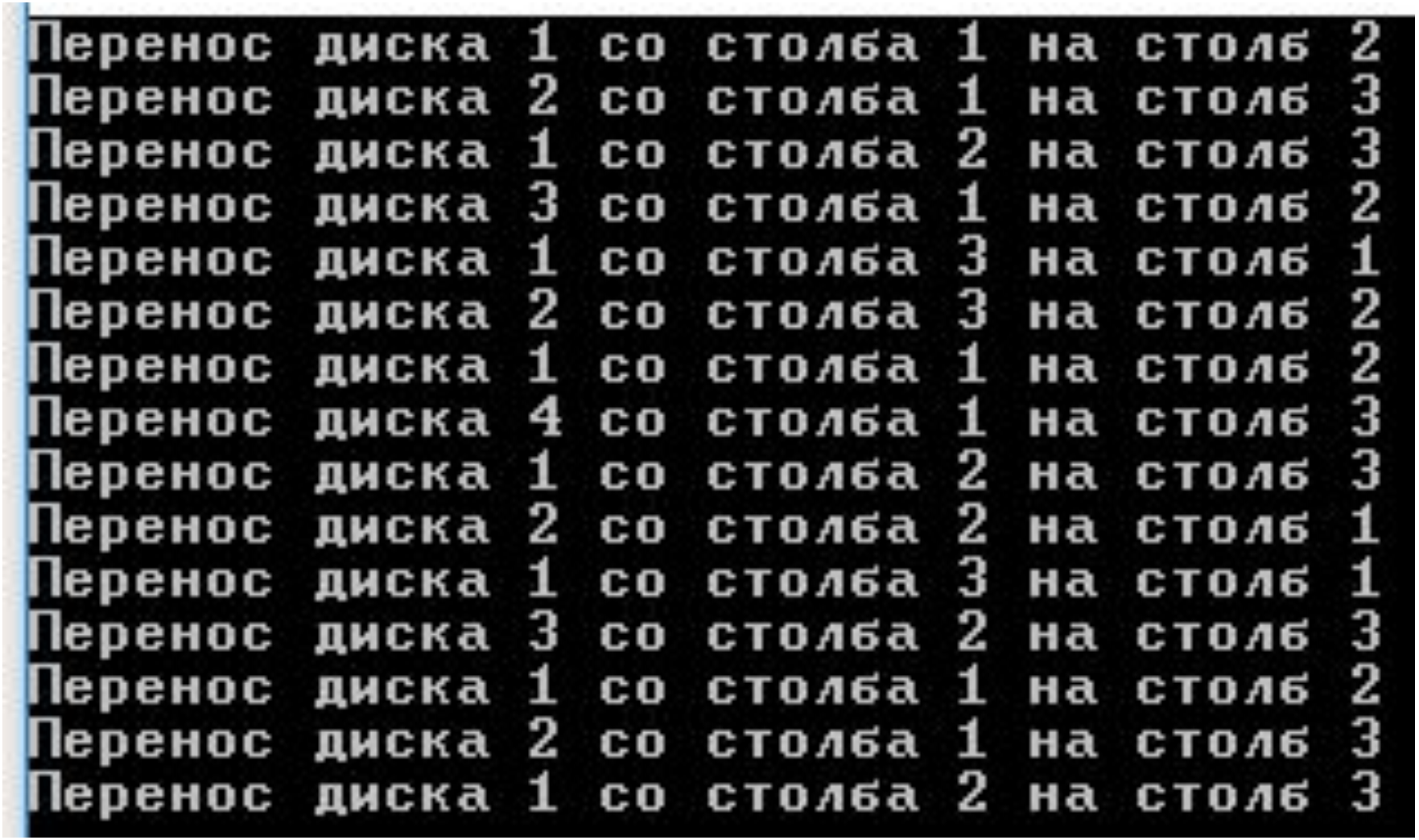
```
Перенос диска 1 со столба 1 на столб 2
Перенос диска 2 со столба 1 на столб 3
Перенос диска 1 со столба 2 на столб 3
```

Ханойская башня

```
class Program {
  static void Main(string[] args) {
    FindSolution(3);
    ReadKey();
  }
}
```

```
Перенос диска 1 со столба 1 на столб 3
Перенос диска 2 со столба 1 на столб 2
Перенос диска 1 со столба 3 на столб 2
Перенос диска 3 со столба 1 на столб 3
Перенос диска 1 со столба 2 на столб 1
Перенос диска 2 со столба 2 на столб 3
Перенос диска 1 со столба 1 на столб 3
```

Ханойская башня



Треугольник Серпинского

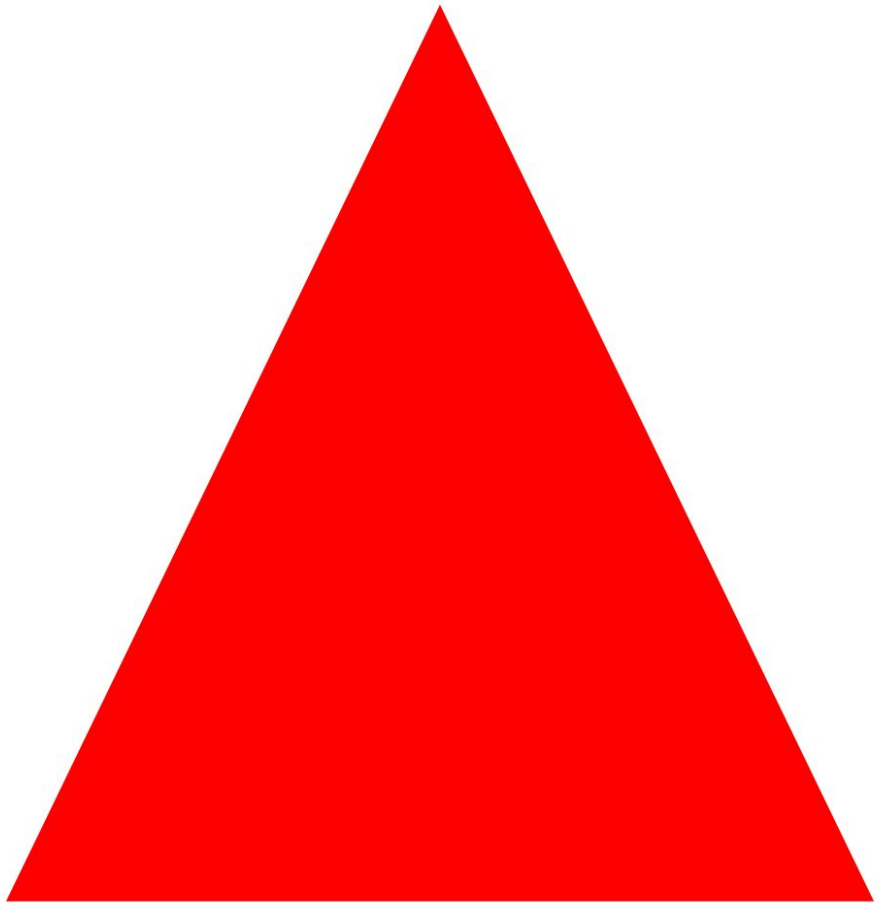
В 1915 году польский математик Вацлав Серпинский придумал занимательный объект, известный как решето Серпинского. Этот треугольник один из самых ранних известных примеров фракталов.

