

Funções - Recursão

Sumário

- Funções
- Recursão

Funções

- Uma função é um conjunto (bloco) de comandos situado fora do programa principal e associado a um nome
 - O bloco de comandos é executado no momento em que função é chamada por meio de seu nome
 - A função pode ser chamada quantas vezes for necessário
 - Vantagens:
 - Melhor organização do programa
 - Reutilização de código
- Em C, o programa principal também é uma função, chamada **main**

Funções

- Quando uma função é chamada:
 - o programa chamador é suspenso e seus dados são empilhados na memória;
 - o bloco de comandos da função é executado;
 - a função termina e retorna para o programa chamador;
 - o programa chamador continua sua execução a partir da próxima instrução.

Funções - Parâmetros

- Uma função pode **receber valores** de entrada através de seus **parâmetros**
 - Também chamados de parâmetros formais
 - A lista de parâmetros (que pode ser vazia) aparece entre () junto ao nome da função
 - Na chamada de uma função, passamos valores (argumentos) para os respectivos parâmetros

função Python

```
1  def addition(a, b):  
2  |      return a + b
```

função C

```
1  int addition(int a, int b){  
2  |      return a + b;  
3  }
```

Funções - Retorno

- Uma função pode **retornar resultados** ao programa que a chamou
 - Instrução **return**
 - Após a execução do return, a função termina
 - O tipo de retorno é especificado na definição da função

função Python

```
1  def addition( a , b ):  
2  |      return a + b
```

função C

```
1  int addition (int a, int b){  
2  |      return a + b;  
3  }
```

Funções - Retorno

- Uma função pode **retornar resultados** ao programa que a chamou
 - Uma função void não retorna nada

função Python

```
1 def funcTeste( a , b ):  
2     print("Recebeu A: %s e B: %s"% (a,b));
```

função C

```
1 void funcTeste (int a, int b){  
2     printf("Recebeu A: %d e B: %d", a,b);  
3 }
```

Funções

- Definição de função
 - Sintaxe

```
1  tipo_retorno nomeFuncao (tipo paramentro, tipo parametro2){  
2      //bloco  
3  }
```

Qualquer tipo: int, float, double, char, struct, ponteiro

- Se **declarar** um **tipo** precisa de **retorno**
- Se colocar **void** não **retorna nada**

```
1  int nomeFuncao (tipo paramentro, tipo parametro2){  
2      //bloco  
3      return valore;  
4  }
```


Funções

- Definição de função
 - Sintaxe

```
1  tipo_retorno nomeFuncao (tipo parametro, tipo parametro2){  
2  |    //bloco  
3  }
```

O nome não pode conter caracteres especiais nem ser uma palavra reservada.

Funções

- Definição de função
 - Sintaxe

```
1  tipo_retorno nomeFuncao (tipo parametro, tipo parametro2) {  
2  |    //bloco  
3  | }
```

Aceita quantos parâmetros forem necessários, sempre seguindo o padrão:
tipo nomeVar

Pode-se deixar vazio ou declarar void se não for necessário nenhum parâmetro

```
1  void nomeFuncao () {  
2  |    //bloco  
3  |    printf("função sem parametro");  
4  |    return; //opcional  
5  | }
```

Funções

- Exemplo
 - A função recebe um valor inteiro e retorna o mesmo elevado ao quadrado
 - return de um valor do tipo int

```
1  #include <stdio.h>
2
3  int elevaAoQuadrado (int val){
4      return val*val;
5  }
6
7  int main(){
8      int a, resultado;
9
10     scanf("%d", &a); //lê valor inteiro
11
12     resultado = elevaAoQuadrado(a); //chamda de função
13
14     printf("%d ^ 2 = %d\n", a, resultado);
15
16     return (0);
17 }
```

Funções

- Exemplo

- A função **parOuImpar** recebe um valor inteiro e diz se ele é par ou ímpar
- O resultado é impresso (printf)
- A função não retorna nada
 - Não possui return
 - Tipo de retorno void

```
1  #include <stdio.h>
2
3  void parOuImpar (int x){
4      if ((x % 2) == 0){
5          printf("Par!\n");
6      } else {
7          printf("Impar!\n");
8      }
9  }
10
11 int main(){
12     int a, resultado;
13
14     scanf("%d", &a); //lê valor inteiro
15     while (a > 0) {
16         parOuImpar(a);
17         scanf("%d", &a); //lê valor inteiro
18     }
19
20     return (0);
21 }
```

Funções

- Exemplo

- A função **imprimeData** recebe como parâmetro um valor do tipo **struct data** e não retorna nada
- A função **constroiData** recebe 3 inteiros e retorna um valor do tipo **struct data**

```
4 struct data {
5     int dia;
6     int mes;
7     int ano;
8 };
9
10 void imprimeData(struct data d) {
11     printf("Dia: %d\n", d.dia);
12     printf("Mes: %d\n", d.mes);
13     printf("Ano: %d\n", d.ano);
14 }
15
16 struct data constroiData(int dia, int mes, int ano) {
17     struct data d;
18     d.dia = dia;
19     d.mes = mes;
20     d.ano = ano;
21     return d;
22 }
23
24 int main() {
25     struct data feriado = constroiData(25, 12, 2022);
26     imprimeData(feriado);
27     return 0;
28 }
```

Funções

- Escopo: regras de visibilidade das variáveis no programa
 - Variáveis locais
 - Declaradas dentro de um bloco
 - Não podem ser usadas ou modificadas fora do respectivo bloco, pois somente existem enquanto este estiver sendo executado
 - Parâmetros formais de funções
 - Comportam-se como variáveis locais da função
 - Variáveis globais
 - Declaradas fora de qualquer bloco
 - São acessíveis e modificáveis em qualquer parte do programa que venha após sua declaração

```
1  #include <stdio.h>
2
3  //declaração de variáveis globais
4
5  void funcao1(parâmetros)
6  {
7      // declaração das variáveis locais da função1
8
9      return;
10 }
11
12 int main()
13 {
14     //declaração das variáveis locais da main()
15
16     return 0;
17 }
```

Funções - Exercícios

1. Crie uma função que receba 2 números inteiros e retorne o maior valor.
2. Crie uma função que receba 3 números inteiros e retorne o maior valor, utilizando uma chamada para função anterior.
3. Crie um programa de conversão entre temperaturas Celsius e Fahrenheit.
 - a. Primeiro, o usuário deve escolher se vai entrar com a temperatura em Celsius ou Fahrenheit, depois a conversão escolhida é realizada.
 - b. Se C é a temperatura em Celsius e F em Fahrenheit, as fórmulas de conversão são:
 - i. $C = 5.(F-32)/9$
 - ii. $F = (9.C/5) + 32$

