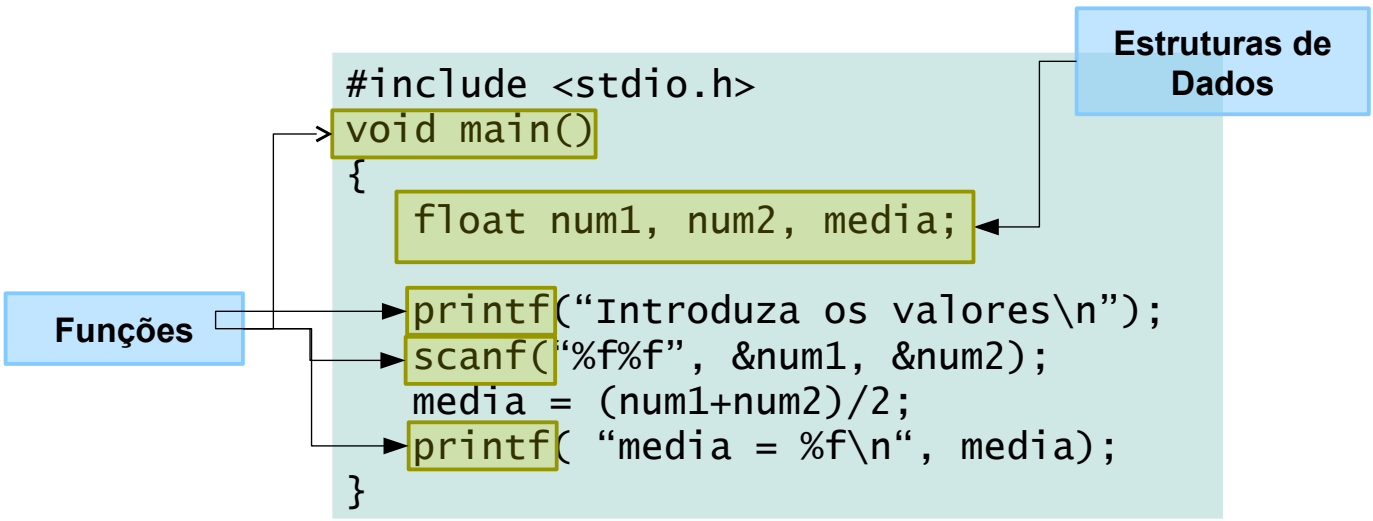
# Algoritmia e Programação

## Linguagem C
## Funções

José Tavares – jrt@isep.ipp.pt
ISEP, 2023/2024

## Programação imperativa

- Dados + funções



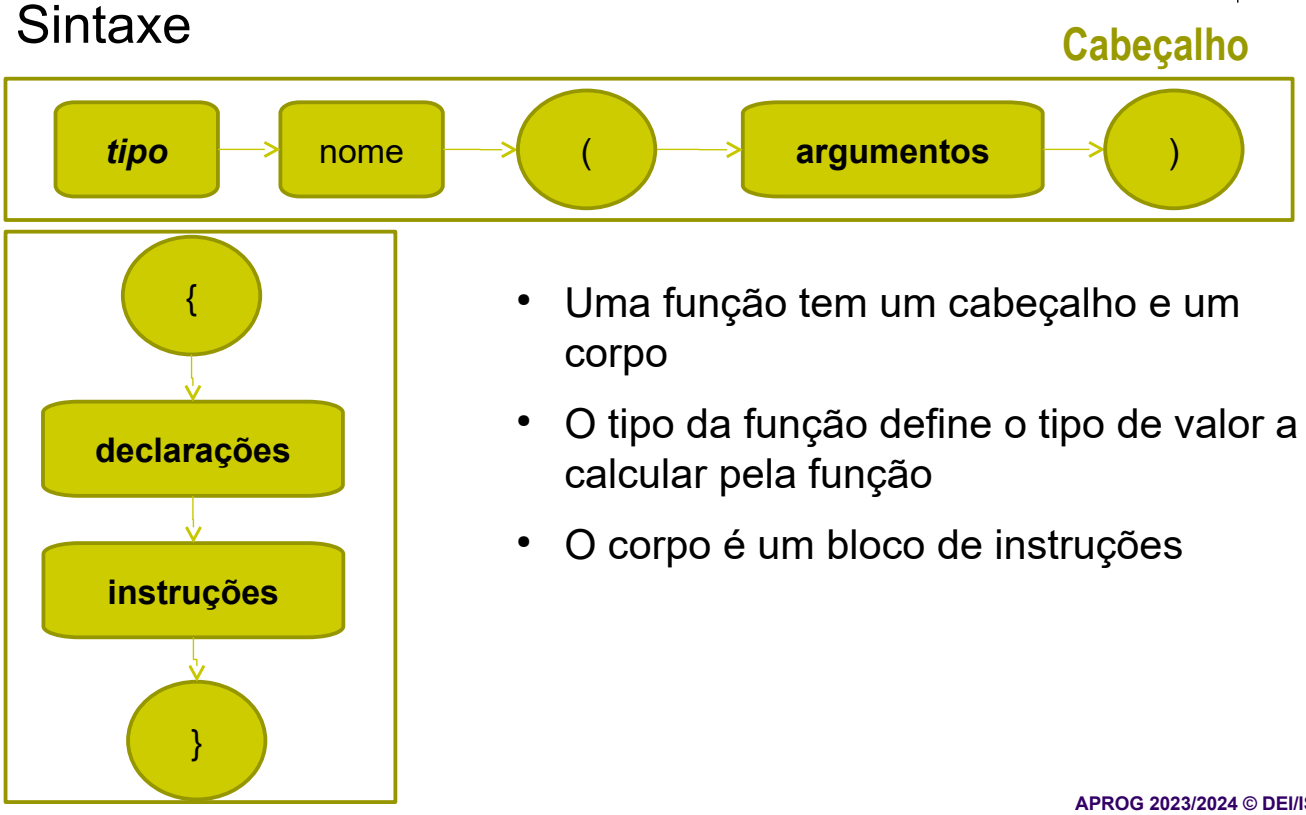
# Funções

- Uso de funções
  - Definição e declaração de funções
  - Tipo da função (retorno de valores)
  - Parâmetros/Argumentos
    - Endereços e Apontadores
- Noção de variáveis globais e variáveis locais



