Linguagem C
Recursividade

Sérgio Campos -- scampos@dcc.ufmg.br

Recursividade

- Algoritmo recursivo é aquele que usa a si mesmo!!!
- Só que com parâmetros diferentes:
  - a cada vez menores.
- Idéia:
  - Determinar a solução para n elementos em função da solução para n - 1 elementos (ou m < n)

Recursividade: Fatorial

- n! =
  - 1 se n = 0
  - n * (n-1)! se n > 0

Recursividade: Fatorial

```
#include<stdio.h>

long int fatorial(int n) {
  int f;
  if (n>=1) f = n * fatorial(n-1);
  else      f = 1;
  return(f);
}

int main() {
  int n;
  printf("Entre um número positivo: ");
  scanf("%d",&n);
  printf("Fatorial de %d = %ld\n\n", n, fatorial(n));
  return 0;
}
```

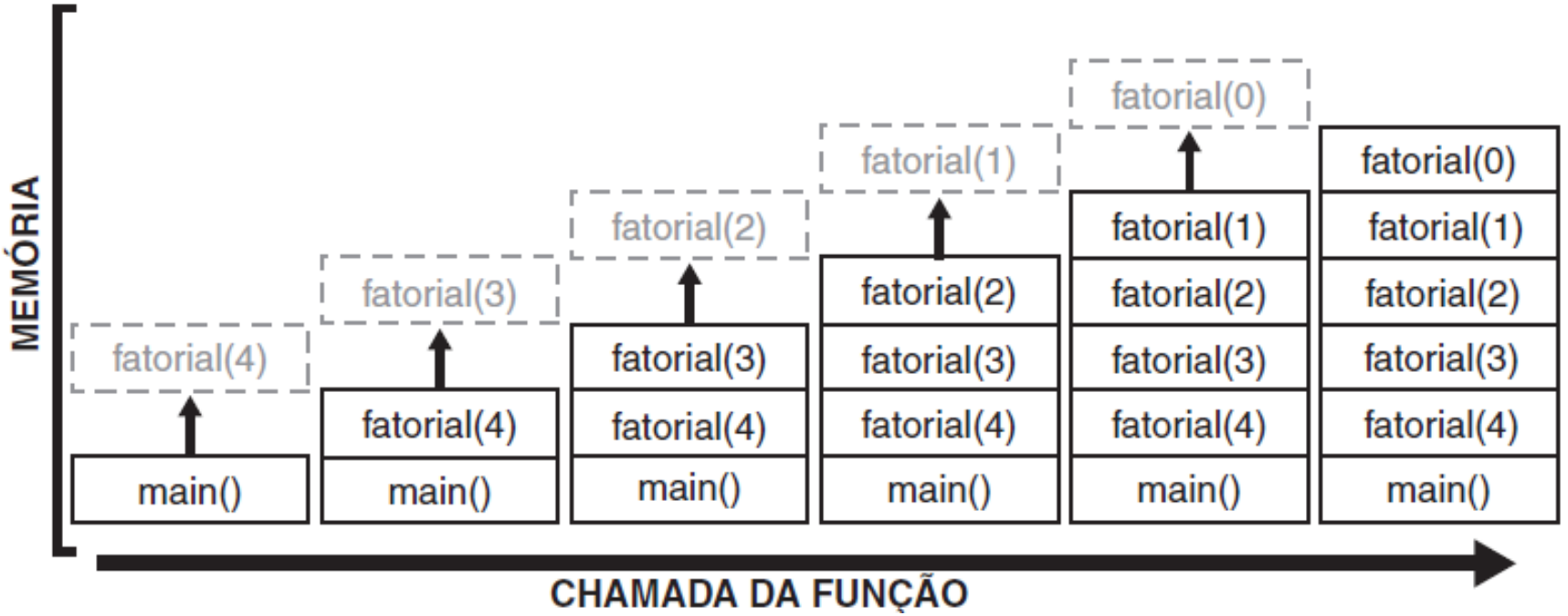
Recursividade: Fatorial

```
#include<stdio.h>

long int fatorial(int n) {
  printf("Cálculo de fat(%d)\n");
  int f;
  if (n>=1) f = n * fatorial(n-1);
  else      f = 1;
  printf("fatorial de %d = %d\n", n, f);
  return(f);
}

int main() {
  int n;
  printf("Entre um número positivo: ");
  scanf("%d",&n);
  printf("Fatorial de %d = %ld\n\n",
    n, fatorial(n));
  return 0;
}
```