Sumário

- Funções
- **Recursão**

Recursividade

- Processo de definir algo utilizando a si mesmo
- Alguns problemas podem ser resolvidos com base na solução de instâncias menores do próprio problema



Recursividade

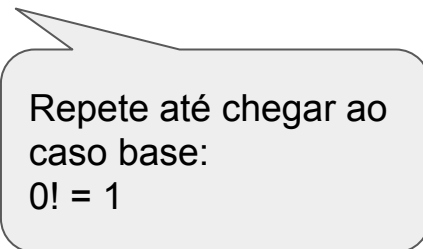
- Exemplo: Fatorial

- Se já conheço $15!$, como calcular $16!$?

$$16! = 16 * 15!$$

- Se já conheço $(n-1)!$, como calcular $n!$?

$$n! = n * (n-1)!$$



Repete até chegar ao
caso base:
 $0! = 1$

Pilha de Execução

- Para melhor compreender a recursividade, vamos entender o que ocorre internamente na memória quando funções são chamadas
 - Você aprenderá isso com mais detalhes em disciplinas futuras
- A memória de um programa contém uma estrutura especial chamada **pilha de execução**, onde cada elemento representa uma função
 - existe também um ponteiro para onde o programa está executando, além de uma área para armazenamento das variáveis (globais e locais)
- Assim que o programa for iniciado, a função *main* é colocada nesta estrutura
- Cada função que for chamada será colocada no topo da pilha
 - Ao ser finalizada, a função sai da pilha e o programa passa a executar a função anterior (que agora está novamente no topo da pilha)

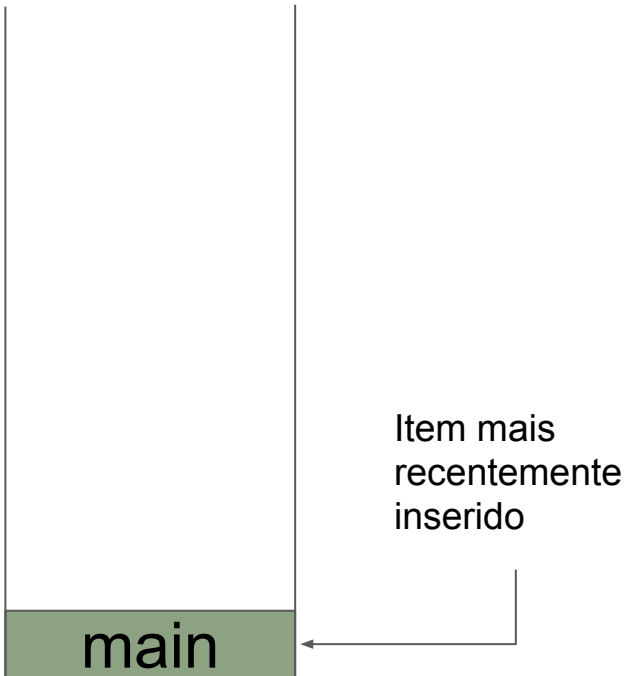
Pilha de Execução

- Exemplo

```
#include <stdio.h>
int soma(int a, int b){
    int r = a+b;
    return r;
}
int dif(int a, int b){
    return a-b;
}
int melega (int a, int b, int c){
    return soma(a, dif(b,c));
}
int main(){
    int s, d, m;
    s = soma(3,2);
    d = dif(3,2);
    m = melega(3,1,5);
    return 0;
}
```



memória

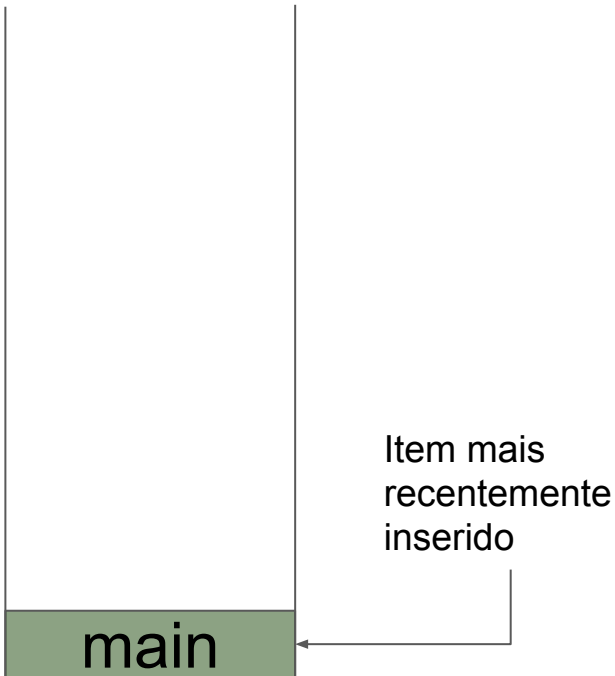


Pilha de Execução

- Exemplo

```
#include <stdio.h>
int soma(int a, int b){
    int r = a+b;
    return r;
}
int dif(int a, int b){
    return a-b;
}
int meleca (int a, int b, int c){
    return soma(a, dif(b,c));
}
int main(){
    int s, d, m;
    s = soma(3,2);
    d = dif(3,2);
    m = meleca(3,1,5);
    return 0;
}
```

memória

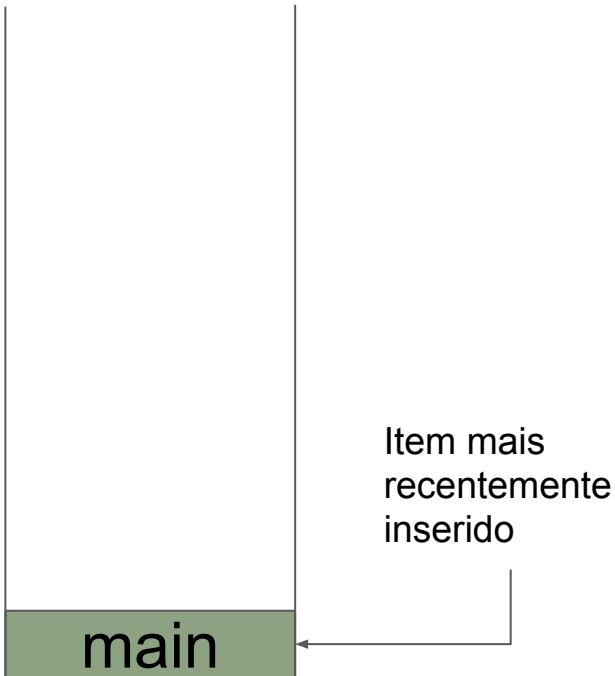


Pilha de Execução

- Exemplo

```
#include <stdio.h>
int soma(int a, int b){
    int r = a+b;
    return r;
}
int dif(int a, int b){
    return a-b;
}
int meleca (int a, int b, int c){
    return soma(a, dif(b,c));
}
int main(){
    int s, d, m;
    s = soma(3,2);
    d = dif(3,2);
    m = meleca(3,1,5);
    return 0;
}
```

