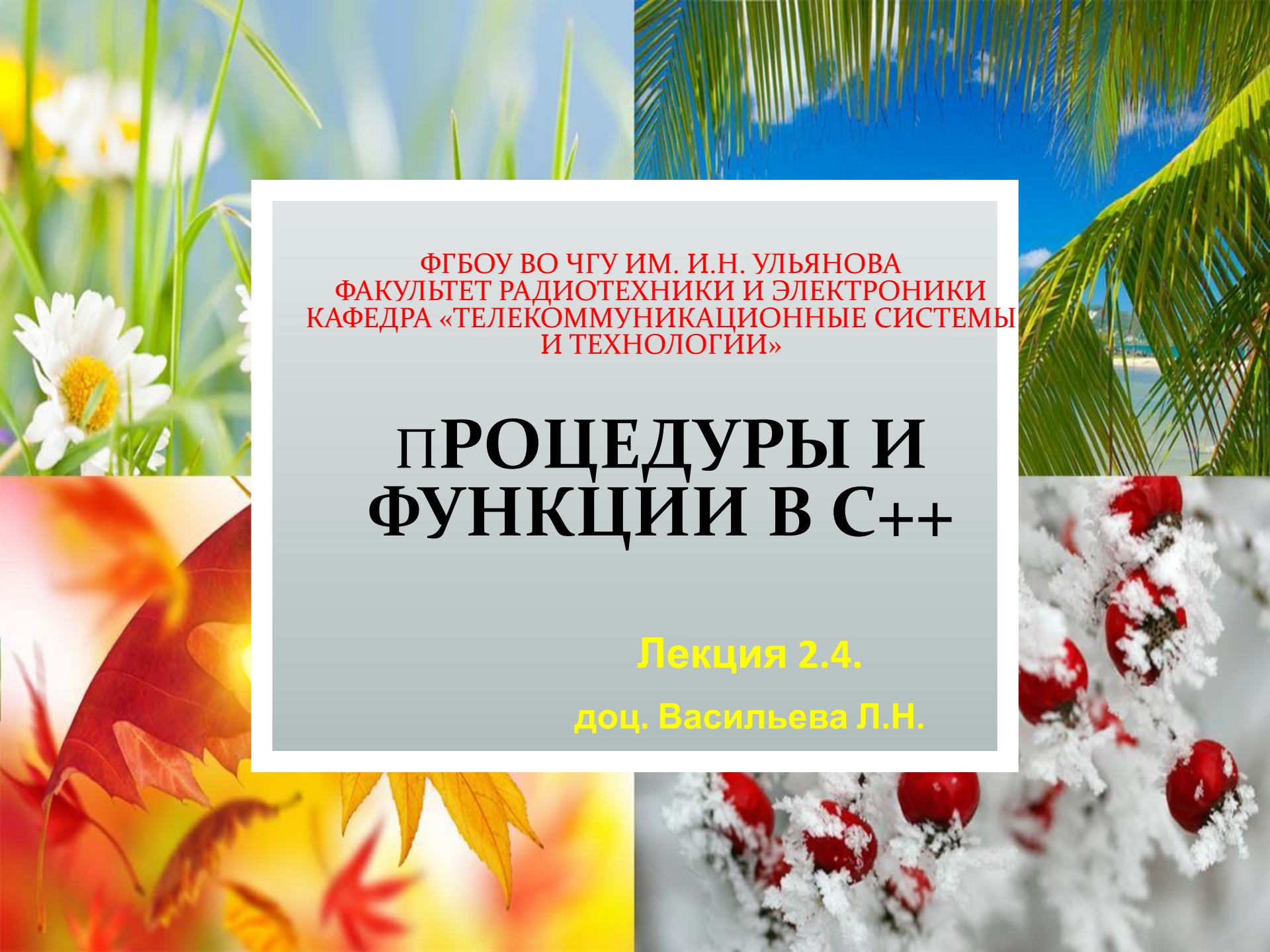


ФГБОУ ВО ЧГУ ИМ. И.Н. УЛЬЯНОВА
ФАКУЛЬТЕТ РАДИОТЕХНИКИ И ЭЛЕКТРОНИКИ
КАФЕДРА «ТЕЛЕКОММУНИКАЦИОННЫЕ СИСТЕМЫ И ТЕХНОЛОГИИ»

ПРОЦЕДУРЫ И ФУНКЦИИ В С++

Лекция 2.4.

доц. Васильева Л.Н.

