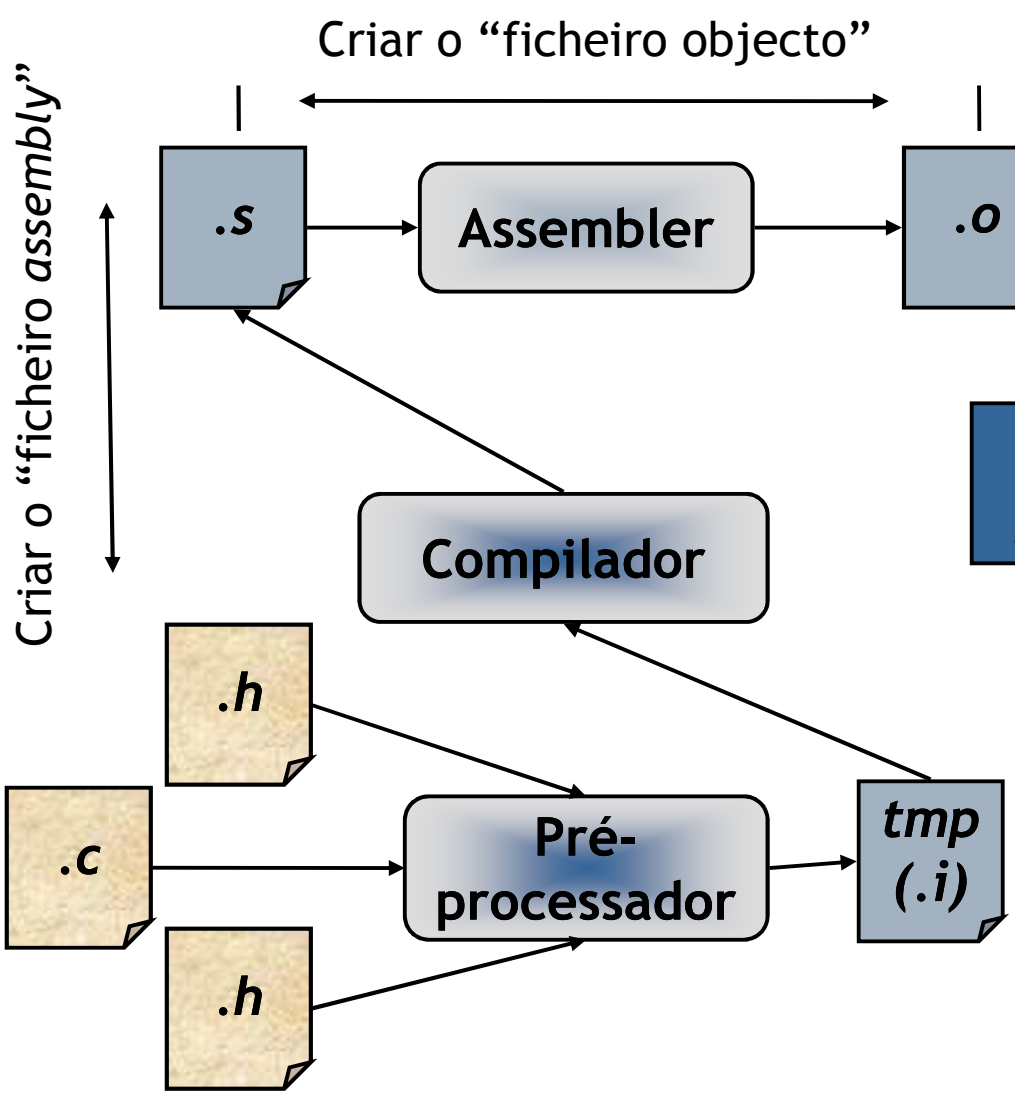
Assembly e a Linguagem C

Introdução

- Compilar um programa com módulos em C e assembly
- Convenções da linguagem C
  - ✓ Passagem de parâmetros
  - ✓ Retorno de resultado
  - ✓ Salvaguarda de registos
  - ✓ Limpar os parâmetros da pilha
- Assembly embebido em código C
- Ambiente de execução das funções em C
- Passagem por referência
- Código reentrante