# Funções

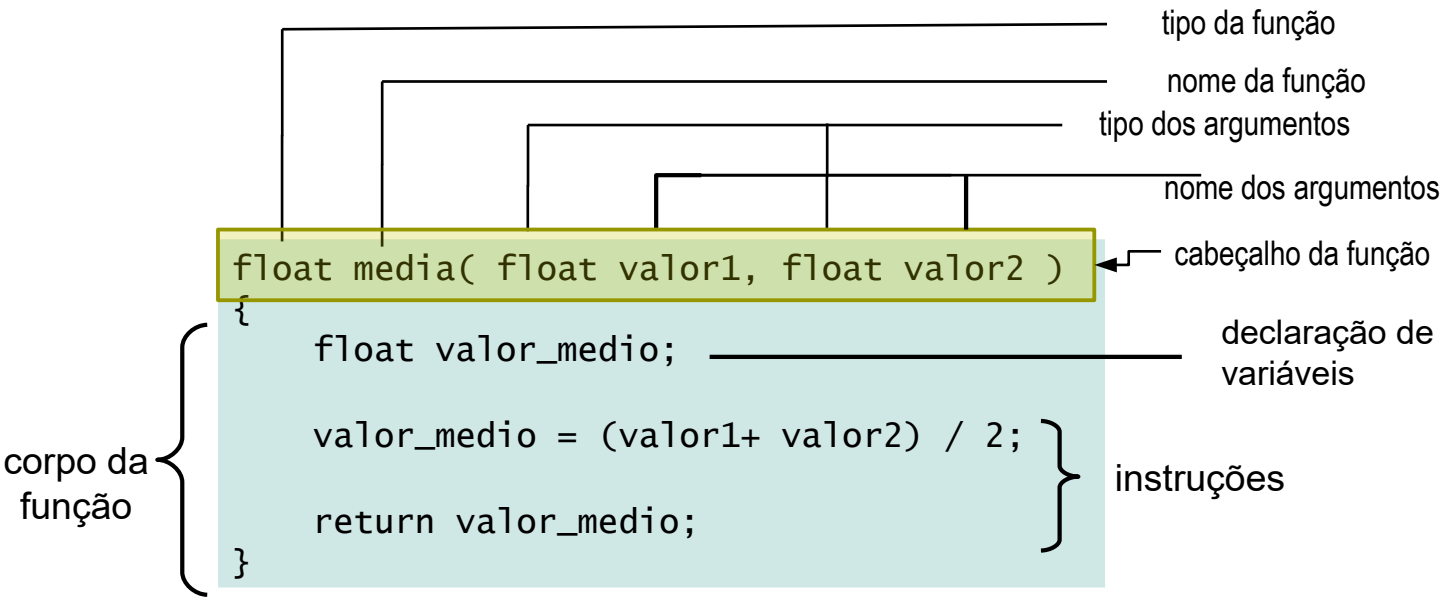
- Sintaxe




- Uma função tem um cabeçalho e um corpo
- O tipo da função define o tipo de valor a calcular pela função
- O corpo é um bloco de instruções

# Funções

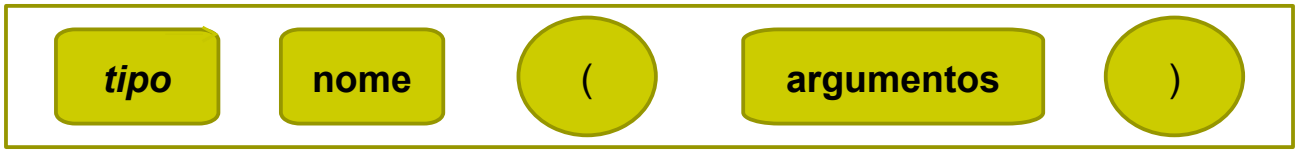
- Exemplo






# Funções

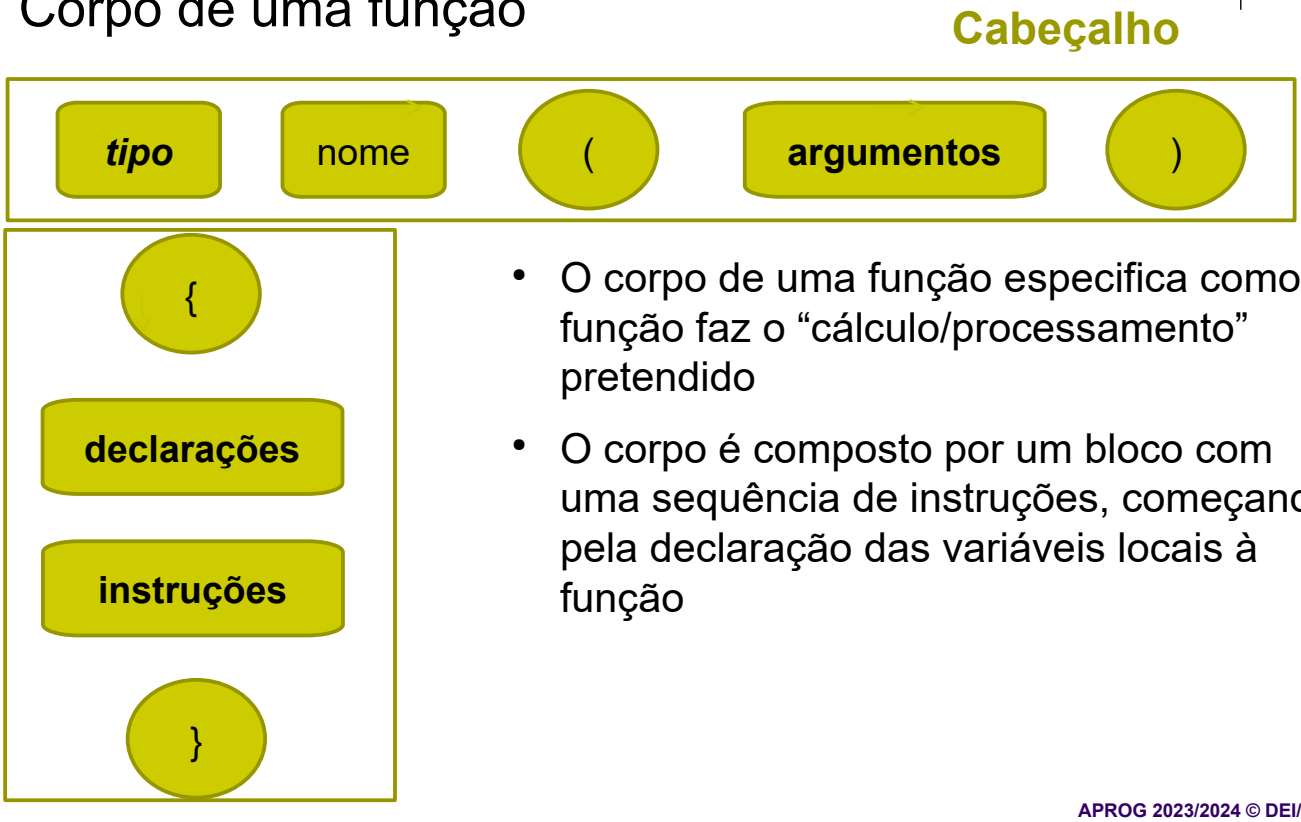
- Cabeçalho de uma função




- O cabeçalho define uma função:
  - Nome da função
  - Dados de entrada (argumentos)
  - Tipo de resultado

# Funções

- Corpo de uma função




- O corpo de uma função especifica como a função faz o "cálculo/processamento" pretendido
- O corpo é composto por um bloco com uma sequência de instruções, começando pela declaração das variáveis locais à função

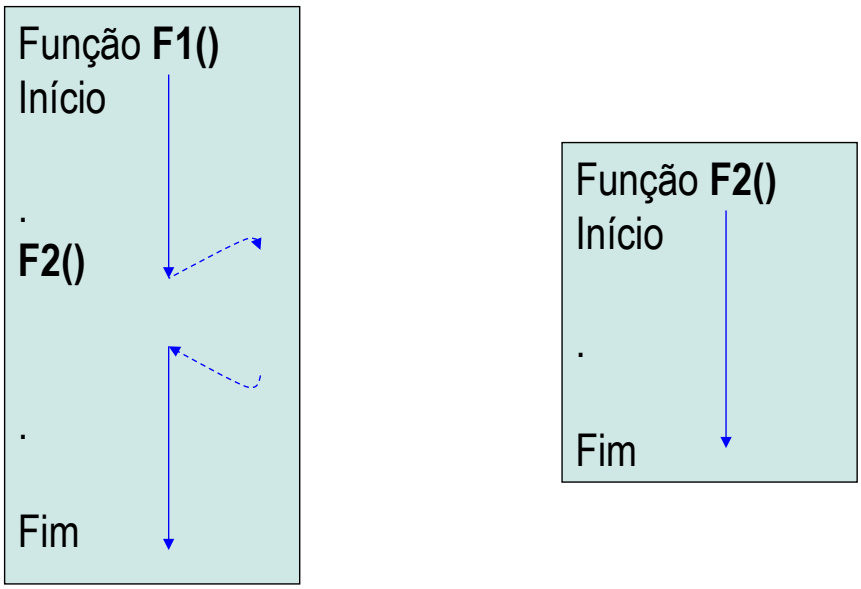


# Funções

- Características
  - Identificação
    - Nome único que identifica a função
  - Unicidade
    - Deve realizar uma tarefa bem definida
  - Funcionamento
    - Funciona como uma caixa preta
  - Reutilização
    - Uma função deve ser genérica e independente de qualquer programa ou projeto, de modo a poder ser reutilizada por outros programas ou projetos

