Структура программы: описание класса.

```
class <Имя класса>
{
    <Описание компонент, которыми обладают все его объекты>
}
```

Структура программы: точка входа.

```
class <Имя класса>
{
    ...
    static void Main()
    {
        <Что делать с самого начала и дальше>
    }
    ...
}
```

Структура программы: простейший вид.

```
class Program
{
    static void Main()
    {
        <Все инструкции>
    }
}
```

Структура программы: namespace (пространство имен) System.

```
using System;
class Program
{
    static void Main()
    {
        <Все инструкции>
    }
}
```

Переменные.

- Тип данных.
- Имя (идентификатор) переменной.
- Размещение в оперативной памяти (где, как, длина и т.п.).

Переменные.

- Тип данных.
- Имя (идентификатор) переменной.
- Размещение в оперативной памяти (где, как, длина и т.п.).

Оператор присваивания.

```
<имя переменной> = <выражение>;
```

Напр.:

```
x = 3.5;    s = "Вася";     z = (3.5 + 0.3*a)*sin(alpha);
n = 1+2;    l = false;      ss = "Вася" + lastname;
```

Оператор присваивания: тонкости.

```
n = 2;
n = n+5;
```

n = ???

Оператор присваивания: тонкости.

```
n = 2;
n = n+5;
```

n = ???

```
n++;          // синоним для  n = n+1;
n--;          // синоним для  n = n-1;
n += 5;       // синоним для  n = n+5;
n -= 5;       // синоним для  n = n-5;
n *= 5;       // синоним для  n = n*5;
```

Тип данных — это что и как с ними можно делать.
Описания переменных:

```
    <тип переменной> <идентификатор переменной>;
```

или

```
    <тип переменной> <идентификатор переменной> = ... ;
```

Тип данных — это что и как с ними можно делать.
Описания переменных:

```
    <тип переменной> <идентификатор переменной>;
```

или

```
    <тип переменной> <идентификатор переменной> = ... ;
```

Напр.:

```
    int m;  int n; double time;
    double a = 3.5;
    string s = "Привет! Хорошая погода!";
    bool b = false;
    int i, j, k=0, l=1;
```

int — действия: + , − , * , / , %

```
(3+7)/4 == 2          17 % 4 == 1
```

int — действия: + , − , * , / , %

```
(3+7)/4 == 2          17 % 4 == 1

    (3+7)/4.0 == 2.5          // делитель и результат в типе double
```

int — действия: + , − , * , / , %

```
(3+7)/4 == 2          17 % 4 == 1
```

```
(3+7)/4.0 == 2.5          // делитель и результат в типе double
```

double — действия: + , − , * , /

```
int n = 5;
double x = (double) n / 2;     // получится 2.5
int m = (int) x;               // получится 2
```

int — действия: + , − , * , / , %

```
(3+7)/4 == 2          17 % 4 == 1

    (3+7)/4.0 == 2.5          // делитель и результат в типе double
```

double — действия: + , − , * , /

```
int n = 5;
double x = (double) n / 2;       // получится 2.5
int m = (int) x;                 // получится 2

double y = Math.Abs(-5.25)       // модуль
double z = Math.Pow(2.2, 3.0)    // возведение в степень 2.2^{3.0}
double w = Math.Sqrt(9.0)        // квадратный корень
```

char — буквы (Unicode, utf-8)

```
char ch1 = 'z';      char ch2 = 'Ы';      char ch3 = ' ';
```

char — буквы (Unicode, utf-8)

```
    char ch1 = 'z';      char ch2 = 'Ы';      char ch3 = ' ';
```

Проверка условий:

```
    char.IsLetter(ch)    char.IsDigit(ch)     char.IsSeparator(ch)
    char.IsUpper(ch)     char.IsLower(ch)
```

Преобразования:

```
    char ch1 = char.ToUpper(ch);     char ch2 = char.ToLower(ch);
```

string — операция +

```
string s = "пар" + 'о' + "воз";
char ch = s[2];                    // ch == 'р'
int n = s.Length;                  //  n == 7
```

string — операция +

```
string s = "пар" + 'о' + "воз";
char ch = s[2];                 // ch == 'р'
int n = s.Length;               //  n == 7
```

Преобразования:

```
int n = int.Parse("123");           // n == 123
double pi = double.Parse("3,14")    // pi == 3.14

<тип> z = ...;
string s = z.ToString();   // изображение значения z в виде stri
```

bool (значения: true и false). Операции:
! (отрицание), && (конъюнкция, “И”), || (дизъюнкция, “ИЛИ”).

```
int n = 5;  bool b;  bool b1 = false;
b = (n > 1) || b1;           // true
n = -3;
b = (n > 1) || b1;           // false
b = !b1;                     // true
```

Условия (предикаты): == (равно), != (не равно),
<, >, <=, >= (сравнения).

Ввод данных с клавиатуры.

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

Ввод данных с клавиатуры.

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

int n = int.Parse (Console.ReadLine());
double x = double.Parse(Console.ReadLine());
```

Ввод данных с клавиатуры.

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

int n = int.Parse (Console.ReadLine());
double x = double.Parse(Console.ReadLine());

s = Console.ReadLine();
n = int.Parse(s);
```

Вывод данных на экран.

```
Console.WriteLine(<выражение>);
```

Вывод данных на экран.

```
Console.WriteLine(<выражение>);
```

```
Console.WriteLine("Всем привет!");
```

Вывод данных на экран.

```
Console.WriteLine(<выражение>);
```

```
Console.WriteLine("Всем привет!");

int n = 3;
Console.WriteLine(n+3);
```

Вывод данных на экран.

```
Console.WriteLine(<выражение>);
```

```
Console.WriteLine("Всем привет!");

int n = 3;
Console.WriteLine(n+3);
Console.WriteLine("пар" + 'о' + "воз");
```

Вывод данных на экран.

```
Console.WriteLine(<выражение>);
```

```
 Console.WriteLine("Всем привет!");

 int n = 3;
 Console.WriteLine(n+3);
 Console.WriteLine("пар" + 'о' + "воз");
```

```
Console.Write(<выражение>);
```

Вывод с форматированием.

```
Console.WriteLine(<формат>, <выражение>, <выражение>, ...);
```

Пример форматирования:

```
x = {0}, его квадрат - {1}
```

```
Console.WriteLine("x = {0}, его квадрат -  {1}", x, x*x );
```

Вывод с форматированием.

```
Console.WriteLine(<формат>, <выражение>, <выражение>, ...);
```

Пример форматирования:

```
x = [ ], его квадрат -  [ ]
```

```
Console.WriteLine("x = {0}, его квадрат -  {1}", x, x*x );
```

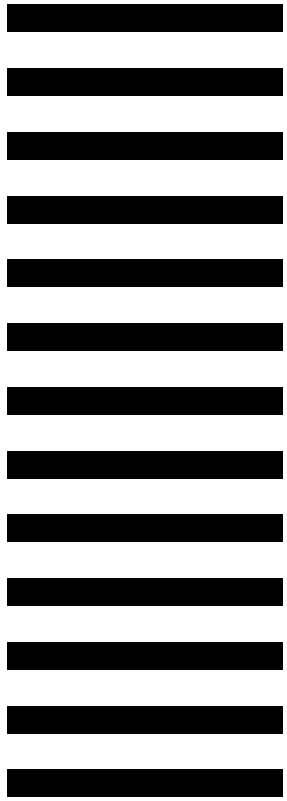
```
x = [      ], его квадрат -  [          ]
```

```
Console.WriteLine("x = {0,5}, его квадрат -  {1,10}", x, x*x );
```

Циклы

Повторить многократно:

Тело цикла



Цикл FOR

```
for( <инициализация счетчиков>; <условие>; <пересчет счетчиков> )
{
    <тело цикла>
}
```

Пример:

```
for( int i = 0; i < n; i++ )
{
    <тело цикла>
}
```

Пример: вычисление $10!=1 \cdot 2 \cdot \ldots \cdot 10$.

```
using System;
class Program
{
    static void Main()
    {









    }
}
```

Пример: вычисление $10!=1 \cdot 2 \cdot \ldots \cdot 10$.

```
using System;
class Program
{
    static void Main()
    {
        int nfact = 1;









    }
}
```

Пример: вычисление $10!=1 \cdot 2 \cdot \ldots \cdot 10$.

```
using System;
class Program
{
    static void Main()
    {
        int nfact = 1;
        for(int i = 1; i <= 10; i++)
        {
            nfact = nfact*i;
        }

    }
}
```

Пример: вычисление $10!=1 \cdot 2 \cdot \ldots \cdot 10$.

```
using System;
class Program
{
    static void Main()
    {
        int nfact = 1;
        for(int i = 1; i <= 10; i++)
        {
            nfact = nfact*i;
        }
        Console.WriteLine(nfact);
    }
}
```

Пример: просуммировать все четные числа от 2 до 100.

```
using System;
class Program
{
    static void Main()
    {









    }
}
```

Пример: просуммировать все четные числа от 2 до 100.

```
using System;
class Program
{
    static void Main()
    {
        int sum = 0;









    }
}
```

Пример: просуммировать все четные числа от 2 до 100.

```
using System;
class Program
{
    static void Main()
    {
        int sum = 0;
        for(int i = 2; i <= 100; i += 2 )
        {

        }

    }
}
```

Пример: просуммировать все четные числа от 2 до 100.

```
using System;
class Program
{
    static void Main()
    {
        int sum = 0;
        for(int i = 2; i <= 100; i += 2 )
        {
            sum += i;       // или sum = sum + i;
        }
        Console.WriteLine(sum);
    }
}
```

Несколько счетчиков.
Пример: распечатать (1,9), (2,8), . . . , (9,1).

```
using System;
class Program
{
    static void Main()
    {
        for (int i = 1, j = 9; i < 10 && j > 0; i++, j-- )
        {
            Console.WriteLine("({0},{1})", i, j);
        }
    }
}
```

Счетчики других типов.

Пример: вычислить $\sqrt{2}$ с точностью 0.0001.

Счетчики других типов.
Пример: вычислить $\sqrt{2}$ с точностью 0.0001.

```
using System;
class Program
{
    static void Main()
    {
        for (double x = 0; x*x < 2; x += 0.0001 )
        {
            Console.WriteLine(x);
        }
    }
}
```

Все вычисления делаются в заголовке, а не в теле цикла!

Циклы WHILE и DO . . . WHILE.

```
while( <условие> )
{
    <тело цикла>
}
```

WHILE есть часто встречающийся частный случай цикла FOR с пустыми разделами инициализации и пересчета счетчиков в заголовке:

```
for( ; <условие>; > ){ <тело цикла> }
```

Пример: подобрать наименьшее натуральное число n, удовлетворяющее условию $n^2 > n + 100$.

```
using System;
class Program
{
    static void Main()
    {









    }
}
```

Пример: подобрать наименьшее натуральное число n, удовлетворяющее условию $n^2 > n + 100$.

```
using System;
class Program
{
    static void Main()
    {
        int n =0;









    }
}
```

Пример: подобрать наименьшее натуральное число n, удовлетворяющее условию $n^2 > n + 100$.

```
using System;
class Program
{
    static void Main()
    {
        int n =0;
        while( n*n <= n+100 )
        {

        }

    }
}
```

Пример: подобрать наименьшее натуральное число n, удовлетворяющее условию $n^2 > n + 100$.

```
using System;
class Program
{
    static void Main()
    {
        int n =0;
        while( n*n <= n+100 )
        {
            n++;
        }

    }
}
```

Пример: подобрать наименьшее натуральное число n, удовлетворяющее условию $n^2 > n + 100$.

```
using System;
class Program
{
    static void Main()
    {
        int n =0;
        while( n*n <= n+100 )
        {
            n++;
        }
        Console.WriteLine(n);
    }
}
```

Цикл DO . . . WHILE.

```
do
{
    <тело цикла>
}
while( <условие> )
```

Отличается тем, что условие проверяется не до, а после каждой итерации.