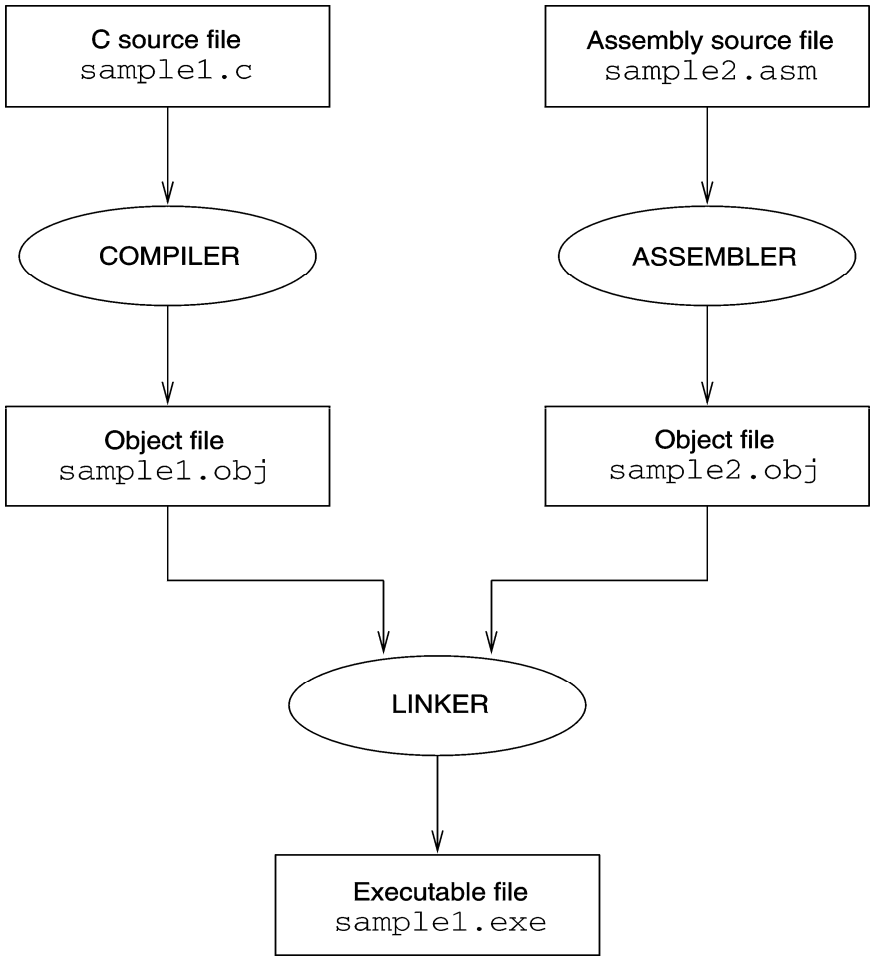
Compilar um Programa C em Linux



- No "mundo" Windows:
  - ✓ .obj (em vez de .o)
  - ✓ .lib (em vez de .a)
  - ✓ .dll (em vez de .so)
  - ✓ .exe (em vez de "sem nada")

Compilar um Programa em C e Assembly

- Modulos em assembly podem ser ligados com módulos desenvolvidos em linguagens de alto nível
- São, no entanto, necessárias convênções na passagem de parâmetros e retorno de resultados
- Vamos usar a linguagem C para ilustrar esses conceitos

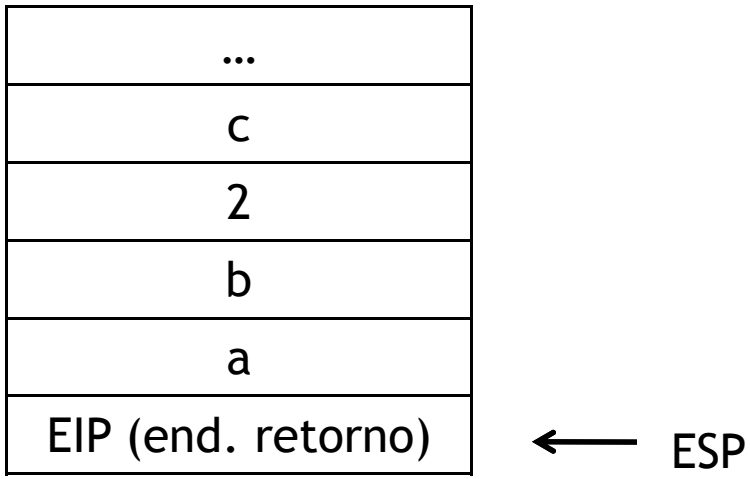


Convenções do C - Passagem de Parâmetros

- Os parâmetros são colocados na pilha da direita para a esquerda
- Exemplo:

sum(a,b,2,c) →

```
push dword [c]
push dword 2
push dword [b]
push dword [a]
call sum
```



Convenções do C – Retorno do Resultado

- Convenção do compilador GCC no IA-32
  - ✓ É a convenção usual no IA-32

Tipo de resultado	Registo usado
inteiro 8 bits - char	AL
inteiro 16 bits - short int	AX
inteiro 32 bits – int, long int e apontadores	EAX
inteiro 64 bits – long long int	EDX:EAX
Vírgula flutuante	ST0

Retorno de Valores de Tipos Não Primitivos

- No C não se retorna o conteúdo de um vector ou de uma estrutura, mas sim o seu endereço
  - ✓ Endereço esse que é colocado no EAX
- Portanto o resultado é um apontador para o tipo do valor a ser retornado
  - ✓ Por exemplo **char*** caso se pretenda retonar um vector de caracteres
- Note porém o endereço a retornar não pode ser o endereço de uma variável local
  - ✓ Essa variável desaparecerá depois da execução da função
- Terá, portanto, de ser o endereço de uma variável global ou reservada dinâmicamente (malloc no C)

Salvarguarda dos Registos numa Chamada

- Por convenção os registos EAX, EDX e ECX são caller-saved

- Portanto é da responsabilidade da subrotina salvaguardar o conteúdo de todos os outros registos que queira utilizar

Limpar os Parâmetros da Pilha

- Por permitir funções com número variável de parâmetros
  - ✓ Exemplo printf:

```
printf (“O valor de x e´ %d\n”, x);  // 2 parâmetros
printf (“O valor de f(%d) e´ %d\n”, x, f(x));  // 3 parâmetros
```

- A linguagem C convenciona que deve ser a subrotina que faz a chamada, a que coloca os parâmetros na pilha, que deve retirá-los

Assembly Embebido no Código C

- A palavra-chave **asm** permite embeber código assembly em código C
- Exemplo GCC:

```
asm(
        "xor   ax, ax"
        "mov   al, dh"
);
```