Берётся сплошной равносторонний треугольник, на первом шаге из центра удаляется перевёрнутый треугольник.

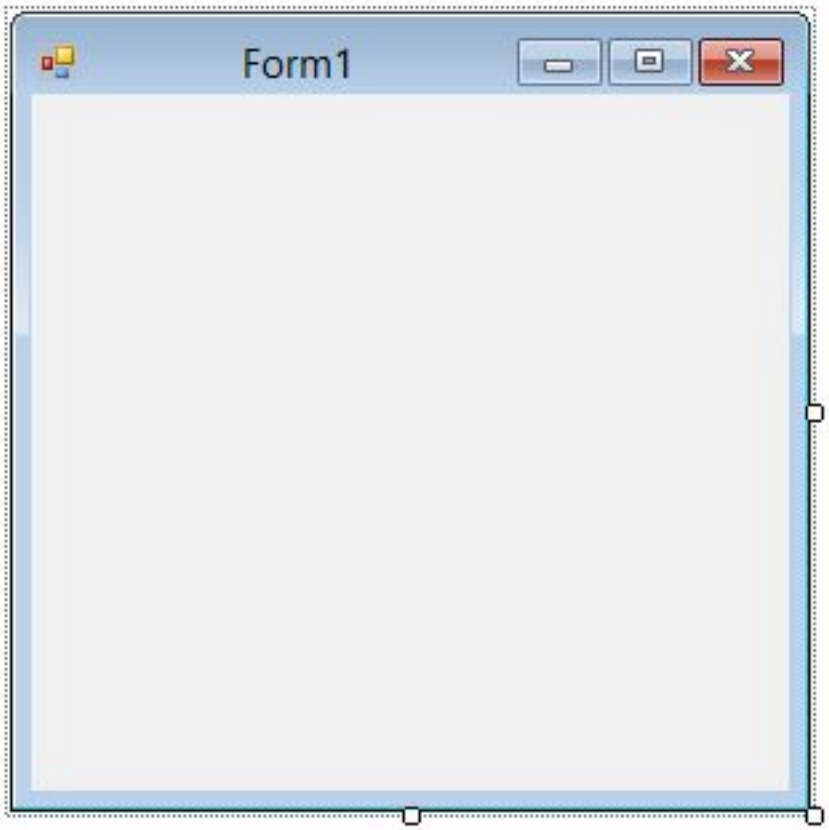
Треугольник Серпинского

На втором шаге удаляется три перевёрнутых треугольника из трёх оставшихся треугольников. Продолжая этот процесс, на n-ом шаге удаляем 3^{n-1} перевёрнутых треугольников из центров 3^{n-1} оставшихся треугольников. Конца этому процессу не будет, и в треугольнике не останется живого места, но и на части он не распадётся - получится объект состоящий из одних только дырок.