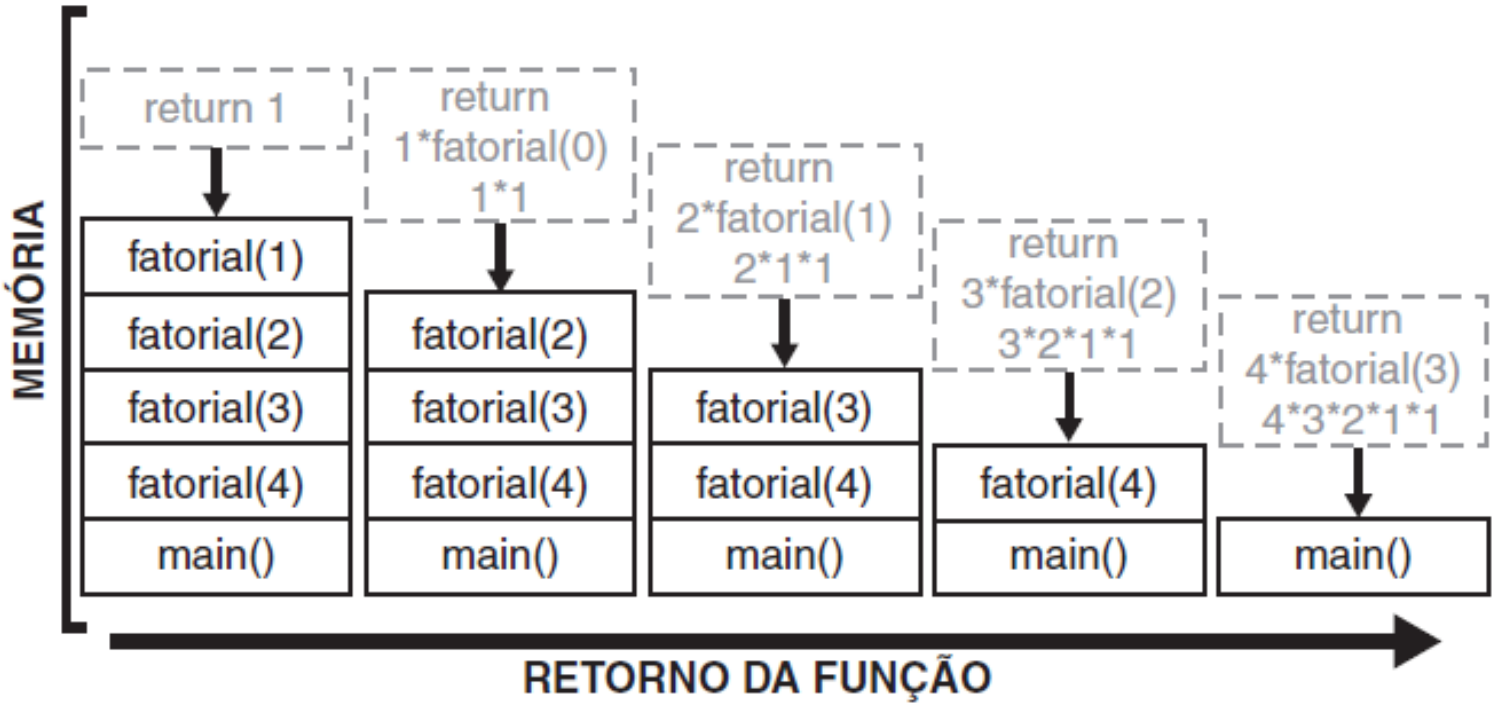
Resultado:

```
Calculo de fat(4)
Calculo de fat(3)
Calculo de fat(2)
Calculo de fat(1)
Calculo de fat(0)
fatorial de 0 = 1
fatorial de 1 = 1
fatorial de 2 = 2
fatorial de 3 = 6
fatorial de 4 = 24
fat(4) = 24
```

Fatorial


MEMÓRIA
fatorial(4)
fatorial(3)
fatorial(2)
fatorial(1)
fatorial(0)
main()
fatorial(4)
fatorial(3)
fatorial(2)
fatorial(1)
main()
fatorial(4)
fatorial(3)
fatorial(2)
main()
fatorial(4)
fatorial(3)
main()
fatorial(4)
main()
CHAMADA DA FUNÇÃO