# Funções

- Interação entre funções
  - À exceção da função principal, a função ***main***, as funções só podem ser usadas/invocadas/chamadas a partir de outra função






# Funções

- Interação entre funções
  - Utilização/invocação/chamada
    - É feita a partir de outra função
    - A função "invocadora" suspende a sua execução no ponto de chamada
    - A função "invocada" é executada
    - A função "invocadora" retoma a sua execução no ponto onde tinha ficado suspensa
  - Execução
    - Entrada de dados ou passagem de parâmetros
    - Execução do bloco de instruções
    - Saída de dados

# Funções

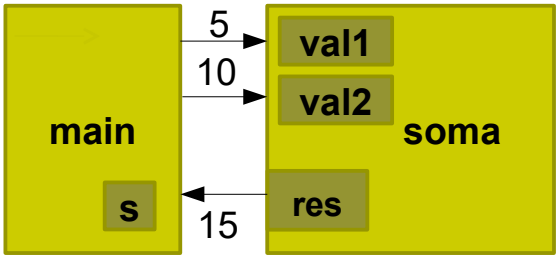
- Comunicação entre funções
  - Entrada de dados na função
    - A comunicação com uma função faz-se através dos argumentos que lhe são enviados e dos parâmetros presentes na função que os recebem
    - Um parâmetro é uma variável local à função a que pertence
  - Retorno / Devolução de valor
    - Saída de dados da função
    - A função devolve sempre algo... Se não for um valor será ***void***...
    - Feita através de instrução ***return***
    - Uma função pode conter várias instruções ***return*** mas apenas uma delas é executada



# Funções

- Comunicação entre funções

```
#include <stdio.h>

int soma(int val1, int val2) {
   int res;

   res= val1 + val2;
   return res;
}

void main() {
   int s;

   s = soma(5, 10);
   printf( "soma = %d\n", s);
}
```



# Funções

- Onde colocar as funções?
  - Podem ser colocadas antes ou depois da função ***main***, antes ou depois de serem invocadas
  - É necessário declarar as funções antes de serem utilizadas
    - Escrita do protótipo da função, por norma, logo a seguir aos ***includes***
    - Protótipo corresponde ao cabeçalho da função seguido de **;**



# Funções

- Onde colocar as funções?
  - Escrever a função antes da função ***main***

```
#include <stdio.h>

float media(float valor1, float valor2) {
    float valor_medio;

    valor_medio = (valor1+ valor2) / 2;

    return valor_medio;
}

void main() {
    float num1, num2, res;

    printf("Introduza os valores\n");
    scanf("%f%f", &num1, &num2);
    res = media(num1, num2);
    printf( "media = %f\n", res);
}
```

# Funções

```
#include <stdio.h>

float media( float valor1, float valor2 ) {
    float valor_medio;

    valor_medio = (valor1+ valor2) / 2;
    return valor_medio;
}

void main() {
   float num1, num2, res;

   printf("Introduza os valores\n");
   scanf("%f%f", &num1, &num2);
   printf( "media = %f\n", media(num1, num2));
}
```



# Funções

- Onde colocar as funções?
  - Se escrever a função depois da função ***main...***
  - … obrigatório definir o protótipo antes da função ***main***

```
#include <stdio.h>

float media(float valor1, float valor2);

void main() {
   float num1, num2, res;

   printf("Introduza os valores\n");
   scanf("%f%f", &num1, &num2);
   res = media(num1, num2);
   printf( "media = %f\n", res);
}

float media(float valor1, float valor2) {
   float valor_medio;

   valor_medio = (valor1+ valor2) / 2;

   return valor_medio;
}
```

# Funções

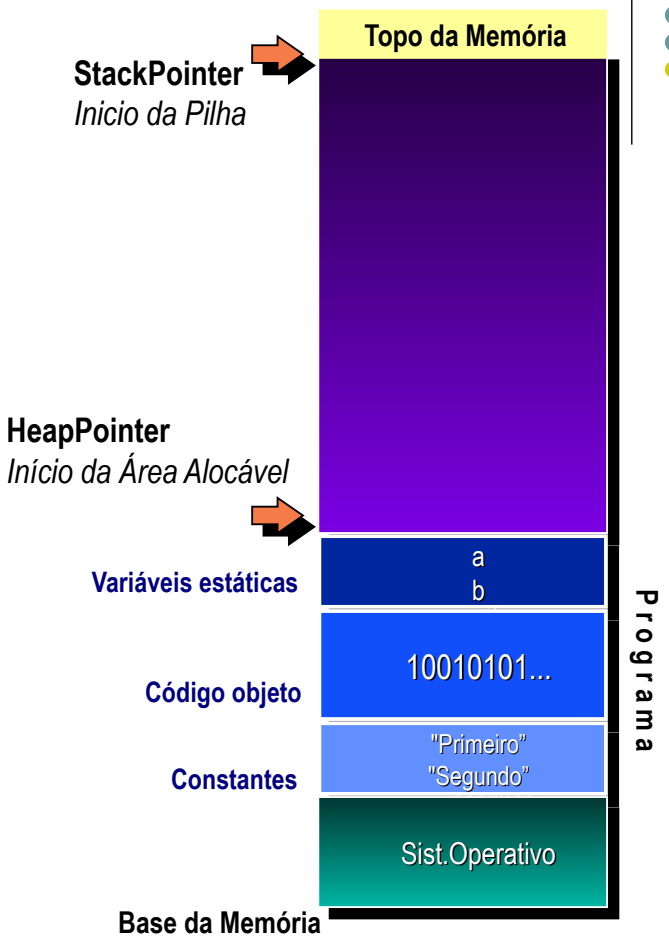
- Variáveis locais
  - Variáveis declaradas dentro do corpo de uma função são conhecidas apenas dentro da própria função
    - A declaração de variáveis dentro de uma função deve ser efetuada antes de qualquer instrução
    - Essas variáveis só podem ser usadas dentro da própria função
  - Depois de terminada a execução de uma função as suas variáveis locais são destruídas
  - Se uma mesma variável for declarada em duas funções distintas não há problema pois o compilador sabe quando utilizar cada uma
    - São variáveis distintas sem qualquer relação

```
#include <stdio.h>

int a = 1; /* VARIÁVEL GLOBAL */

void xpto() {
   a++;
   printf("Valor da variavel 'a': %d", a);
}

void main() {
   int a = 3, b = 2;
   /* VARIÁVEIS LOCAIS */

   printf("Valor da variavel 'a': %d", a);
   printf("Valor da variavel 'b': %d", b);

   xpto();
}
```



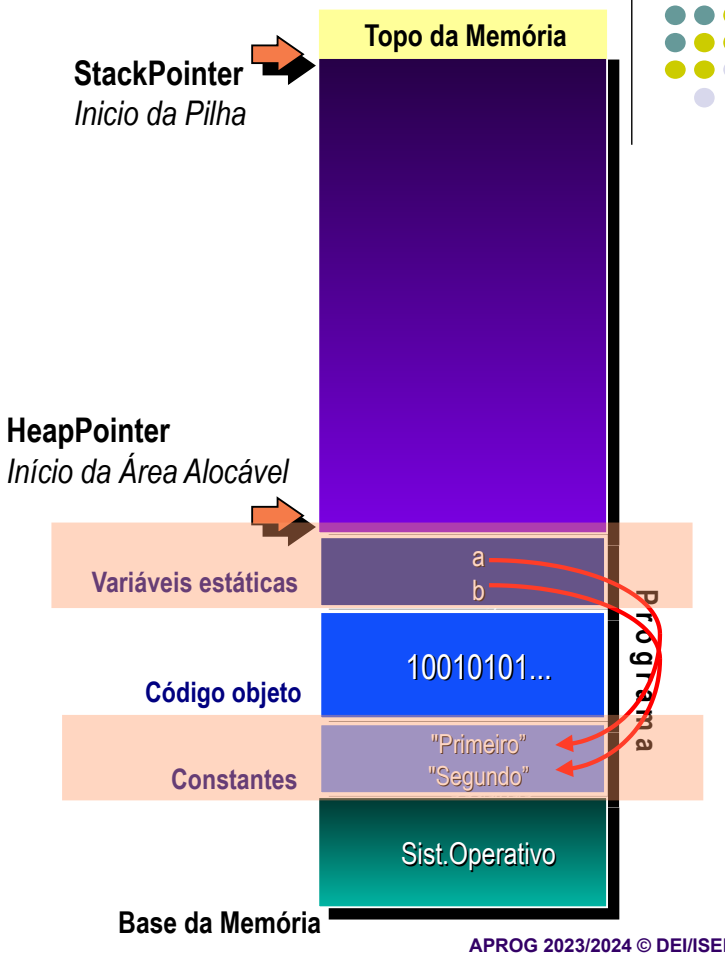
# Funções

```
#include <stdio.h>
char *a, *b;

int func_A()
{
   int local1, local2;
   .....
}
void func_B()
{
   int localA, localB;
   localA = func_A();
   localB = func_A();
}
main()
{
   a = "Primeiro";
   b = "Segundo"
   func_B();
}
```



