Алгоритмизация и программирование. Программирование на языке C++

Процедуры.
Функции.
Рекурсия.

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 **перед основной программой**
- в программе может быть **много процедур**
- чтобы процедура заработала, нужно **вызвать** её по имени из основной программы или из другой процедуры

Процедура с параметрами

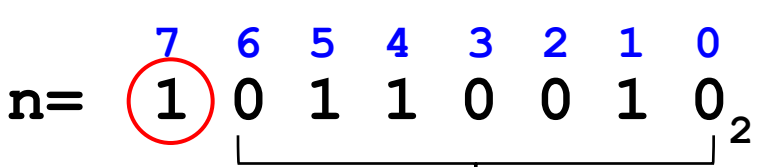
Задача. Вывести на экран запись целого числа (0..255) в 8-битном двоичном коде.

много раз!

Алгоритм:

$178 \Rightarrow 10110010_2$

? Как вывести первую цифру?



`n / 128`

`n % 128`

? Как вывести вторую цифру?

`n1 / 64`

Процедура с параметрами

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

Результат зависит от **n**!

Процедура с параметрами

```
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.;
}
```

Задачи

«A»: Напишите процедуру, которая принимает параметр – натуральное число N – и выводит на экран линию из N символов '–'.

Пример:

```
Введите N:
10
----------
```

«B»: Напишите процедуру, которая выводит на экран в столбик все цифры переданного ей числа, начиная с первой.

Пример:

```
Введите натуральное число:
1234
1
2
3
4
```

Задачи

«С»: Напишите процедуру, которая выводит на экран запись переданного ей числа в римской системе счисления.

Пример:

```
Введите натуральное число:
2013
MMXIII
```

Изменяемые параметры

Задача. Написать процедуру, которая меняет местами значения двух переменных.

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

передача по значению

Процедура работает с копиями переданных значений параметров!

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

Почему не работает?

```
2 3
```

Изменяемые параметры

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
```

Задачи

«A»: Напишите процедуру, которая переставляет три переданные ей числа в порядке возрастания.

Пример:

```
Введите три натуральных числа:
10 15 5
5 10 15
```

«B»: Напишите процедуру, которая сокращает дробь вида M/N. Числитель и знаменатель дроби передаются как изменяемые параметры.

Пример:

```
Введите числитель и знаменатель дроби:
25 15
После сокращения: 5/3
```

Задачи

«С»: Напишите процедуру, которая вычисляет наибольший общий делитель и наименьшее общее кратное двух натуральных чисел и возвращает их через изменяемые параметры.

Пример:

```
Введите два натуральных числа:
10 15
НОД(10,15)=5
НОК(10,15)=30
```

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
```

Функция – это именованная последовательность описаний и операторов, выполняющее какое-либо законченное действие. Функция может принимать параметры и возвращать значение.

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!

Объявление и определение функций

Объявление функции:

```
int sum (int a, int b);
```

Определение функции:

```
int sum (int a, int b)
{
    return (a+b);
}
```

Обмен информации между функциями

При совместной работе функции должны обмениваться информацией. Это можно осуществить с помощью:

- глобальных переменных;
- через параметры;
- через возвращаемое функцией значение.

Пример функции

```
#include <iostream>
int sum (int a, int b); //объявление функции

int main(){
    int a = 2, b = 3, c, d;
    c = sum(a, b); //вызов функции
    cin >> d;
    cout << sum(c, d); //вызов функции
    return 0;
}
```

Возвращаемое значение

Возврат из функции в вызвавшую
ее функцию реализуется оператором return:

```
return [выражение];
```

Примеры:

```
int function_1 {
    return 1;
}

double function_2 {
return 1; //1 преобразуется к типу double
}
```

Параметры функции

Формальные параметры – параметры, перечисленные в заголовке описания функции.

Фактические параметры (аргументы) – параметры, записанные в операторе вызова функции .

