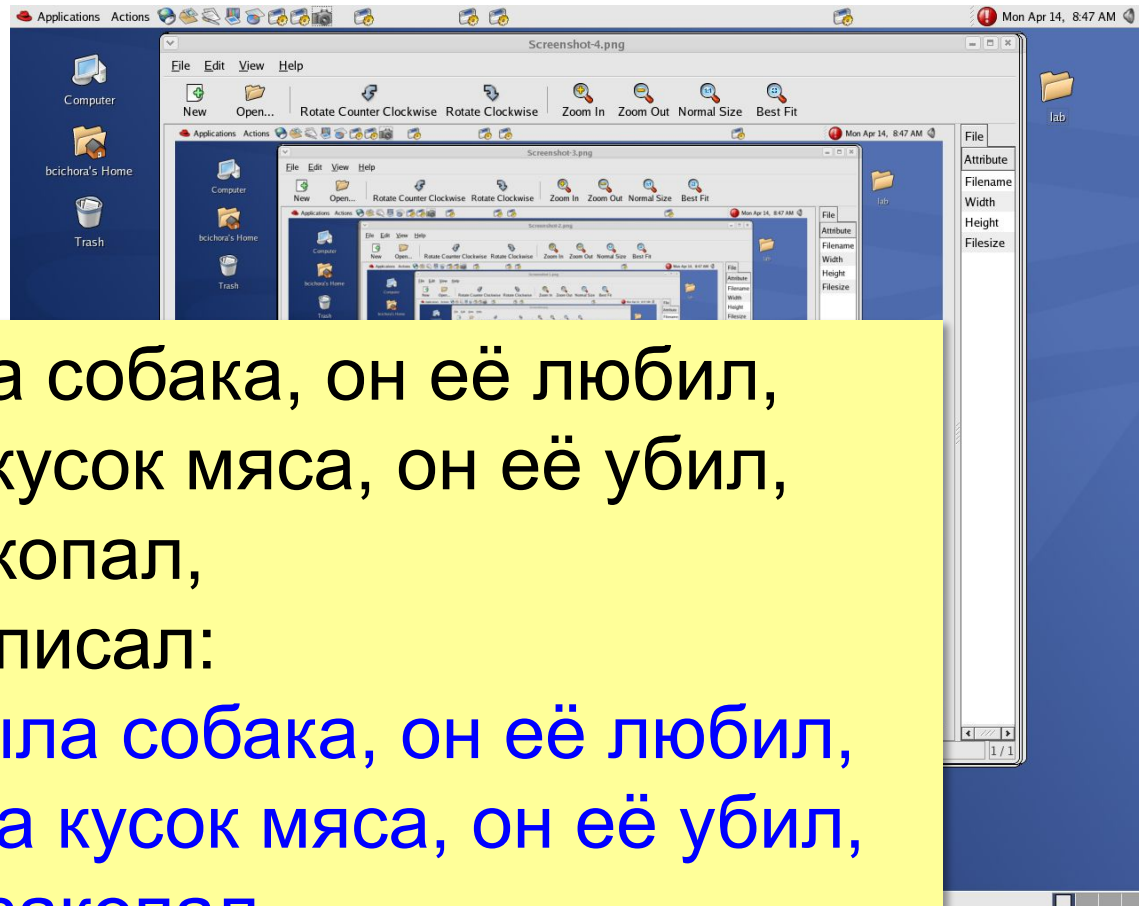
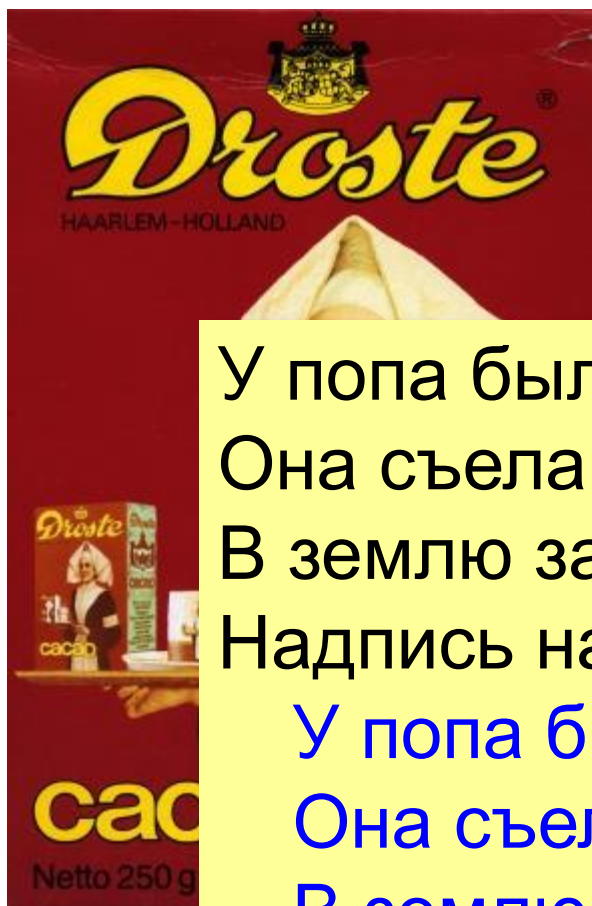