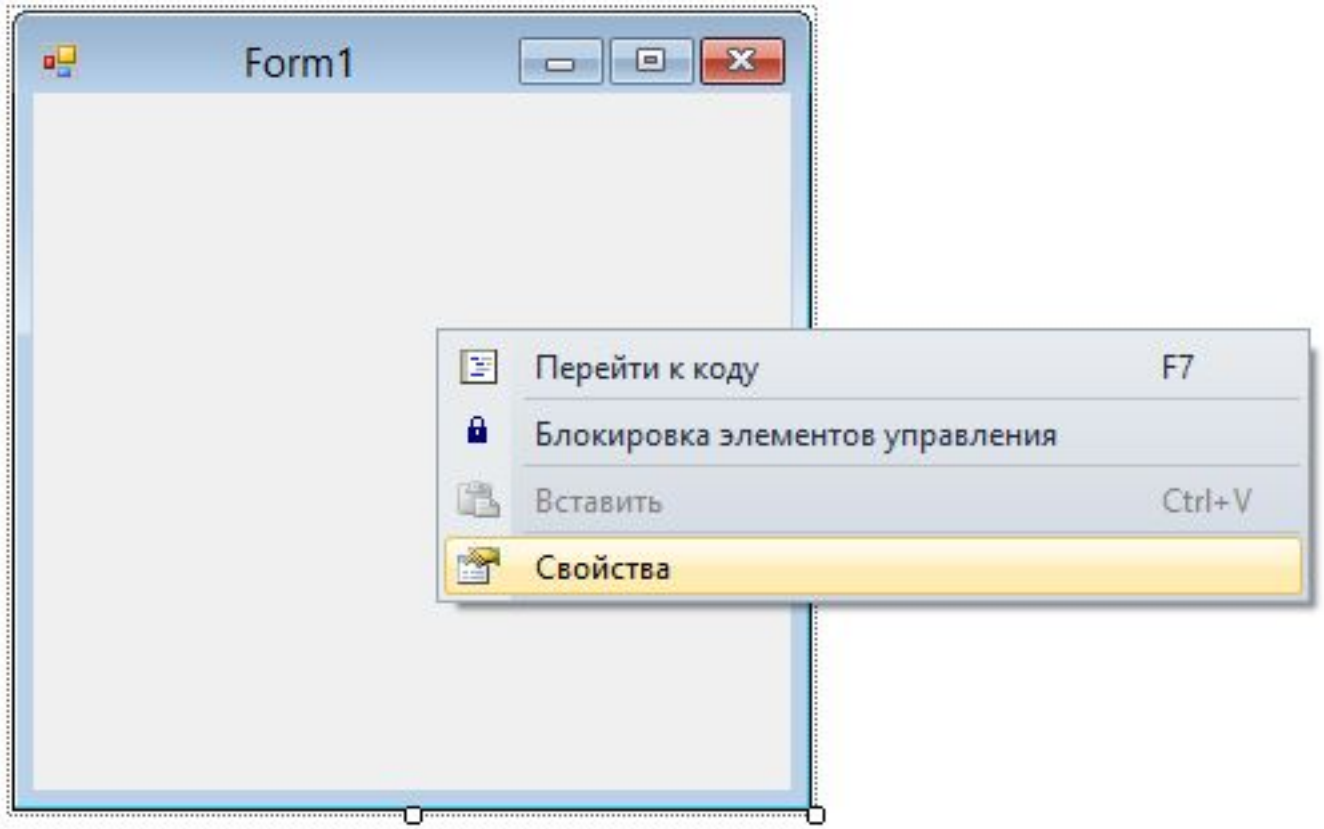
Треугольник Серпинского



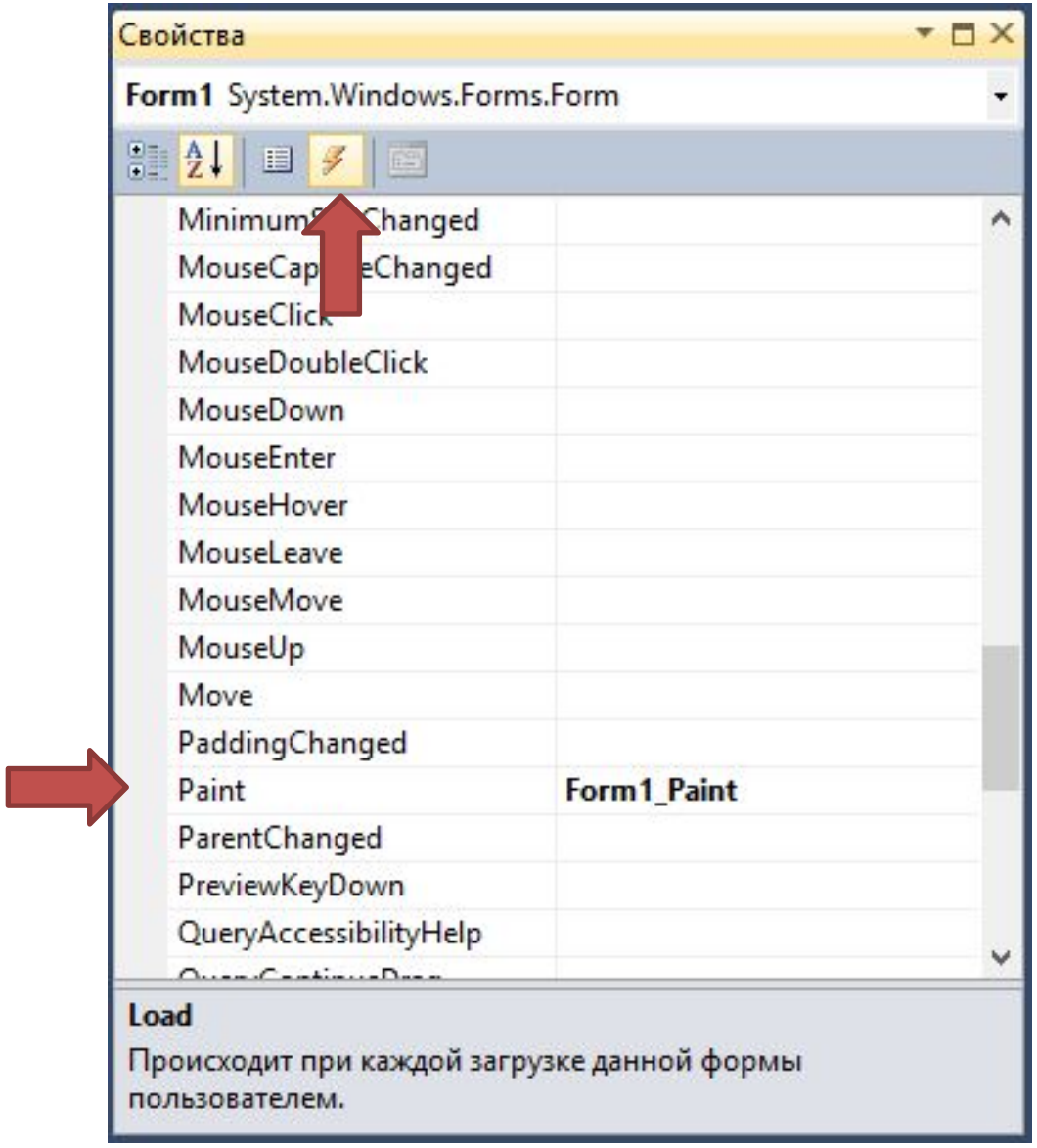
Треугольник Серпинского



Треугольник Серпинского



Треугольник Серпинского



Треугольник Серпинского

```
public partial class Form1 : Form
{
  public Form1()
  {
    InitializeComponent();
  }
  private void Form1_Paint(object sender,
                           PaintEventArgs e)
  {
  }
}
```