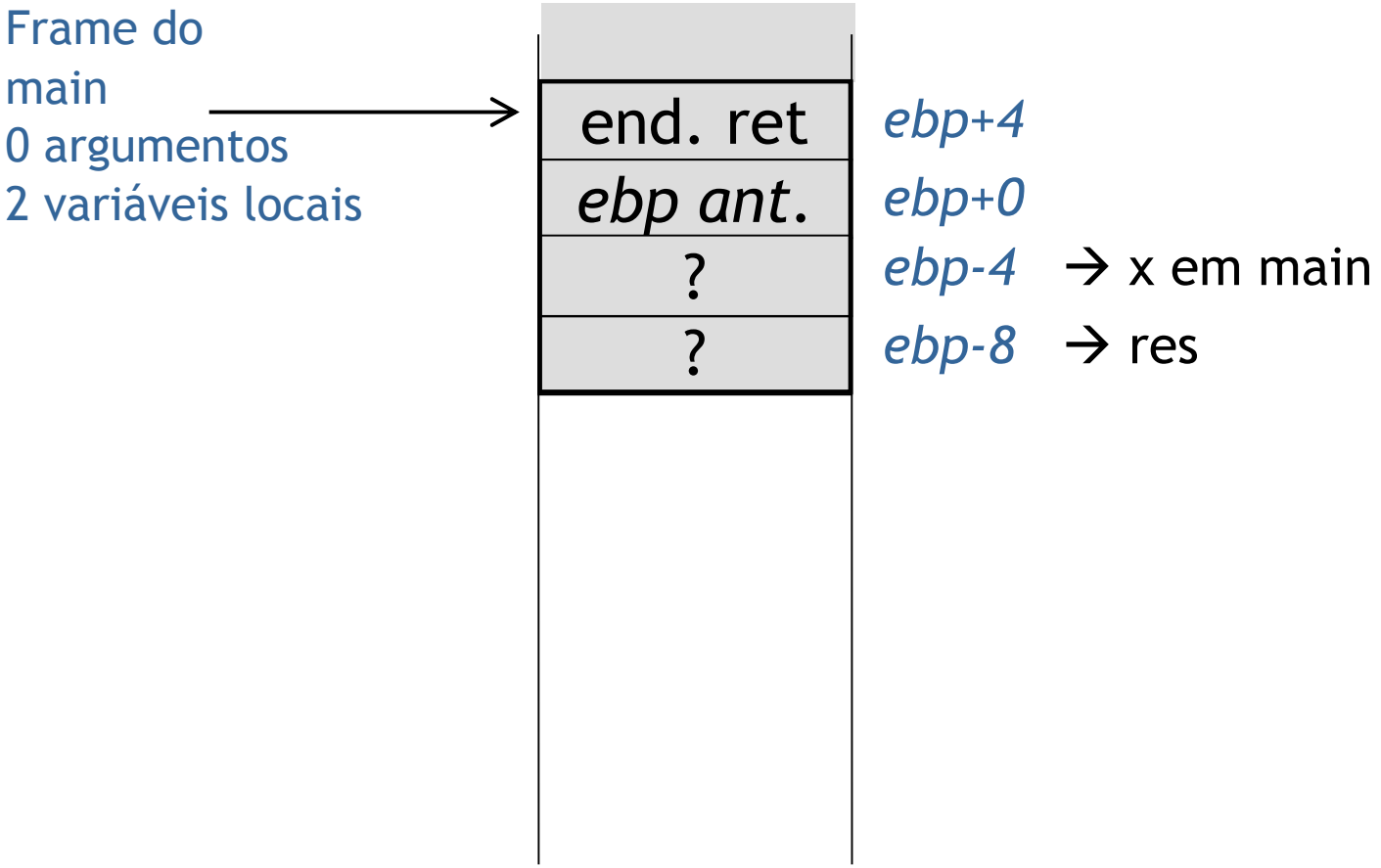
- Exemplo compilador Borland C:

```
asm {
        xor   ax, ax
        mov   al, dh
}
```