Передача параметров функции

```
#include <iostream>
void f(int i, int* j, int& k); //описание функции
int main(){
	int i = 1, j = 2, k = 3;
	cout << “i j k\n”;
	cout << i << ‘ ‘<< j <<  ‘ ‘ << k << ‘\n’;
	f(i, &j, k);
	cout << i << ‘ ‘<< j <<  ‘ ‘ << k;
	return 0;
}
//определение функции
void f(int i, int* j, int& k){
	i++;
	(*j)++;
	k++;
}
```

- по значению;
- по адресу:
  - с использованием указател
  - по ссылке.

Передача массивов в качестве параметров

```
#include <iostream>
int sum(const int* mas, const int n); //описание функции
int const n = 10;
int main{
	int marks[n] = {3, 4, 5, 4, 4};
	cout << “Сумма элементов массива: “ << sum(marks, n);
	return 0;
}
int sum(const int* mas, const int n){ //определение функции
	int s = 0;
	for (int i = 0; i < n; i++)
		s +=mas[i];
	return s;
}
```

Передача массивов в качестве параметров

```
#include <iostream>
int sum(const int* mas, const int n); //описание функции
int const n = 10;
int main{
	int marks[n] = {3, 4, 5, 4, 4};
	cout << “Сумма элементов массива: “ << sum(marks, n);
	return 0;
}
int sum(const int* mas, const int n){ //определение функции
	int s = 0;
	for (int i = 0; i < n; i++)
		s +=mas[i];
	return s;
}
```

Передача имен функций в качестве параметров

```
void f(int a){ //определение функции
    …
}
void (*pf)(int); //указатель на функцию
…
pf = &f; //указателю присваивается адрес функции
pf(10); //функция f вызывается через указатель pf
```

Параметры со значениями по умолчанию

```
int f(int a, int b = 0);
…
f(100);
f(a, 100); //варианты вызова функции f

void f_1(int, int = 100, char* = 0);
…
f_1(a);
f_1(a, 10);
f_1(a, 10, “Hello!”); //варианты вызова функции f_1
```

Функции с переменным числом параметров

```
int printf(const char*, …);
```

Пример:

```
//один параметр
    int printf(“Введите исходные данные”);
//два параметра
    int printf(“Сумма: ”, sum);
//пять параметров
    int printf(“Оценки: ”, mark_1, mark_2, mark_3, mark_4 );
```

Задачи

«A»: Напишите функцию, которая находит наибольший общий делитель двух натуральных чисел.

Пример:

```
Введите два натуральных числа:
7006652 112307574
НОД(7006652,112307574) = 1234.
```

«B»: Напишите функцию, которая определяет сумму цифр переданного ей числа.

Пример:

```
Введите натуральное число:
123
Сумма цифр числа 123 равна 6.
```

Задачи

«С»: Напишите функцию, которая «переворачивает» число, то есть возвращает число, в котором цифры стоят в обратном порядке.

Пример:

```
Введите натуральное число:
1234
После переворота: 4321.
```

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?

? Какой алгоритм?

```
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;
   }
```

Задачи

«А»: Напишите логическую функцию, которая определяет, является ли переданное ей число совершенным, то есть, равно ли оно сумме своих делителей, меньших его самого.

Пример:

```
Введите натуральное число:
28
Число 28 совершенное.
```

Пример:

```
Введите натуральное число:
29
Число 29 не совершенное.
```

Задачи

«B»: Напишите логическую функцию, которая определяет, являются ли два переданные ей числа взаимно простыми, то есть, не имеющими общих делителей, кроме 1.

Пример:

```
Введите два натуральных числа:
28 15
Числа 28 и 15 взаимно простые.
```

Пример:

```
Введите два натуральных числа:
28 16
Числа 28 и 16 не взаимно простые.
```

Задачи

«С»: Простое число называется гиперпростым, если любое число, получающееся из него откидыванием нескольких цифр, тоже является простым. Например, число 733 – гиперпростое, так как и оно само, и числа 73 и 7 – простые. Напишите логическую функцию, которая определяет, верно ли, что переданное ей число – гиперпростое. Используйте уже готовую функцию **`isPrime`**, которая приведена в учебнике.

