Функция

**Функция** являет собой небольшую **подпрограмму**. Если просто программа - это решение какой-то прикладной задачи, то функция – это тоже решение, только уже в рамках программы и, соответственно, она выполняет задачу «попроще». Функции позволяют **уменьшить размер программы** за счет того, что не нужно повторно писать какой-то фрагмент кода - мы просто вызываем сколько угодно и где нужно объявленную функцию.

Объявление функции имеет следующую структуру:

static [тип возвращаемого значения] [имя функции] ([аргументы])  
{  
  // тело функции  
}

Функция может возвращать значение или не возвращать. Если функция, например, возвращает целое число, то нужно указать тип возвращаемого значения **int**. Если функция не возвращает никакого значения, то для этого используется ключевое слово **void**.

**Аргументы** – это те данные, которые необходимы для выполнения функции. Аргументы записываются в формате [тип] [имя]. Если аргументов несколько, они отделяются запятой. Аргументы могут отсутствовать, тогда после имени функции будут пустые круглые скобки.

Пример функции, которая не возвращает значение

Напишем простую функцию "**Hello**", которая будет здороваться с тем, чье имя ему передали, то есть функция принимает 1 аргумент строкового типа "**name**". Так как функция **не** будет возвращать значение, указываем тип **void** перед именем функции.

static void Hello(string name)  
{  
   Console.WriteLine("Hello, " + name);  
}

Функция написана, и теперь используем ее:

static void Hello(string name)  
{  
   Console.WriteLine("Hello, " + name);  
}

public static void Main()  
{  
   string name = "Саша";  
   Hello(name);   
}

То есть при вызове функции **не** надо писать типы аргументов, а надо передавать значения. **Заметим, что функция пишется над главной функцией Main.**

Пример функции, которая возвращает значения

Напишем функцию **Sum**, которая будет находить **сумму двух чисел**. Аргументами у этой функции будут **два целых числа**. Тип возвращаемого значения – целое число **int**, так как сумма двух целых чисел - целое число.

static int Sum(int a, int b)  
 {  
    int sum = a + b;  
    return sum;  
 }

Логика у функции проста. Создаем переменную **sum**, в которую записываем сумму переданных значений в переменные **a** и **b**. Используем оператор **return**, чтобы вернуть результат.

**Оператор return должен быть обязательно в функции, которая возвращает значение.**

Используем нашу функцию:

static int Sum(int a, int b)  
{  
 int sum = a + b;  
 return sum;  
}

static void Main(string[] args)  
{  
 int a = 3, b = 6;  
   int sum = Sum(a, b);   
   Console.WriteLine(sum);  
}

Пример функции, которая не принимает аргументов

Также функции могут **не принимать аргументов**,тогда после имени функции будут идти **пустые** круглые скобки. Например, напишем функцию, которая выводит "**Hello, BEEGEEK!**":

static void Hello()  
{  
  Console.WriteLine("Hello, BEEGEEK!");  
}

Функция **Hello** ничего не принимает, так как внутри круглых скобок никаких аргументов нет, и ничего не возвращает. Она просто выводит на экран "Hello, BEEGEEK!". Давайте ее вызовем:

static void Hello()  
{  
  Console.WriteLine("Hello, BEEGEEK!");  
}

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

Оператор return

Когда встречается этот оператор, происходит **выход из функции** и код ниже (если он есть) выполняться не будет (например, в функцию передан такой аргумент, при котором нет смысла выполнять функцию). Он похож на оператор **break**, который используется для выхода из циклов. Этот оператор также можно использовать и в функциях, которые не возвращают значение. Оператор return допустимо использовать несколько раз в функции.

Разбор задачи

По данным натуральным *n* и *k* вычислите значение *Ckn*=*n*!*k*!⋅(*n*−*k*)!число сочетаний из *n* элементов по *k*.

Давайте разбираться. Заметим, что нам нужно посчитать **факториал** 3 раза: *n*!,*k*!,(*n*−*k*)!. А далее разделить значение *n*! на произведение *k*!⋅(*n*−*k*)!.

Давайте решать по порядку. Факториал *n* - это произведение чисел от 1 до *n*. Чтобы посчитать его, нужно воспользоваться **мультипликатором**. Так и сделаем:

int n = Convert.ToInt32(Console.ReadLine());

int k = Convert.ToInt32(Console.ReadLine());  
  
int factN = 1;  
for (int i = 1; i <= n; i++)  
{  
  factN = factN \* i;  
}

Теперь в переменной **factN** хранится факториал числа *n*. Таким же образом посчитаем факториал и двух других чисел, а именно факториал *k* и факториал *n*−*k*:

int factK = 1;  
for (int i = 1; i <= k; i++)  
{  
  factK = factK \* i;  
}

int nSubtractK = n - k;  
int factNSubtractK = 1;  
for (int i = 1; i <= nSubtractK; i++)  
{  
  factNSubtractK = factNSubtractK \* i;  
}

Затем, имея значения трех факториалов, нужно вывести:

Console.WriteLine(factN / (factK \* factNSubtractK));

Таким образом получилась следующая программа:

int n = Convert.ToInt32(Console.ReadLine());  
int k = Convert.ToInt32(Console.ReadLine());  
int factN = 1;

for (int i = 1; i <= n; i++)  
{  
  factN = factN \* i;  
}  
  
int factK = 1;

for (int i = 1; i <= k; i++)  
{  
  factK = factK \* i;  
}  
  
int nSubtractK = n - k;  
int factNSubtractK = 1;

for (int i = 1; i <= nSubtractK; i++)  
{  
  factNSubtractK = factNSubtractK \* i;  
}  
  
Console.WriteLine(factN / (factK \* factNSubtractK));

Заметим, что **вычисление факториала** происходит 3 **раза**, меняя **только** значение числа, чей факториал нужно вычислить. Такая программа выглядит **длинной**, **запутанной** и **понять** ее становится **сложнее**.

Давайте напишем **функцию** для вычисления факториала 1 раз, и будем ее использовать 3 раза или сколько угодно раз. Итак, функция вычисления факториала будет принимает **одно целое число**, чей факториал нужно вычислить, и возвращать тоже **целое число**, так как факториал целого числа есть целое число. Функция будет выглядеть следующим образом

static int Factorial(int n)  
{  
  int fact = 1;  
  for (int i = 1; i <= n; i++)  
  {  
    fact = fact \* i;  
  }  
  
  return fact;  
}

То есть мы посчитали факториал и вернули его с функции. Теперь давайте ее использовать

static int Factorial(int n)  
{  
  int fact = 1;  
  for (int i = 1; i <= n; i++)  
  {  
    fact = fact \* i;  
  }  
  
  return fact;  
}

static void Main(string[] args)  
{  
  int n = Convert.ToInt32(Console.ReadLine());  
  int k = Convert.ToInt32(Console.ReadLine());  
  
  int factN = Factorial(n);  
  
  int factK = Factorial(k);  
  
  int nSubtractK = n - k;  
  int factNSubtractK = Factorial(nSubtractK);  
    
  Console.WriteLine(factN / (factK \* factNSubtractK));  
}

Заметим, насколько **меньше** и **понятней** стала наша программа. За счет **удаления повторного кода**(вычисление факториала), мы сэкономили время, и понять нашу программу теперь легко.