ПРОЦЕДУРЫ В C++

ЗАЧЕМ НУЖНЫ ПРОЦЕДУРЫ?

```
cout <<  "Ошибка программы";
```

много раз!

```
void Error()
{
   cout << "Ошибка программы";
}
```

```
main()
{
   int n;
   cin >> n;
   if ( n<0 ) Error();
   ...
}
```

вызов процедуры

ЧТО ТАКОЕ ПРОЦЕДУРА?

Процедура – вспомогательный алгоритм, который выполняет некоторые действия.

- текст (расшифровка) процедуры записывается **после основной программы**
- в программе может быть **много процедур**
- чтобы процедура заработала, нужно **вызвать** её по имени из основной программы или из другой процедуры

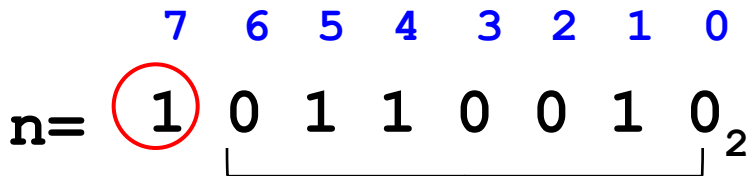
ПРОЦЕДУРА С ПАРАМЕТРАМИ

Задача. Вывести на экран запись целого числа (0..255) в 8-битном двоичном коде.

Алгоритм:

$178 \Rightarrow 10110010_2$

? Как вывести первую цифру?



`n / 128`

`n % 128`

`n1 / 64`

? Как вывести вторую цифру?

Задача. Вывести на экран запись целого числа (0..255) в 8-битном двоичном коде.

Решение:

```
k = 128;
while ( k > 0 )
   {
   cout <<  n / k;
   n = n % k;
   k = k / 2;
   }
```

178 ⇒ 10110010

n	k	вывод
178	128	1

```
void printBin ( int n )
{
   int k;
   k = 128;
   while ( k > 0 )
      {
      cout <<  n / k;
      n = n % k;
      k = k / 2;
      }
}
```

локальные переменные

Параметры – данные, изменяющие работу процедуры.

```
main()
{
   printBin ( 99 );
}
```

значение параметра (**аргумент**)

НЕСКОЛЬКО ПАРАМЕТРОВ

```
void printSred(int a, int b)
{
   cout << (a+b)/2.;
}
```

ИЗМЕНЯЕМЫЕ ПАРАМЕТРЫ

Написать процедуру, которая меняет местами значения двух переменных.

передача по значению

```
void Swap ( int a, int b )
{
  int c;
  c=a; a=b; b=c;
}
```

Процедура работает с копиями переданных значений параметров!

```
main()
{
  int x=2, y=3;
  Swap ( x,y );
  cout << x << " " << y;
}
```

Почему не работает?

```
2 3
```

переменные могут изменяться

```
void Swap ( int & a, int & b )
{
  int c;
  c=a; a=b; b=c;
}
```

передача по **ссылке**

Вызов:

```
int a, b;
Swap(a, b);    // правильно
Swap(2, 3);    // неправильно
Swap(a, b+3); // неправильно
```

ФУНКЦИИ В С++

ЧТО ТАКОЕ ФУНКЦИЯ?

Функция – это вспомогательный алгоритм, который возвращает *значение-результат* (число, символ или объект другого типа).

Задача. Написать функцию, которая вычисляет сумму цифр числа.

Алгоритм:

```
сумма = 0
пока n != 0
    сумма = сумма + n % 10
    n = n / 10
```

СУММА ЦИФР ЧИСЛА

```
int sumDigits ( int n )
{
   int sum = 0;
   while ( n != 0 )
      {
      sum += n % 10;
      n /= 10;
      }
  return sum;
}
```

тип результата

передача результата

```
main()
{
 cout << sumDigits(12345);
}
```

ИСПОЛЬЗОВАНИЕ ФУНКЦИЙ

```
x = 2*sumDigits(n+5);
z = sumDigits(k) + sumDigits(m);
if ( sumDigits(n) % 2 == 0 )
   {
   cout << "Сумма цифр чётная\n";
   cout << "Она равна " << sumDigits(n);
   }
```

Функция, возвращающая целое число, может использоваться везде, где и целая величина!