Пример: попугай Кеша.

```
using System;
class Program
{
    static void Main()
    {









    }
}
```

Пример: попугай Кеша.

```
using System;
class Program
{
    static void Main()
    {
        string s;
        do
        {


        }
        while( s != "Кеша" )
        Console.WriteLine("Кеша хороший!");
    }
}
```

Пример: попугай Кеша.

```
using System;
class Program
{
    static void Main()
    {
        string s;
        do
        {
            Console.WriteLine("Угадай, как меня зовут?");
            s = Console.Readline();
        }
        while( s != "Кеша" )
        Console.WriteLine("Кеша хороший!");
    }
}
```

Вложенные циклы. Пример: распечатать всевозможные пары цифр — (0,0), (0,1), . . . , (9,9).

```
using System;
class Program
{
    static void Main()
    {









    }
}
```

Вложенные циклы. Пример: распечатать всевозможные пары цифр — (0,0), (0,1), . . . , (9,9).

```
using System;
class Program
{
    static void Main()
    {
        for( int i = 0; i < 10; i++)
        {









        }
    }
}
```

Вложенные циклы. Пример: распечатать всевозможные пары цифр — (0,0), (0,1), . . . , (9,9).

```
using System;
class Program
{
    static void Main()
    {
        for( int i = 0; i < 10; i++)
        {
            for( int j = 0; j < 10; j++)
            {

            }
        }
    }
}
```

Вложенные циклы. Пример: распечатать всевозможные пары цифр — (0,0), (0,1), . . . , (9,9).

```
using System;
class Program
{
    static void Main()
    {
        for( int i = 0; i < 10; i++)
        {
            for( int j = 0; j < 10; j++)
            {
                Console.WriteLine("({0},{1}) ", i, j);
            }
        }
    }
}
```

Пример: распечатать то же самое в виде таблицы.

```
using System;
class Program
{
    static void Main()
    {
        for( int i = 0; i < 10; i++)
        {
            for( int j = 0; j < 10; j++)
            {
                Console.Write("({0},{1}) ", i, j);
            }
            Console.WriteLine();
        }
    }
}
```

Что напечатает следующий код?

```
using System;
class Program
{
    static void Main()
    {
        for( int i = 0; i < 10; i++)
        {
            for( int j = 0; j < i+1; j++)
            {
                Console.Write("({0},{1}) ", i, j);
            }
            Console.WriteLine();
        }
    }
}
```

```
(0,0)
(1,0) (1,1)
(2,0) (2,1) (2,2)
(3,0) (3,1) (3,2) (3,3)
(4,0) (4,1) (4,2) (4,3) (4,4)
(5,0) (5,1) (5,2) (5,3) (5,4) (5,5)
(6,0) (6,1) (6,2) (6,3) (6,4) (6,5) (6,6)
(7,0) (7,1) (7,2) (7,3) (7,4) (7,5) (7,6) (7,7)
(8,0) (8,1) (8,2) (8,3) (8,4) (8,5) (8,6) (8,7) (8,8)
(9,0) (9,1) (9,2) (9,3) (9,4) (9,5) (9,6) (9,7) (9,8) (9,9)
```

Структурное ветвление

Оператор IF

```
if (<условие>)
{
    <операторы>
}
else
{
    <операторы>
}
```

Сокращенная форма:

```
if (<условие>) { <операторы> }
```

Пример распознавания того, какая клавиша нажата:

```
Console.Write("Enter a character: ");
char ch = (char)Console.Read();
```

Пример распознавания того, какая клавиша нажата:

```
Console.Write("Enter a character: ");
char ch = (char)Console.Read();
if (Char.IsLetter(ch))
{









}
```

Пример распознавания того, какая клавиша нажата:

```
Console.Write("Enter a character: ");
char ch = (char)Console.Read();
if (Char.IsLetter(ch))
{
    if (Char.IsLower(ch))
    {
        Console.WriteLine("The character is lowercase.");
    }




}
```

Пример распознавания того, какая клавиша нажата:

```
Console.Write("Enter a character: ");
char ch = (char)Console.Read();
if (Char.IsLetter(ch))
{
    if (Char.IsLower(ch))
    {
        Console.WriteLine("The character is lowercase.");
    }
    else
    {
        Console.WriteLine("The character is uppercase.");
    }
}
```

```
else
{
    Console.WriteLine("Not an alphabetic character.");
}
```

Пример: попугай Кеша.

```
static void Main()
{
    string s;
    do
    {
        Console.WriteLine("Угадай, как меня зовут?");
        s = Console.Readline();

    }
    while( s != "Кеша" )
    Console.WriteLine("Кеша хороший!");
}
```

Попугай Кеша отругивается:

```
static void Main()
{
    string s;
    do
    {
        Console.WriteLine("Угадай, как меня зовут?");
        s = Console.Readline();
        if ( s != "Кеша" ){ Console.WriteLine("Сам дурак!");
    }
    while( s != "Кеша" )
    Console.WriteLine("Кеша хороший!");
}
```

Пример. Решение квадратного уравнения $ax^2 + bx + c = 0$.

Оператор SWITCH Синтаксис:

```
switch (<переменная>)
    {
        case <значение1>: <операторы>; break;
        case <значение2>: <операторы>; break;
        ...
        default: <операторы>; break;  // необязательная часть
    }
```

```
static void Main()                    // Простейший калькулятор
        {   char ch; double x; Console.WriteLine("Ready!");









        }
```

```
static void Main()
        {   char ch; double x; Console.WriteLine("Ready!");
            double r = double.Parse(Console.ReadLine());









        }
```

```
static void Main()
        {   char ch; double x; Console.WriteLine("Ready!");
            double r = double.Parse(Console.ReadLine());
            do
            {









                    }
           while(ch != '=');

        }
```

```
static void Main()
        {   char ch; double x; Console.WriteLine("Ready!");
            double r = double.Parse(Console.ReadLine());
            do
            {   ch = char.Parse(Console.ReadLine());
                if (ch != '=')
                {









                }   }
           while(ch != '=');

        }
```

```
static void Main()
        {   char ch; double x; Console.WriteLine("Ready!");
            double r = double.Parse(Console.ReadLine());
            do
            {   ch = char.Parse(Console.ReadLine());
                if (ch != '=')
                {   x = double.Parse(Console.ReadLine());









                }   }
           while(ch != '=');

        }
```

```
static void Main()
        {   char ch; double x; Console.WriteLine("Ready!");
            double r = double.Parse(Console.ReadLine());
            do
            {   ch = char.Parse(Console.ReadLine());
                if (ch != '=')
                {   x = double.Parse(Console.ReadLine());
                    switch (ch)
                    {   case '+': r = r + x; break;
                        case '-': r = r - x; break;
                        case '*': r = r * x; break;
                        case '/': r = r / x; break;
           }    }   }
           while(ch != '=');

        }
```

```
static void Main()
        {   char ch; double x; Console.WriteLine("Ready!");
            double r = double.Parse(Console.ReadLine());
            do
            {   ch = char.Parse(Console.ReadLine());
                if (ch != '=')
                {   x = double.Parse(Console.ReadLine());
                    switch (ch)
                    {   case '+': r = r + x; break;
                        case '-': r = r - x; break;
                        case '*': r = r * x; break;
                        case '/': r = r / x; break;
            }   }   }
            while(ch != '=');
            Console.WriteLine("-------\n {0}", r);
        }
```

Массивы: `a[0],a[1],...,a[99]`

```
b[0,0],b[0,1],...,b[0,9]
b[1,0],b[1,1],...,b[1,9]
...
b[9,0],b[9,1],...,b[9,9]
```

Описание, создание заполнение одномерных массивов

```
int[ ] a;                    // описание
char[ ] b;
```

Описание, создание заполнение одномерных массивов

```
int[ ] a;                    // описание
char[ ] b;
int[ ] a1 = new int[5];  //описание и создание
```

Описание, создание заполнение одномерных массивов

```
int[ ] a;                       // описание
char[ ] b;
int[ ] a1 = new int[5];  //описание и создание
int[ ] a2 = new int[ ] { 1, 3, 5, 7, 9 };  // описание, создание
                                            // и заполнение
```

Описание, создание заполнение одномерных массивов

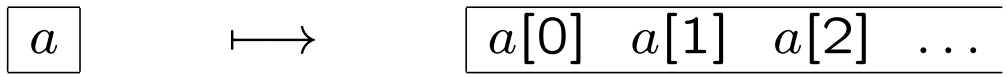
```
int[ ] a;                    // описание
char[ ] b;
int[ ] a1 = new int[5];  //описание и создание
int[ ] a2 = new int[ ] { 1, 3, 5, 7, 9 };  // описание, создание
                                           // и заполнение
    // Альтернативный синтаксис:
int[ ] array3 = { 1, 2, 3, 4, 5, 6 };
string[ ]  =  { "Мама", "мыла", "раму" }
```

Описание, создание заполнение одномерных массивов

```
int[ ] a;                    // описание
char[ ] b;
int[ ] a1 = new int[5];  //описание и создание
int[ ] a2 = new int[ ] { 1, 3, 5, 7, 9 };  // описание, создание
                                           // и заполнение
    // Альтернативный синтаксис:
int[ ] array3 = { 1, 2, 3, 4, 5, 6 };
string[ ]  =  { "Мама", "мыла", "раму" }

a = new int[3];  // создание описанного ранее массива
a[2] = 55;       // заполнение созданного ранее массива
```

Массивы — переменные-ссылки



Описание массива создает лишь хранилище (`a`) для ссылки.

new — создает хранилище для элементов массива и записывает его адрес в `a`.

Только после всего этого в тексте программы можно использовать выражения с индексами `a[...]`

Пример. Ввести 10 чисел и распечатать их в обратном порядке:

```
static void Main()
{
  int[ ] a = new int[10];









}
```

Пример. Ввести 10 чисел и распечатать их в обратном порядке:

```
static void Main()
{
  int[ ] a = new int[10];
  for(int i = 0; i < 10; i++)
  {
    a[i] = int.Parse(Console.ReadLine());
  }




}
```

Пример. Ввести 10 чисел и распечатать их в обратном порядке:

```
static void Main()
{
  int[ ] a = new int[10];
  for(int i = 0; i < 10; i++)
  {
    a[i] = int.Parse(Console.ReadLine());
  }
  for(int i = 9; i >= 0; i--)
  {
    Console.WriteLine(a[i]);
  }
}
```

Пример, проясняющий специфику переменных-ссылок:

```
{
  int[ ] ar1 = new int[ ] {1, 2, 3}
  int[ ] ar2 = ar1;
  ar2[0] = 55;
  Console.WritLine(ar1[0]);
}
```

Что будет напечатано, 1 или 55? Почему?

Многомерные массивы

```
int[,] b = new int[2, 3];

int[,] c = { { 1, 2, 3 }, { 4, 5, 6 } };

b[1, 2] = 25 + c[0, 2];
```

$\boxed{b} \longmapsto \boxed{\begin{array}{lll} b[0,0] & b[0,1] & b[0,2] \\ b[1,0] & b[1,1] & b[1,2] \end{array}}$

$\boxed{c} \longmapsto \boxed{\begin{array}{lll} c[0,0]=1 & c[0,1]=2 & c[0,2]=3 \\ c[1,0]=4 & c[1,1]=5 & c[1,2]=6 \end{array}}$

Заполнение двухмерного массива с клавиатуры:

```
int[ ] a = new a[2,3];
Console.WriteLine("Введите 6 чисел:");
```

Заполнение двухмерного массива с клавиатуры:

```
int[ ] a = new a[2,3];
Console.WriteLine("Введите 6 чисел:");
for(int i=0; i<2; i++)
{




}
```

Заполнение двухмерного массива с клавиатуры:

```
int[ ] a = new a[2,3];
Console.WriteLine("Введите 6 чисел:");
for(int i=0; i<2; i++)
{
    for(int j=0; j<3; j++)
    {

    }
}
```

Заполнение двухмерного массива с клавиатуры:

```
int[ ] a = new a[2,3];
Console.WriteLine("Введите 6 чисел:");
for(int i=0; i<2; i++)
{
    for(int j=0; j<3; j++)
    {
        a[i,j] = int.Parse(Console.ReadLine() );
    }
}
```

Вывод двухмерного массива в виде таблицы:

```
for(int i=0; i<2; i++)
{
    for(int j=0; j<3; j++)
    {
        Console.Write(" {0,5} ",a[i,j]);
    }
    Console.WriteLine();
}
```

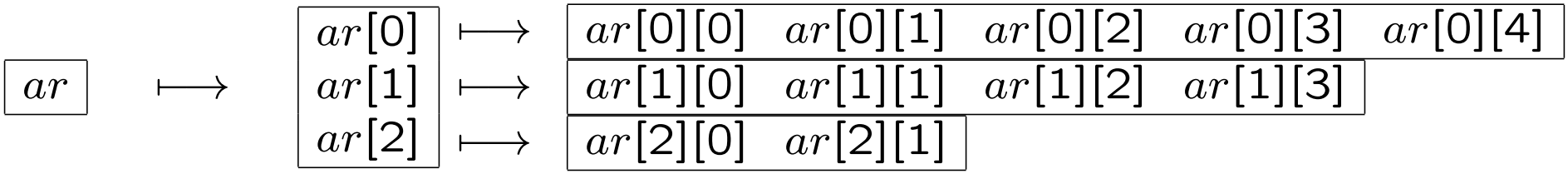
Массивы массивов — длины одномерных массивов могут быть разными:

```
int[ ][ ] ar = new int[3][ ];

ar[0] = new int[ ] { 1, 3, 5, 7, 9 };
ar[1] = new int[ ] { 0, 2, 4, 6 };
ar[2] = new int[ ] { 11, 22 };

ar[2][1] += 10;
```

Массив массивов, организация хранения



Цикл foreach — заготовленная модификация цикла FOR для циклической обработки всех элементов массива.

```
string[] s = {"Мама", "мыла", "раму" };
foreach(string w in s)
{
    Console.WritLine(w);
}
```

Длина массива:

```
string[] s = {"Мама", "мыла", "раму" };

int n = s.Length;     // n = 3
```

Строки символов (string) — похожи на массивы.

Обращение к буквам: `s[i]`, где `i = 0,1,..., s.Length - 1`,
но нельзя `s[...]=... ;`

Строки символов (string) — похожи на массивы.

Обращение к буквам: `s[i]`, где `i = 0,1,..., s.Length - 1`,
но нельзя `s[...]=... ;`

Основная операция на строках: `s1 + s2`.

Строки символов (string) — похожи на массивы.

Обращение к буквам: `s[i]`, где `i = 0,1,..., s.Length - 1`,
но нельзя `s[...]=... ;`

Основная операция на строках: `s1 + s2`.