# Funções

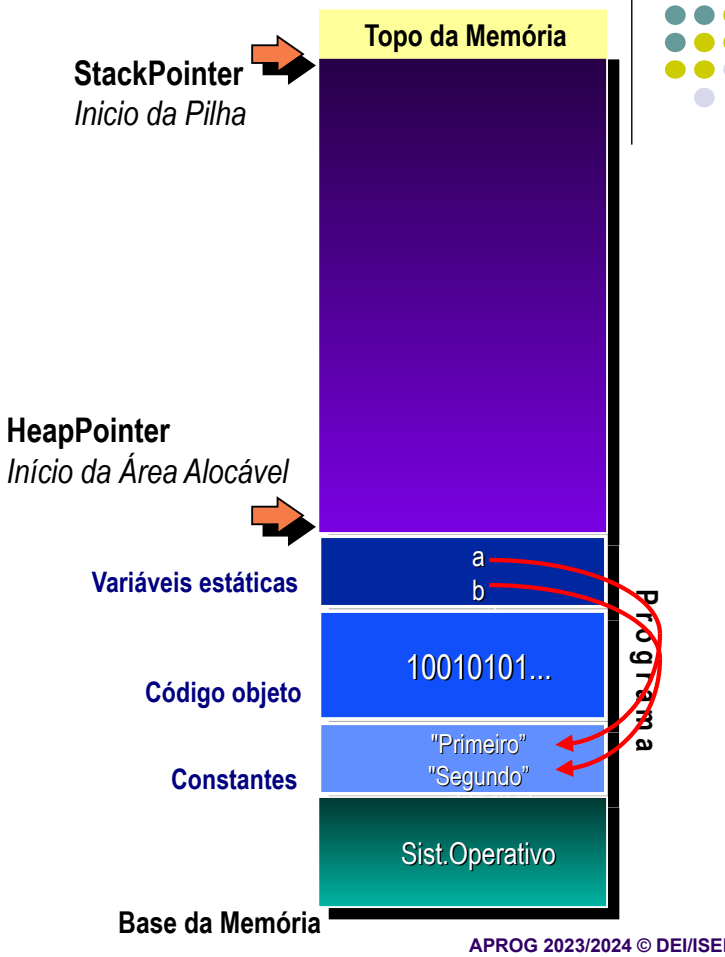
```
#include <stdio.h>
char *a, *b;

int func_A()
{
   int local1, local2;
   .....
}
void func_B()
{
   int localA, localB;
   localA = func_A();
   localB = func_A();
}
int main()
{
   a = "Primeiro";
   b = "Segundo"
   func_B();
}
```





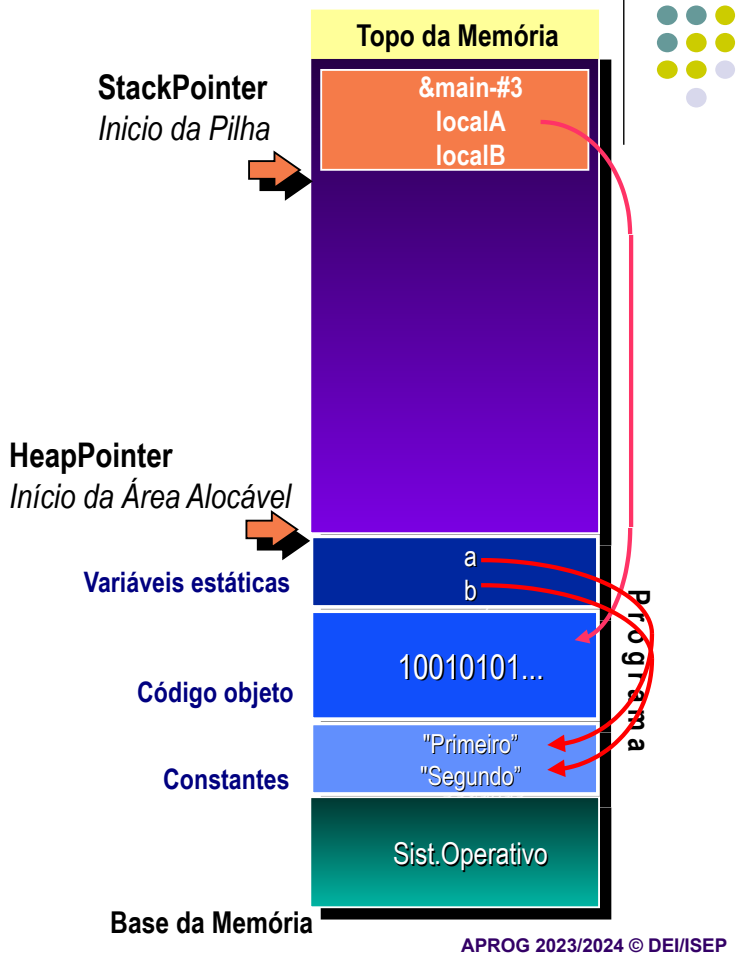

# Funções

```
#include <stdio.h>
char *a, *b;

int func_A()
{
   int local1, local2;
   .....
}
void func_B()
{
   int localA, localB;
   localA = func_A();
   localB = func_A();
}
int main()
{
   a = "Primeiro";
   b = "Segundo"
   func_B();
}
```



# Funções

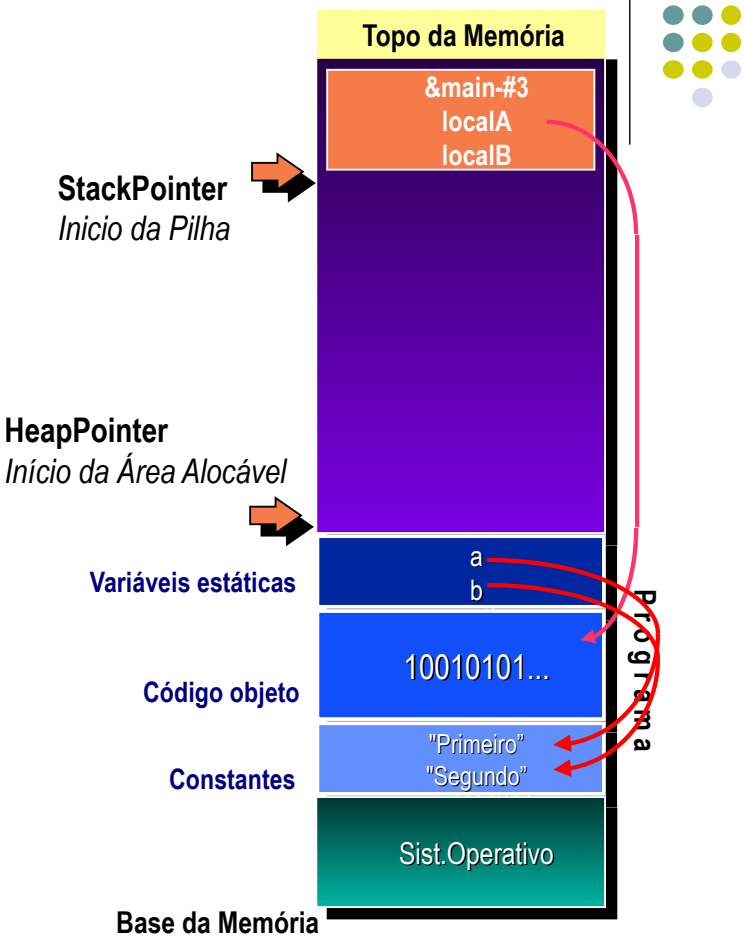
```
#include <stdio.h>
char *a, *b;

int func_A()
{
   int local1, local2;
   .....
}
void func_B()
{
  int localA, localB;
  localA = func_A();
  localB = func_A();
}
int main()
{
   a = "Primeiro";
   b = "Segundo"
   func_B();
}
```