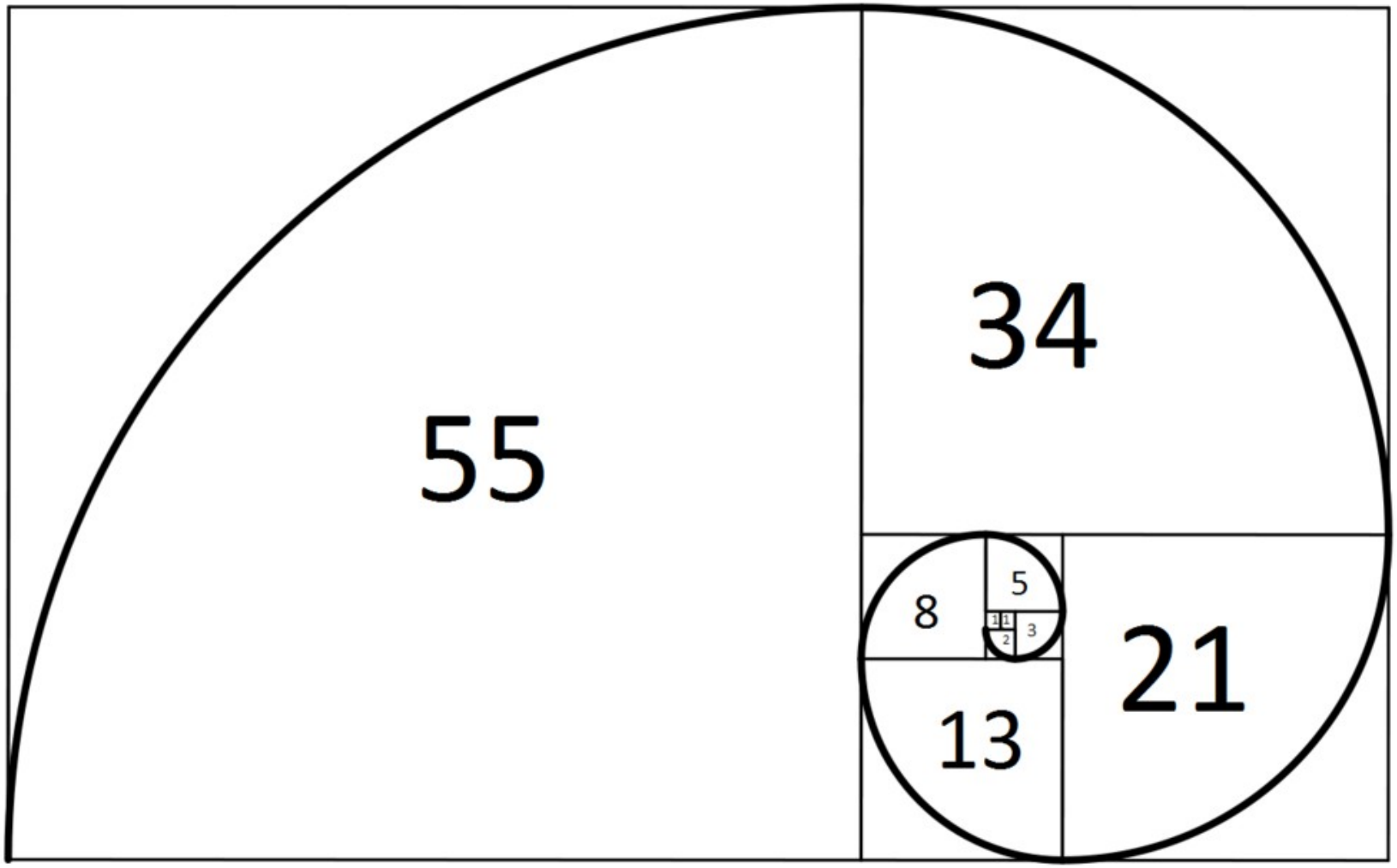
Fatorial


MEMÓRIA
return 1
return 1*fatorial(0) 1*1
return 2*fatorial(1) 2*1*1
return 3*fatorial(2) 3*2*1*1
return 4*fatorial(3) 4*3*2*1*1
fatorial(1)
fatorial(2)
fatorial(3)
fatorial(4)
main()
fatorial(2)
fatorial(3)
fatorial(4)
main()
fatorial(3)
fatorial(4)
main()
fatorial(4)
main()
main()
RETORNO DA FUNÇÃO