У переменных других типов есть `ToString()` — метод преобразовать их значения в “изображения”.

```
int n = 2007;
string s = "7 февраля "+n.ToString()+" года";
```

Дополнительные средства: выделение подстроки и замены одной подстроки на другую:

```
string s1 = "orange";
string s2 = s1.Substring(2,4);        // s2 <- "ange"
string s3 = s1.Replace("an","XYZ");   // s3 <- "orXYZge"
```

Все они устроены так, чтобы исходная строка оставалась неизменной, а результат записывался в новую переменную.

Дополнительные средства: поиск подстроки (-1 означает “не нашел”)

```
string s = "orange";
int n = s.IndexOf("an");   // n <- 2
```

Дополнительные средства: поиск подстроки (-1 означает “не нашел”)

```
string s = "orange";
int n = s.IndexOf("an");   // n <- 2
```

Сравнение строк с данной по порядку ("abc"< "ABC"< "abcd"<"bb" возвращает 1, 0, -1):

```
string s = "ABC";
Console.WriteLine(s.CompareTo("abc"));   //   1
Console.WriteLine(s.CompareTo("ABC"));   //   0
Console.WriteLine(s.CompareTo("abcd"));  //  -1
Console.WriteLine(s.CompareTo("bb"));    //  -1
```

Split — метод, который разрезает фразу на слова и создает соответствующий массив слов. Ему на вход следует подать массив букв-разделителей:

```
char[] delimit = new char[] { ' ', '.','!','?' };

string s = "Hello! What is your name? My name is Nick.";
```

Split — метод, который разрезает фразу на слова и создает соответствующий массив слов. Ему на вход следует подать массив букв-разделителей:

```
char[] delimit = new char[] { ' ', '.','!','?' };

string s = "Hello! What is your name? My name is Nick.";

string[] words = s.Split(delimit);
foreach(string w in words)
{
     Console.WriteLine(w);
}
```

Подпрограммы как методы

Пример: вычислить, в каком из двух слов буква 'z' встречается чаще.

Выделение подзадачи:

- вызывающая программа (метод Main) выбирает слово s и передает его подпрограмме;

- подпрограмма (метод CountZ) должна подсчитать, сколько раз буква 'z' входит в s, и вернуть это число вызывающей программе;

- и так два раза, после чего Main печатает ответ.

Схема программы:

```
using System;
class Program
{
    static int CountZ(string s)
    {

    }
    static void Main()
    {   ...
        int n1 = CountZ(s1);
        ...
        int n2 = CountZ(s2);
        ...
    }
}
```

```
using System;
class Program
{
 ...
 static void Main()
 {
   string s1 = Console.ReadLine();









 }
}
```

```
using System;
class Program
{
 ...
 static void Main()
 {
   string s1 = Console.ReadLine();
   int n1 = CountZ(s1);              //первый вызов









 }
}
```

```
using System;
class Program
{
 ...
 static void Main()
 {
   string s1 = Console.ReadLine();
   int n1 = CountZ(s1);              //первый вызов

   string s2 = Console.ReadLine();
   int n2 = CountZ(s2);              //второй вызов




 }
}
```

```
using System;
class Program
{
 ...
 static void Main()
 {
   string s1 = Console.ReadLine();
   int n1 = CountZ(s1);              //первый вызов

   string s2 = Console.ReadLine();
   int n2 = CountZ(s2);              //второй вызов

   if (n1 > n2) { Console.WriteLine("В первом слове букв z больше.")
   if (n1 == n2) { Console.WriteLine("Букв z поровну."); }
   if (n1 < n2) { Console.WriteLine("Во втором слове букв z больше."
 }
}
```

```
using System;
class Program
{
    static int CountZ(string s) // аргумент s -- формальный параметр
    {









    }
    static void Main() ...
}
```

```
using System;
class Program
{
    static int CountZ(string s) // аргумент s -- формальный параметр
    {
        int n = 0;









    }
    static void Main() ...
}
```

```
using System;
class Program
{
    static int CountZ(string s) // аргумент s -- формальный параметр
    {
        int n = 0;
        for (int i = 0; i < s.Length ; i++ )
        {
            if (s[i] == 'z') { n++; };
        }


    }
    static void Main() ...
}
```

```
using System;
class Program
{
    static int CountZ(string s) // аргумент s -- формальный параметр
    {
        int n = 0;
        for (int i = 0; i < s.Length ; i++ )
        {
            if (s[i] == 'z') { n++; };
        }
        return n;  // возврат результата

    }
    static void Main() ...
}
```

```
using System;
class Program
{
    static int CountZ(string s) // аргумент s -- формальный параметр
    {
        int n = 0;
        for (int i = 0; i < s.Length ; i++ )
        {
            if (s[i] == 'z') { n++; };
        }
        return n;  // возврат результата
        // Переменные s, n, i - локальные
    }
    static void Main() ...
}
```

Пример: вычислить, в каком из двух слов буква 'z' встречается чаще.

Другой вариант подзадачи:

- вызывающая программа (метод Main) ничего не передает подпрограмме, а только вызывает ее;

- подпрограмма (метод ReadAndCount) сама спрашивает слово s у пользователя, считает, сколько раз буква 'z' входит в s, и возвращает это число вызывающей программе;

- и так два раза, после чего Main печатает ответ.

Схема программы:

```
using System;
class Program
{
    static int ReadAndCount()
    {

    }
    static void Main()
    {   ...
        int n1 = ReadAndCount();
        ...
        int n2 = ReadAndCount();
        ...
    }
}
```

```
using System;
class Program
{
 ...
 static void Main()
 {
   int n1 = ReadAndCount();              //первый вызов

   int n2 = ReadAndCount();              //второй вызов

   if (n1 > n2) { Console.WriteLine("В первом слове букв z больше.")
   if (n1 == n2) { Console.WriteLine("Букв z поровну."); }
   if (n1 < n2) { Console.WriteLine("Во втором слове букв z больше.")
 }
}
```

```
using System;
class Program
{
    static int ReadAndCount()
    {
        string s = Console.ReadLine();









    }
    static void Main() ...
}
```

```
using System;
class Program
{
    static int ReadAndCount()
    {
        string s = Console.ReadLine();
        int n = 0;
        for (int i = 0; i < s.Length ; i++ )
        {
            if (s[i] == 'z') { n++; };
        }
        return n;  // возврат результата
    }
    static void Main() ...
}
```

Пример: вычислить, в каком из двух слов буква 'z' встречается чаще.

Две подзадачи:

- метод ReadAndCount

- метод PrintRes

Схема программы:

```
using System;
class Program
{
    static int ReadAndCount()
    {
    }
    static void PrintRes(int n1, int n2) // тип результата void,
    {                                    //  нет return
    }

    static void Main()
    {
    }
}
```

Main — только управляет:

```
using System;
class Program
{
    ...
    static void Main()
    {
        int n1 = ReadAndCount();            //первый вызов
        int n2 = ReadAndCount();            //второй вызов
        PrintRes(n1,n2);            //печать
    }
}
```

```
using System;
class Program
{
 static int ReadAndCount() ...

 static void PrintRes(int n1, int n2) // тип результата void, нет ret
 {
   if (n1 > n2) { Console.WriteLine("В первом слове букв z больше.")
   if (n1 == n2) { Console.WriteLine("Букв z поровну."); }
   if (n1 < n2) { Console.WriteLine("Во втором слове букв z больше.")
   Console.WriteLine("-------------------------");
 }

 static void Main() ...
}
```

Main можно переписать еще короче:

```
static void Main();
{
    PrintRes(ReadAndCount(), ReadAndCount());
}
```

Но реально никакой экономии не будет.

Выдача результата через формальные параметры:

```
static void ReadAndCount(out int n)
{
    string s = Console.ReadLine();
    n = 0;
    for (int i = 0; i < s.Length ; i++ )
    {
        if (s[i] == 'z') { n++; };
    }
}
```

Не требует `return`.

Вызов не требует присваивания:

```
...
ReadAndCount(out n1);
...
ReadAndCount(out n2);
...
```

В вызовах `n1, n2` — переменные, а не выражения!

Этот способ позволяет выдавать сразу несколько результатов, а также может быть совмещен с обычным — через значение функции. Например, функция

```
static int f(out bool b, out s string, int x)
{
    ...
    return ...;
}
```

возвращает свое значение и содержимое переменных `b, s`.

```
using System;
class Program
    {
        static int f(out bool b, out string s, int x)
        {   b = false;
            s = "!!!";
            return x;
        }









    }
```

```
using System;
class Program
    {
        static int f(out bool b, out string s, int x)
        {   b = false;
            s = "!!!";
            return x;
        }
        static void Main()
        {   bool bb; string ss; int y = 5;
            int n = f(out bb, out ss, 2 + 3 * y);
            Console.WriteLine(" {0} {1} {2}", bb, ss, n);
        }
        //  вернет False  !!! 17
    }
```

Классы

Классы — основной вид описаний, задают одновременно:

- тип (ссылочный),
- схему (образец) всех объектов, населяющих этот тип.

```
class MyClass
{
    //Members - все, чем должны обладать объекты этого класса
}
```

Это — “шаблон для штамповки однотипных роботов”.
Сам класс не выполняет никаких действий — все делают созданные по этому шаблону объекты.

Основные вопросы:

• Как проектировать?
Состав и устройство членов класса (members), т.е. что может быть пунктом проекта.

• Как создавать объекты?
Это делает оператор new <конструктор класса>.

• Как управлять всем этим?

Простейший состав класса: поля и методы.

Простейший состав класса: поля и методы.

• Поля — переменные, т.е. хранилище данных каждого объекта данного класса (багажник робота-исполнителя).

Простейший состав класса: поля и методы.

• Поля — переменные, т.е. хранилище данных каждого объекта данного класса (багажник робота-исполнителя).

• Методы — это подпрограммы-функции, которыми объект обладает (инструменты робота).

Простейший состав класса: поля и методы.

• Поля — переменные, т.е. хранилище данных каждого объекта данного класса (багажник робота-исполнителя).

• Методы — это подпрограммы-функции, которыми объект обладает (инструменты робота).

Управление багажником: в теле метода можно прямо обращаться к полям.

Простейший состав класса: поля и методы.

• Поля — переменные, т.е. хранилище данных каждого объекта данного класса (багажник робота-исполнителя).
• Методы — это подпрограммы-функции, которыми объект обладает (инструменты робота).

Управление багажником: в теле метода можно прямо обращаться к полям.

Но через формальные параметры методу можно также передавать любые данные извне.

Простейший состав класса: поля и методы.

• Поля — переменные, т.е. хранилище данных каждого объекта данного класса (багажник робота-исполнителя).
• Методы — это подпрограммы-функции, которыми объект обладает (инструменты робота).

Управление багажником: в теле метода можно прямо обращаться к полям.

Но через формальные параметры методу можно также передавать любые данные извне.

Пример: класс сумматоров.

```
class Summator
{
    int sum = 0;      // поле, хранит сумму

    void Add(int x)  // метод, добавляет слагаемое
    {
        sum += x;
    }
}
```

Пример: класс сумматоров (защищен до неуправляемости).

```
class Summator
{
    int sum = 0;      // поле, хранит сумму

    void Add(int x)  // метод, добавляет слагаемое
    {
        sum += x;
    }
}
```

Не будет работать потому, что все содержимое класса по умолчанию имеет модификатор private.
Следует явно указывать public, где надо открыть доступ извне.

```
using System;
class Summator
{   public int sum = 0;
    public void Add(int x)
    {
        sum += x;
    }
}
```

```
using System;
class Summator
{   public int sum = 0;
    public void Add(int x)
    {
        sum += x;
    }
}
class Program
{
    static void Main()
    {



    }
}
```

```
using System;
class Summator
{   public int sum = 0;
    public void Add(int x)
    {
        sum += x;
    }
}
class Program
{
    static void Main()
    {
        Summator sm = new Summator();  //создание объекта sm


    }
}
```

```
using System;
class Summator
{   public int sum = 0;
    public void Add(int x)
    {
        sum += x;
    }
}
class Program
{
    static void Main()
    {
        Summator sm = new Summator();  //создание объекта sm
        for ( int i = 1; i < 100; i++) { sm.Add(i) }

    }
}
```

```
using System;
class Summator
{   public int sum = 0;
    public void Add(int x)
    {
        sum += x;
    }
}
class Program
{
    static void Main()
    {
        Summator sm = new Summator();  //создание объекта sm
        for ( int i = 1; i < 100; i++) { sm.Add(i) }
        Console.WriteLine(sm.sum);
    }
}
```

Тонкости: `static` — разместить “на базе”.

```
using System;
class Summator
{   public static int sum = 0;
    public void Add(int x) { sum += x;}
}
```

Тонкости: `static` — разместить “на базе”.

```
using System;
class Summator
{   public static int sum = 0;
    public void Add(int x) { sum += x;}
}
class Program
{
    static void Main()
    {




    }
}
```

Тонкости: `static` — разместить “на базе”.

```
using System;
class Summator
{   public static int sum = 0;
    public void Add(int x) { sum += x;}
}
class Program
{
    static void Main()
    {   Summator s1 = new Summator();
        Summator s2 = new Summator();



    }
}
```

Тонкости: `static` — разместить “на базе”.

```
using System;
class Summator
{   public static int sum = 0;
    public void Add(int x) { sum += x;}
}
class Program
{
    static void Main()
    {   Summator s1 = new Summator();
        Summator s2 = new Summator();
        s1.Add(3);
        s2.Add(2);

    }
}
```

Тонкости: `static` — разместить “на базе”.

```
using System;
class Summator
{   public static int sum = 0;
    public void Add(int x) { sum += x;}
}
class Program
{
    static void Main()
    {   Summator s1 = new Summator();
        Summator s2 = new Summator();
        s1.Add(3);
        s2.Add(2);
        Console.WriteLine(Summator.sum); // 5
    }
}
```

Тонкости: программирование конструкторов для new.

```
class Summator
{
    public int sum = 0;
    public Summator()           // конструктор по умолчанию
    {
        sum = 0;
    }
    public Summator(int n)      // дополнительный конструктор
    {
        sum = n;
    }
    public void Add(int x) { sum += x; }
}
```

Теперь создавать новые объекты класса `Summator` можно двумя способами:

```
Summator sm1 = new Summator();
Summator sm2 = new Summator(25);
```

Сумматор `sm2` начнет свою работу, уже имея число 25 в поле `sum`.