Программирование на языке C++

§ 61. Рекурсия

Что такое рекурсия?



У попа была собака, он её любил,
Она съела кусок мяса, он её убил,
В землю закопал,
Надпись написал:

У попа была собака, он её любил,
Она съела кусок мяса, он её убил,
В землю закопал,
Надпись написал:

...

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:

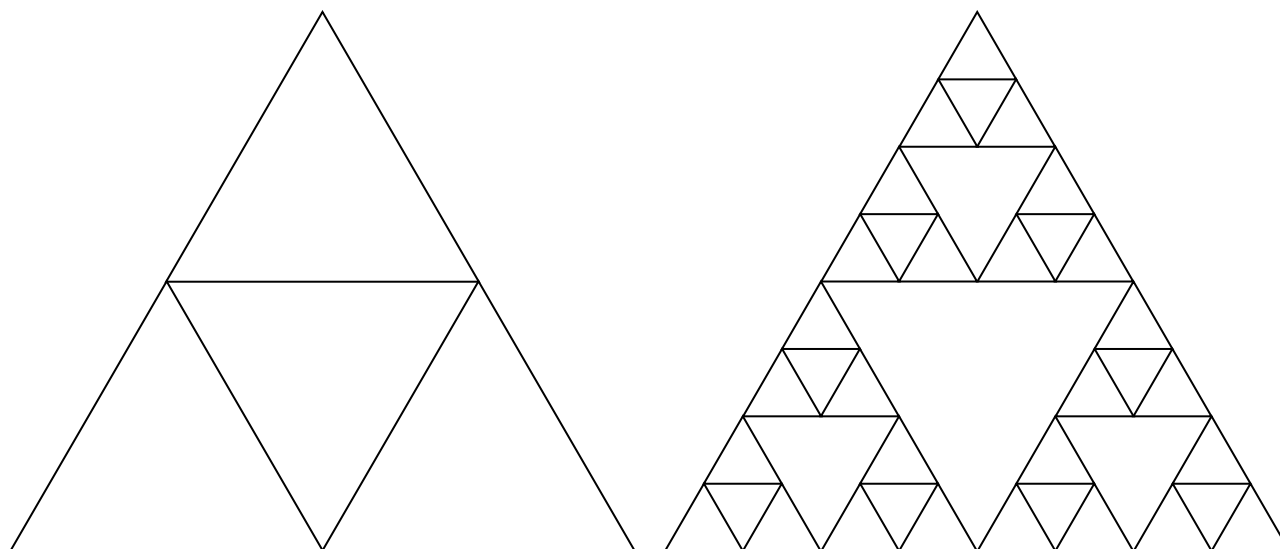
- $F_1 = F_2 = 1$
- $F_n = F_{n-1} + F_{n-2}$ при $n > 2$

1, 1, 2, 3, 5, 8, 13, 21, 34, ...