Ambiente das Funções na Linguagem C

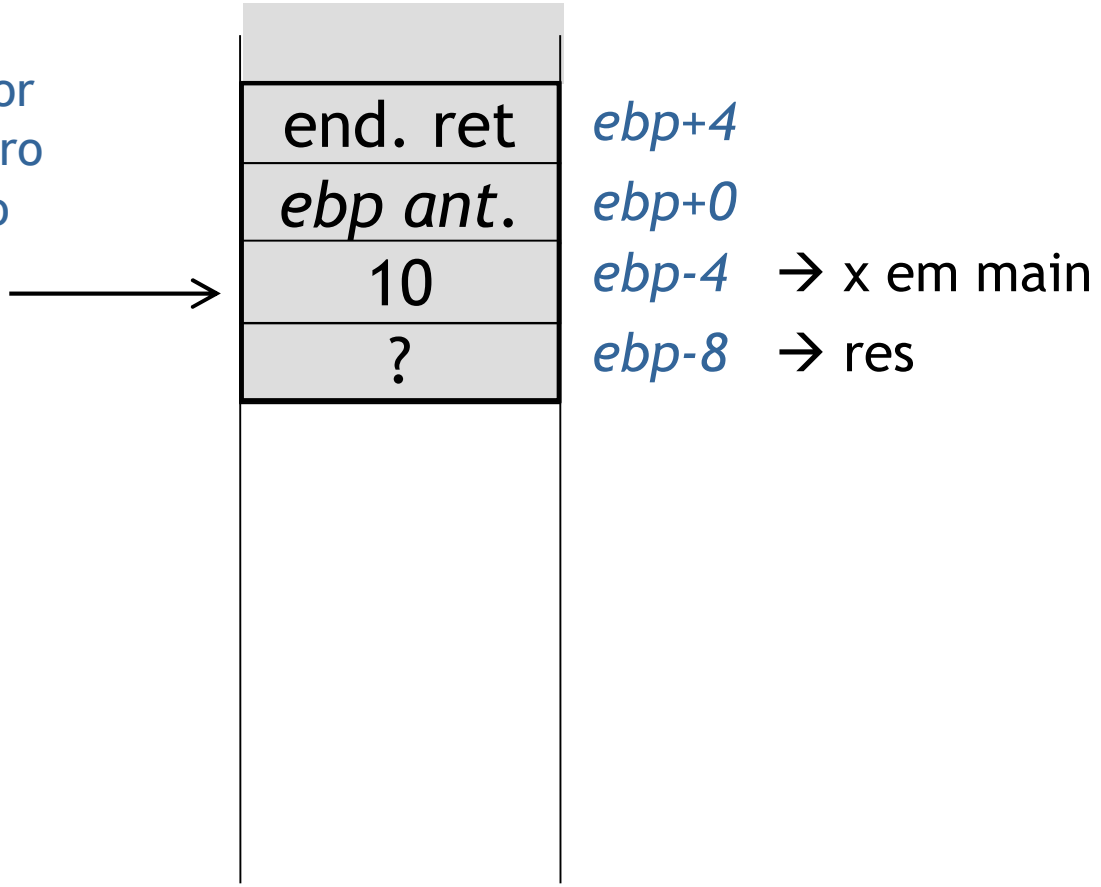
```
#include <stdio.h>
int par(int);

int main() {
   int x;
   scanf("%d", &x);
   int res = par(x);
   if (res)
       printf("%d e' par\n", x);
   else
       printf("%d e' impar\n", x);
   return 0;
}

int par (int x) {
   return !(x%2);
}
```

Ambiente da função main



Ambiente da função main

Suponhamos que o utilizador insere o número 10 aquando do scanf



Chamada de Funções

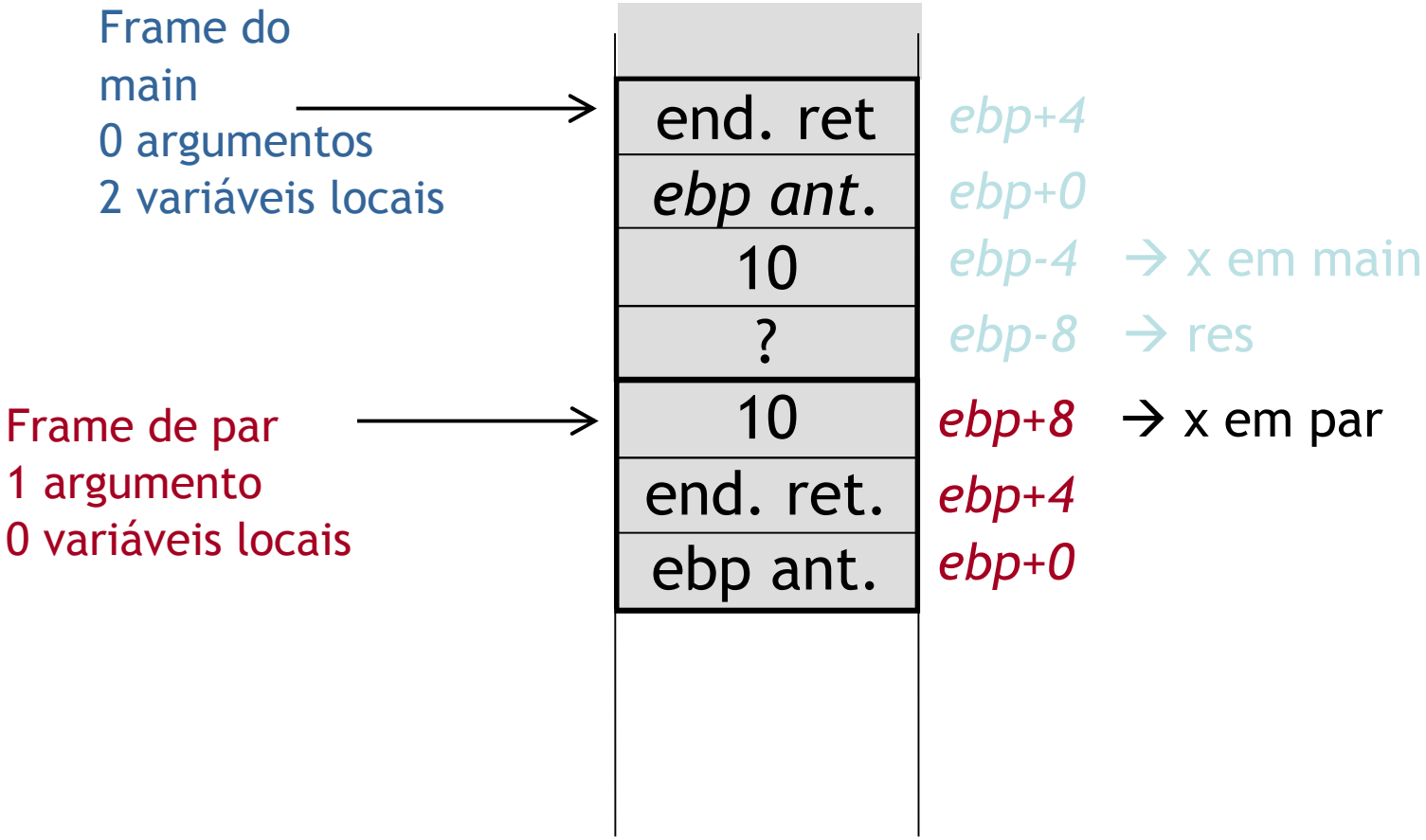
- `res = par(x)` é compilado para (ou equivalente):

```
push dword [ebp-4]  ; 1ª variável local de main → x
call par
add esp, 4          ; Desempilhar o argumento
mov [ebp-8], eax    ; Guardar resultado em res
```