Série de Fibonacci

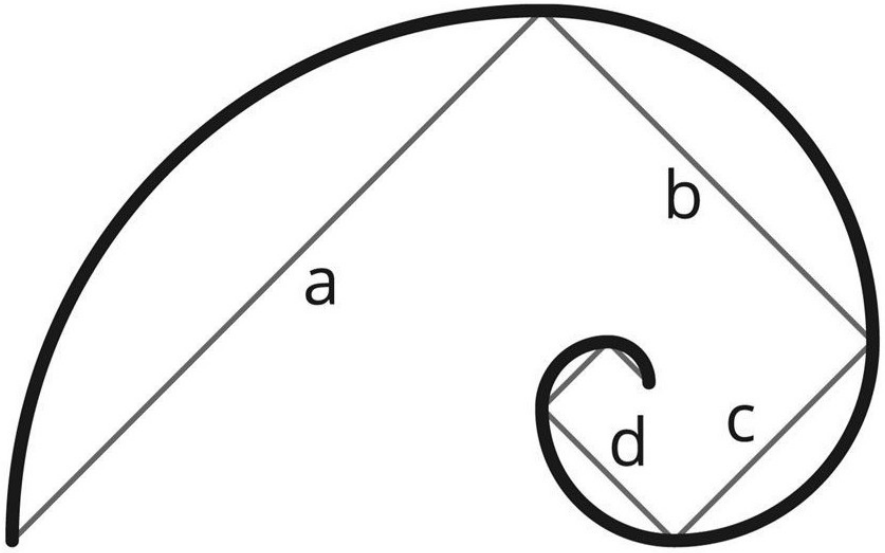
- O i-ésimo número é a soma dos dois anteriores:
  - fib(1)=1;
  - fib(2)=1;
  - fib(i)=fib(i-1) + fib(i-2), i>2
  - 1; 1; 2; 3; 5; 8; ...