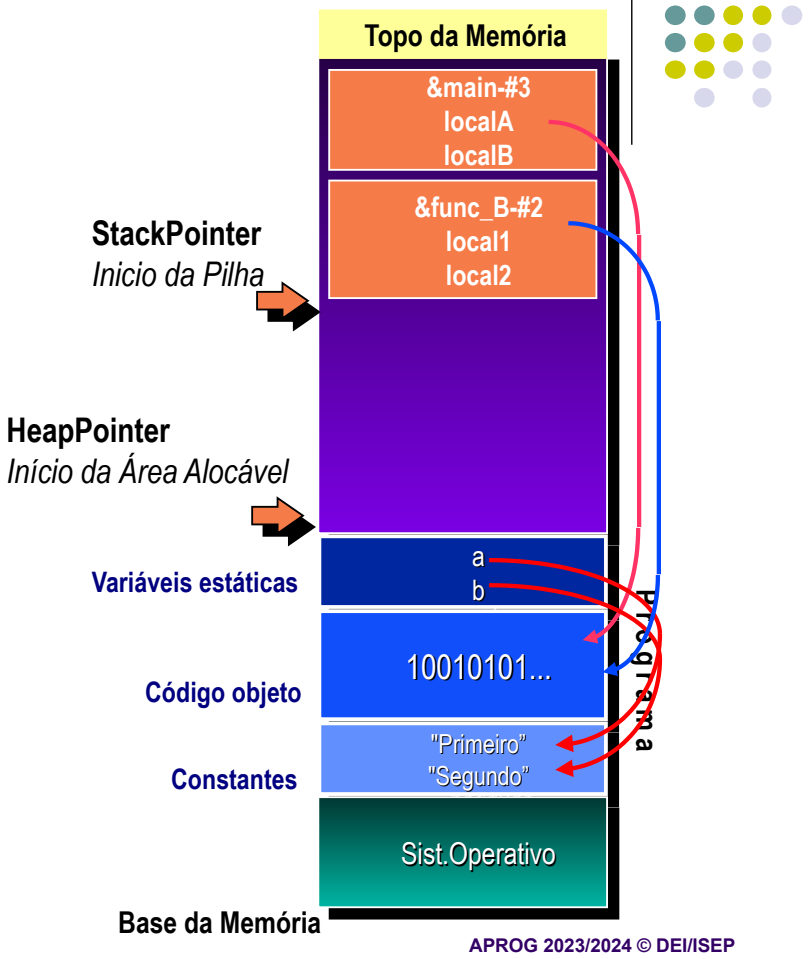

# Funções

```
#include <stdio.h>
char *a, *b;

int func_A()
{
   int local1, local2;
   .....
}
void func_B()
{
  int localA, localB;
  localA = func_A();
  localB = func_A();
}
int main()
{
   a = "Primeiro";
   b = "Segundo"
   func_B();
}
```



# Funções

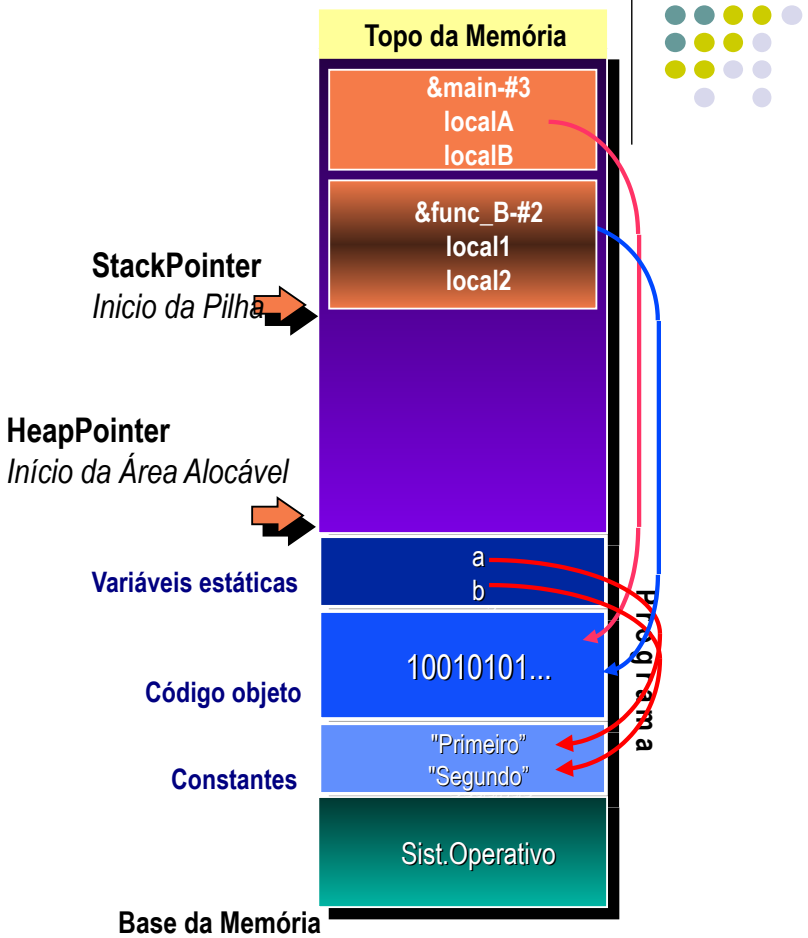
```
#include <stdio.h>
char *a, *b;

int func_A()
{
    int local1, local2;
    .....
}
void func_B()
{
  int localA, localB;
  localA = func_A();
  localB = func_A();
}
int main()
{
    a = "Primeiro";
    b = "Segundo"
    func_B();
}
```