memória



Pilha de Execução

- Exemplo

```
#include <stdio.h>
int soma(int a, int b){
    int r = a+b;
    return r;
}
int dif(int a, int b){
    return a-b;
}
int meleca (int a, int b, int c){
    return soma(a, dif(b,c));
}
int main(){
    int s, d, m;
    s = soma(3,2);
    d = dif(3,2);
    m = meleca(3,1,5);
    return 0;
}
```

memória

Item mais
recentemente
inserido

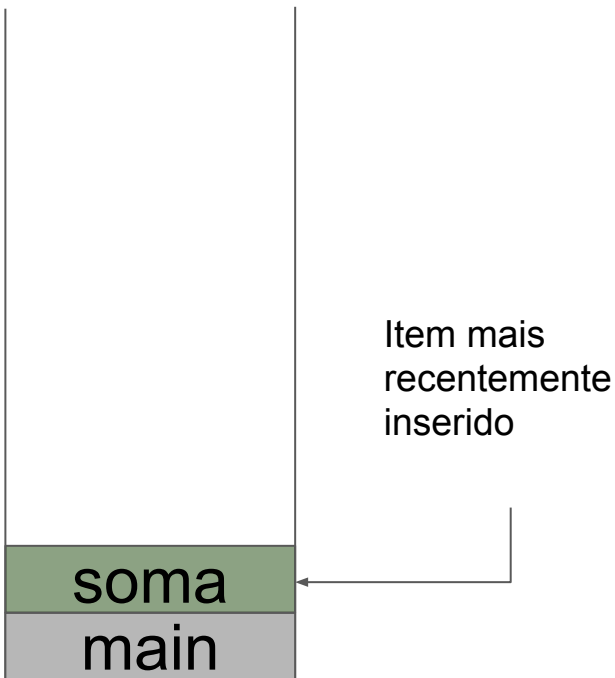
main

Pilha de Execução

- Exemplo

```
#include <stdio.h>
int soma(int a, int b){
    int r = a+b;
    return r;
}
int dif(int a, int b){
    return a-b;
}
int meleca (int a, int b, int c){
    return soma(a, dif(b,c));
}
int main(){
    int s, d, m;
    s = soma(3,2);
    d = dif(3,2);
    m = meleca(3,1,5);
    return 0;
}
```

memória

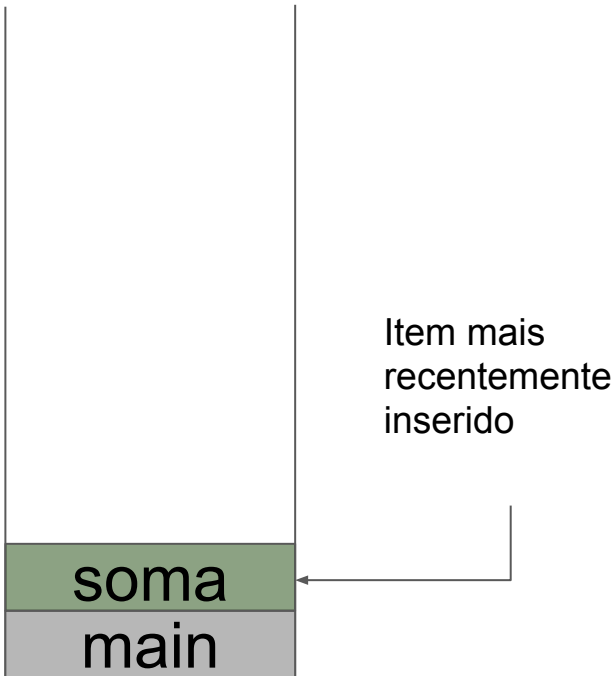


Pilha de Execução

- Exemplo

```
#include <stdio.h>
int soma(int a, int b){
    int r = a+b;
    return r;
}
int dif(int a, int b){
    return a-b;
}
int meleca (int a, int b, int c){
    return soma(a, dif(b,c));
}
int main(){
    int s, d, m;
    s = soma(3,2);
    d = dif(3,2);
    m = meleca(3,1,5);
    return 0;
}
```

memória

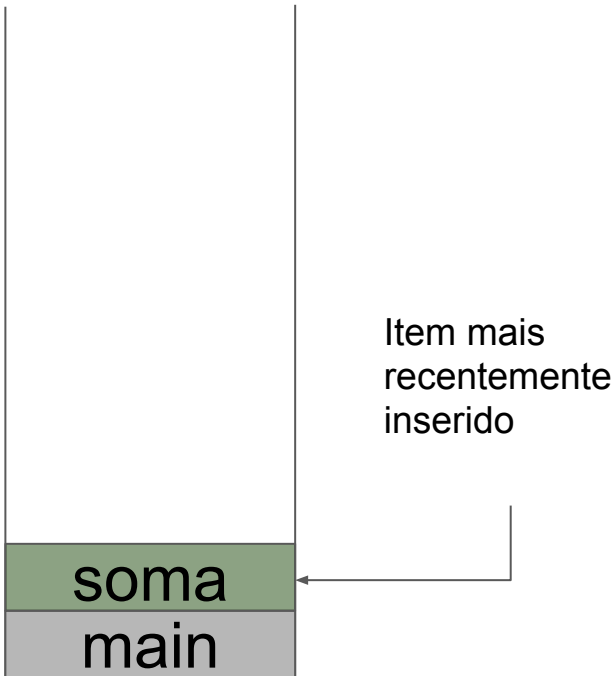


Pilha de Execução

- Exemplo

```
#include <stdio.h>
int soma(int a, int b){
    int r = a+b;
    return r;
}
int dif(int a, int b){
    return a-b;
}
int meleca (int a, int b, int c){
    return soma(a, dif(b,c));
}
int main(){
    int s, d, m;
    s = soma(3,2);
    d = dif(3,2);
    m = meleca(3,1,5);
    return 0;
}
```