- A função `par` é compilada para (ou equivalente):

```
par:
  push ebp
  mov ebp, esp
  mov eax, [ebp+8]    ; 1º argumento
  …                   ; Cálculo do resultado e
  …                   ; colocá-lo em eax
  pop ebp
  ret
```

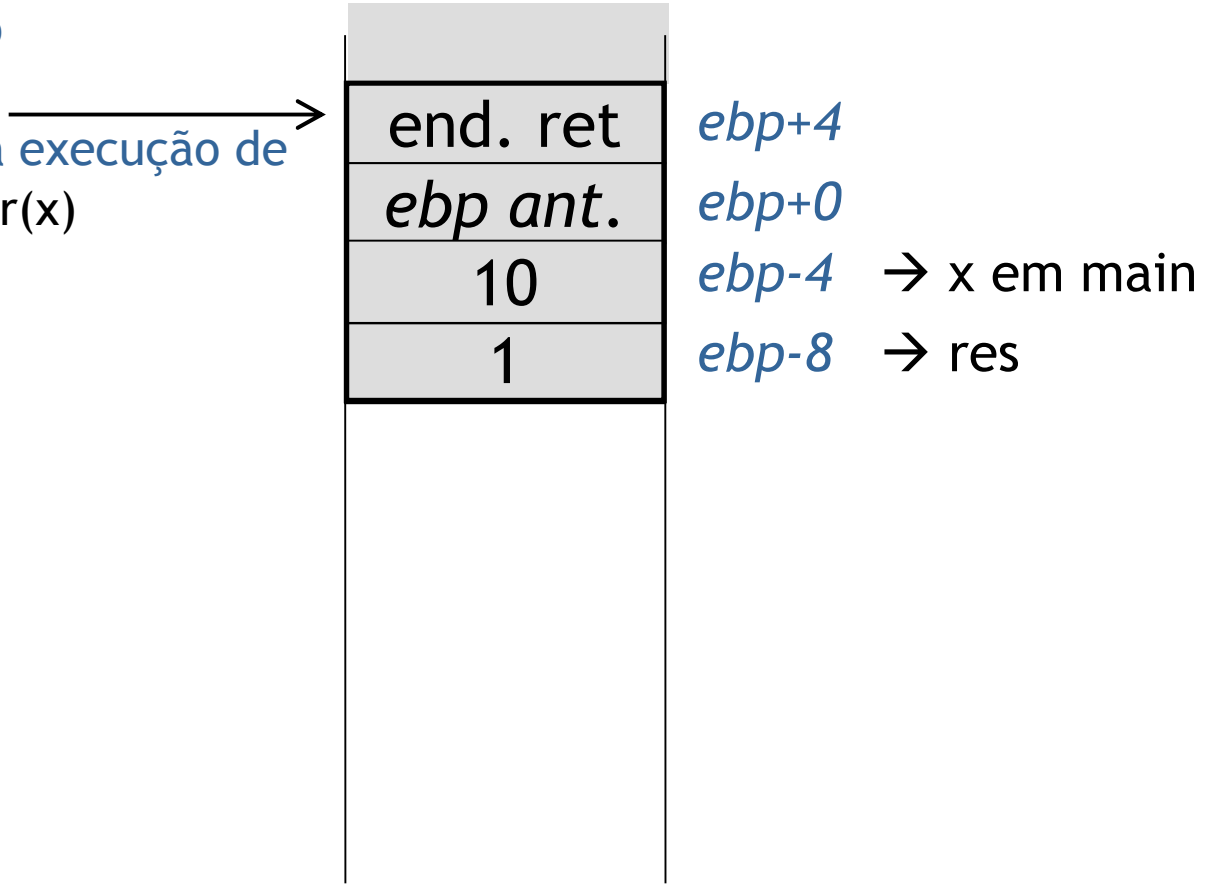
Chamada de Funções: a Pilha



Nota: a função par pode ter uma variável local onde é guardado o resultado do cálculo de x%2. Depende do código gerado pelo compilador.

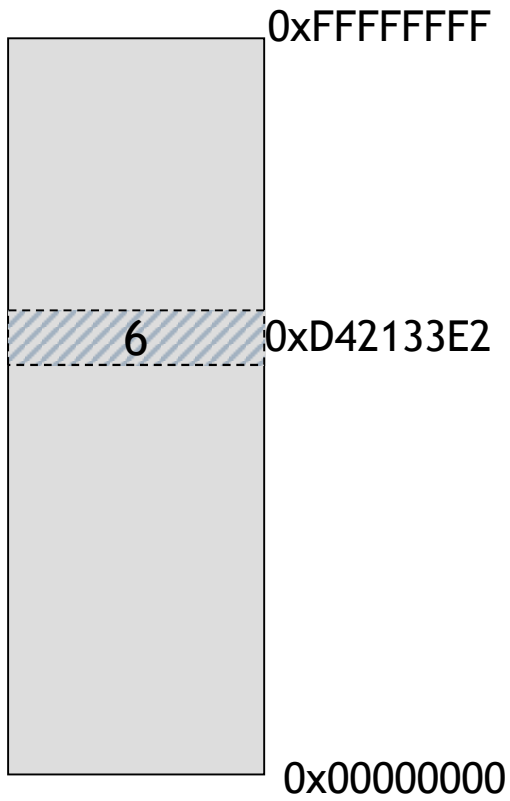
Chamada de Funções: a Pilha

Frame do
main
depois da execução de
res = par(x)