Пример: игра “чёт-нечет”.

Капитал игрока — 10 единиц, у компьютера — бесконечный.

Ставка в игре — 1.

Игрок выбирает 0 или 1, компьютер — тоже, только случайным образом. Если они выбрали одно и то же, то выигрывает пользователь, иначе — компьютер. Игра продолжается до разорения игрока.

(см. проект “Чет-нечет”)

Свойства — это члены класса, являющиеся программной имитацией полей (т.е. обычных переменных).

• Свойству можно как присвоить значение, так и прочитать уже имеющееся.

• Но эти действия лишь имитируются вызовами двух методов `set` и `get`, которые программируются при разработке класса. Что, где и как хранится на самом деле — знают только они.

```
class MyClass
{   ...
    public double MyProperty   // имитирует поле типа double
    {
        get
        {   // тут программируют вычисление текущего значения
            return <значение>;
        }
        set
        {   // что сделать, когда требуют присвоить значение
            // ( MyProperty = value)
            // value - стандартное обозначение для правой
            // части оператора присваивания
        }
    }
    ...
}
```

```
class MyClass                          // get и set могут быть какими угод
{   ...
    double mysum;   // поле private
    double otkat;   // поле private
    public double MySum
    {   get
        {
            return mysum;
        }
        set
        {
            mysum = 0.9 * value;
            otkat = 0.1 * value;
        }
    }
    ...
}
```

Извне класса непосредственный доступ к реальному хранилищу `mysum` закрыт (поле private), но его можно посмотреть и изменить через свойство `MySum`.

```
MyClass t = new MyClass();
...
t.MySum = 10.0;
Console.WriteLine("На счете {0}", t.MySum);   //  9,0  !!!
```

Сделать поле доступным только для чтения:

1. Заводим свойство `Sum` для доступа к private полю `sum`.
2. Если метод `set` не задан, то присвоить свойству `Sum` какое-либо значение невозможно. А других способов извне воздействовать на поле `sum` не программируем.

```
class MyClass
{
    ...
    double sum;              // поле private
    public double Sum
    {
            get { return sum; }

    }
    ...
}
```

Свойства для представления вычисляемых значений.

```
class Rectangle
{
    double w;  // хранится, private
    double h;  // хранится, private
```

Свойства для представления вычисляемых значений.

```
class Rectangle
{
    double w;  // хранится, private
    double h;  // хранится, private
    public double Square   // значение не хранится, а вычисляется
    {









    }
```

Свойства для представления вычисляемых значений.

```
class Rectangle
{
    double w;  // хранится, private
    double h;  // хранится, private
    public double Square   // значение не хранится, а вычисляется
    {
            get { return w*h; }




    }
```

Свойства для представления вычисляемых значений.

```
class Rectangle
{
    double w;  // хранится, private
    double h;  // хранится, private
    public double Square   // значение не хранится, а вычисляется
    {
            get { return w*h; }
            set
            {
                w = Math.Sqrt(value);
                h = w;
            }
    }
```

// продолжение описания класса Rectangle:

```
    public double Width         // для единообразия
    {
            get { return w; }
            set { w = value; }

    }




}
```

// продолжение описания класса Rectangle:

```
    public double Width         // для единообразия
    {
            get { return w; }
            set { w = value; }

    }
    public double Height        // для единообразия
    {
            get { return h; }
            set { h = value; }
    }
}
```

Как использовать тип Rectangle (см. проект Rectangle):

```
class Program
{
    static void Main()
    {   Rectangle q1 = new Rectangle();
        Rectangle q2 = new Rectangle();
        q1.Height = 2.0;
        q1.Width = 3.0;
        Console.WriteLine(q1.Width);    // 3.0
        Console.WriteLine(q1.Square);   // 6.0
        q1.Height += 2.0;
        Console.WriteLine(q1.Square);   // 12.0
        q2.Square = 12.0;
        Console.WriteLine(q2.Height);   // 3.4...
    }
}
```

Классы с параметрами (Generics).

В описании класса можно использовать вместо фиксированного имени типа (`int`, `string`, `...`) типовую переменную (`T`). Получается схема похожих классов, отличающихся лишь значениями этой переменной T.

Классы с параметрами (Generics).

В описании класса можно использовать вместо фиксированного имени типа (`int`, `string`, `...`) типовую переменную (`T`). Получается схема похожих классов, отличающихся лишь значениями этой переменной T.

Пример: в описании класса Array определяется тип `T[]` — массивы элементов типа `T`. Но когда создается конкретный массив, мы задаем значение этой переменной:

```
int[ ] a = new int[10];
string[ ] b = new string[ ] {"aaa", "bbbb", "cc"}
```

Синтаксис описания классов с параметрами:

```
class MyClass<T>
{
    // В описаниях членов можно использовать тип T
}
```

Всюду вне класса вместо T надо подставлять конкретные типы:

```
class Program
{
    static void Main()
    {
        ...
        MyClass<int> a = new MyClass<int>();
        ...
        MyClass<char> b = new MyClass<char>();
        ...
    }
}
```

Пример: класс односвязных списков.

Списки элементов типа T строятся из пустого списка с помощью операции “добавления головы”:

• `null` — список (пустой);

• если l — список, a — элемент типа T, то $l; a$ — новый список. Элемент a наз. головой, а список l — хвостом полученного списка $l; a$.

Основные операции с односвязными списками:

`AddHead(a,l), Head(), Tail().`

(см. проект Mylist)

```
class MyList<T>
{
    T data;                 // голова
    MyList<T> next;         // хвост
```

```
class MyList<T>
{
    T data;
    MyList<T> next;

    public static MyList<T> AddHead(T h, MyList<T> t)
    {




    }
```

```
class MyList<T>
{
    T data;
    MyList<T> next;

    public static MyList<T> AddHead(T h, MyList<T> t)
    {
        MyList<T> tt = new MyList<T>();    // создаем список,



    }
```

```
class MyList<T>
{
    T data;
    MyList<T> next;

    public static MyList<T> AddHead(T h, MyList<T> t)
    {
        MyList<T> tt = new MyList<T>();
        tt.data = h;                          // заполняем голову
        tt.next = t;                          // и хвост

    }
```

```
class MyList<T>
{
    T data;
    MyList<T> next;

    public static MyList<T> AddHead(T h, MyList<T> t)
    {
        MyList<T> tt = new MyList<T>();
        tt.data = h;
        tt.next = t;
        return tt;
    }
```

```
public T Head()
{
    return data;
}
```

```
public T Head()
{
    return data;
}

public MyList<T> Tail()
{
    return next;
}
```

```
public static void PrL(MyList<T> l)  // печать списка
{




}
```

```
public static void PrL(MyList<T> l)
{
    if(l != null)
    {


    }
}
```

```
public static void PrL(MyList<T> l)
{
    if(l != null)
    {
        Console.WriteLine(l.data.ToString());

    }
}
```

```
public static void PrL(MyList<T> l)
{
    if(l != null)
    {
        Console.WriteLine(l.data.ToString());
        MyList<T>.PrL(l.next);
    }
}
```

```
public static void PrL(MyList<T> l)
{
    if(l != null)
    {
        Console.WriteLine(l.data.ToString());
        MyList<T>.PrL(l.next);
    }
}
public static void SetNull(out MyList<T> l)   // опустошение
{

}
```

```
    public static void PrL(MyList<T> l)
    {
        if(l != null)
        {
            Console.WriteLine(l.data.ToString());
            MyList<T>.PrL(l.next);
        }
    }
    public static void SetNull(out MyList<T> l)
    {
            l = null;
    }
}
```

Стандартный класс List<T> обладает гораздо большими возможностями. См. проект generic_list и numerolog.

Стандартный класс Queue<T> — проект peremeshivanie.

Использование классов — проекты Fractions и lemmings (наследование).

Управление консолью

```
Console.SetCursorPosition(x,y);                    //положение курсора
```

```
Console.SetCursorPosition(x,y);              //положение курсора

Console.ForegroundColor = ConsoleColor.Blue; //цвет
Console.BackgroundColor = ConsoleColor.White;
```

```
Console.SetCursorPosition(x,y);              //положение курсора

Console.ForegroundColor = ConsoleColor.Blue; //цвет
Console.BackgroundColor = ConsoleColor.White;

if (Console.KeyAvailable)    //проверка нажатия клавиши
{




}
```

```
Console.SetCursorPosition(x,y);              //положение курсора

Console.ForegroundColor = ConsoleColor.Blue; //цвет
Console.BackgroundColor = ConsoleColor.White;

if (Console.KeyAvailable)    //проверка нажатия клавиши
{  ConsoleKeyInfo keyInfo = Console.ReadKey(); // чтение клавиши









}
```

```
Console.SetCursorPosition(x,y);              //положение курсора

Console.ForegroundColor = ConsoleColor.Blue; //цвет
Console.BackgroundColor = ConsoleColor.White;

if (Console.KeyAvailable)    //проверка нажатия клавиши
{  ConsoleKeyInfo keyInfo = Console.ReadKey(); // чтение клавиши
   switch (keyInfo.Key)
   {case ConsoleKey.LeftArrow:  ... break;  // реакция на нажатие
    case ConsoleKey.RightArrow: ... break;
    case ConsoleKey.UpArrow:    ... break;
    case ConsoleKey.DownArrow:  ... break;
    case ConsoleKey.Spacebar:   ... break;
   }
}
```

generic list

```
using System;
using System.Collections.Generic;

public class Example {
 public static void Main()
 {
   List<string> li = new List<string>();

   li.Add("Мама");
   li.Add("мыла");
   li.Add("раму");

   foreach (string s in li)
   {
     Console.WriteLine(s);
   }
   Console.WriteLine("\nЭлементов: {0}", li.Count);
   Console.WriteLine("Их индексация - в порядке добавления:");
   for (int i = 0; i < 3; i++)
   {
```

```
  Console.WriteLine("li[{0}]= \"{1}\"", i, li[i]);
}

Console.WriteLine("\nДобавим еще одно слово \"мылом\":");
li.Add("мылом");
foreach (string s in li){Console.WriteLine(s);}
Console.WriteLine("\nЭлементов: {0}", li.Count);

Console.WriteLine("\nВставим слово \"чисто\" под номером 1:");
li.Insert(1,"чисто");
foreach (string s in li) { Console.WriteLine(s); }
Console.WriteLine("\nЭлементов: {0}", li.Count);

Console.WriteLine("\nУдалим слово \"чисто\":");
li.Remove("чисто");
foreach (string s in li) { Console.WriteLine(s); }
Console.WriteLine("\nЭлементов: {0}", li.Count);

Console.WriteLine("\nУдалим слово под номером 3:");
li.RemoveAt(3);
foreach (string s in li) { Console.WriteLine(s); }
```

```
   Console.WriteLine("\nЭлементов: {0}", li.Count);

   Console.WriteLine("\nДобавим для нарушения порядка и отсортируем:");
   li.Add("окна");
   li.Sort();
   foreach (string s in li) { Console.WriteLine(s); }

   Console.WriteLine("\nНайдем слово \"окна\" в списке, потом удалим:");
   int n = li.IndexOf("окна");
   Console.WriteLine(n); li.RemoveAt(n);
   foreach (string s in li) { Console.WriteLine(s); }
   Console.WriteLine("\nЭлементов: {0}", li.Count);
 }
}
```

numerolog

```
/* Нумерология: 2011 - простое и есть сумма простого (11)
 * количества последовательных простых чисел
 * 157 + ... + 211 = 2011. Проверить.
 */

using System;
using System.Collections.Generic;

public class Example // Решето Эратосфена. Сумма простых 156<x<212.
{
    public static void Main()
    {
        List<int> p = new List<int>();
        List<int> q = new List<int>();
        for (int i = 2; i < 2012; i++)
        {
            q.Add(i);
        }
        while (q.Count > 0)
        {
```

```
    int n=q[0];
    p.Add(n);

    for (int i = 0; i < q.Count; i++ )
    {
        if (q[i] % n == 0) { q.RemoveAt(i); }
        // есть тонкость с изменением индексации!!!
        // два числа подряд не могут делиться на n
    }
    /*
    int i=0;
    while (i < q.Count)
    {
        if (q[i] % n == 0) { q.RemoveAt(i); } else { i++; }
    }
    */
}

Console.WriteLine("\n------------{0}", p.Count);
int s = 0;
foreach (int i in p)
```

```
        {
            Console.Write("{0} ", i);
            if ((i > 156)&&(i<212)) { s += i; }
        }
        Console.WriteLine("\n s={0}",s);
    }
}
```