Треугольник Серпинского

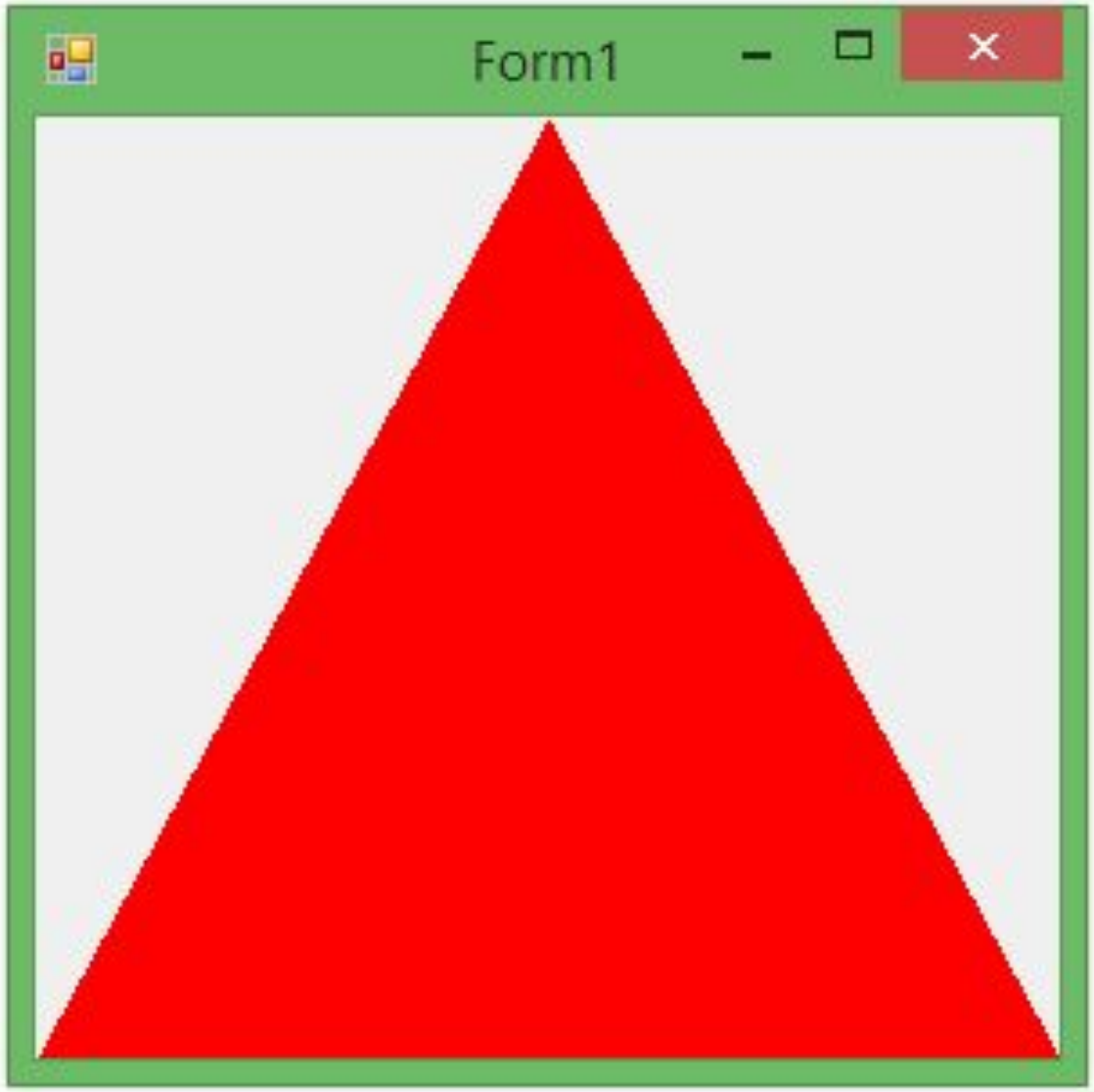
```
private void Form1_Paint(object sender,
                         PaintEventArgs e)
{
  Rectangle bounds = e.ClipRectangle;
  Point a = new Point(bounds.Width/2, 0);
  Point b = new Point(0, bounds.Height);
  Point c = new Point(bounds.Width,
                      bounds.Height);
  Graphics g = e.Graphics;
  g.FillPolygon(Brushes.Red,
                new Point[] { a, b, c });
}
```

Треугольник Серпинского

```
private v                                                    r,
                                                            gs e)
{
   Rectang
   Point a                                                    0);
   Point b                                                    t);
   Point c

   Graphic
   g.FillP

                                                            });
}
```



```
void Serpinski(Graphics g, Point a, Point b,
                           Point c, int n)
{
  if (n > 0) {
    Point newA = Center(a, b);
    Point newB = Center(b, c);
    Point newC = Center(c, a);

    g.FillPolygon(Brushes.White,
          new Point[] { newA, newB, newC });

    Serpinski(g, a, newA, newC, n - 1);
    Serpinski(g, newA, b, newB, n - 1);
    Serpinski(g, newC, newB, c, n - 1);
  }
}
```

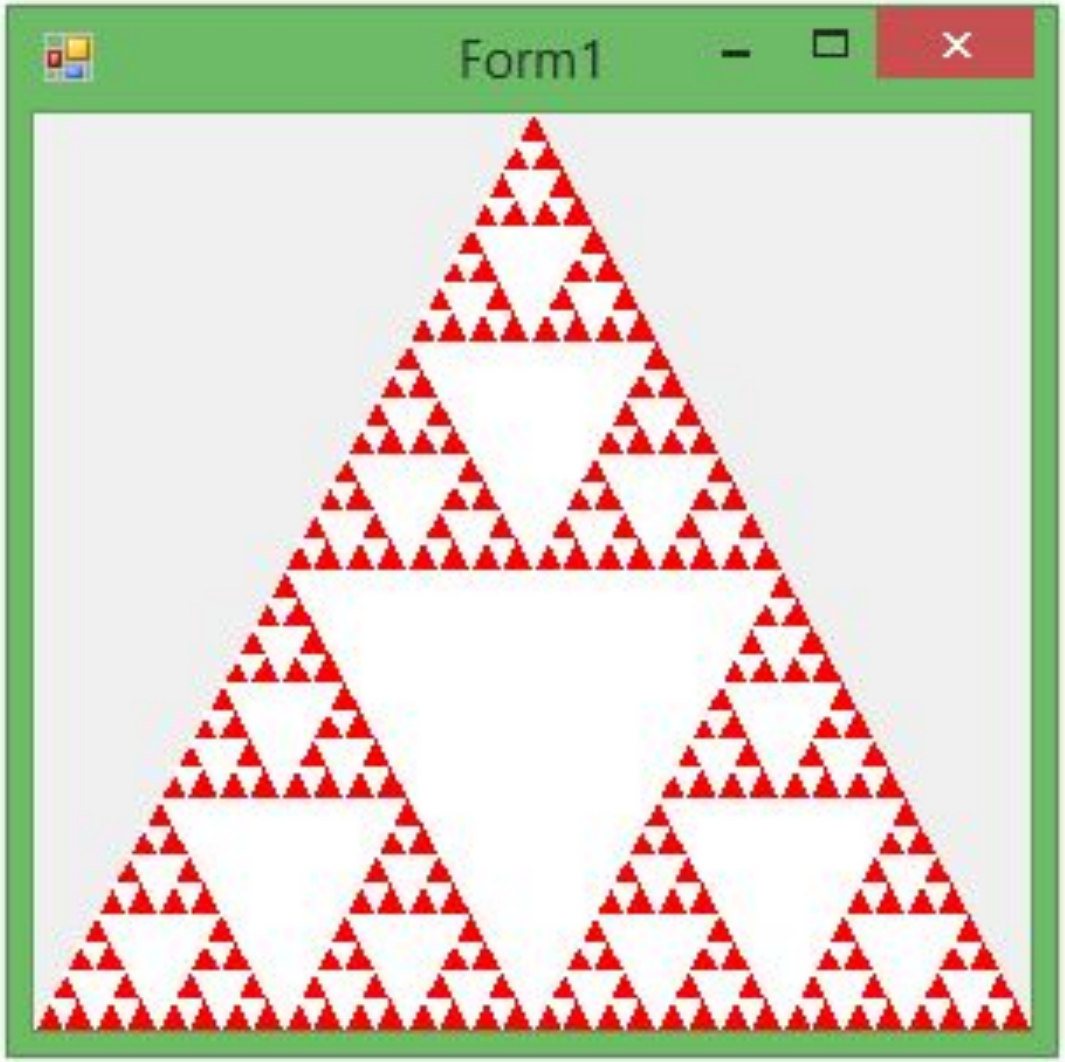
Треугольник Серпинского

```
Point Center(Point a, Point b)
{
    return new Point(
        (a.X + b.X)/2,
        (a.Y + b.Y)/2
    );
}
```

```
private void Form1_Paint(object sender,
                         PaintEventArgs e)
{
  Rectangle bounds = e.ClipRectangle;
  Point a = new Point(bounds.Width/2, 0);
  Point b = new Point(0, bounds.Height);
  Point c = new Point(bounds.Width,
                      bounds.Height);
  Graphics g = e.Graphics;
  g.FillPolygon(Brushes.Red,
                new Point[] { a, b, c });

  Serpinski(g, a, b, c, 5);
}
```