Passagem de Parâmetros: Modo

- Suponhamos a existência de uma variável X com endereço `0xD42133E2` e valor 6



- Passagem por valor:

  `push` `[X]` ou **`push`** `[0xD42133E2]`

  - ✓ A subrotina recebe o valor 6

- Passagem por referência:

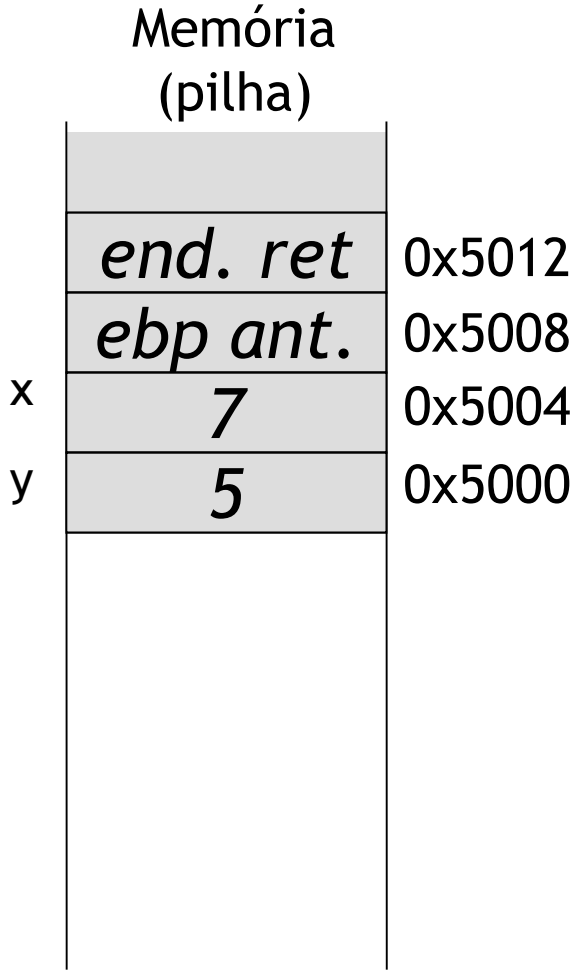
  `push dword` `X` ou

  `push dword` `0xD42133E2`

  - ✓ A subrotina recebe o endereço 0xD42133E2 onde está colocado o valor

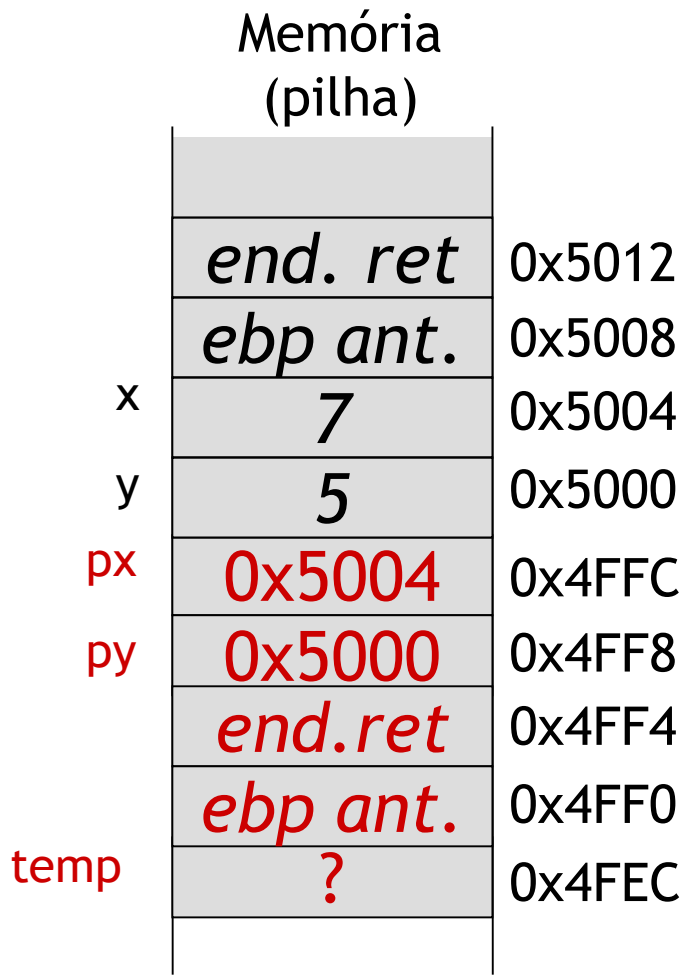
Passagem por Referência no C

```
int main() {
  int x = 7, y = 5;
  printf("Valor de X: %d\n", x);
  printf("Valor de Y: %d\n", y);
  troca(&x, &y);
  printf("Valor de X: %d\n", x);
  printf("Valor de Y: %d\n", y);
  return 0;
}

void troca(int *px, int *py) {
  int temp;
  temp = *px;
  *px = *py;
  *py = temp;
}
```