peremeshivanie

```
// Перемешивание колоды карт многократной раздачей на 3 руки.
using System;
using System.Collections.Generic;


class Program {
   static void Main()
   {
       List<int> l = new List<int>();   // колода
       for (int i = 0; i < 36; i++) { l.Add(i); }

       List<int>[] h = new List<int>[3]; // 3 руки
       for (int i = 0; i < 3; i++)
       {
           h[i] = new List<int>();       // создаются пустыми
       }

       int k;
       for (int j = 0; j < 100; j++)
```

```
{
    // Раздаем колоду:
    k = 0;
    // Random r = new Random();
    // k = r.Next(3);
    while (l.Count > 0)
    {
        h[k].Add(l[l.Count - 1]);
        l.RemoveAt(l.Count - 1);
        k = (k + 1) % 3;
       // k = r.Next(3);
    }

    // Собираем колоду
    for (int i = 0; i < 3; i++)
    {
        l.AddRange(h[i]);   // добавляется весь список
        h[i].Clear();       // очищаются руки
    }
}
// Что получилось:
```

```
foreach (int i in l) { Console.Write("{0}, ",i); }
Console.WriteLine("\n----------------------");


// Игра в "пьяницу":
Queue<int>[] a = new Queue<int>[2]; // 2 игрока
a[0] = new Queue<int>();
a[1] = new Queue<int>();

k = 0;               // раздача
while (l.Count > 0)
{
    a[k].Enqueue(l[l.Count - 1]);
    l.RemoveAt(l.Count - 1);
    k = (k + 1) % 2;
}
//Распечатка
Console.WriteLine();
foreach (int i in a[0]) { Console.Write("{0,2} ", i); }
Console.WriteLine();
foreach (int i in a[1]) { Console.Write("{0,2} ", i); }
```

```
Console.WriteLine();


//Розыгрыш
while (a[0].Count > 0 && a[1].Count > 0)
{
    int u = a[0].Dequeue();
    int v = a[1].Dequeue();
    if (u > v)
    {
        a[0].Enqueue(u); //сначала свою карту, старшую!
        a[0].Enqueue(v);
        Console.WriteLine("{0,2}>{1,2} => I", u, v);
        //Console.ReadLine();
    }
    else
    {
        a[1].Enqueue(v); //сначала свою карту, старшую!
        a[1].Enqueue(u);
        Console.WriteLine("{0,2}<{1,2} => II", u, v);
    }
```

```
 Console.ReadLine();
}

if (a[0].Count > 0) //Объявление победы
{
    Console.WriteLine("Победил I");
}
else
{
    Console.WriteLine("Победил II");
}


/*
// Рассказать о хэштаблицах:
// Dictionary (hashtable)
Dictionary<string, string> d = new Dictionary<string, string>();
d.Add("стол", "table");
d.Add("стул", "chair");
d.Add("кровать", "bed");
string w,s;
```

```
        w = "кровать";
        //w = "табуретка";
        if (d.TryGetValue(w, out s))
        {
            Console.WriteLine("{0} - {1}",w,s);
        }
        else
        {
            Console.WriteLine("Слова \"{0}\" нет в словаре",w);
        }
        */
    }
}
```

Fractions

```
// Новый тип Fraction (дроби) - новый класс.

// 1. Создание, доступ, равенство.
/* using System;

namespace Fractions {
    class Fraction
    {
        int p;   // числитель, private
        public int P { get { return p; } }
        int q;   // знаменатель, private
        public int Q { get { return q; } }


        //конструктор -- создает дробь,
        //не дает обратить знаменатель в 0.
        public Fraction(int a, int b)
        {
            if (b != 0) { p = a; q = b; }
            else { p = 0; q = 1; }
```

```
        }

        // проверка дробей на равенство
        public static bool Eq(Fraction fr1, Fraction fr2)
        {
            return fr1.P * fr2.Q == fr1.Q * fr2.P;
        }

    }

    class Program
    {
        static void Main() // для отладки
        { //сделать 2 дроби и проверить их равенство,
          //распечатать.

        }
    }
} */

// 2. Добавить ToString() для удобства печати.
```

```
/* using System;

namespace Fractions {
    class Fraction
    {
        int p;   // числитель, private
        public int P { get { return p; } }
        int q;   // знаменатель, private
        public int Q { get { return q; } }


        //конструктор -- создает дробь,
        //не дает обратить знаменатель в 0.
        public Fraction(int a, int b)
        {
            if (b != 0) { p = a; q = b; }
            else { p = 0; q = 1; }
        }

        // проверка дробей на равенство
        public static bool Eq(Fraction fr1, Fraction fr2)
```

```
        {
            return fr1.P * fr2.Q == fr1.Q * fr2.P;
        }

        // подмена (override)  стандартного ToString()
        public override string ToString()
        {
            return '(' + P.ToString() + '/' + Q.ToString() + ')';
        }

    }

    class Program
    {
        static void Main() // для отладки
        { //сделать дробь и распечатать


        }
    }
} */
```

```
// 3. Научить сокращать дроби. Понадобится НОД.
/* using System;

namespace Fractions {
    class Fraction
    {
        int p;   // числитель, private
        public int P { get { return p; } }
        int q;   // знаменатель, private
        public int Q { get { return q; } }


        //конструктор -- создает дробь,
        //не дает обратить знаменатель в 0.
        public Fraction(int a, int b)
        {
            if (b != 0) { p = a; q = b; }
            else { p = 0; q = 1; }
        }
```

```
// проверка дробей на равенство
public static bool Eq(Fraction fr1, Fraction fr2)
{
    return fr1.P * fr2.Q == fr1.Q * fr2.P;
}

// подмена (override)  стандартного ToString()
public override string ToString()
{
    return '(' + P.ToString() + '/' + Q.ToString() + ')';
}

// При a > b >0  работает Алгоритм Евклида:
// NOD(a,b)=NOD(b, a%b).
// Это рекурсия !!!
public static int NOD(int a, int b)
{
    if (a < b)
    {
        int tmp = a; a = b; b = tmp;
    }
```

```
        if (b == 0) { return a; }
        else { return NOD(b, a % b); }
    }

    // Сократить на НОД и перенести знак в числитель
    public void Reduce()
    {
        int p1, q1;
        if (p < 0) { p1 = -p; } else { p1 = p; }
        if (q < 0) { q1 = -q; } else { q1 = q; }
        int d = NOD(p1, q1);
        p1 = p1 / d; q1 = q1 / d;
        if (p * q < 0) { p = -p1; q = q1; }
        else { p = p1; q = q1; }
    }

}

class Program
{
    static void Main() // для отладки
```

```
        {   //отладить NOD,
            //сделать дробь, распечатать,
            //сократить и опять распечатать


        }
    }
} */

// 4. Добавить сложение, вычитание и умножение
/* using System;

namespace Fractions {
    class Fraction
    {
        int p;   // числитель, private
        public int P { get { return p; } }
        int q;   // знаменатель, private
        public int Q { get { return q; } }
```

```
//конструктор -- создает дробь,
//не дает обратить знаменатель в 0.
public Fraction(int a, int b)
{
    if (b != 0) { p = a; q = b; }
    else { p = 0; q = 1; }
}

// проверка дробей на равенство
public static bool Eq(Fraction fr1, Fraction fr2)
{
    return fr1.P * fr2.Q == fr1.Q * fr2.P;
}

// подмена (override)  стандартного ToString()
public override string ToString()
{
    return '(' + P.ToString() + '/' + Q.ToString() + ')';
}

// При a > b >0  работает Алгоритм Евклида:
```

```
// NOD(a,b)=NOD(b, a%b).
// Это рекурсия !!!
public static int NOD(int a, int b)
{
    if (a < b)
    {
        int tmp = a; a = b; b = tmp;
    }
    if (b == 0) { return a; }
    else { return NOD(b, a % b); }
}

// Сократить на НОД и перенести знак в числитель
public void Reduce()
{
    int p1, q1;
    if (p < 0) { p1 = -p; } else { p1 = p; }
    if (q < 0) { q1 = -q; } else { q1 = q; }
    int d = NOD(p1, q1);
    p1 = p1 / d; q1 = q1 / d;
    if (p * q < 0) { p = -p1; q = q1; }
```

```
    else { p = p1; q = q1; }
}

// сложение + сокращение
public static Fraction Add(Fraction fr1, Fraction fr2)
{
    Fraction fr =
        new Fraction(fr1.P * fr2.Q + fr2.P * fr1.Q, fr1.Q * fr2.Q);
    fr.Reduce();
    return fr;
}
// вычитание + сокращение
public static Fraction Subsr(Fraction fr1, Fraction fr2)
{
    Fraction fr =
        new Fraction(fr1.P * fr2.Q - fr2.P * fr1.Q, fr1.Q * fr2.Q);
    fr.Reduce();
    return fr;
}
// умножение + сокращение
public static Fraction Mult(Fraction fr1, Fraction fr2)
```

```
        {
            Fraction fr = new Fraction(fr1.P * fr2.P, fr1.Q * fr2.Q);
            fr.Reduce();
            return fr;
        }

    }

    class Program
    {
        static void Main() // для отладки
        {   // выполнить действия


        }
    }
} */

// 5. Добавить деление, прерывание при делении на 0.
/* using System;
```

```
namespace Fractions {
    class Fraction
    {
        int p;   // числитель, private
        public int P { get { return p; } }
        int q;   // знаменатель, private
        public int Q { get { return q; } }


        //конструктор -- создает дробь,
        //не дает обратить знаменатель в 0.
        public Fraction(int a, int b)
        {
            if (b != 0) { p = a; q = b; }
            else { p = 0; q = 1; }
        }

        // проверка дробей на равенство
        public static bool Eq(Fraction fr1, Fraction fr2)
        {
            return fr1.P * fr2.Q == fr1.Q * fr2.P;
```

```
}

// подмена (override)  стандартного ToString()
public override string ToString()
{
    return '(' + P.ToString() + '/' + Q.ToString() + ')';
}

// При a > b >0  работает Алгоритм Евклида:
// NOD(a,b)=NOD(b, a%b).
// Это рекурсия !!!
public static int NOD(int a, int b)
{
    if (a < b)
    {
        int tmp = a; a = b; b = tmp;
    }
    if (b == 0) { return a; }
    else { return NOD(b, a % b); }
}
```

```
// Сократить на НОД и перенести знак в числитель
public void Reduce()
{
    int p1, q1;
    if (p < 0) { p1 = -p; } else { p1 = p; }
    if (q < 0) { q1 = -q; } else { q1 = q; }
    int d = NOD(p1, q1);
    p1 = p1 / d; q1 = q1 / d;
    if (p * q < 0) { p = -p1; q = q1; }
    else { p = p1; q = q1; }
}

// сложение + сокращение
public static Fraction Add(Fraction fr1, Fraction fr2)
{
    Fraction fr =
         new Fraction(fr1.P * fr2.Q + fr2.P * fr1.Q, fr1.Q * fr2.Q);
    fr.Reduce();
    return fr;
}
```

```
// вычитание + сокращение
public static Fraction Subsr(Fraction fr1, Fraction fr2)
{
    Fraction fr =
         new Fraction(fr1.P * fr2.Q - fr2.P * fr1.Q, fr1.Q * fr2.Q);
    fr.Reduce();
    return fr;
}

// умножение + сокращение
public static Fraction Mult(Fraction fr1, Fraction fr2)
{
    Fraction fr = new Fraction(fr1.P * fr2.P, fr1.Q * fr2.Q);
    fr.Reduce();
    return fr;
}

// деление + прерывание + сокращение
public static Fraction Div(Fraction fr1, Fraction fr2)
{
    if (fr2.P == 0) { throw new ArgumentException(); }
```

```
            Fraction fr = new Fraction(fr1.P * fr2.Q, fr1.Q * fr2.P);
            fr.Reduce();
            return fr;
        }


    }

    class Program
    {
        static void Main() // для отладки
        {   // выполнить деление, проверить прерывание


        }
    }
} */

// 6. Добавить преобразование в double
/* using System;
```

```
namespace Fractions {
    class Fraction
    {
        int p;   // числитель, private
        public int P { get { return p; } }
        int q;   // знаменатель, private
        public int Q { get { return q; } }


        //конструктор -- создает дробь,
        //не дает обратить знаменатель в 0.
        public Fraction(int a, int b)
        {
            if (b != 0) { p = a; q = b; }
            else { p = 0; q = 1; }
        }

        // проверка дробей на равенство
        public static bool Eq(Fraction fr1, Fraction fr2)
        {
            return fr1.P * fr2.Q == fr1.Q * fr2.P;
```

```
}

// подмена (override)  стандартного ToString()
public override string ToString()
{
    return '(' + P.ToString() + '/' + Q.ToString() + ')';
}

// При a > b >0  работает Алгоритм Евклида:
// NOD(a,b)=NOD(b, a%b).
// Это рекурсия !!!
public static int NOD(int a, int b)
{
    if (a < b)
    {
        int tmp = a; a = b; b = tmp;
    }
    if (b == 0) { return a; }
    else { return NOD(b, a % b); }
}
```

```
// Сократить на НОД и перенести знак в числитель
public void Reduce()
{
    int p1, q1;
    if (p < 0) { p1 = -p; } else { p1 = p; }
    if (q < 0) { q1 = -q; } else { q1 = q; }
    int d = NOD(p1, q1);
    p1 = p1 / d; q1 = q1 / d;
    if (p * q < 0) { p = -p1; q = q1; }
    else { p = p1; q = q1; }
}

// сложение + сокращение
public static Fraction Add(Fraction fr1, Fraction fr2)
{
    Fraction fr =
      new Fraction(fr1.P * fr2.Q + fr2.P * fr1.Q, fr1.Q * fr2.Q);
    fr.Reduce();
    return fr;
}
```

```
// вычитание + сокращение
public static Fraction Subsr(Fraction fr1, Fraction fr2)
{
    Fraction fr =
      new Fraction(fr1.P * fr2.Q - fr2.P * fr1.Q, fr1.Q * fr2.Q);
    fr.Reduce();
    return fr;
}

// умножение + сокращение
public static Fraction Mult(Fraction fr1, Fraction fr2)
{
    Fraction fr = new Fraction(fr1.P * fr2.P, fr1.Q * fr2.Q);
    fr.Reduce();
    return fr;
}

// деление + прерывание + сокращение
public static Fraction Div(Fraction fr1, Fraction fr2)
{
    if (fr2.P == 0) { throw new ArgumentException(); }
```

```
        Fraction fr = new Fraction(fr1.P * fr2.Q, fr1.Q * fr2.P);
        fr.Reduce();
        return fr;
    }

    // преобразование в double
    public double ToDouble()
    {
        return (double)p / q;
    }


}
class Program
{
    static void Main() // для отладки
    {   // 1) преобразовать в double;
        // 2) подсчитать точно 1+1/2+1/3+...+1/n
        // При каких n это получится?

}}} */
```