Пример:

```
Введите натуральное число:
733
Число 733 гиперпростое.
```

Пример:

```
Введите натуральное число:
19
Число 19 не гиперпростое.
```

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, …

Рекурсивные функции

Рекурсивной называется функция, которая вызывает сама себя.

Рекурсия:

- прямая;
- косвенная.

Вычисление факториала:

```
long fact (long n){
        if (n == 0 || n == 1) return 1;
        return (n * fact(n - 1));
}
```

Перегрузка функций

```
//возвращает наибольшее из двух целых
        int max(int, int);
//возвращает подстроку наибольшей длины
        char* max(char*, char*);
//возвращает наибольшее из первого параметра и длины второго
        int max(int, char*);
//возвращает наибольшее из второго параметра и длины первого
        int max(char*, int);
```

Неоднозначность может проявиться при:

- преобразовании типа;
- использовании параметров-ссылок;
- использовании аргументов по умолчанию.

Правила описания перегруженных функций

- Перегруженные функции должны находиться в одной области видимости, иначе произойдёт закрытие аналогично одинаковым именам переменных во вложенных блоках.

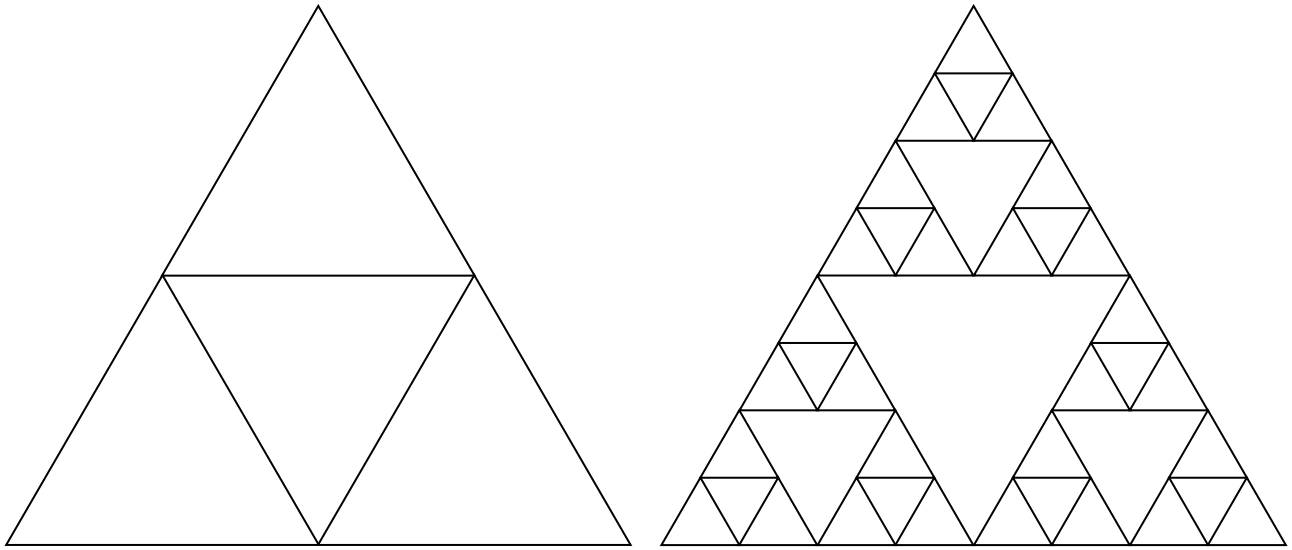
- Перегруженные функции могут иметь параметры по молчанию, при этом значение одного и того же параметра разных функций должны совпадать. В различных вариантах перегруженных функций может быть различное количество параметров по умолчанию.

- Функции не могут быть перегружены, если описание их параметров отличается только модификатором **const** или использованием ссылки.