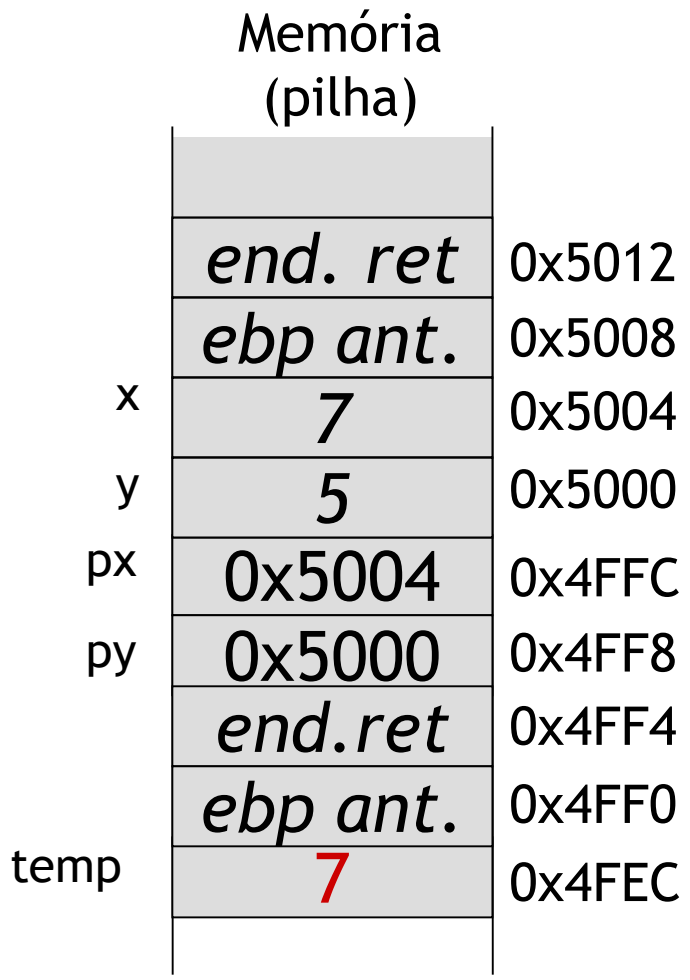
Passagem por Referência no C

```
int main() {
  int x = 7, y = 5;
  printf("Valor de X: %d\n", x);
  printf("Valor de Y: %d\n", y);
  troca(&x, &y);
  printf("Valor de X: %d\n", x);
  printf("Valor de Y: %d\n", y);
  return 0;
}

void troca(int *px, int *py) {
  int temp;
  temp = *px;
  *px = *py;
  *py = temp;
}
```



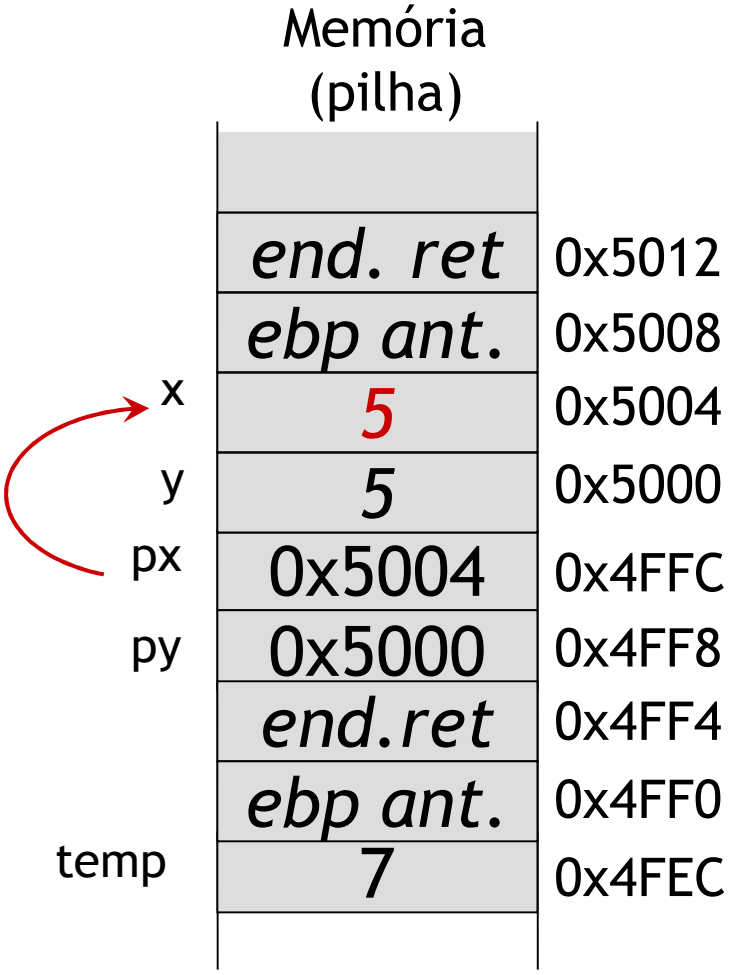
Passagem por Referência no C

```
int main() {
  int x = 7, y = 5;
  printf("Valor de X: %d\n", x);
  printf("Valor de Y: %d\n", y);
  troca(&x, &y);
  printf("Valor de X: %d\n", x);
  printf("Valor de Y: %d\n", y);
  return 0;
}

void troca(int *px, int *py) {
  int temp;
  temp = *px;
  *px = *py;
  *py = temp;
}
```