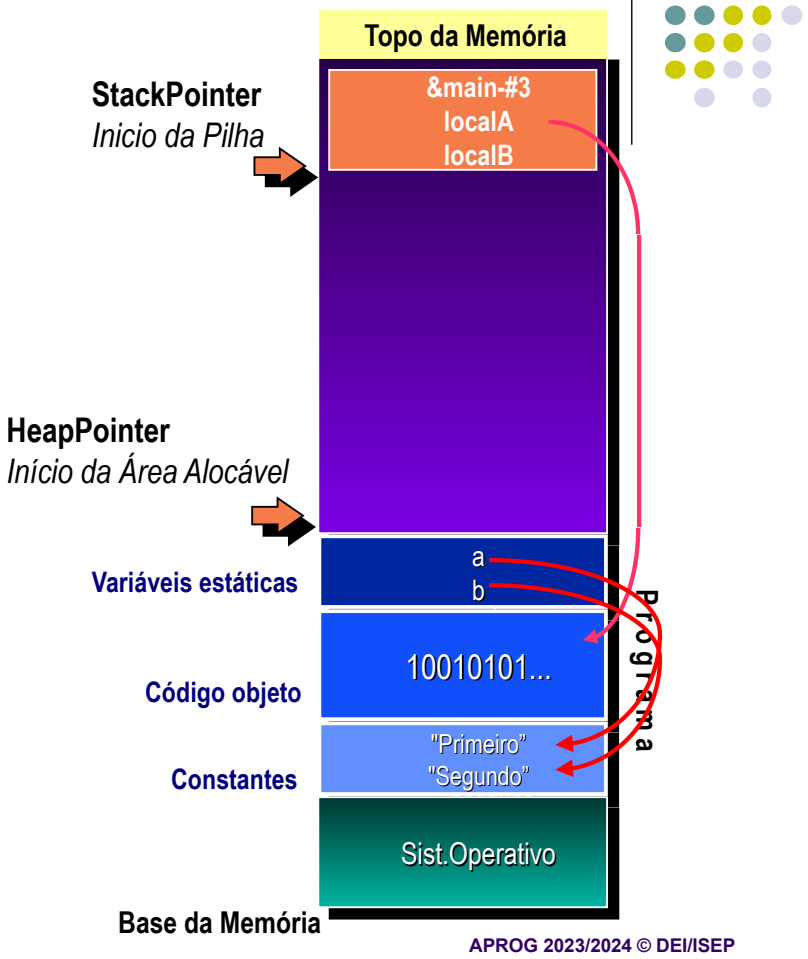

# Funções

```
#include <stdio.h>
char *a, *b;

int func_A()
{
    int local1, local2;
    .....
}
void func_B()
{
  int localA, localB;
  localA = func_A();
  localB = func_A();
}
int main()
{
    a = "Primeiro";
    b = "Segundo"
    func_B();
}
```



# Funções

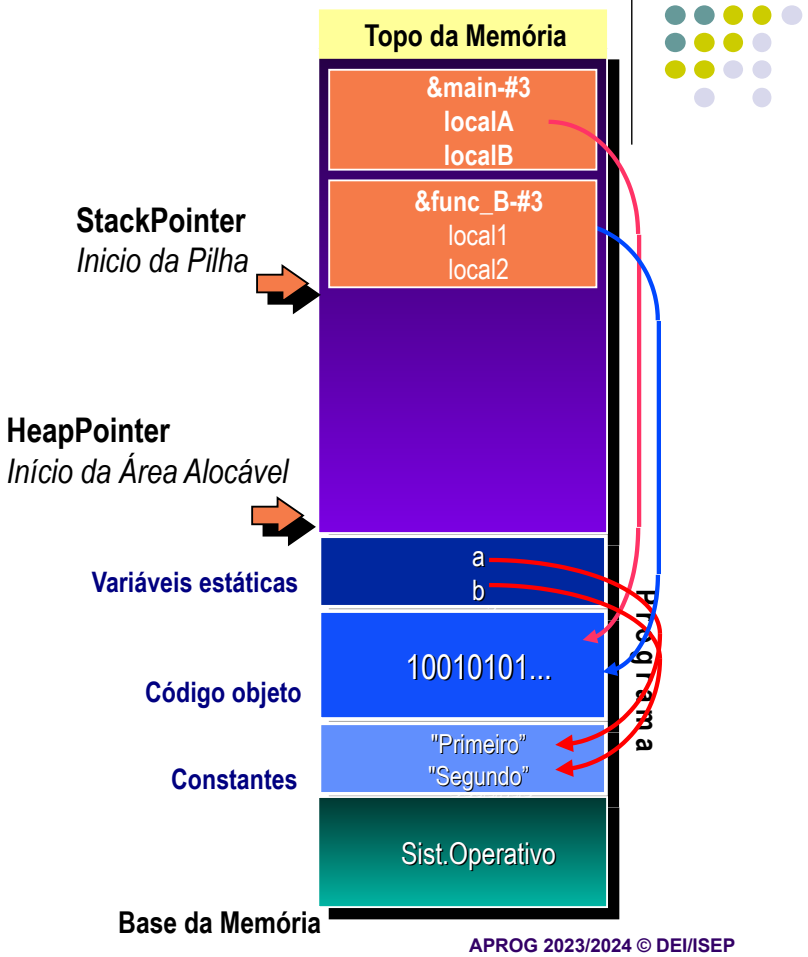
```
#include <stdio.h>
char *a, *b;

int func_A()
{
   int local1, local2;
   .....
}
void func_B()
{
  int localA, localB;
  localA = func_A();
  localB = func_A();
}
int main()
{
   a = "Primeiro";
   b = "Segundo"
   func_B();
}
```





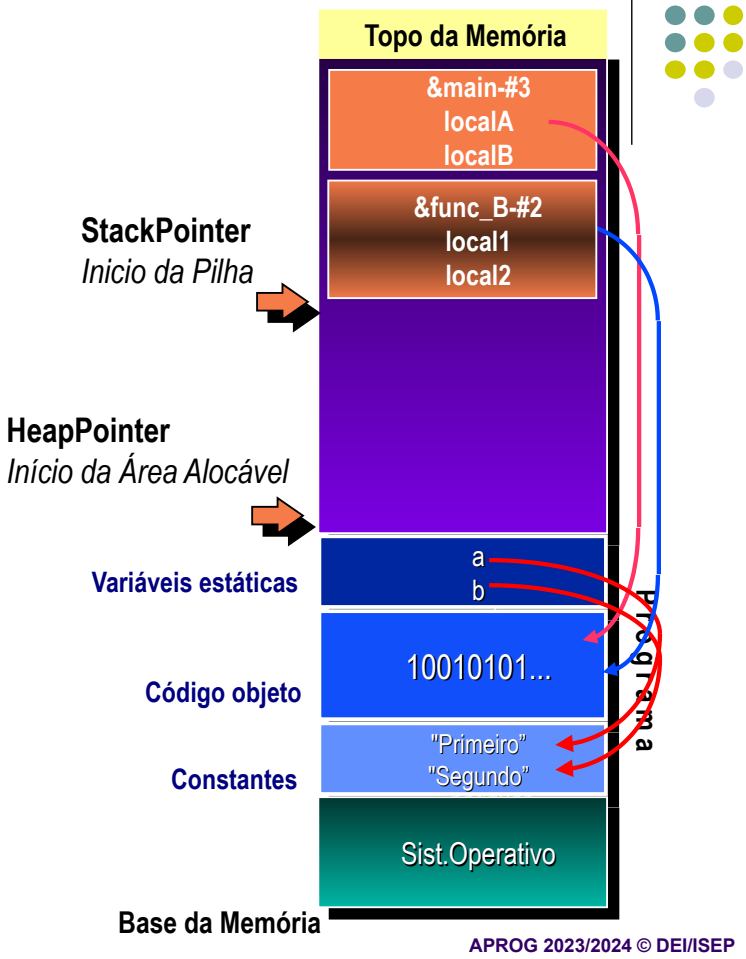

# Funções

```
#include <stdio.h>
char *a, *b;

int func_A()
{
   int local1, local2;
   .....
}
void func_B()
{
  int localA, localB;
  localA = func_A();
  localB = func_A();
}
int main()
{
   a = "Primeiro";
   b = "Segundo"
   func_B();
}
```