memória



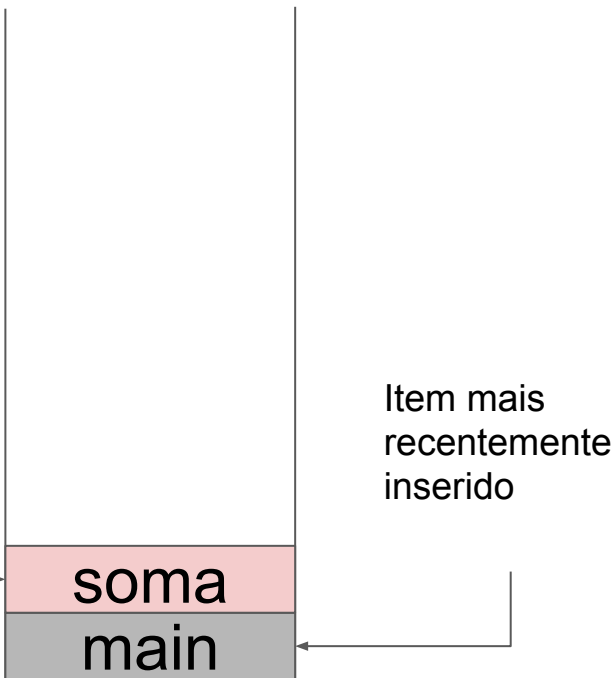
Pilha de Execução

- Exemplo

```
#include <stdio.h>
int soma(int a, int b){
    int r = a+b;
    return r;
}
int dif(int a, int b){
    return a-b;
}
int meleca (int a, int b, int c){
    return soma(a, dif(b,c));
}
int main(){
    int s, d, m;
    s = soma(3,2);
    d = dif(3,2);
    m = meleca(3,1,5);
    return 0;
}
```

Remove da
memória e
volta de onde
parou

memória



Pilha de Execução

- Exemplo

```
#include <stdio.h>
int soma(int a, int b){
    int r = a+b;
    return r;
}
int dif(int a, int b){
    return a-b;
}
int meleca (int a, int b, int c){
    return soma(a, dif(b,c));
}
int main(){
    int s, d, m;
    s = soma(3,2);
    d = dif(3,2);
    m = meleca(3,1,5);
    return 0;
}
```

memória

Item mais
recentemente
inserido

main

Pilha de Execução

- Exemplo

```
#include <stdio.h>
int soma(int a, int b){
    int r = a+b;
    return r;
}
int dif(int a, int b){
    return a-b;
}
int meleca (int a, int b, int c){
    return soma(a, dif(b,c));
}
int main(){
    int s, d, m;
    s = soma(3,2);
    d = dif(3,2);
    m = meleca(3,1,5);
    return 0;
}
```

memória

Item mais
recentemente
inserido

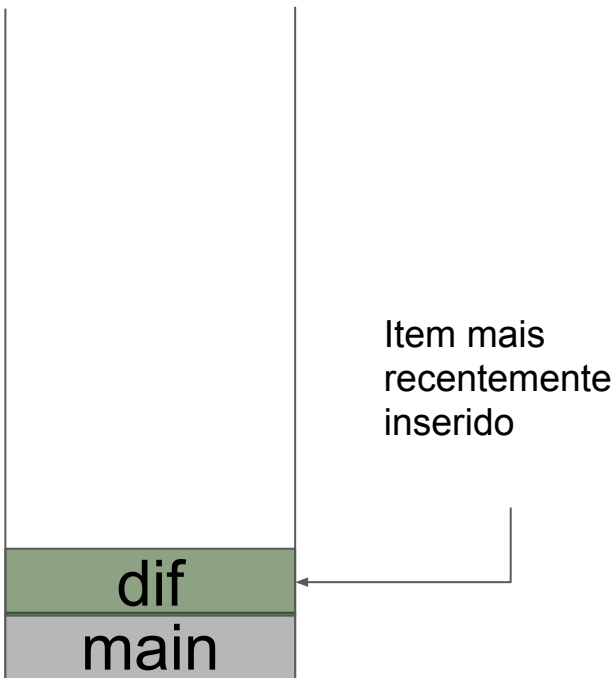
main

Pilha de Execução

- Exemplo

```
#include <stdio.h>
int soma(int a, int b){
    int r = a+b;
    return r;
}
int dif(int a, int b){
    return a-b;
}
int meleca (int a, int b, int c){
    return soma(a, dif(b,c));
}
int main(){
    int s, d, m;
    s = soma(3,2);
    d = dif(3,2);
    m = meleca(3,1,5);
    return 0;
}
```

memória



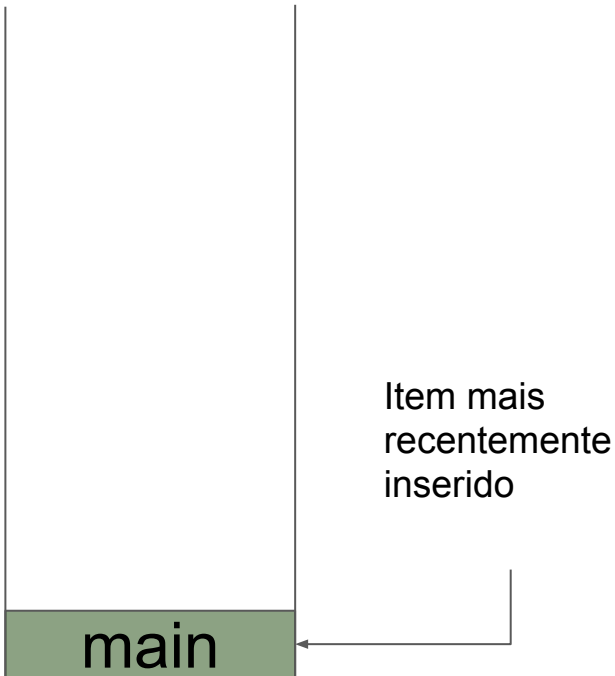
Pilha de Execução

- Exemplo

```
#include <stdio.h>
int soma(int a, int b){
    int r = a+b;
    return r;
}
int dif(int a, int b){
    return a-b;
}
int meleca (int a, int b, int c){
    return soma(a, dif(b,c));
}
int main(){
    int s, d, m;
    s = soma(3,2);
    d = dif(3,2);
    m = meleca(3,1,5);
    return 0;
}
```