lemmings

```
/* Наследование и полиморфизм.
 * В предлагаемой заготовке для игры Лемминги
 * требуется использовать 2 сорта леммингов,
 * отличающихся стратегией выбора очередного хода.
 * Называем их методы делать ход одинаково - void step().
 * Теперь хочется сделать массив всех леммингов
 * и заставить их всех ходить с помощью вызова этого метода
 * в цикле.
 * Но все элементы массива обязаны иметь одинаковый тип,
 * а у нас они разные. Для двух классов леммингов вводим
 * общего родителя. Тогда каждый лемминг будет иметь 2 типа:
 * собственный и родительский. Массив можно описать, как
 * массив элементов родительского типа, а заполнить леммингами
 * обоих типов.
 */

/* using System; using System.Threading;    //для задержки

namespace lemmings {
    class Map
```

```
{
    static char[,] a=new char[20,20]; // игровое поле
    static Random r = new Random();
    Lem[] lemmings;  // массив леммингов


    // Определяем родителя Lem и 2 наследника LemL и LemR
    class Lem  // родитель
    {
        // protected -- доступно себе и наследникам
        protected int direction; // направление движения
        protected int x;         // координаты
        protected int y;

        public Lem()         //общий конструктор
        {                    // для себя и наследников
            direction=r.Next(4);
            do
            {
                x=r.Next(20);
                y=r.Next(20);
```

```
        }
        while(a[x,y]=='#');
        a[x,y]='+';
    }

    public virtual void step(){} // для переопределения
                                 // наследниками
}

class LemR:Lem // наследование - правые лемминги (+):
{              // когда некуда идти - поворот направо
    public override void step()
    {
        a[x, y] = ' ';        //убрать фишку с поля
        switch (direction)
        {
            case 0:   //вправо
                if (a[x+1,y]==' '){x++;}
                else{direction=3;}
                break;
            case 1:  //вверх
```

```
                if (a[x,y+1]==' '){y++;}
                else{direction=0;}
                break;
            case 2:  //влево
                if (a[x-1,y]==' '){x--;}
                else{direction=1;}
                break;
            case 3:  //вниз
                if (a[x,y-1]==' '){y--;}
                else{direction=2;}
                break;
        }
        a[x,y]='+';    //поставить фишку
    }
}

class LemL:Lem // наследование - левые лемминги (*):
{          // когда некуда идти - поворот налево
        public override void step()
        {
            a[x, y] = ' ';    //убрать фишку
```

```
    switch (direction)
    {
        case 0:
            if (a[x+1,y]==' '){x++;}
            else{direction=1;}
            break;
        case 1:
            if (a[x,y+1]==' '){y++;}
            else{direction=2;}
            break;
        case 2:
            if (a[x-1,y]==' '){x--;}
            else{direction=3;}
            break;
        case 3:
            if (a[x,y-1]==' '){y--;}
            else{direction=0;}
            break;
    }
    a[x, y] = '*';      //поставить фишку
}
```

```
}


// Конструктор всего класса Map
// n - количество леммингов
public Map(int n)
{
    // препятствия по границе поля:
    for (int i = 0; i < 20; i++)
    {
        for (int j = 0; j < 20; j++)
        {
            if (i == 0 || i == 19 || j == 0 || j == 19) { a[i, j] = '#'; }
            else { a[i, j] = ' '; }
        }
    }
    // можно вставить доп. препятствия
    // a[10,1]='#';
```

```
    lemmings = new Lem[n];  // созание и заполнение масива
                            // разными леммингами
    for (int i = 0; i < n; i++)
    {
        if (i % 2 == 0) { lemmings[i] = new LemL(); }
        else { lemmings[i] = new LemR(); }
    }
}

// изображение  текущего расположения в цвете
void ShowMap()
{
    Console.SetCursorPosition(0, 0);
    for (int i = 0; i < 20; i++)
    {
        for (int j = 0; j < 20; j++)
        {
            switch (a[i, j]) //выбор цвета
            {
                case '+':
                    Console.ForegroundColor = ConsoleColor.Blue;
```

```
                    break;
                case '*':
                    Console.ForegroundColor = ConsoleColor.Red;
                    break;
                case '#':
                    Console.ForegroundColor = ConsoleColor.White;
                    break;
            }
            Console.Write(a[i, j]);
        }
        Console.WriteLine();
    }
}

public void OneStep()  // все лемминги ходят 1 раз
{
    foreach (Lem l in lemmings)
    {
        l.step();

    }
```

```
            ShowMap();   //перерисует все поле
        }


    }
    class Program
    {
        static void Main()
        {
            Map m = new Map(10);
            for (int k = 0; k < 200; k++)
            {
                m.OneStep();
                Thread.Sleep(100); //задержка
            }
        }
    }
} */


// Добавим реакцию на клавиши - проставлять доп. препятствия.
```

```
// Стадия 1: перехват клавиши --
// Console.KeyAvailable,
// Console.ReadKey(),
// ConsoleKeyInfo,
// ConsoleKey.LeftArrow, ...
/* using System; using System.Threading;    //для задержки

namespace lemmings {
    class Map
    {
        static char[,] a = new char[20, 20]; // игровое поле
        static Random r = new Random();
        Lem[] lemmings;  // массив леммингов


        // Определяем родителя Lem и 2 наследника LemL и LemR
        class Lem  // родитель
        {
            // protected -- доступно себе и наследникам
            protected int direction; // направление движения
            protected int x;         // координаты
```

```
    protected int y;

    public Lem()        //общий конструктор
    {                   // для себя и наследников
        direction = r.Next(4);
        do
        {
            x = r.Next(20);
            y = r.Next(20);
        }
        while (a[x, y] == '#');
        a[x, y] = '+';
    }

    public virtual void step() { } // для переопределения
    // наследниками
}

class LemR : Lem // наследование - правые лемминги (+):
{               // когда некуда идти - поворот направо
    public override void step()
```

```
{
    a[x, y] = ' ';        //убрать фишку с поля
    switch (direction)
    {
        case 0:   //вправо
            if (a[x + 1, y] == ' ') { x++; }
            else { direction = 3; }
            break;
        case 1:  //вверх
            if (a[x, y + 1] == ' ') { y++; }
            else { direction = 0; }
            break;
        case 2:  //влево
            if (a[x - 1, y] == ' ') { x--; }
            else { direction = 1; }
            break;
        case 3:  //вниз
            if (a[x, y - 1] == ' ') { y--; }
            else { direction = 2; }
            break;
    }
```

```
        a[x, y] = '+';    //поставить фишку
    }
}

class LemL : Lem // наследование - левые лемминги (*):
{          // когда некуда идти - поворот налево
    public override void step()
    {
        a[x, y] = ' ';    //убрать фишку
        switch (direction)
        {
            case 0:
                if (a[x + 1, y] == ' ') { x++; }
                else { direction = 1; }
                break;
            case 1:
                if (a[x, y + 1] == ' ') { y++; }
                else { direction = 2; }
                break;
            case 2:
                if (a[x - 1, y] == ' ') { x--; }
```

```
                else { direction = 3; }
                break;
            case 3:
                if (a[x, y - 1] == ' ') { y--; }
                else { direction = 0; }
                break;
        }
        a[x, y] = '*';      //поставить фишку
    }
}


// Конструктор всего класса Map
// n - количество леммингов
public Map(int n)
{
    // препятствия по границе поля:
    for (int i = 0; i < 20; i++)
    {
        for (int j = 0; j < 20; j++)
        {
```

```
            if (i == 0 || i == 19 || j == 0 || j == 19) { a[i, j] = '#'; }
            else { a[i, j] = ' '; }
        }
    }
    // можно вставить доп. препятствия
    // a[10,1]='#';



    lemmings = new Lem[n];  // созание и заполнение масива
    // разными леммингами
    for (int i = 0; i < n; i++)
    {
        if (i % 2 == 0) { lemmings[i] = new LemL(); }
        else { lemmings[i] = new LemR(); }
    }
}

// изображение  текущего расположения в цвете
void ShowMap()
{
```

```
Console.SetCursorPosition(0, 0);
for (int i = 0; i < 20; i++)
{
    for (int j = 0; j < 20; j++)
    {
        switch (a[i, j]) //выбор цвета
        {
            case '+':
                Console.ForegroundColor = ConsoleColor.Blue;
                break;
            case '*':
                Console.ForegroundColor = ConsoleColor.Red;
                break;
            case '#':
                Console.ForegroundColor = ConsoleColor.White;
                break;
        }
        Console.Write(a[i, j]);
    }
    Console.WriteLine();
}
```

```
}

public void OneStep()  // все лемминги ходят 1 раз
{
    foreach (Lem l in lemmings)
    {
        l.step();

    }
    ShowMap();   //перерисует все поле
    KeyboardAction(); // реакция на клавиши
}

void KeyboardAction()  // перехват клавиши
{
    if (Console.KeyAvailable)
    {
        ConsoleKeyInfo keyInfo = Console.ReadKey();
        Console.SetCursorPosition(0, 21);
        switch (keyInfo.Key)
```

```
            {
                case ConsoleKey.LeftArrow:
                    Console.WriteLine("<-"); break;
                case ConsoleKey.RightArrow:
                    Console.WriteLine("->"); break;
                case ConsoleKey.UpArrow:
                    Console.WriteLine(@"/\"); break;
                case ConsoleKey.DownArrow:
                    Console.WriteLine(@"\/"); break;
                case ConsoleKey.Spacebar:
                    Console.WriteLine("  "); break;
            }


        }
    }
}
class Program
{
    static void Main()
    {
```

```
            Map m = new Map(2);

            for (int k = 0; k < 200; k++)
            {
                m.OneStep();
                Thread.Sleep(50); //задержка
            }
            Console.SetCursorPosition(0, 20);
        }
    }
} */

// Стадия 2 -- добавление курсора в состав Map.
// KeyboardAction() управляет его перемещением.
/* using System; using System.Threading;    //для задержки

namespace lemmings {
    class Map
    {
        static char[,] a = new char[20, 20]; // игровое поле
        static Random r = new Random();
```

```
Lem[] lemmings;  // массив леммингов
int curX=10, curY=10; // координаты курсора

// Определяем родителя Lem и 2 наследника LemL и LemR
class Lem  // родитель
{
    // protected -- доступно себе и наследникам
    protected int direction; // направление движения
    protected int x;         // координаты
    protected int y;

    public Lem()        //общий конструктор
    {                   // для себя и наследников
        direction = r.Next(4);
        do
        {
            x = r.Next(20);
            y = r.Next(20);
        }
        while (a[x, y] == '#');
        a[x, y] = '+';
```

```
    }

    public virtual void step() { } // для переопределения
    // наследниками
}

class LemR : Lem // наследование - правые лемминги (+):
{               // когда некуда идти - поворот направо
    public override void step()
    {
        a[x, y] = ' ';        //убрать фишку с поля
        switch (direction)
        {
            case 0:   //вправо
                if (a[x + 1, y] == ' ') { x++; }
                else { direction = 3; }
                break;
            case 1:  //вверх
                if (a[x, y + 1] == ' ') { y++; }
                else { direction = 0; }
                break;
```

```
            case 2:  //влево
                if (a[x - 1, y] == ' ') { x--; }
                else { direction = 1; }
                break;
            case 3:  //вниз
                if (a[x, y - 1] == ' ') { y--; }
                else { direction = 2; }
                break;
        }
        a[x, y] = '+';    //поставить фишку
    }
}

class LemL : Lem // наследование - левые лемминги (*):
{          // когда некуда идти - поворот налево
    public override void step()
    {
        a[x, y] = ' ';    //убрать фишку
        switch (direction)
        {
            case 0:
```

```
                if (a[x + 1, y] == ' ') { x++; }
                else { direction = 1; }
                break;
            case 1:
                if (a[x, y + 1] == ' ') { y++; }
                else { direction = 2; }
                break;
            case 2:
                if (a[x - 1, y] == ' ') { x--; }
                else { direction = 3; }
                break;
            case 3:
                if (a[x, y - 1] == ' ') { y--; }
                else { direction = 0; }
                break;
        }
        a[x, y] = '*';      //поставить фишку
    }
}
```

```
// Конструктор всего класса Map
// n - количество леммингов
public Map(int n)
{
    // препятствия по границе поля:
    for (int i = 0; i < 20; i++)
    {
        for (int j = 0; j < 20; j++)
        {
            if (i == 0 || i == 19 || j == 0 || j == 19) { a[i, j] = '#'; }