Espiral de Fibonacci



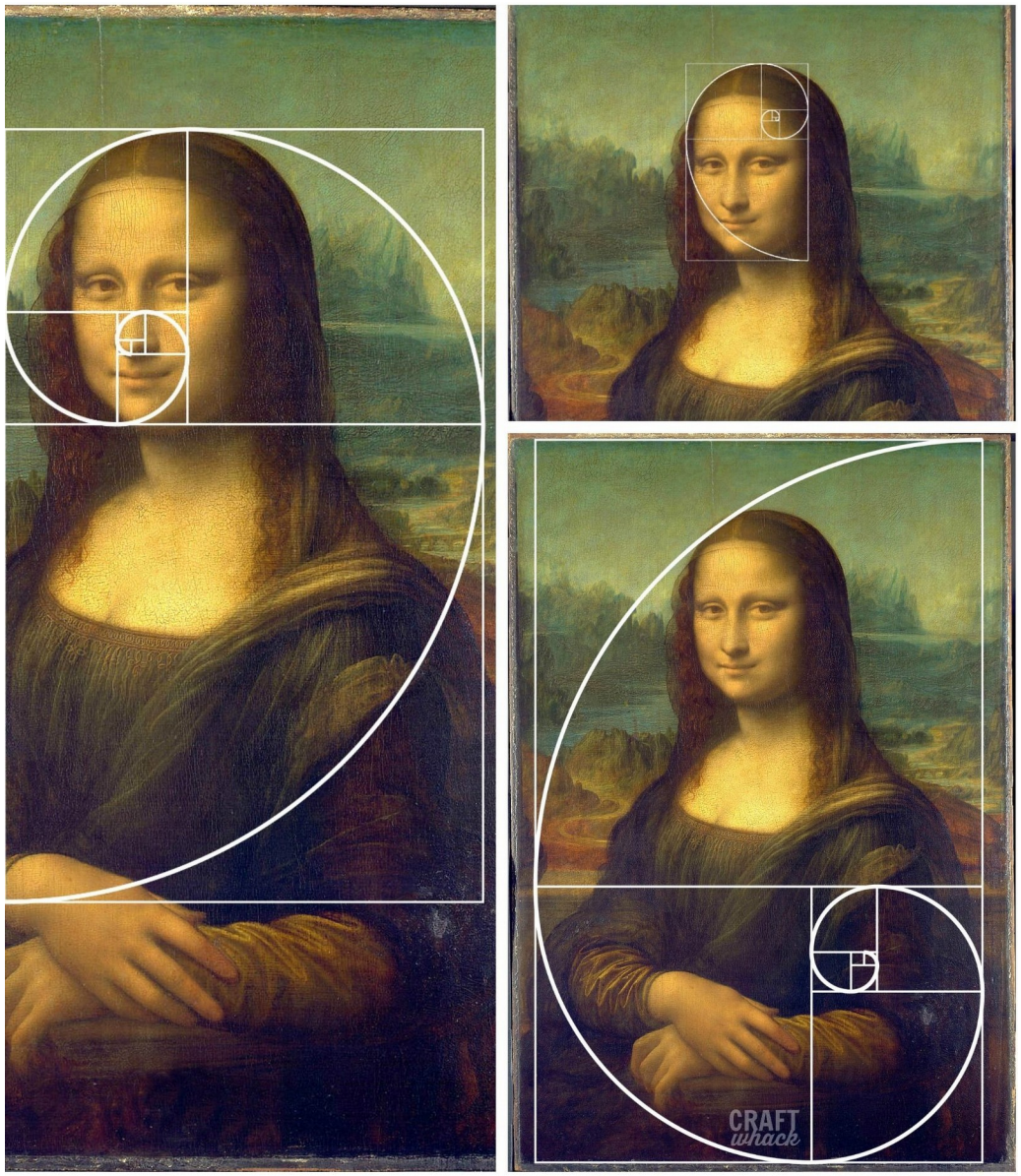
Fibonacci na Natureza

Fibonacci e a Razão Dourada



$$\frac{a+b}{a} \approx \frac{b+c}{b} \approx \frac{c+d}{c} \approx 1{,}618$$

Fibonacci nas Artes



Fibonacci

```
#include<stdio.h>

long int fib(int n) {
  if ((n == 1) || (n == 2))
    return 1;
  else
    return fib(n - 1) + fib(n - 2);
}

int main() {
  int n;

  printf("Entre um número positivo: ");
  scanf("%d",&n);
  printf("Fibonacci de %d = %ld\n\n", n, fib(n));
  return 0;
}
```

Torres de Hanói

- Objetivo: Mover n discos de A para C, um de cada vez, sem que um disco maior fique sobre um menor.

Torres de Hanói

- Diz a lenda que em Hanói, Vietnã existe um mosteiro com 3 torres e 64 discos em uma delas
- A tarefa dos monges é mover os discos da primeira torre para a terceira usando as regras vistas
- E, quando eles concluírem o trabalho...
  - O MUNDO ACABA!!!!!

Torres de Hanói

```
#include <stdio.h>
void movedisco(int origem, int destino) {
  printf("%d -> %d\n", origem, destino);
}
void movetorre(int altura, int de, int para, int uso)
{
  if (altura > 0) {
    movetorre(altura-1, de, uso, para);
    movedisco(de, para);
    movetorre(altura-1, uso, para, de);
  };
}
int main () {
  int n;
  printf("Entre com o número de discos: ");
  scanf("%d", &n);
  movetorre(n, 1, 3, 2);
}
```

Torres de Hanói

- Entrada: 3
- Saída:

1 -> 3
1 -> 2
3 -> 2
1 -> 3
2 -> 1
2 -> 3
1 -> 3

Torres de Hanói

- É possível calcular quantos passos são necessários para mover os discos:
  - 2^n-1
- Se cada movimentação de disco gastar 1 segundo, para 64 discos o tempo será:
  - 585 bilhões de anos!
- UFA!!!...

Custo da Recursividade

- Variáveis locais são geradas dinâmicamente a cada chamada do subprograma.
  - Alto custo em termos de memória!