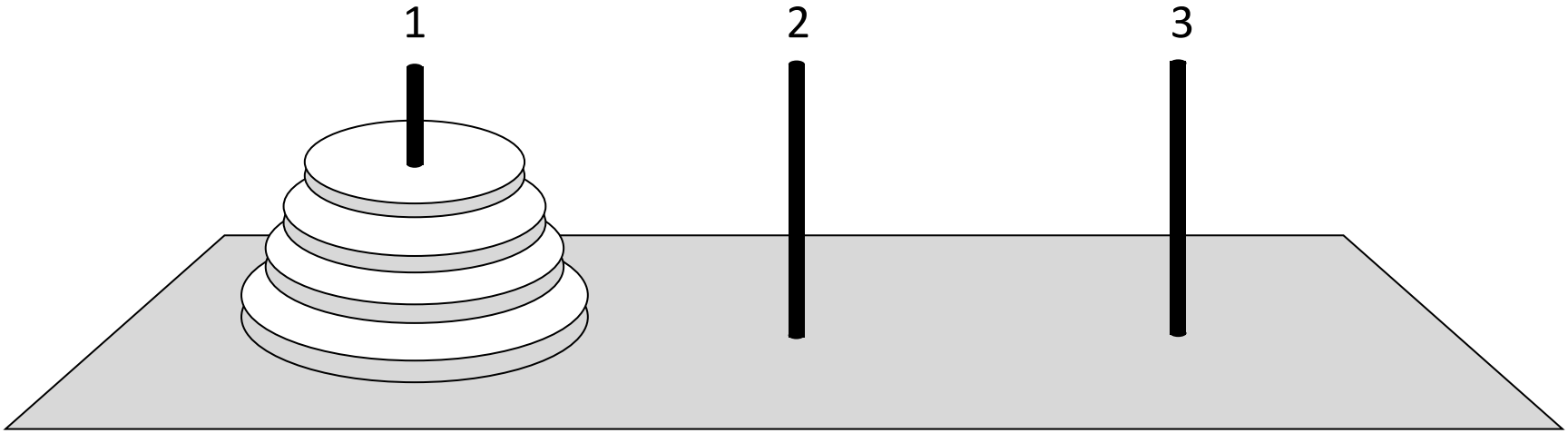
Фракталы

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!**

Ханойские башни – процедура

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);
}
```

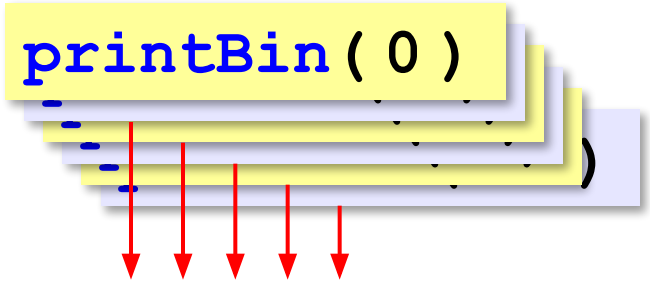
Вывод двоичного кода числа

```
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



? Как без рекурсии?

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

? Где условие окончания рекурсии?

```
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

Задачи

«A»: Напишите рекурсивную функцию, которая вычисляет НОД двух натуральных чисел, используя модифицированный алгоритм Евклида.

Пример:

```
Введите два натуральных числа:
7006652 112307574
НОД(7006652,112307574)=1234.
```

«B»: Напишите рекурсивную функцию, которая раскладывает число на простые сомножители.

Пример:

```
Введите натуральное число:
378
378 = 2*3*3*3*7
```

Задачи

«С»: Дано натуральное число N. Требуется получить и вывести на экран количество всех возможных *различных* способов представления этого числа в виде суммы натуральных чисел (то есть, 1 + 2 и 2 + 1 – это один и тот же способ разложения числа 3). Решите задачу с помощью рекурсивной процедуры.

Пример:

```
Введите натуральное число:
4
Количество разложений: 4.
```

Как работает рекурсия?

Факториал:

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

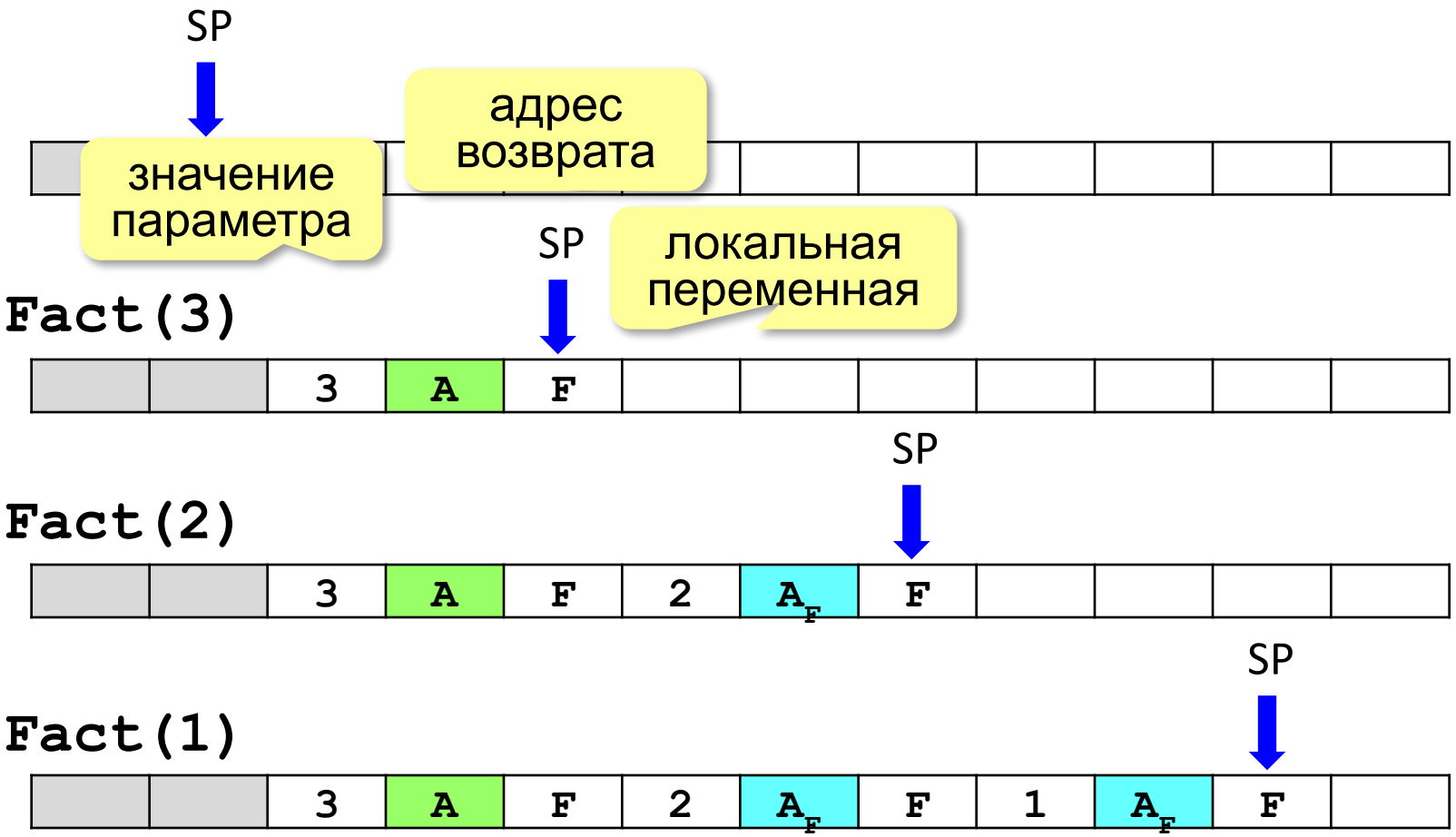
```
int Fact ( int N )
{
  int F;
  cout << "-> N=" << N << endl;
  if ( N <= 1 )
        F = 1;
  else F = N * Fact(N - 1);
  cout << "<- N=" << N << endl;
  return F;
}
```

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

? Как сохранить состояние функции перед рекурсивным вызовом?

Стек

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;
}
```