Треугольник Серпинского



Значения параметров по умолчанию

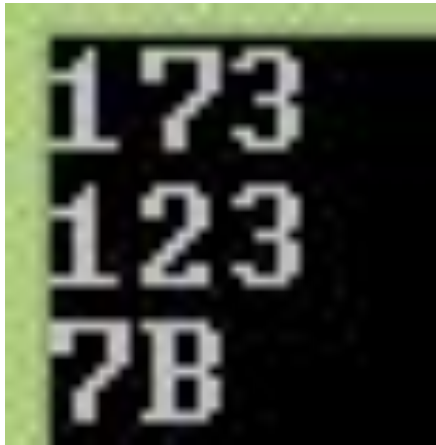
У функции может быть больше параметров, чем в самых простых и наиболее часто используемых случаях

Значения параметров по умолчанию

```
static void Print(int n, int b)
{
    switch (b) {
    case 8: Console.WriteLine(
                    Convert.ToString(n, 8));
       break;
    case 10: Console.WriteLine(n); break;
    case 16: Console.WriteLine("{0:X}", n);
       break;
    }
}
```

Значения параметров по умолчанию

```
static void Main(string[] args)
{
    Print(123, 8);
    Print(123, 10);
    Print(123, 16);
}
```



Значения параметров по умолчанию

```
static void Print(int n, int b = 10) {
    switch (b) {
    case 8: Console.WriteLine(
                        Convert.ToString(n, 8));
       break;
    case 10: Console.WriteLine(n); break;
    case 16: Console.WriteLine("{0:X}", n);
       break;
    }
}
```

```
Print(123, 10);
```

```
Print(123);
```

Значения параметров по умолчанию

Параметры по умолчанию должны идти в конце функции:

```
// нормально
int F(int x, int y = 0, int z = 0);
// ошибка
int G(int x = 0, int y = 0, int z);
// ошибка
int H(int x = 0, int y, int z = 0);
```

Значения параметров по умолчанию

```
static int Sum(int x=1, int y=2, int z=3)
{
    return x + y + z;
}
static void Main(string[] args)
{
    Console.WriteLine( Sum() );
    Console.WriteLine( Sum(10) );
    Console.WriteLine( Sum(10, 20) );
    Console.WriteLine( Sum(z:30) );
    Console.WriteLine( Sum(z:30, x:10) );
}
```

Значения параметров по умолчанию

```
static int Sum(int x=1, int y=2, int z=3)
{
    return x + y + z;
}
static void Main(string[] args)
{
    Console.WriteLine( Sum() );
    Console.WriteLine( Sum(10) );
    Console.WriteLine( Sum(10, 20) );
    Console.WriteLine( Sum(z:30) );
    Console.WriteLine( Sum(z:30, x:10) );
}
```

Перегрузка функций

```
static void Main() {
   for(int i = 0; i < 10; i++)
      Console.WriteLine(Pow(2, i));
}
```

Перегрузка функций

```
static int Pow(int x, int n) {
  if(n < 0) throw
    new
ArgumentException("ошибка");
  switch(n)
  {
  case 0: return 1;
  case 1: return x;
  default: return x*Pow(x, n-1);
  }
}
```

Перегрузка функций

В большинстве ранних языков программирования, для упрощения процесса трансляции существовало ограничение, согласно которому одновременно в программе **не может быть доступно более одной процедуры с одним и тем же именем.**

В соответствии этому ограничению, все функции, видимые в данной точке программы, должны иметь различные имена.

Перегрузка функций

Разные функции обычно имеют разные имена, но функциям, выполняющим сходные операции над объектами разных типов, лучше дать одно имя.

Если типы параметров таких функций различны, то транслятор всегда может разобраться, какую функцию нужно вызывать.