            else { a[i, j] = ' '; }
        }
    }
    // можно вставить доп. препятствия
    // a[10,1]='#';



    lemmings = new Lem[n];  // созание и заполнение масива
    // разными леммингами
    for (int i = 0; i < n; i++)
```

```
    {
        if (i % 2 == 0) { lemmings[i] = new LemL(); }
        else { lemmings[i] = new LemR(); }
    }
}

// изображение  текущего расположения в цвете
void ShowMap()
{
    Console.SetCursorPosition(0, 0);
    for (int i = 0; i < 20; i++)
    {
        for (int j = 0; j < 20; j++)
        {
            switch (a[i, j]) //выбор цвета
            {
                case '+':
                    Console.ForegroundColor = ConsoleColor.Blue;
                    break;
                case '*':
                    Console.ForegroundColor = ConsoleColor.Red;
```

```
                    break;
                case '#':
                    Console.ForegroundColor = ConsoleColor.White;
                    break;
            }
            Console.Write(a[i, j]);
        }
        Console.WriteLine();
    }
    Console.SetCursorPosition(curX, curY);
}

public void OneStep()  // все лемминги ходят 1 раз
{
    foreach (Lem l in lemmings)
    {
        l.step();

    }
    ShowMap();   //перерисует все поле
    KeyboardAction(); // реакция на клавиши
```

```
}

void KeyboardAction()  // перехват клавиши
{
    if (Console.KeyAvailable)
    {
        ConsoleKeyInfo keyInfo = Console.ReadKey();
        //Console.SetCursorPosition(0, 21);
        switch (keyInfo.Key)
        {
            case ConsoleKey.LeftArrow:
                if (curX > 1) { curX--; }; break;
            case ConsoleKey.RightArrow:
                if (curX < 18) { curX++; }; break;
            case ConsoleKey.UpArrow:
                if (curY > 1) { curY--; }; break;
            case ConsoleKey.DownArrow:
                if (curY < 18) { curY++; }; break;
            case ConsoleKey.Spacebar:
                a[curY, curX] = '#'; break;// карту нарисовали
                                           // перевернутой :-)
```

```
                }


            }
        }
    }
    class Program
    {
        static void Main()
        {
            Map m = new Map(2);

            for (int k = 0; k < 500; k++)
            {
                m.OneStep();
                Thread.Sleep(100); //задержка
            }
            Console.SetCursorPosition(0, 20); // в самый низ
        }
    }
} */
```

```
/* 4. Теперь пусть леминги дерутся друг с друга, причем оборона сильнее атаки.
 * Тот, кто напал, погибает (съеден). Изменения следующие
 * 1) Теперь lemmings -- не массив, а список (чтобы удалять съеденных, lemmings.Re
 * Изменяется цикл начального его заполнения
 * и цикл foreach по нему в OneStep (не работает после Remove) меняем на while.
 * 2) У лемингов изменяем step -- если налетел на другого,
 * то съеден и удален из lemmings. Его изображение удаляется с карты.
 */
/* using System; using System.Threading;    //для задержки using
System.Collections.Generic;

namespace lemmings {
    class Map
    {
        static char[,] a = new char[20, 20]; // игровое поле
        static Random r = new Random();
        //Lem[] lemmings;  // массив лемингов
        static List<Lem> lemmings;

        // Определяем родителя Lem и 2 наследника LemL и LemR
```

```
class Lem  // родитель
{
    // protected -- доступно себе и наследникам
    protected int direction; // направление движения
    protected int x;         // координаты
    protected int y;

    public Lem()         //общий конструктор
    {                    // для себя и наследников
        direction = r.Next(4);
        do
        {
            x = r.Next(20);
            y = r.Next(20);
        }
        while (a[x, y] == '#');
        a[x, y] = '+';
    }

    public virtual void step() { } // для переопределения
    // наследниками
```

```
}

class LemR : Lem // наследование - правые лемминги (+):
{               // когда некуда идти - поворот направо
    public override void step()
    {
        a[x, y] = ' ';        //убрать фишку с поля
        switch (direction)
        {
            case 0:   //вправо
                if (a[x + 1, y] == ' ') { x++; a[x, y] = '+'; }
                else if (a[x + 1, y] != '#') { lemmings.Remove(this); }
                else { direction = 3; a[x, y] = '+'; }
                break;
            case 1:  //вверх
                if (a[x, y + 1] == ' ') { y++; a[x, y] = '+'; }
                else if (a[x, y+1] != '#') { lemmings.Remove(this); }
                else { direction = 0; a[x, y] = '+'; }
                break;
            case 2:  //влево
                if (a[x - 1, y] == ' ') { x--; a[x, y] = '+'; }
```

```
                else if (a[x - 1, y] != '#') { lemmings.Remove(this); }
                else { direction = 1; a[x, y] = '+'; }
                break;
            case 3:  //вниз
                if (a[x, y - 1] == ' ') { y--; a[x, y] = '+'; }
                else if (a[x, y -1] != '#') { lemmings.Remove(this); }
                else { direction = 2; a[x, y] = '+'; }
                break;
        }
       // a[x, y] = '+';    //поставить фишку
    }
}

class LemL : Lem // наследование - левые лемминги (*):
{          // когда некуда идти - поворот налево
    public override void step()
    {
        a[x, y] = ' ';    //убрать фишку
        switch (direction)
        {
            case 0:
```

```
            if (a[x + 1, y] == ' ') { x++; a[x, y] = '*'; }
            else if (a[x + 1, y] != '#') { lemmings.Remove(this); }
            else { direction = 1; a[x, y] = '*'; }
            break;
        case 1:
            if (a[x, y + 1] == ' ') { y++; a[x, y] = '*'; }
            else if (a[x, y+1] != '#') { lemmings.Remove(this); }
            else { direction = 2; a[x, y] = '*'; }
            break;
        case 2:
            if (a[x - 1, y] == ' ') { x--; a[x, y] = '*'; }
            else if (a[x - 1, y] != '#') { lemmings.Remove(this); }
            else { direction = 3; a[x, y] = '*'; }
            break;
        case 3:
            if (a[x, y - 1] == ' ') { y--; a[x, y] = '*'; }
            else if (a[x, y-1] != '#') { lemmings.Remove(this); }
            else { direction = 0; a[x, y] = '*'; }
            break;
    }
    // a[x, y] = '*';      //поставить фишку
```

```
    }
}


// Конструктор всего класса Map
// n - количество леммингов
public Map(int n)
{
    // препятствия по границе поля:
    for (int i = 0; i < 20; i++)
    {
        for (int j = 0; j < 20; j++)
        {
            if (i == 0 || i == 19 || j == 0 || j == 19) { a[i, j] = '#'; }
            else { a[i, j] = ' '; }
        }
    }
    // можно вставить доп. препятствия
    // a[10,1]='#';
```

```
    lemmings = new List<Lem>();  // созание и заполнение масива
    // разными леммингами
    for (int i = 0; i < n; i++)
    {
        if (i % 2 == 0) { lemmings.Add( new LemL()); }
        else { lemmings.Add(new LemR()); }
    }
}

// изображение  текущего расположения в цвете
void ShowMap()
{
    Console.SetCursorPosition(0, 0);
    for (int i = 0; i < 20; i++)
    {
        for (int j = 0; j < 20; j++)
        {
            switch (a[i, j]) //выбор цвета
            {
                case '+':
```

```
                    Console.ForegroundColor = ConsoleColor.Green ;
                    break;
                case '*':
                    Console.ForegroundColor = ConsoleColor.Yellow ;
                    break;
                case '#':
                    Console.ForegroundColor = ConsoleColor.White;
                    break;
            }
            Console.Write(a[i, j]);
        }
        Console.WriteLine();
    }
}

public void OneStep()  // все лемминги ходят 1 раз
{
    //foreach (Lem l in lemmings)
    int i = 0;
    int k = lemmings.Count;
    while(i<k)
```

```
        {
            lemmings[i].step();
            if (lemmings.Count < k) { k = lemmings.Count; } else { i++; }
        }
        ShowMap();   //перерисует все поле
    }


}
class Program
{
    static void Main()
    {
        Map m = new Map(20);
        for (int k = 0; k < 200; k++)
        {
            m.OneStep();
            Thread.Sleep(100); //задержка
        }
    }
}
```

```
}

*/

/*
// Вариант "своих не едят".
using System; using System.Threading;    //для задержки using
System.Collections.Generic;

namespace lemmings {
    class Map
    {
        static char[,] a = new char[20, 20]; // игровое поле
        static Random r = new Random();
        //Lem[] lemmings;  // массив леминго в
        static List<Lem> lemmings;

        // Определяем родителя Lem и 2 наследника LemL и LemR
        class Lem  // родитель
        {
            // protected -- доступно себе и наследникам
```

```
    protected int direction; // направление движения
    protected int x;         // координаты
    protected int y;

    public Lem()        //общий конструктор
    {                   // для себя и наследников
        direction = r.Next(4);
        do
        {
            x = r.Next(20);
            y = r.Next(20);
        }
        while (a[x, y] == '#');
        a[x, y] = '+';
    }

    public virtual void step() { } // для переопределения
    // наследниками
}

class LemR : Lem // наследование - правые лемминги (+):
```

```
{               // когда некуда идти - поворот направо
    public override void step()
    {
        a[x, y] = ' ';        //убрать фишку с поля
        switch (direction)
        {
            case 0:   //вправо
                if (a[x + 1, y] == ' ') { x++; a[x, y] = '+'; }
                else if (a[x + 1, y] == '*') { lemmings.Remove(this); }
                else { direction = 3; a[x, y] = '+'; }
                break;
            case 1:  //вверх
                if (a[x, y + 1] == ' ') { y++; a[x, y] = '+'; }
                else if (a[x, y + 1] == '*') { lemmings.Remove(this); }
                else { direction = 0; a[x, y] = '+'; }
                break;
            case 2:  //влево
                if (a[x - 1, y] == ' ') { x--; a[x, y] = '+'; }
                else if (a[x - 1, y] == '*') { lemmings.Remove(this); }
                else { direction = 1; a[x, y] = '+'; }
                break;
```

```
            case 3:  //вниз
                if (a[x, y - 1] == ' ') { y--; a[x, y] = '+'; }
                else if (a[x, y - 1] == '*') { lemmings.Remove(this); }
                else { direction = 2; a[x, y] = '+'; }
                break;
        }
        // a[x, y] = '+';    //поставить фишку
    }
}

class LemL : Lem // наследование - левые лемминги (*):
{          // когда некуда идти - поворот налево
    public override void step()
    {
        a[x, y] = ' ';    //убрать фишку
        switch (direction)
        {
            case 0:
                if (a[x + 1, y] == ' ') { x++; a[x, y] = '*'; }
                else if (a[x + 1, y] == '+') { lemmings.Remove(this); }
                else { direction = 1; a[x, y] = '*'; }
```

```
                    break;
                case 1:
                    if (a[x, y + 1] == ' ') { y++; a[x, y] = '*'; }
                    else if (a[x, y + 1] == '+') { lemmings.Remove(this); }
                    else { direction = 2; a[x, y] = '*'; }
                    break;
                case 2:
                    if (a[x - 1, y] == ' ') { x--; a[x, y] = '*'; }
                    else if (a[x - 1, y] == '+') { lemmings.Remove(this); }
                    else { direction = 3; a[x, y] = '*'; }
                    break;
                case 3:
                    if (a[x, y - 1] == ' ') { y--; a[x, y] = '*'; }
                    else if (a[x, y - 1] == '+') { lemmings.Remove(this); }
                    else { direction = 0; a[x, y] = '*'; }
                    break;
            }
            // a[x, y] = '*';      //поставить фишку
        }
    }
```

```
// Конструктор всего класса Map
// n - количество леммингов
public Map(int n)
{
    // препятствия по границе поля:
    for (int i = 0; i < 20; i++)
    {
        for (int j = 0; j < 20; j++)
        {
            if (i == 0 || i == 19 || j == 0 || j == 19) { a[i, j] = '#'; }
            else { a[i, j] = ' '; }
        }
    }
    // можно вставить доп. препятствия
    // a[10,1]='#';



    lemmings = new List<Lem>();  // созание и заполнение масива
    // разными леммингами
```

```
    for (int i = 0; i < n; i++)
    {
        if (i % 2 == 0) { lemmings.Add(new LemL()); }
        else { lemmings.Add(new LemR()); }
    }
}

// изображение  текущего расположения в цвете
void ShowMap()
{
    Console.SetCursorPosition(0, 0);
    for (int i = 0; i < 20; i++)
    {
        for (int j = 0; j < 20; j++)
        {
            switch (a[i, j]) //выбор цвета
            {
                case '+':
                    Console.ForegroundColor = ConsoleColor.Green;
                    break;
                case '*':
```

```
                    Console.ForegroundColor = ConsoleColor.Yellow;
                    break;
                case '#':
                    Console.ForegroundColor = ConsoleColor.White;
                    break;
            }
            Console.Write(a[i, j]);
        }
        Console.WriteLine();