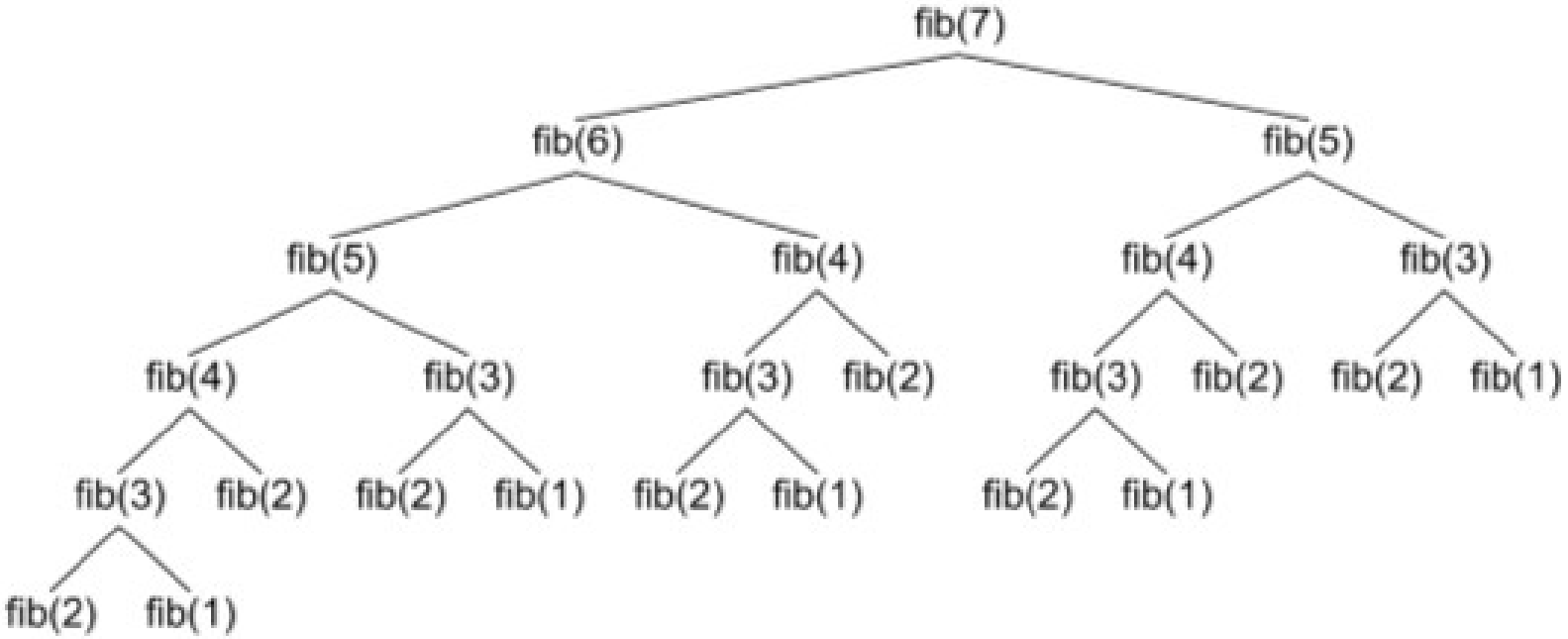
Fibonacci

```
#include<stdio.h>

long int fib(int n) {
  if ((n == 1) || (n == 2))
    return 1;
  else
    return fib(n - 1) + fib(n - 2);
}

int main() {
  int n;

  printf("Entre um número positivo: ");
  scanf("%d",&n);
  printf("Fibonacci de %d = %ld\n\n", n, fib(n));
  return 0;
}
```

A cada chamada 2 variáveis são alocadas.