Перегрузка функций

```
static double Pow(double x, int n) {
   if(n < 0) throw
      new
ArgumentException("ошибка");
   switch(n)
   {
   case 0: return 1;
   case 1: return x;
   default: return x*Pow(x, n-1);
   }
}
```

Перегрузка функций

```
int pow(int, int);
double pow(double, int);

// вызов pow(int, int)
x = pow(2, 10);

// вызов pow(double,
int)
y = pow(2.0, 10);
```

Перегрузка функций

```
int pow(int, int);
```

```
int __pow_i_i(int, int)
```

```
double pow(double, int);
```

```
double __pow_d_i(double, int)
```

Перегрузка функций

```
void Print(int n, int base = 10)
```

Эквивалентная запись при помощи перегрузки:

```
void Print(int n) { Print(n, 10); }
```

Работа со структурами

Рассмотрим на примере простой программы по работе с адресами

Функции программы:

- Чтение входных данных
- Хранение данных
- Вывод данных

Пример: почтовый адрес

```
using System;

class Program {
  static void Main() {

    // адрес
    string name;      // имя
    string town;      // город
    string street;    // улица
    long number;      // номер дома
    int zip;          // почтовый
индекс
```

Пример: почтовый адрес

```
    // чтение данных
    Console.WriteLine("Введите параметры
адреса");
    Console.Write("Имя: ");
    name = Console.ReadLine();
    Console.Write("Город: ");
    town = Console.ReadLine();
    Console.Write("Улица: ");
    street = Console.ReadLine();
    Console.Write("Номер дома: ");
    number = long.Parse(Console.ReadLine());
    Console.Write("Индекс: ");
    zip = int.Parse(Console.ReadLine());
```

Пример: почтовый адрес

```
// чтение данных
Console.WriteLine("Введите параметры
адр
number = long.Parse(Console.ReadLine());
Console.Write("Индекс: ");
zip = int.Parse(Console.ReadLine());
```

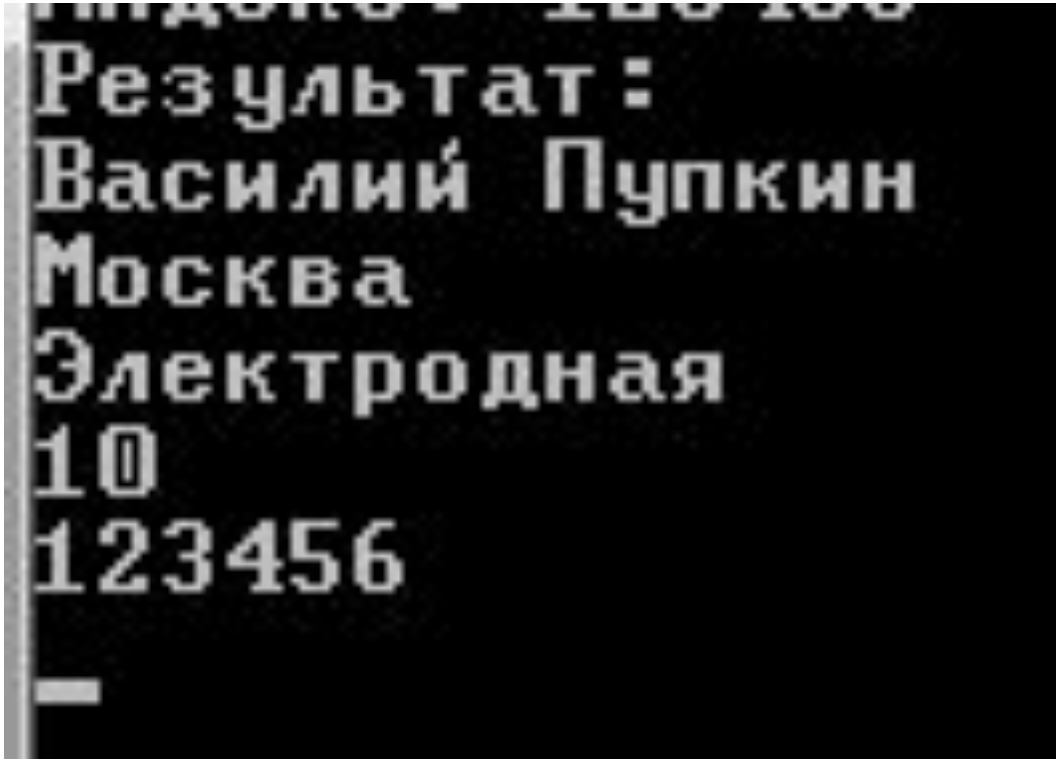
```
Введите параметры адреса
Имя: Василий Пупкин
Город: Москва
Улица: Электродная
Номер дома: 10
Индекс: 123456
```

Пример: почтовый адрес

```
        // вывод данных
        Console.WriteLine("Результат: ");
        Console.WriteLine(name);
        Console.WriteLine(town);
        Console.WriteLine(street);
        Console.WriteLine(number);
        Console.WriteLine(zip);

        Console.ReadKey();
    }
}
```

Пример: почтовый адрес




```
");
    }
}
```

Основным недостатком полученной программы является отсутствие возможности повторно использовать уже написанный код

Переместим часть кода в функции (вывод)

```
// вывод данных
Console.WriteLine("Результат: ");
Console.WriteLine(name);
Console.WriteLine(town);
Console.WriteLine(street);
Console.WriteLine(number);
Console.WriteLine(zip);
```

```
// вывод данных
PrintAddress(name, town, street, number, zip);
```

Переместим часть кода в функции (вывод)

```
static void PrintAddress(string name,
                         string town,
                         string street,
                         long number,
                         int zip)
{
  Console.WriteLine("Результат: ");
  Console.WriteLine(name);
  Console.WriteLine(town);
  Console.WriteLine(street);
  Console.WriteLine(number);
  Console.WriteLine(zip);
}
```

Переместим часть кода в функции (вывод)

Результат не изменился:

```
Результат:
Василий Пупкин
Москва
Электродная
10
123456
```

Попытаемся аналогичным образом создать функцию чтения данных

Переместим часть кода в функции (ввод)

```
    // чтение данных
    Console.WriteLine("Введите параметры
адреса");
    Console.Write("Имя: ");
    name = Console.ReadLine();
    Console.Write("Город: ");
    town = Console.ReadLine();
    Console.Write("Улица: ");
    street = Console.ReadLine();
    Console.Write("Номер дома: ");
    number = long.Parse(Console.ReadLine());
    Console.Write("Индекс: ");
    zip = int.Parse(Console.ReadLine());
```

```
    // чтение данных
    ReadAddress(name, town, street, number,
 zip);
```

Переместим часть кода в функции (ввод)

```
static void ReadAddress(string name, string town,
                        string street, long number,
                        int zip) {
  Console.WriteLine("Введите параметры адреса");
  Console.Write("Имя: ");
  name = Console.ReadLine();
  Console.Write("Город: ");
  town = Console.ReadLine();
  Console.Write("Улица: ");
  street = Console.ReadLine();
  Console.Write("Номер дома: ");
  number = long.Parse(Console.ReadLine());
  Console.Write("Индекс: ");
  zip = int.Parse(Console.ReadLine());
}
```