    }
}

public void OneStep()  // все лемминги ходят 1 раз
{
    //foreach (Lem l in lemmings)
    int i = 0;
    int k = lemmings.Count;
    while (i < k)
    {
        lemmings[i].step();
        if (lemmings.Count < k) { k = lemmings.Count; } else { i++; }
```

```
            }
            ShowMap();   //перерисует все поле
        }


    }
    class Program
    {
        static void Main()
        {
            Map m = new Map(20);
            for (int k = 0; k < 200; k++)
            {
                m.OneStep();
                Thread.Sleep(200); //задержка
            }
        }
    }
} */

// Бродячие лемминги в графическом режиме. Размер поля d задается в классе Form1.
```

```
/* using System; using System.Drawing; using System.Windows.Forms; using
System.Threading;

namespace kak_leming {
    class Map
    {
        public static char[,] a;        // = new char[20,20]; // игровое поле
        static Random r = new Random();
        public Lem[] lemmings;  // массив лемингов


        // Определяем родителя Lem и 2 наследника LemL и LemR
        public class Lem  // родитель
        {
            // protected -- доступно себе и наследникам
            protected int direction; // направление движения
            protected int x;         // координаты
            protected int y;
```

```
    public virtual void step(){} // для переопределения
                                 // наследниками
}

public class LemR:Lem // наследование - правые лемминги (+):
{               // когда некуда идти - поворот направо
    public LemR(int dim)        //общий конструктор
    {                    // для себя и наследников
        direction=r.Next(4);
        do
        {
            x=r.Next(dim);
            y=r.Next(dim);
        }
        while(a[x,y]=='#');
        a[x,y]='+';
    }

    public override void step()
    {
```

```
a[x, y] = ' ';        //убрать фишку с поля
switch (direction)
{
    case 0:   //вправо
        if (a[x+1,y]==' '){x++;}
        else{direction=3;}
        break;
    case 1:  //вверх
        if (a[x,y+1]==' '){y++;}
        else{direction=0;}
        break;
    case 2:  //влево
        if (a[x-1,y]==' '){x--;}
        else{direction=1;}
        break;
    case 3:  //вниз
        if (a[x,y-1]==' '){y--;}
        else{direction=2;}
        break;
}
a[x,y]='+';    //поставить фишку
```

```
    }
}

public class LemL:Lem // наследование - левые лемминги (*):
{          // когда некуда идти - поворот налево
    public LemL(int dim)        //общий конструктор
    {                    // для себя и наследников
        direction=r.Next(4);
        do
        {
            x=r.Next(dim);
            y=r.Next(dim);
        }
        while(a[x,y]=='#');
        a[x,y]='+';
    }

    public override void step()
        {
            a[x, y] = ' ';    //убрать фишку
            switch (direction)
```

```
            {
                case 0:
                    if (a[x+1,y]==' '){x++;}
                    else{direction=1;}
                    break;
                case 1:
                    if (a[x,y+1]==' '){y++;}
                    else{direction=2;}
                    break;
                case 2:
                    if (a[x-1,y]==' '){x--;}
                    else{direction=3;}
                    break;
                case 3:
                    if (a[x,y-1]==' '){y--;}
                    else{direction=0;}
                    break;
            }
            a[x, y] = '*';      //поставить фишку
        }
}
```

```
// Конструктор всего класса Map
// n - количество леммингов
public Map(int n, int dim)
{
    a = new char[dim, dim];
    // препятствия по границе поля:
    for (int i = 0; i < dim; i++)
    {
        for (int j = 0; j < dim; j++)
        {
            if (i == 0 || i == dim-1 || j == 0 || j == dim-1) { a[i, j] =
            else { a[i, j] = ' '; }
        }
    }

    lemmings = new Lem[n];  // созание и заполнение масива
                            // разными леммингами
    for (int i = 0; i < n; i++)
    {
```

```
            if (i % 2 == 0) { lemmings[i] = new LemL(dim); }
            else { lemmings[i] = new LemR(dim); }
        }
    }
}


class Form1 : Form
{
    int d = 40;  // размер поля
    Label[,] l;
    Button b = new Button();
    Map m;

    public Form1()
    {
        m = new Map(10,d);   // 10 -- количество леммингов
        l = new Label[d, d];


        b.Location = new Point((int)(4.6*d), 5*d);
```

```
        b.Text = "Start";
        b.Click += new EventHandler(b_Click);
        this.Controls.Add(b);

        for (int i = 0; i < d; i++)
            for (int j = 0; j < d; j++)
            {
                l[i, j] = new Label();
                l[i, j].Size = new Size(10, 10);
                l[i, j].Location = new Point(10 + 11 * j, 10 + 11 * i);
                l[i, j].BackColor = Color.Coral;
                this.Controls.Add(l[i, j]);

            }
        this.Size = new Size(l[d - 1, d - 1].Right + 15, l[d - 1, d - 1].Botto
        this.FormBorderStyle = FormBorderStyle.FixedDialog;
        this.StartPosition = FormStartPosition.CenterScreen;

    }
```

```
void b_Click(object sender, EventArgs e)
{
    b.Hide();
    for (int k = 0; k < 200; k++)
    {
        OneStep();
    }
    b.Show();
}

public void OneStep()  // все лемминги ходят 1 раз
{
    foreach (Map.Lem l in m.lemmings)
    {
        l.step();

    }
    ShowMap();   //перерисует все поле
    Thread.Sleep(100);
}
```

```
void ShowMap()
{
    for (int i = 0; i < d; i++)
    {
        for (int j = 0; j < d; j++)
        {
            switch (Map.a[i, j]) //выбор цвета
            {
                case '+':
                    l[i, j].BackColor = Color.BlueViolet;
                    break;
                case '*':
                    l[i, j].BackColor = Color.Red;
                    break;
                case '#':
                    l[i, j].BackColor = Color.Brown;
                    break;
                default:
                    l[i, j].BackColor = Color.Coral;
                    break;
            }
```

```
                }
            }
            Application.DoEvents();

        }

    }
    class Program
    {
        [STAThread]
        static void Main()
        {
            Form1 f = new Form1();
            Application.Run(f);
        }
    }
} */


// Дерущиеся лемминги в графическом режиме. Размер поля d и количество лемингов n
```

```
using System; using System.Drawing; using System.Windows.Forms; using System.Threa
using System.Collections.Generic;


namespace kak_leming {
    class Map
    {
        public static char[,] a;        // = new char[20,20]; // игровое поле
        static Random r = new Random();
        public static List<Lem> lemmings;
        // массив леминго в -- теперь список


        // Определяем родителя Lem и 2 наследника LemL и LemR
        public class Lem  // родитель
        {
            // protected -- доступно себе и наследникам
            protected int direction; // направление движения
            protected int x;         // координаты
            protected int y;
```

```
    public virtual void step() { } // для переопределения
    // наследниками
}

public class LemR : Lem // наследование - правые лемминги (+):
{               // когда некуда идти - поворот направо
    public LemR(int dim)        //общий конструктор
    {                     // для себя и наследников
        direction = r.Next(4);
        do
        {
            x = r.Next(dim);
            y = r.Next(dim);
        }
        while (a[x, y] == '#');
        a[x, y] = '+';
    }
```

```
public override void step()
{
    a[x, y] = ' ';        //убрать фишку с поля
    switch (direction)
    {
        case 0:   //вправо
            if (a[x + 1, y] == ' ') { x++; a[x, y] = '+'; }
            else if (a[x + 1, y] != '#') { lemmings.Remove(this); }
            else { direction = 3; a[x, y] = '+'; }
            break;
        case 1:  //вверх
            if (a[x, y + 1] == ' ') { y++; a[x, y] = '+'; }
            else if (a[x, y + 1] != '#') { lemmings.Remove(this); }
            else { direction = 0; a[x, y] = '+'; }
            break;
        case 2:  //влево
            if (a[x - 1, y] == ' ') { x--; a[x, y] = '+'; }
            else if (a[x - 1, y] != '#') { lemmings.Remove(this); }
            else { direction = 1; a[x, y] = '+'; }
            break;
        case 3:  //вниз
```

```
                if (a[x, y - 1] == ' ') { y--; a[x, y] = '+'; }
                else if (a[x, y - 1] != '#') { lemmings.Remove(this); }
                else { direction = 2; a[x, y] = '+'; }
                break;
            }
            // a[x, y] = '+';    //поставить фишку
        }
    }

    public class LemL : Lem // наследование - левые лемминги (*):
    {          // когда некуда идти - поворот налево
        public LemL(int dim)        //общий конструктор
        {                    // для себя и наследников
            direction = r.Next(4);
            do
            {
                x = r.Next(dim);
                y = r.Next(dim);
            }
            while (a[x, y] == '#');
            a[x, y] = '*';
```

```
}

public override void step()
{
    a[x, y] = ' ';    //убрать фишку
    switch (direction)
    {
        case 0:
            if (a[x + 1, y] == ' ') { x++; a[x, y] = '*'; }
            else if (a[x + 1, y] == '+') { lemmings.Remove(this); }
            else { direction = 1; a[x, y] = '*'; }
            break;
        case 1:
            if (a[x, y + 1] == ' ') { y++; a[x, y] = '*'; }
            else if (a[x, y + 1] == '+') { lemmings.Remove(this); }
            else { direction = 2; a[x, y] = '*'; }
            break;
        case 2:
            if (a[x - 1, y] == ' ') { x--; a[x, y] = '*'; }
            else if (a[x - 1, y] == '+') { lemmings.Remove(this); }
            else { direction = 3; a[x, y] = '*'; }
```

```
                break;
            case 3:
                if (a[x, y - 1] == ' ') { y--; a[x, y] = '*'; }
                else if (a[x, y - 1] == '+') { lemmings.Remove(this); }
                else { direction = 0; a[x, y] = '*'; }
                break;
        }
        // a[x, y] = '*';      //поставить фишку
    }
}

// Конструктор всего класса Map
// n - количество леммингов
public Map(int n, int dim)
{
    a = new char[dim, dim];
    // препятствия по границе поля:
    for (int i = 0; i < dim; i++)
    {
        for (int j = 0; j < dim; j++)
        {
```

```
                if (i == 0 || i == dim - 1 || j == 0 || j == dim - 1) { a[i, j
                else { a[i, j] = ' '; }
            }
        }

        lemmings = new List<Lem>();  // созание и заполнение масива
        // разными леммингами
        for (int i = 0; i < n; i++)
        {
            if (i % 2 == 0) { lemmings.Add(new LemL(dim)); }
            else { lemmings.Add(new LemR(dim)); }
        }
    }
}


class Form1 : Form
{
    int d = 40;  // размер поля
    int n = 20; // количество леммингов
    Label[,] l;
```

```
Button b = new Button();
Map m;


public Form1()
{
    l = new Label[d, d];


    b.Location = new Point((int)(4.6 * d), 5 * d);
    b.Text = "Start";
    b.Click += new EventHandler(b_Click);
    this.Controls.Add(b);

    for (int i = 0; i < d; i++)
        for (int j = 0; j < d; j++)
        {
            l[i, j] = new Label();
            l[i, j].Size = new Size(10, 10);
            l[i, j].Location = new Point(10 + 11 * j, 10 + 11 * i);
            l[i, j].BackColor = Color.Coral;
```

```
            this.Controls.Add(l[i, j]);

        }
    this.Size = new Size(l[d - 1, d - 1].Right + 15, l[d - 1, d - 1].Botto
    this.FormBorderStyle = FormBorderStyle.FixedDialog;
    this.StartPosition = FormStartPosition.CenterScreen;

}


void b_Click(object sender, EventArgs e)
{
    b.Hide();
    m = new Map(n, d);
    for (int k = 0; k < 200; k++)
    {
        OneStep();
    }
    b.Show();
}
```

```
public void OneStep()  // все лемминги ходят 1 раз
{
    int i = 0;
    int k = Map.lemmings.Count;
    while (i < k)
    {
        Map.lemmings[i].step();
        if (Map.lemmings.Count < k) { k = Map.lemmings.Count; } else { i++
    }
    ShowMap();   //перерисует все поле
    Thread.Sleep(100);
}

void ShowMap()
{
    for (int i = 0; i < d; i++)
    {
        for (int j = 0; j < d; j++)
        {
            switch (Map.a[i, j]) //выбор цвета
            {
```

```
                    case '+':
                        l[i, j].BackColor = Color.BlueViolet;
                        break;
                    case '*':
                        l[i, j].BackColor = Color.Red;
                        break;
                    case '#':
                        l[i, j].BackColor = Color.Brown;
                        break;
                    default:
                        l[i, j].BackColor = Color.Coral;
                        break;
                }
            }
        }
        Application.DoEvents();

    }

}
class Program
```

```
    {
        [STAThread]
        static void Main()
        {
            Form1 f = new Form1();
            Application.Run(f);
        }
    }
}
```