memória

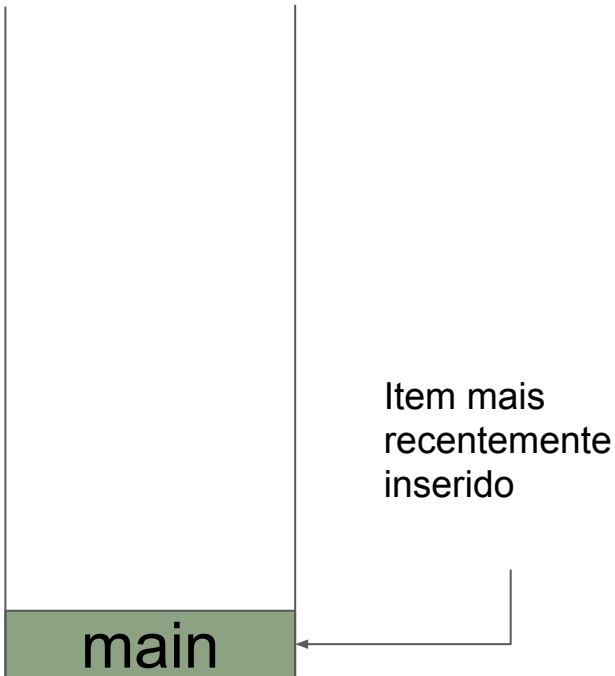


Pilha de Execução

- Exemplo

```
#include <stdio.h>
int soma(int a, int b){
    int r = a+b;
    return r;
}
int dif(int a, int b){
    return a-b;
}
int melega (int a, int b, int c){
    return soma(a, dif(b,c));
}
int main(){
    int s, d, m;
    s = soma(3,2);
    d = dif(3,2);
    m = melega(3,1,5);
    return 0;
}
```

memória

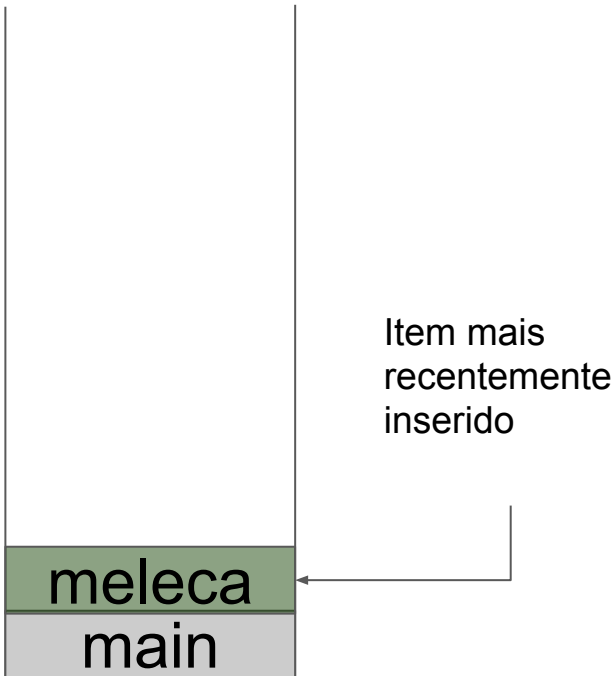


Pilha de Execução

- Exemplo

```
#include <stdio.h>
int soma(int a, int b){
    int r = a+b;
    return r;
}
int dif(int a, int b){
    return a-b;
}
int meleca (int a, int b, int c){
    return soma(a, dif(b,c));
}
int main(){
    int s, d, m;
    s = soma(3,2);
    d = dif(3,2);
    m = meleca(3,1,5);
    return 0;
}
```

memória

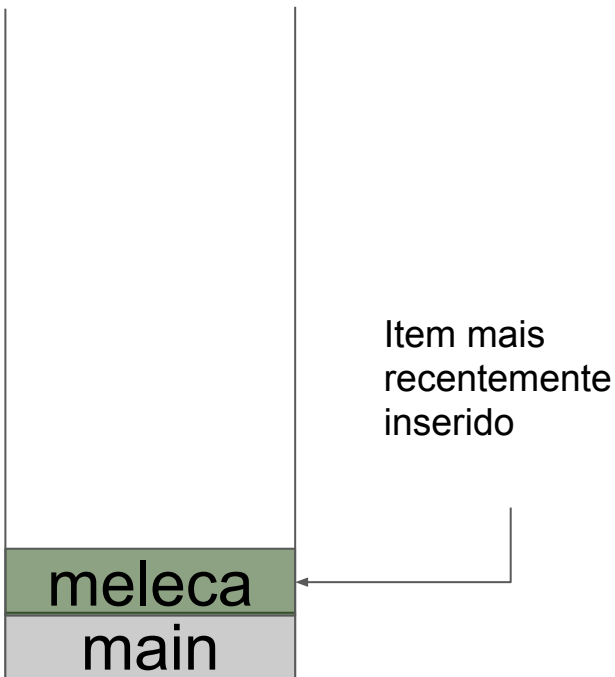


Pilha de Execução

- Exemplo

```
#include <stdio.h>
int soma(int a, int b){
    int r = a+b;
    return r;
}
int dif(int a, int b){
    return a-b;
}
int meleca (int a, int b, int c){
    return soma(a, dif(b,c));
}
int main(){
    int s, d, m;
    s = soma(3,2);
    d = dif(3,2);
    m = meleca(3,1,5);
    return 0;
}
```

memória

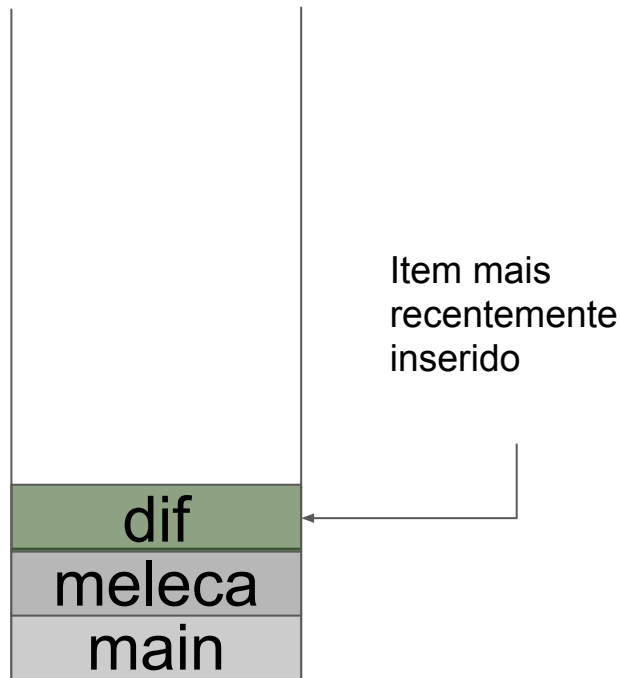


Pilha de Execução

- Exemplo

```
#include <stdio.h>
int soma(int a, int b){
    int r = a+b;
    return r;
}
int dif(int a, int b){
    return a-b;
}
int meleca (int a, int b, int c){
    return soma(a, dif(b,c));
}
int main(){
    int s, d, m;
    s = soma(3,2);
    d = dif(3,2);
    m = meleca(3,1,5);
    return 0;
}
```