# Funções

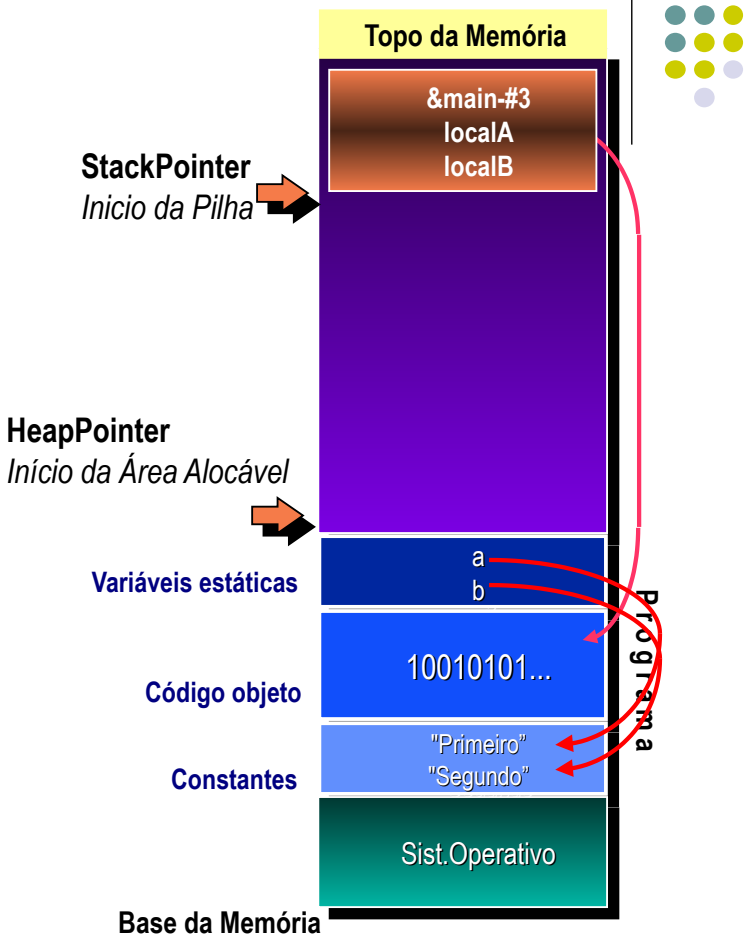
```
#include <stdio.h>
char *a, *b;

int func_A()
{
   int local1, local2;
   .....
}
void func_B()
{
  int localA, localB;
  localA = func_A();
  localB = func_A();
}
int main()
{
   a = "Primeiro";
   b = "Segundo"
   func_B();
}
```





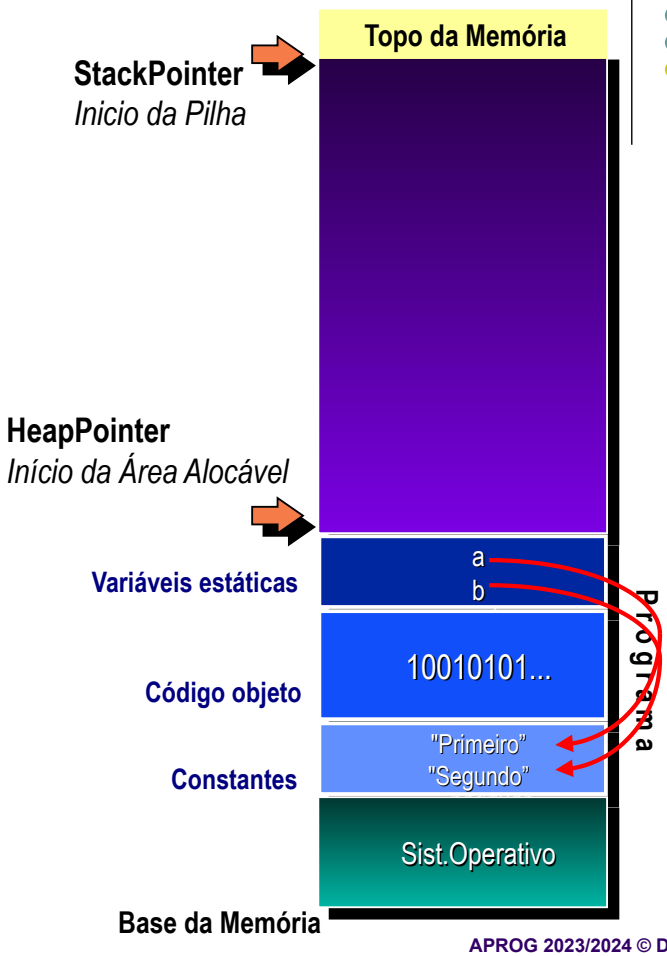

# Funções

```
#include <stdio.h>
char *a, *b;

int func_A()
{
   int local1, local2;
   .....
}
void func_B()
{
  int localA, localB;
  localA = func_A();
  localB = func_A();
}
int main()
{
   a = "Primeiro";
   b = "Segundo"
   func_B();
}
```



# Funções

```
#include <stdio.h>
char *a, *b;

int func_A()
{
   int local1, local2;
   .....
}
void func_B()
{
  int localA, localB;
  localA = func_A();
  localB = func_A();
}
int main()
{
   a = "Primeiro";
   b = "Segundo"
   func_B();
}
```






# Funções – Exemplo

- Descreva um algoritmo que determine o fatorial de uma sequência de números inteiros introduzidos pelo utilizador. A sequência termina com -1

```
Algoritmo Fatorial2
ED: N, F, I inteiros
INÍCIO
    Escrever("Escreva um numero")
    Ler(N)
    Enquanto N <> -1 Fazer
        F ← 1
        Para I=1 Até N Passo 1 Fazer
            F ← F * I
        FimPara
        Escrever("O factorial de ", N, " é ", F)
        Escrever("Escreva um numero")
        Ler(N)
    FimEnquanto
Fim
```

# Funções – Exemplo

- Descreva um algoritmo que determine o fatorial de uma sequência de números inteiros introduzidos pelo utilizador. A sequência termina com -1

```
Algoritmo Fatorial2
ED: N, F, I inteiros
Inicio
   Escrever("Escreva um numero")
   Ler(N)
   Enquanto N <> -1 Fazer
      Escrever("O fatorial de ", N, " é ", fatorial(N))
      Escrever("Escreva um numero")
      Ler(N)
   FimEnquanto
Fim
```



# Funções – Exemplo

```
#include <stdio.h>

long int fatorial(int num) {
    long int res = 1, i;
    for(i = 1; i <= num; i++)
        res = res * i;
    return res;
}

void main() {
    int n;

    printf("Introduza um numero:\n");
    scanf("%d", &n);
    while(n != -1)
    {
        printf("O fatorial do numero %d e %ld\n", n, fatorial(n));
        printf("Introduza um numero:\n");
        scanf("%d", &n);
    }
}
```

# Funções – Exercício

- Escreva uma função que determina se um número inteiro é par:
  - Argumentos (informação de entrada)
    - Número inteiro
  - Cálculo
    - Analisar resto da divisão por 2
  - Tipo (resultado do cálculo)
    - Verdade ou Falso



# Funções – Exercício

- Escreva uma função que determina se um número inteiro é par

```
enum boolean {FALSE, TRUE};
/* em C++ existe o tipo bool com valores
   true ou false, mas em C não... */

enum boolean par(int num)
{
   if(num%2==0)
      return TRUE;
   else
      return FALSE;
}
```

# Funções – Exercício

- Escreva um programa que lê uma sequência de números inteiros, terminada pelo 0, e usa a função anterior para determinar quantos desses valores são pares.