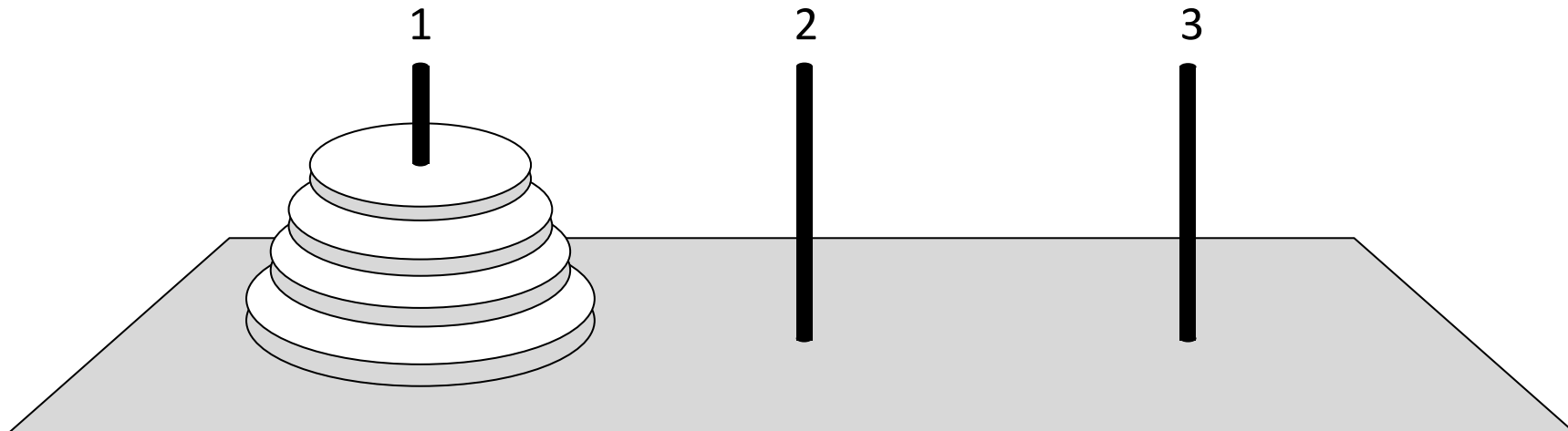
Фракталы

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 );  
}
```

рекурсия

рекурсия

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

?

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

printBin (0)



? Как без рекурсии?

Вычисление суммы цифр числа

```
int sumDig ( int n )
```

```
{
```

```
    int sum;
```

последняя цифра

```
    sum = n % 10;
```

```
    if ( n >= 10 )
```

рекурсивный вызов

```
        sum += sumDig ( n / 10 );
```

```
    return sum;
```

```
}
```

?

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

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

Пример:

Введите два натуральных числа:

7006652 112307574

$\text{НОД}(7006652, 112307574) = 1234$.

«В»: 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;  
}
```