memória

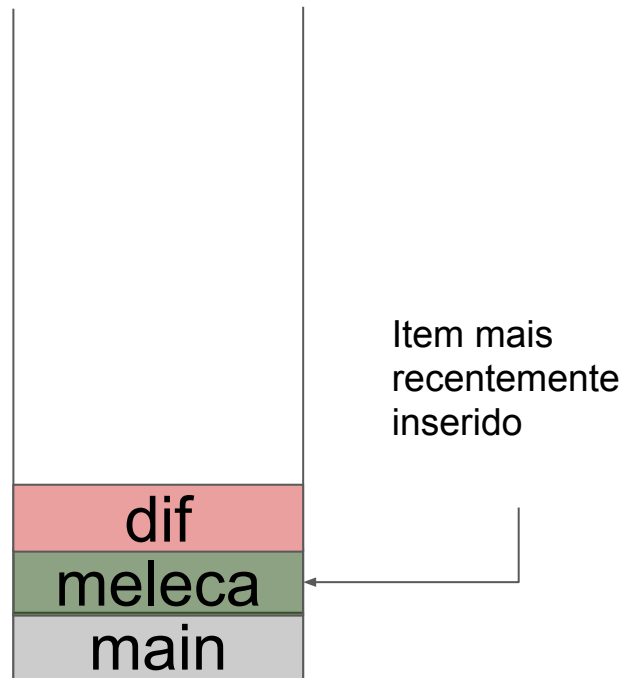


Pilha de Execução

- Exemplo

```
#include <stdio.h>
int soma(int a, int b){
    int r = a+b;
    return r;
}
int dif(int a, int b){
    return a-b;
}
int meleca (int a, int b, int c){
    → return soma(a, dif(b,c));
}
int main(){
    int s, d, m;
    s = soma(3,2);
    d = dif(3,2);
    → m = meleca(3,1,5);
    return 0;
}
```

memória

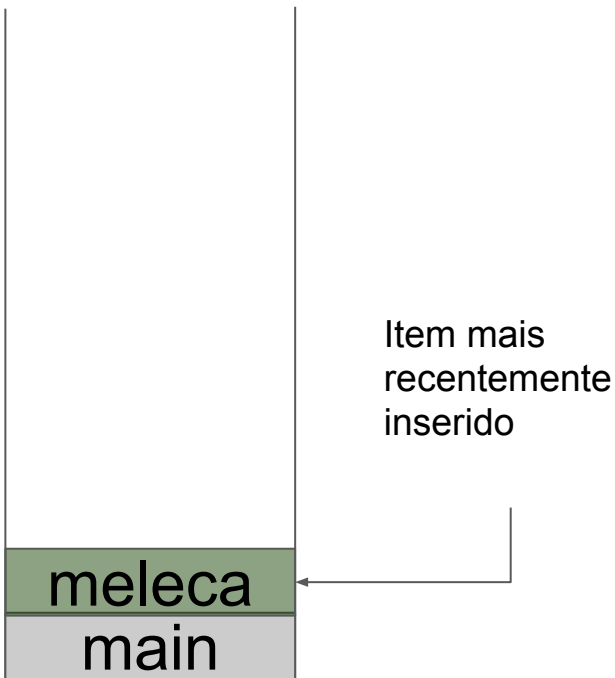


Pilha de Execução

- Exemplo

```
#include <stdio.h>
int soma(int a, int b){
    int r = a+b;
    return r;
}
int dif(int a, int b){
    return a-b;
}
int melega (int a, int b, int c){
    return soma(a, dif(b,c));
}
int main(){
    int s, d, m;
    s = soma(3,2);
    d = dif(3,2);
    m = melega(3,1,5);
    return 0;
}
```

memória

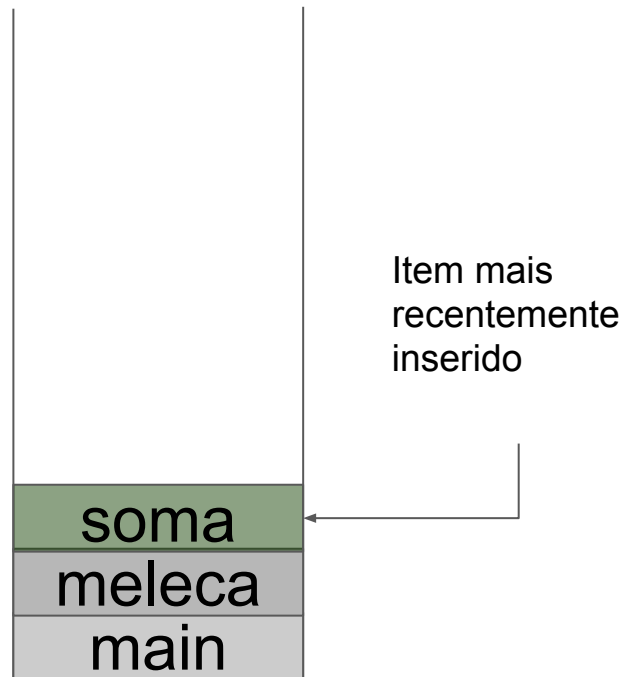


Pilha de Execução

- Exemplo

```
#include <stdio.h>
int soma(int a, int b){
    int r = a+b;
    return r;
}
int dif(int a, int b){
    return a-b;
}
int meleca (int a, int b, int c){
    return soma(a, dif(b,c));
}
int main(){
    int s, d, m;
    s = soma(3,2);
    d = dif(3,2);
    m = meleca(3,1,5);
    return 0;
}
```