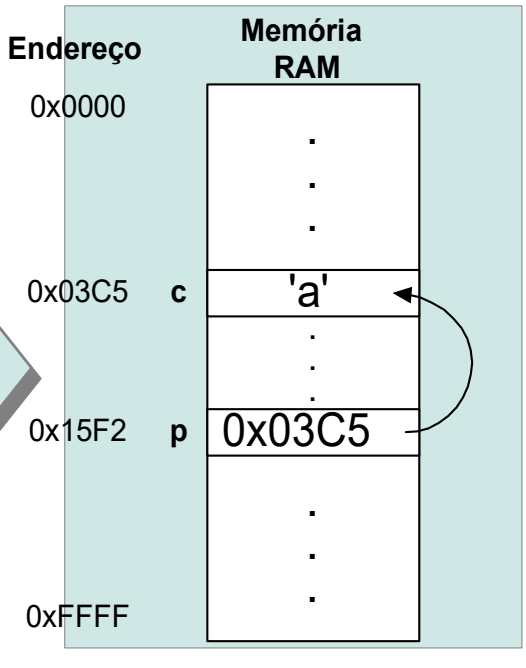
```
void main() {
    int nr, cont = 0;

    do {
        printf("Insira um numero. (para terminar insira 0)\n");
        scanf("%d", &nr);
        if(nr!=0){
            if ( par(nr) == TRUE ) {
                printf("O numero %d e par.\n", nr);
                cont++;
            }
            else printf("O numero %d e impar.\n", nr);
        }
    } while(nr!=0);

    printf("Inseriu %d numeros pares.\n", cont);
}
```



# Variáveis do tipo Apontador

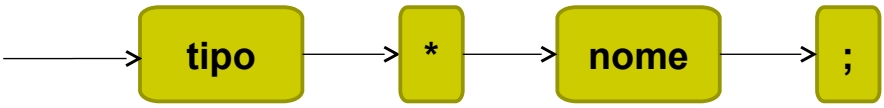
- Qualquer variável tem um nome, um valor e um endereço
- Uma variável do tipo apontador guarda um endereço de memória

```
char c;  //c é um carácter
char *p; //p é um apontador para carácter.

p = &c; // p agora guarda o endereço de c

*p = 'a'; // c = 'a'
```



# Apontadores

- Conceitos Básicos
  - O compilador associa a cada variável uma posição única em memória, capaz de suportar os dados do tipo dessa variável;
  - Os apontadores são um mecanismo particularmente fléxivel de manipulação de dados, pois permitem manipular diretamente dados contidos em endereços específicos de memória;
- Sintaxe





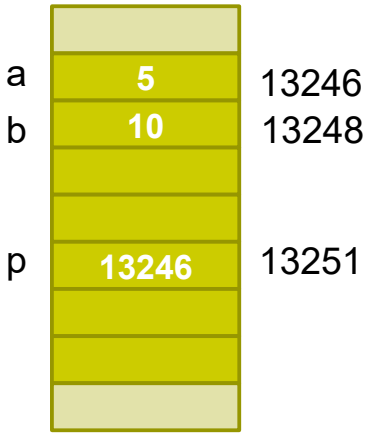

# Apontadores

- Um apontador é uma variável como outra qualquer, cujo objetivo é armazenar um endereço de memória
  - Um endereço é um valor, é um número, entre 0 e o total de *bytes* de memória, que indica a posição que a variável ocupa em memória
- Inicialização
  - Faz-se através do operador &
    - Para evitar problemas de programação é bom hábito inicializar sempre os apontadores
    - A constante simbólica NULL, quando colocada num apontador, significa que este não aponta para nenhuma variável

# Apontadores

- Acesso ao valor e ao endereço de uma variável
  - O acesso ao endereço de uma variável faz-se através do operador &
    - Acesso direto: através do nome da variável
    - Acesso indireto: através do endereço da variável

```
int a = 5, b;
int * p;

b = 10;
p = &a;
```




- Acesso
  - `a -> 5`
  - `&a -> 13246`
  - `b -> 10`
  - `&b -> 13248`
  - `p -> 13246`
  - `&p -> 13251`
  - `*p -> 5`



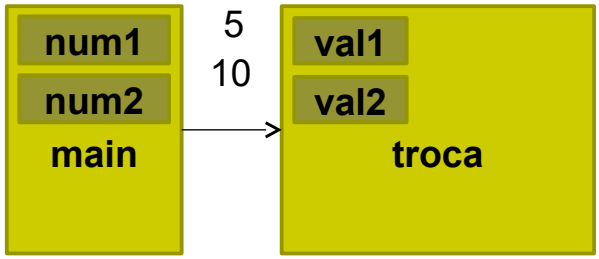
# Apontadores

- Passagem de parâmetros
  - **Por valor**
    - Numa passagem de parâmetros por valor, o que é enviado à função é a cópia dos valores de que necessita
  - **Por referência**
    - Numa passagem de parâmetros por referência, o que é enviado à função não é uma cópia do valor da variável, mas a própria variável, ou uma referência para esta

- EM C SÓ EXISTE PASSAGEM DE PARÂMETROS POR VALOR!

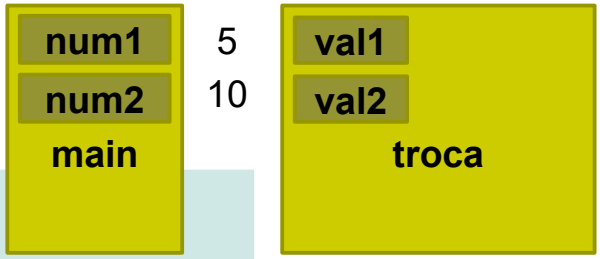
# Funções – passagem de parâmetros

```
#include <stdio.h>
void troca(int val1, int val2)
{
    int res;

    res= val1;
    val1 = val2;
    val2 = res;
}

void main()
{
   int num1=5, num2=10;

    troca(num1, num2);
    printf( num1= %d\nnum2=%d", num1, num2);

}
```

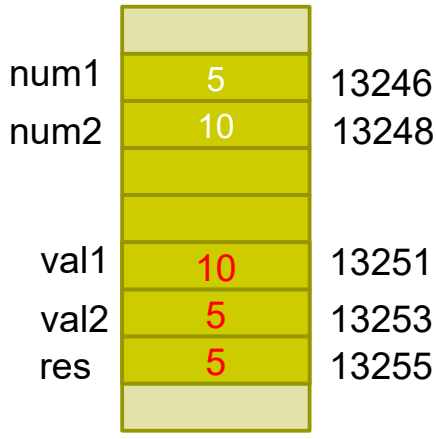





# Funções




```
#include <stdio.h>

void troca( int val1, int val2 )
{
    int res;

    res= val1;
    val1 = val2;
    val2 = res;
}

void main()
{
   int num1=5, num2=10;

    troca(num1, num2);
    printf( num1= %d\nnum2=%d", num1, num2);
}
```

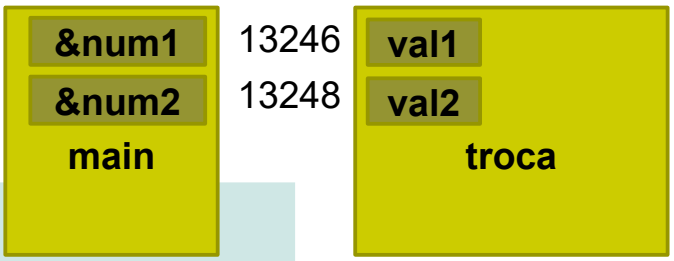


# Apontadores

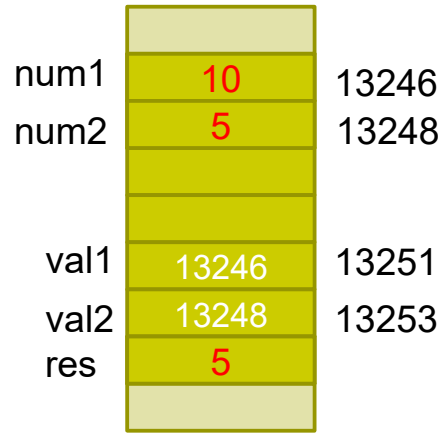
- Passagem de parâmetros
  - Para alterar o valor de uma variável dentro de uma função, e como em C não é possível criar referências, podemos passar à função o Endereço (usando o operador &) da variável que se pretende alterar
    - Se passamos o endereço de uma variável, a função recebe um apontador para a variável
    - Se dentro da função usamos apontadores, então podemos alterar os locais para onde eles apontam e que correspondem às variáveis originais

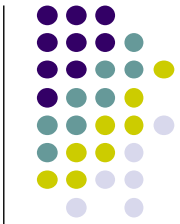


# Funções




```
#include <stdio.h>

void troca( int *val1, int *val2 )
{
    int res;

    res= *val1;
    *val1 = *val2;
    *val2 = res;
}

void main()
{
    int num1=5, num2=10;

    troca(&num1, &num2);
    printf( num1= %d\nnum2=%d", num1, num2);

}
```





# Questões