ЛОГИЧЕСКИЕ ФУНКЦИИ

Задача. Найти все простые числа в диапазоне от 2 до 100.

```
main()
{
   int i;
   for (i = 2; i <= 100; i++)
      if ( isPrime(i) )
         cout <<  i << endl;
}
```

функция, возвращающая логическое значение (true/false)

ФУНКЦИЯ: ПРОСТОЕ ЧИСЛО ИЛИ НЕТ?

```
bool isPrime ( int n )
{
   int count = 0, k = 2;
   while ( k*k <= n && count == 0 )
      {
      if ( n % k == 0 )
          count ++;
      k ++;
      }
   return (count == 0);
}
```

```
if( count == 0 )
      return true;
else return false;
```

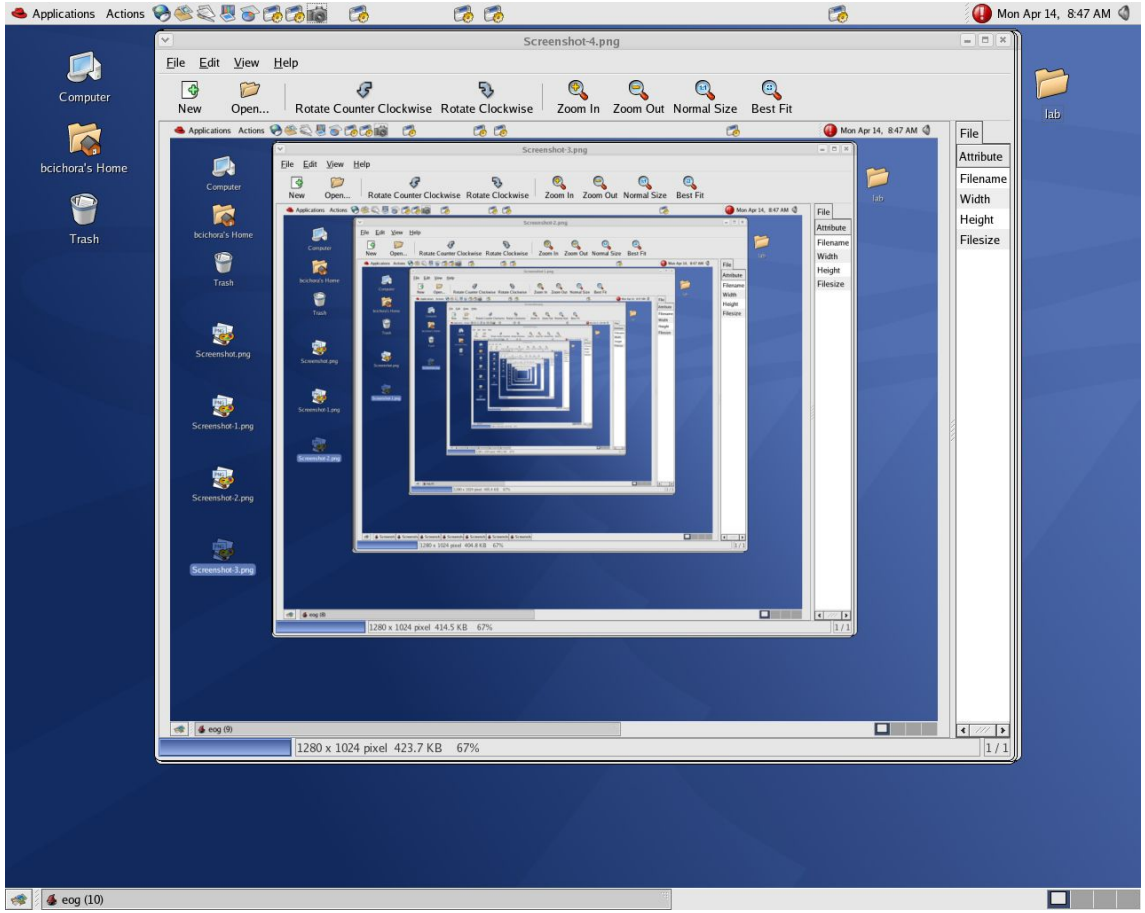
ЛОГИЧЕСКИЕ ФУНКЦИИ: ИСПОЛЬЗОВАНИЕ

Функция, возвращающая логическое значение, может использоваться везде, где и логическая величина!

```
cin >> n;
while ( isPrime(n) )
   {
   cout << "простое число\n";
   cin >> n;
   }
```

РЕКУРСИЯ В С++

ЧТО ТАКОЕ РЕКУРСИЯ?



Натуральные числа:

- 1 – натуральное число
- если n – натуральное число, то $n+1$ – натуральное число

индуктивное определение

Рекурсия — это способ определения множества объектов через само это множество на основе заданных простых базовых случаев.