memória

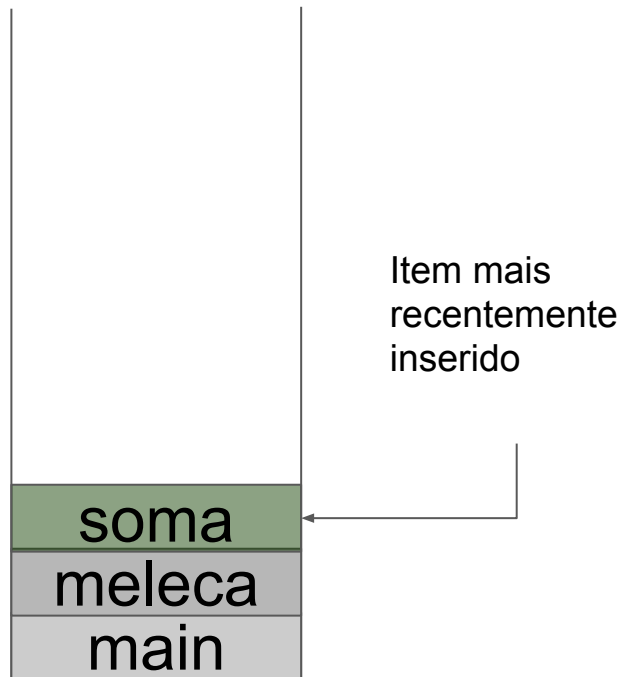


Pilha de Execução

- Exemplo

```
#include <stdio.h>
int soma(int a, int b){
    int r = a+b;
    return r;
}
int dif(int a, int b){
    return a-b;
}
int meleca (int a, int b, int c){
    return soma(a, dif(b,c));
}
int main(){
    int s, d, m;
    s = soma(3,2);
    d = dif(3,2);
    m = meleca(3,1,5);
    return 0;
}
```

memória

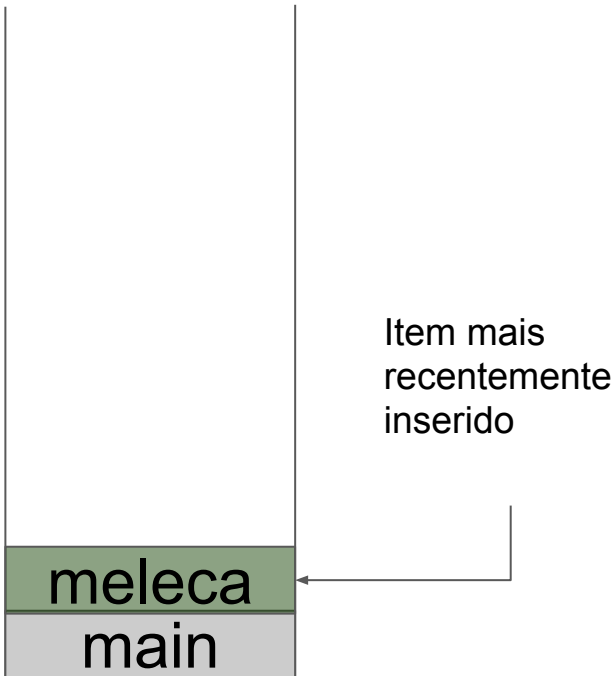


Pilha de Execução

- Exemplo

```
#include <stdio.h>
int soma(int a, int b){
    int r = a+b;
    return r;
}
int dif(int a, int b){
    return a-b;
}
int melega (int a, int b, int c){
    return soma(a, dif(b,c));
}
int main(){
    int s, d, m;
    s = soma(3,2);
    d = dif(3,2);
    m = melega(3,1,5);
    return 0;
}
```

memória

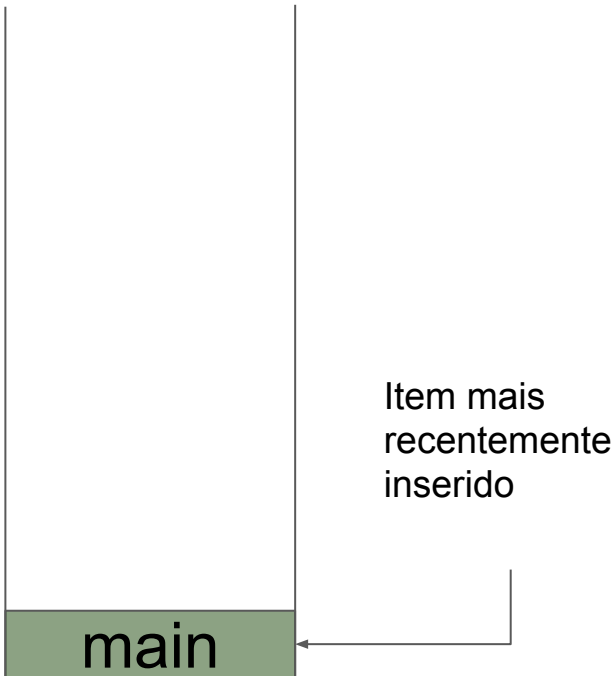


Pilha de Execução

- Exemplo

```
#include <stdio.h>
int soma(int a, int b){
    int r = a+b;
    return r;
}
int dif(int a, int b){
    return a-b;
}
int melega (int a, int b, int c){
    return soma(a, dif(b,c));
}
int main(){
    int s, d, m;
    s = soma(3,2);
    d = dif(3,2);
    m = melega(3,1,5);
    return 0;
}
```

memória

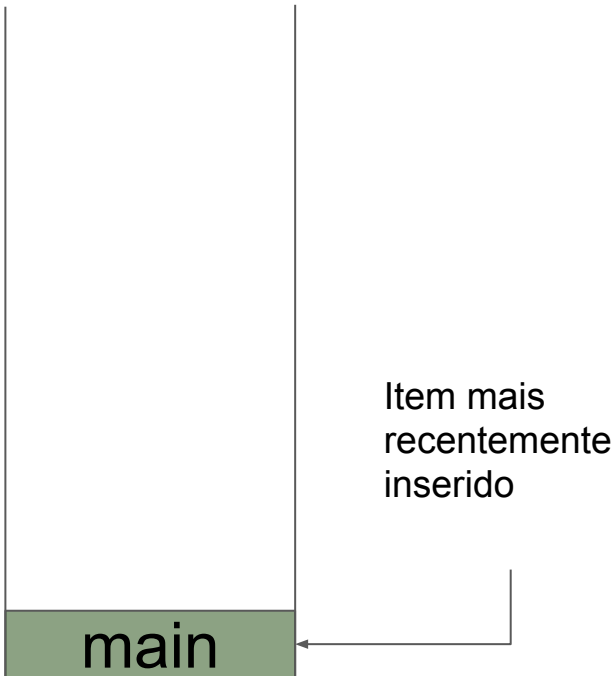


Pilha de Execução

- Exemplo

```
#include <stdio.h>
int soma(int a, int b){
    int r = a+b;
    return r;
}
int dif(int a, int b){
    return a-b;
}
int melega (int a, int b, int c){
    return soma(a, dif(b,c));
}
int main(){
    int s, d, m;
    s = soma(3,2);
    d = dif(3,2);
    m = melega(3,1,5);
    return 0;
}
```