Ошибки компиляции

```
static void Main()
{
    // адрес
    string name;            // имя
    string town;            // город
    string street;          // улица
    long number;            // номер дома
    int zip;                // почтовый индекс

    // чтение данных
    ReadAddress(name, town, street, number, zip);

    // вывод дан
    PrintAddress                                    p);

    Console.Read
}
```

(local variable) int zip

Error:
Use of unassigned local variable 'zip'

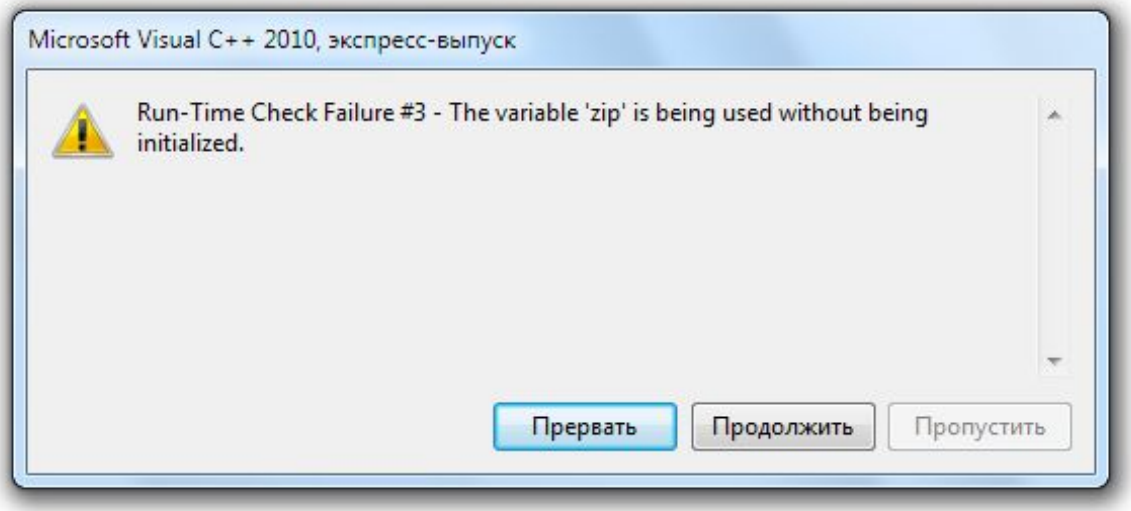
Ошибки компиляции

error CS0165: Use of unassigned local variable 'name'
error CS0165: Use of unassigned local variable 'town'
error CS0165: Use of unassigned local variable 'street'
error CS0165: Use of unassigned local variable 'number'
error CS0165: Use of unassigned local variable 'zip'

В отличие от C++, компилятор C# считает использование неинициализированных переменных ошибкой

Ошибки компиляции

В C++ мы получали ошибку во время выполнения программы:



Ошибки компиляции

Проблема решалась при помощи указателей:

```
// чтение данных
void ReadAddress(wstring* name,
                 wstring* town,
                 wstring* street,
                 long* number,
                 int* zip)
```

В C# указатели можно использовать только в режиме неуправляемого кода

В безопасном режиме все немного сложнее

Типы данных C#

Система типов языка C# включает следующие категории:

- Указатели
- **Ссылочные типы**
- **Типы значений**

Две последние категории рассмотрим более подробно

Ссылочные типы

Переменные ссылочных типов хранят ссылки на фактические данные.

Ссылочными типами являются:

- **Классы** (class)
- Делегаты (delegate)
- Интерфейсы (interface)
- *Строки (string)*
- *Массивы*

Типы значений

Тип значений хранит свое содержимое в памяти, выделенной в стеке.

Типы значений состоят из двух основных категорий :

- **Структуры** (struct)
- Перечисления (enum)

Все базовые типы C#, за исключением string, представляют собой типы значений

Особенности типов значений

Переменные, основанные на типах значений, содержат непосредственно значения. При присвоении переменной одного типа значений другому создается копия присваиваемого значения

В этом заключается отличие от переменных ссылочного типа, при присвоении которых копируются ссылки на объекты, но не сами объекты

Особенности типов значений

- В отличие от ссылочных типов тип значения не может содержать значение **null**

- Для каждого типа значений существует неявный конструктор по умолчанию, инициализирующий значение по умолчанию для данного типа

Ключевые слова out и ref

Для передачи типов значений по ссылке в безопасном режиме были введены специальные ключевые слова out и ref

```
static void Read(out C c) {
  c.x = int.Parse(Console.ReadLine());
}
```

```
static void Read(ref C c) {
  c.x = int.Parse(Console.ReadLine());
}
```

Отличие out от ref

ref указывает компилятору, что передаваемый объект был инициализирован до вызова функции.

out сообщает, что переменная будет инициализированна внутри функции.

Т.е. **ref** обеспечивает передачу значения в обе стороны, а **out** только из функции наружу.

Отличие out от ref: ошибка при использовании out

```
static void Read(out C c)
{
    int y = c.x;
    c.x = in                                    );
}

static voi
{
```

(parameter) C c

Error:
Use of possibly unassigned field 'x'

В таких случаях нужно использовать ref

Итог: функция ReadAddress

```
static void ReadAddress(out string name,
                        out string town, out string street,
                        out long number, out int zip) {
  Console.WriteLine("Введите параметры адреса");
  Console.Write("Имя: ");
  name = Console.ReadLine();
  Console.Write("Город: ");
  town = Console.ReadLine();
  Console.Write("Улица: ");
  street = Console.ReadLine();
  Console.Write("Номер дома: ");
  number = long.Parse(Console.ReadLine());
  Console.Write("Индекс: ");
  zip = int.Parse(Console.ReadLine());
}
```

При вызове функции, по аналогии с C++, перед каждой переменной указываем ключевое слово out

```
ReadAddress(out name,
            out town,
            out street,
            out number,
            out zip);
```