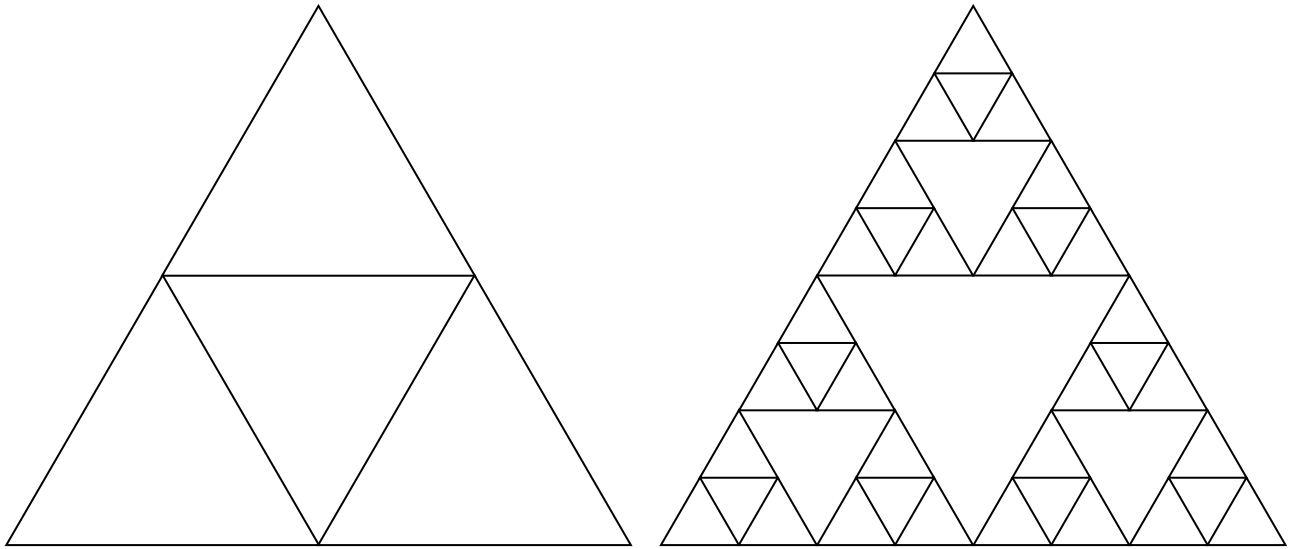
Числа Фибоначчи:

- $F_1 = F_2 = 1$
- $F_n = F_{n-1} + F_{n-2}$ при $n > 2$

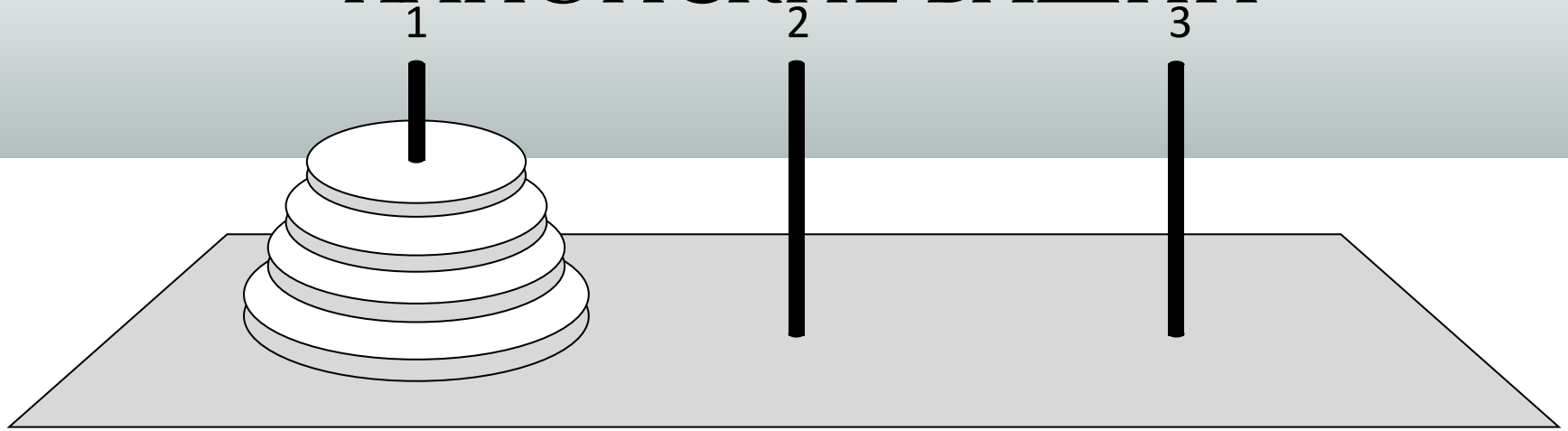
1, 1, 2, 3, 5, 8, 13, 21, 34, …

Фракталы – геометрические фигуры, обладающие самоподобием.