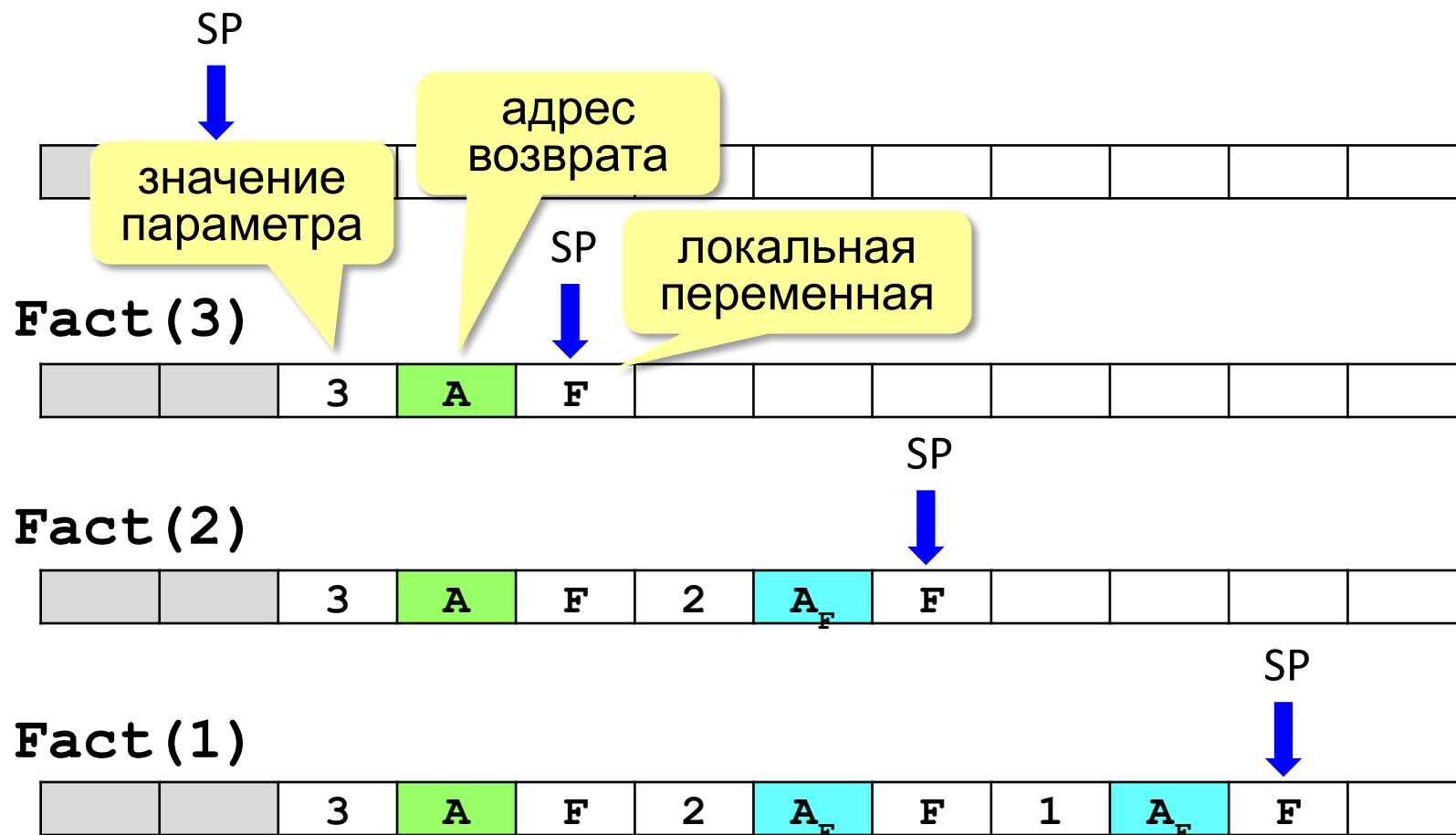
```
-> N = 3  
    -> N = 2  
        -> N = 1  
            <- N = 1  
        <- N = 2  
    <- N = 3
```






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 итерационным!

итерационный
алгоритм

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

Конец фильма

ПОЛЯКОВ Константин Юрьевич

д.т.н., учитель информатики

ГБОУ СОШ № 163, г. Санкт-Петербург

kpolyakov@mail.ru

ЕРЕМИН Евгений Александрович

к.ф.-м.н., доцент кафедры мультимедийной

дидактики и ИТО ПГГПУ, г. Пермь

eremin@pspu.ac.ru

Источники иллюстраций

1. old-moneta.ru
2. www.random.org
3. www.allruletk.ru
4. www.lotterypros.com
5. logos.cs.uic.edu
6. ru.wikipedia.org
7. иллюстрации художников издательства «Бином»
8. авторские материалы