Итог

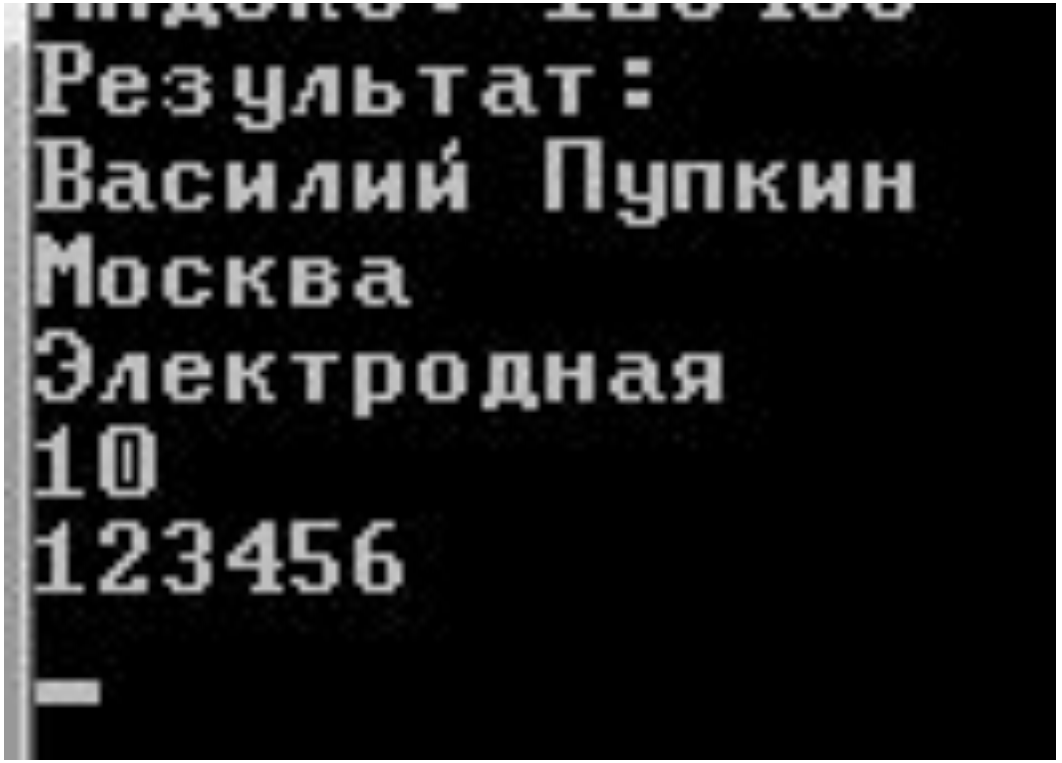
```
static void Main()
{
  // адрес
  string name;       // имя
  string town;       // город
  string street;     // улица
  long number;       // номер дома
  int zip;           // почтовый индекс

  // чтение данных
  ReadAddress(out name, out town, out street, out number, out zip);

  // вывод данных
  PrintAddress(name, town, street, number, zip);

  Console.ReadKey();
}
```

Результат не изменился



Слишком большие заголовки функций, избыток переменных

```
static void PrintAddress(string name,
  string town, string street, long number,
  int zip)

static void ReadAddress(out string name,
  out string town, out string street,
  out long number, out int zip)
```

Структура – множество логически связанных данных

```
// адрес
struct Address
{
   public string name;     // имя
   public string town;     // город
   public string street;   // улица
   public long number;     // номер дома
   public int zip;         // почтовый
индекс
};
```

Отличия от C++

- Область видимости по умолчанию – **private**
- Нельзя объявлять конструкторы без параметорв
- Инициализируется значениями по умолчанию
- Нельзя использовать наследование

Доступ к полям при помощи оператора (.)

```
static void PrintAddress(Address addr)
{
  Console.WriteLine("Результат: ");
  Console.WriteLine(addr.name);
  Console.WriteLine(addr.town);
  Console.WriteLine(addr.street);
  Console.WriteLine(addr.number);
  Console.WriteLine(addr.zip);
}
```

```
addr.town
```

Создание переменной структуры

```
static void Main()
{
    Address addr = new Address();

    // чтение данных
    ReadAddress(out addr);

    // вывод данных
    PrintAddress(addr);
    Console.ReadKey();
}
```

Инициализация структуры (1)

```
  static void Main()
  {
    // адрес
    Address addr = new Address();
    addr.name = "Василий
Пупкин";
    addr.town = "Москва";
    addr.street = "Электродная";
    addr.number = 1;
    addr.zip = 123456;

    // вывод данных
    PrintAddress(addr);
    Console.ReadKey();
  }
```

Результат:
Василий Пупкин
Москва
Электродная
1
123 456

Инициализация структуры (2)

```
static void Main() {
  // адрес
  Address addr = new Address(){
    name = "Василий Пупкин",
    town = "Москва",
    street = "Электродная",
    number = 1,
    zip = 123456
  };

  // вывод данных
  PrintAddress(addr);
  Console.ReadKey();
}
```

Результат:
Василий Пупкин
Москва
Электродная
1
123 456

Инициализация структуры (3)

```
struct Address {
  public string name;       // имя
  public string town;       // город
  public string street;     // улица
  public long number;       // номер дома
  public int zip;           // почтовый индекс

  public Address(string name, string town,
                 string street, long number, int zip) {
    this.name = name;
    this.town = town;
    this.street = street;
    this.number = number;
    this.zip = zip;
  }
};
```

Инициализация структуры (3)

```
static void Main()
{
  // адрес
  Address addr = new Address("Василий Пупкин",
                "Москва", "Электродная", 1,
123456);

  // вывод данных
  PrintAddress(addr);
  Console.ReadKey();
}
```

```
Результат:
Василий Пупкин
Москва
Электродная
1
123 456
```

Копирование структур

```
static void Main()
{
  // адрес
  Address addr = new Address("Василий Пупкин",
                 "Москва", "Электродная", 1,
123456);

  Address addr2 = addr;

  // вывод данных
  PrintAddress(addr2);
  Console.ReadKey();
}
```

```
Address addr2 = addr;
```

Массивы структур

```
static void Main()
{
  // адрес
  Address[] addr = new Address[5] {
    new Address("Имя5", "A", "B", 0,
0),
    new Address("Имя4", "A", "B", 0,
0),
    new Address("Имя3", "A", "B", 0,
0),
    new Address("Имя2", "A", "B", 0,
0),
    new Address("Имя1", "A", "B", 0,
0),
  };
  // вывод данных
  for(int i = 0; i < 5; i++)
    PrintAddress(addr[i]);
```

```
Результат:
Имя5
A
B
0
0

Результат:
Имя4
A
B
0
0

Результат:
Имя3
...
```