Треугольник Серпинского:



ХАНОЙСКИЕ БАШНИ



- за один раз переносится один диск
- класть только меньший диск на больший
- третий стержень вспомогательный

```
перенести (n, 1, 3)
    перенести (n-1, 1, 2)
    1 -> 3
    перенести (n-1, 2, 3)
```

ХАНОЙСКИЕ БАШНИ – ПРОЦЕДУРА

сколько

откуда

куда

```
void Hanoi ( int n, int k, int m )
{
   int p;
   p = 6 - k - m;
   Hanoi ( n-1, k, p );
   cout << k << " -> " << m << endl;
   Hanoi ( n-1, p, m );
}
```

номер вспомогательного стержня (1+2+3=6!)

рекурсия

рекурсия

? Что плохо?

! **Рекурсия никогда не остановится!**

Рекурсивная процедура (функция) — это процедура (функция), которая вызывает сама себя напрямую или через другие процедуры и функции.

```
void Hanoi ( int n, int k, int m )
{
  int p;
  if ( n == 0 ) return;
  p = 6 - k - m;
  Hanoi ( n - 1, k, p );
  cout << k << " -> " << m << endl;
  Hanoi ( n - 1, p, m );
}
```

условие выхода из рекурсии

```
main()
{
  Hanoi(4, 1, 3);
}
```

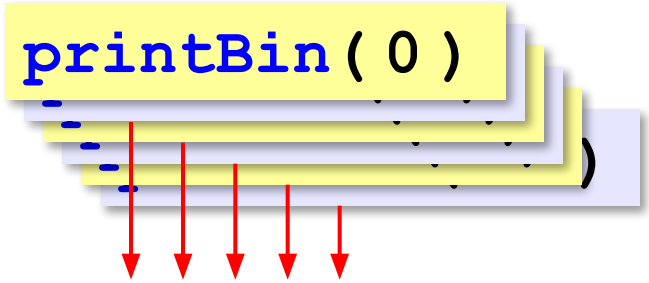
ВЫВОД ДВОИЧНОГО КОДА ЧИСЛА

```
void printBin( int n )
{
  if (n == 0) return;
  printBin(n / 2);
  cout << n % 2;
}
```

условие выхода из рекурсии

напечатать все цифры, кроме последней

вывести последнюю цифру



ВЫЧИСЛЕНИЕ СУММЫ ЦИФР ЧИСЛА

```
int sumDig ( int n )
{
  int sum;
  sum = n %10;
  if ( n >= 10 )
    sum += sumDig ( n / 10 );
  return sum;
}
```

последняя цифра

рекурсивный вызов

```
sumDig(1234)
    4 + sumDig(123)
        4 + 3 + sumDig(12)
            4 + 3 + 2 + sumDig(1)
                4 + 3 + 2 + 1
```

АЛГОРИТМ ЕВКЛИДА

Алгоритм Евклида. Чтобы найти НОД двух натуральных чисел, нужно вычитать из большего числа меньшее до тех пор, пока меньшее не станет равно нулю. Тогда второе число и есть НОД исходных чисел.

```
int NOD ( int a, int b )
{
  if ( a == 0 || b == 0 )
    return a+b;
  if ( a>b )
        return NOD( a-b, b );
  else return NOD( a, b-a );
}
```