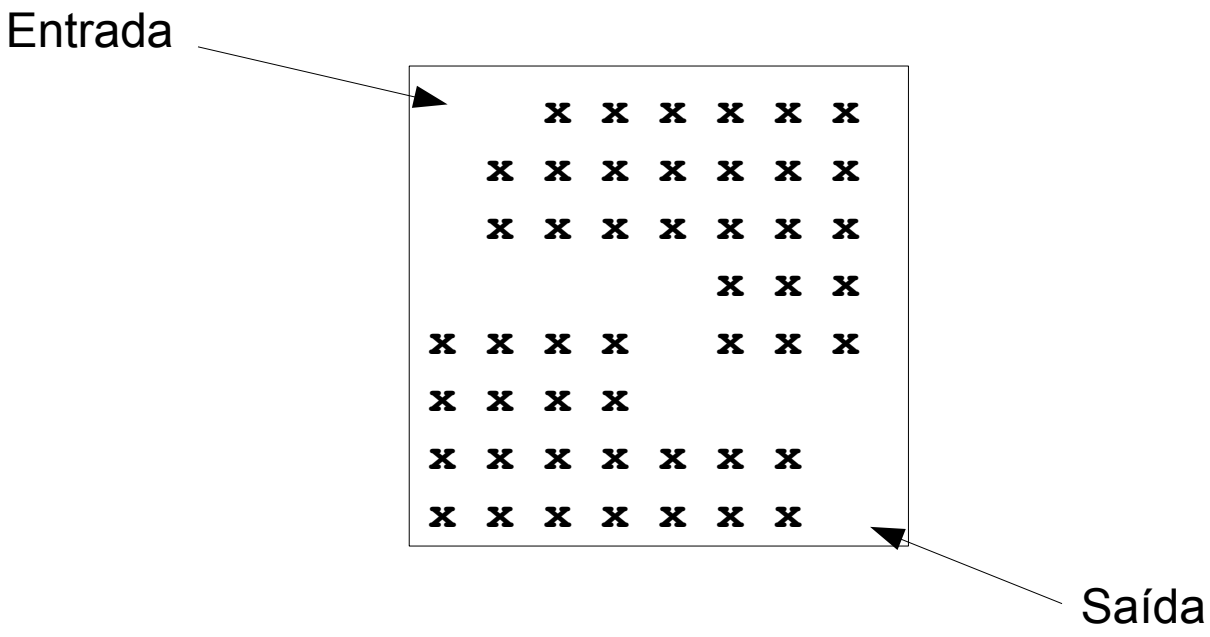
Custo da Recursividade

- Mas... o número de chamadas é grande!



Labirinto

O labirinto consiste de:



Labirinto

```
#include <stdio.h>
#include <stdbool.h>

#define W (8) /* labyrinth width */
#define H (8) /* labyrinth height */

void printMaze(char maze[W][H]) {
  for (int i = 0; i < W; i++) {
    for (int j = 0; j < H; j++) {
      printf("%c ", maze[i][j]);
    }
    putchar('\n');
  }
}
```

Labirinto

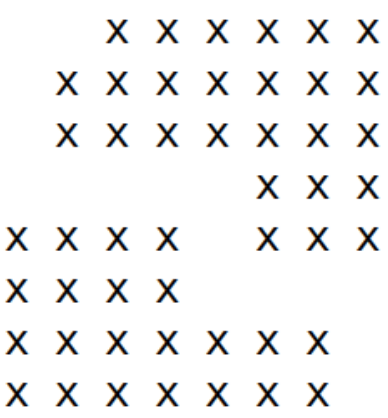
```
bool findPath(char maze[W][H], int x, int y) {
    maze[x][y] = '!';
    if (x == W - 1 && y == H - 1) return true;

    if (x + 1 < W && maze[x + 1][y] == ' ')
        if (findPath(maze, x + 1, y)) return true;

    if (x - 1 >= 0 && maze[x - 1][y] == ' ')
        if (findPath(maze, x - 1, y)) return true;

    if (y + 1 < H && maze[x][y + 1] == ' ')
        if (findPath(maze, x, y + 1)) return true;

    if (y - 1 >= 0 && maze[x][y - 1] == ' ')
        if (findPath(maze, x, y - 1)) return true;

    maze[x][y] = ' ';
    return false;
}
```

Labirinto

```
int main(void) {
    char maze[W][H] = {
        {' ', ' ', 'x', 'x', 'x', 'x', 'x', 'x'},
        {' ', 'x', 'x', 'x', 'x', 'x', 'x', 'x'},
        {' ', 'x', 'x', 'x', 'x', 'x', 'x', 'x'},
        {' ', ' ', ' ', ' ', ' ', 'x', 'x', 'x'},
        {'x', 'x', 'x', 'x', ' ', 'x', 'x', 'x'},
        {'x', 'x', 'x', 'x', ' ', ' ', ' ', ' '},
        {'x', 'x', 'x', 'x', 'x', 'x', 'x', ' '},
        {'x', 'x', 'x', 'x', 'x', 'x', 'x', ' '},
    };
    printMaze(maze);
    if (findPath(maze, 0, 1)) {
        printf("Maze completed!\n");
        printMaze(maze);
    } else {
        printf("No path found!");
    }
}
```

Labirinto

O labirinto consiste de:



```
Maze completed!
 ! ! x x x x x x
 ! x x x x x x x
 ! x x x x x x x
 ! ! ! ! ! x x x
 x x x x ! x x x
 x x x x ! ! ! !
 x x x x x x x !
 x x x x x x x !
```