memória



Pilha de Execução

- Exemplo

```
#include <stdio.h>
int soma(int a, int b){
    int r = a+b;
    return r;
}
int dif(int a, int b){
    return a-b;
}
int meleca (int a, int b, int c){
    return soma(a, dif(b,c));
}
int main(){
    int s, d, m;
    s = soma(3,2);
    d = dif(3,2);
    m = meleca(3,1,5);
    return 0;
}
```

memória

Fim do programa

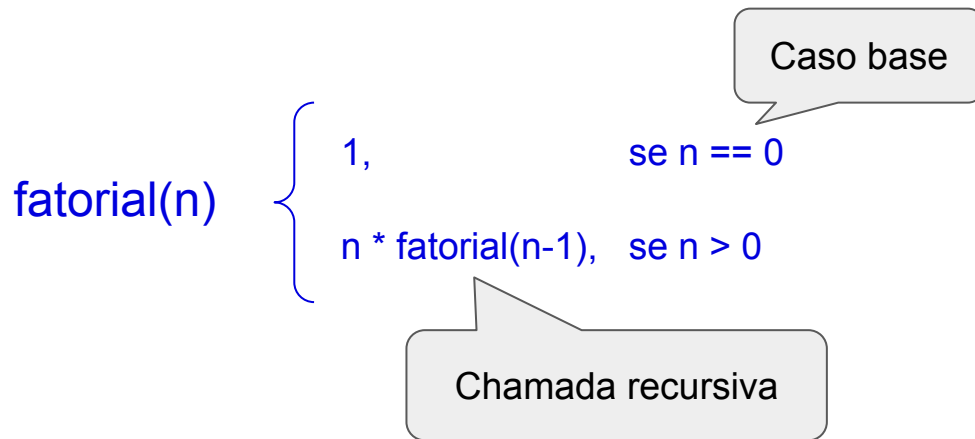
Recursividade

- Por que usar recursão?
 - Programas recursivos são, em geral, mais simples de escrever, analisar e entender
 - Quebramos o problema em partes menores, deixando-o mais simples, e chamamos a função várias vezes até chegar à forma mais simples de resolvê-lo



Recursividade

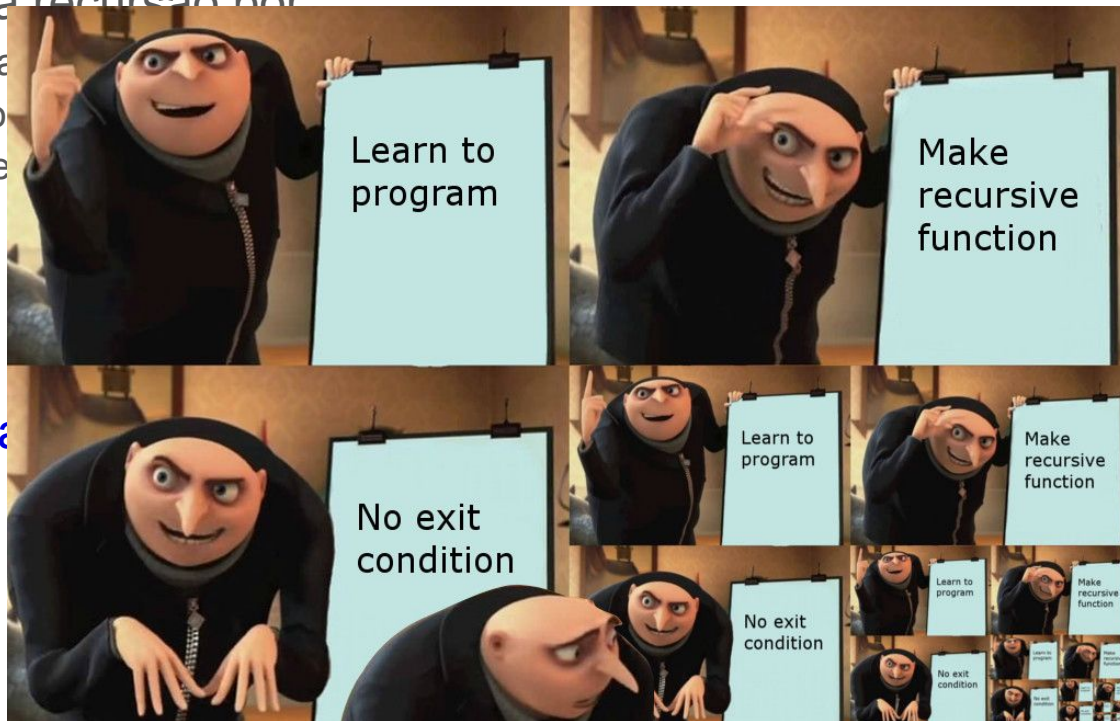
- Podemos decompor uma recursão por:
 - **Caso base:** uma instância do problema solucionada facilmente, sem recursão
 - **Chamadas recursivas:** onde a função é definida em termos de si própria, realizando uma redução em direção ao seu caso base



Recursividade

- Podemos decompor uma recursão por:

- **Caso base:** uma instância
 - **Chamadas recursivas:** o
- redução em direção ao se



Recursividade

- Exemplo - Fatorial
 - $n! = n \cdot (n-1) \cdot (n-2) \cdot \dots \cdot 1$;

```
#include <stdio.h>

int fatorial(int n){
    if (n == 0)
        return 1;
    return n*fatorial(n-1);
}

int main(){
    printf("%d", fatorial(10));
    return 0;
}
```

Recursividade

- Exemplo - Imprimir os números naturais menores ou iguais a N

```
#include <stdio.h>

void imprimeNaturais(int N) {
    if (N == 0) {
        printf("%d", N);
        return;
    }
    imprimeNaturais(N-1);
    printf(" %d", N);
}

int main() {
    imprimeNaturais(10);
    printf("\n");
    return 0;
}
```


Recursividade

- Exemplo - Imprimir os números inteiros maiores ou iguais a X e menores que Y

```
#include <stdio.h>

void imprimeInteiros(int X, int Y){
    if (X >= Y)
        return;
    printf(" %d ", X);
    imprimeInteiros(X+1, Y);
}

int main(){
    imprimeInteiros(-2,8);
    printf("\n");
    return 0;
}
```

Exercícios

1. Implemente uma função recursiva que, dado um número inteiro n , calcula o valor de 2^n .
2. Implemente uma função recursiva que, dados dois números inteiros x e n , calcula o valor de x^n .
3. Implemente uma função que imprime um número natural em base binária (deve fazer a conversão).
4. Implemente uma função recursiva que calcula o somatório de um vetor passado por parâmetro.
5. Implemente uma função recursiva que inverte uma string.