условие окончания рекурсии

рекурсивные вызовы

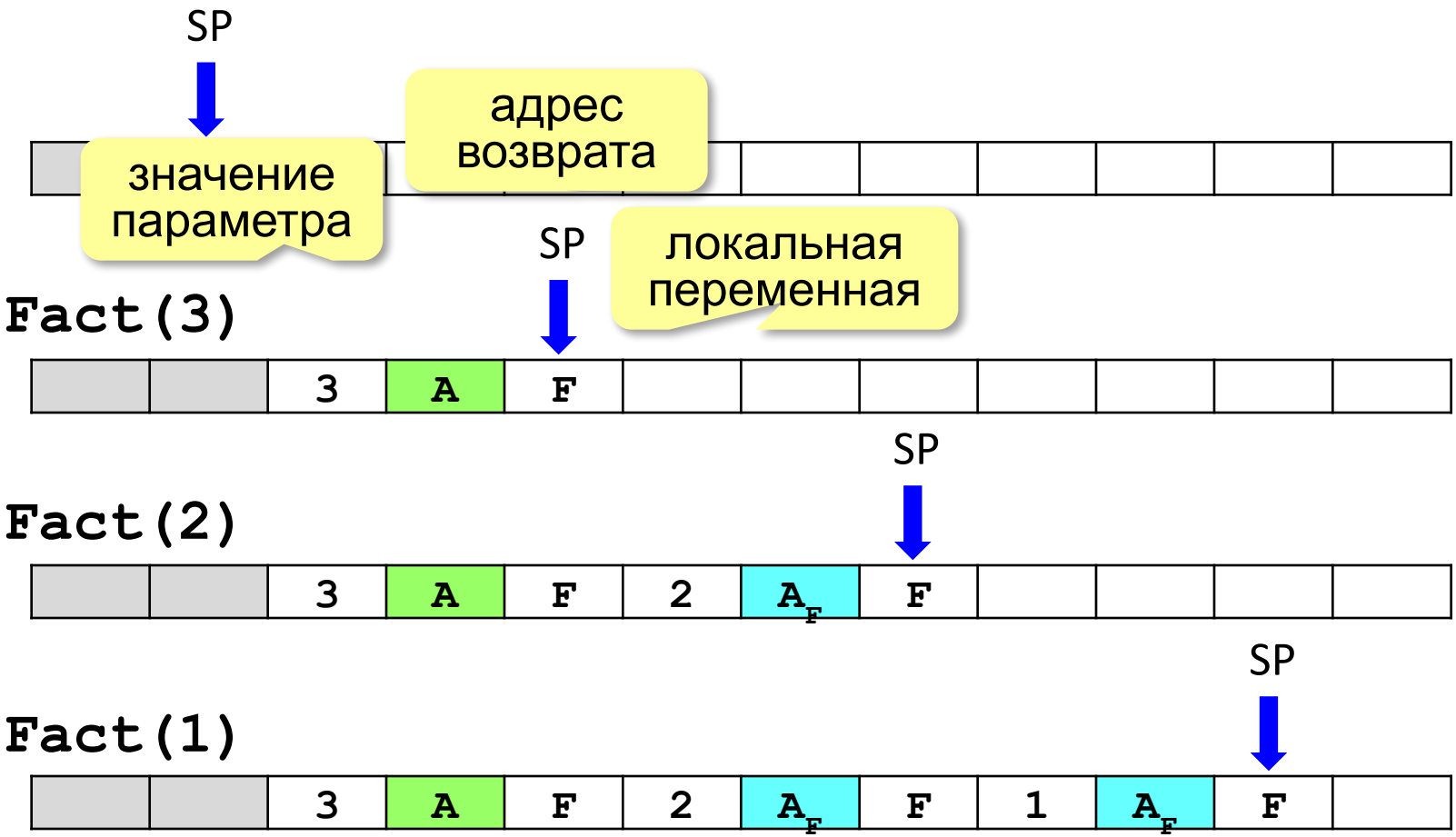
КАК РАБОТАЕТ РЕКУРСИЯ?

Факториал: $$N! = \begin{cases} 1, & N = 1 \\ N \cdot (N-1)!, & N > 1 \end{cases}$$

```
int Fact ( int N )
{
  int F;
  cout << "-> N=" << N << endl;
  if ( N <= 1 )
        F = 1;
  else F = N * Fact(N - 1);
  cout << "<- N=" << N << endl;
  return F;
}
```

```
-> N = 3
  -> N = 2
    -> N = 1
    <- N = 1
  <- N = 2
<- N = 3
```

Стек – область памяти, в которой хранятся локальные переменные и адреса возврата.



РЕКУРСИЯ – «ЗА» И «ПРОТИВ»

- с каждым новым вызовом расходуется память в стеке (возможно переполнение стека)
- затраты на выполнение служебных операций при рекурсивном вызове

- программа становится более короткой и понятной

- возможно переполнение стека
- замедление работы

Любой рекурсивный алгоритм можно заменить нерекурсивным!

итерационный алгоритм

```
int Fact ( int N )
{
 int F;
 F=1;
 for(i=2;i<=N;i++)
   F=F*i;
 return F;
}
```