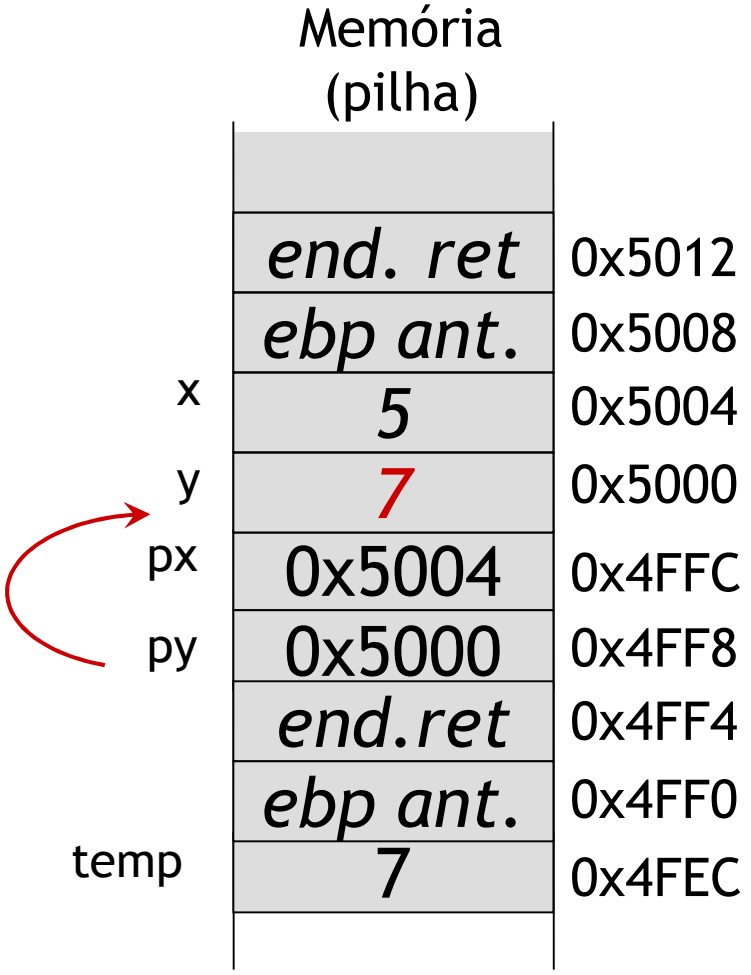
Passagem por Referência no C

```
int main() {
  int x = 7, y = 5;
  printf("Valor de X: %d\n", x);
  printf("Valor de Y: %d\n", y);
  troca(&x, &y);
  printf("Valor de X: %d\n", x);
  printf("Valor de Y: %d\n", y);
  return 0;
}

void troca(int *px, int *py) {
  int temp;
  temp = *px;
  *px = *py;
  *py = temp;
}
```



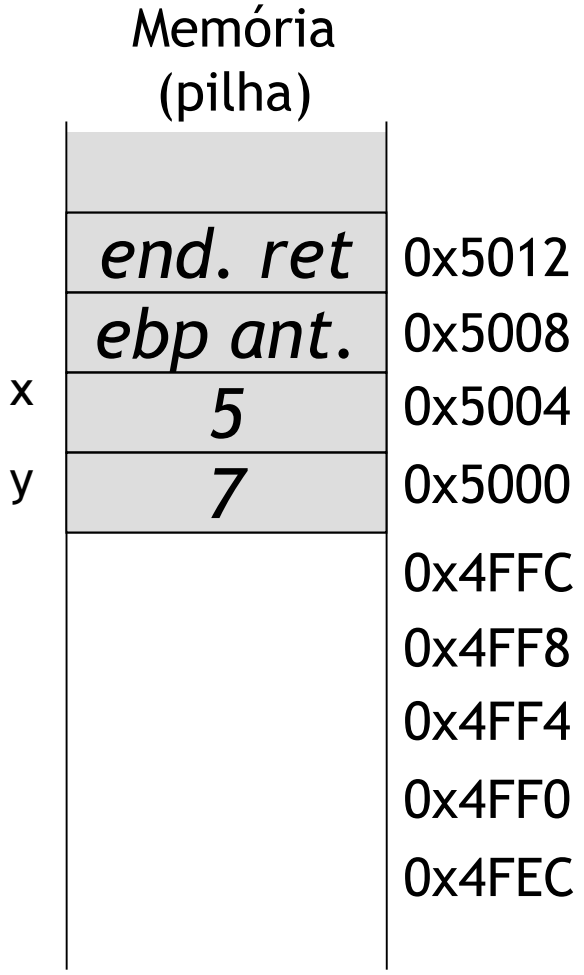
Passagem por Referência no C

```
int main() {
  int x = 7, y = 5;
  printf("Valor de X: %d\n", x);
  printf("Valor de Y: %d\n", y);
  troca(&x, &y);
  printf("Valor de X: %d\n", x);
  printf("Valor de Y: %d\n", y);
  return 0;
}

void troca(int *px, int *py) {
  int temp;
  temp = *px;
  *px = *py;
  *py = temp;
}
```



Passagem por Referência no C

```
int main() {
  int x = 7, y = 5;
  printf("Valor de X: %d\n", x);
  printf("Valor de Y: %d\n", y);
  troca(&x, &y);
  printf("Valor de X: %d\n", x);
  printf("Valor de Y: %d\n", y);
  return 0;
}

void troca(int *px, int *py) {
  int temp;
  temp = *px;
  *px = *py;
  *py = temp;
}
```



Código Reentrante

- Código que depende apenas dos valores passados por parâmetro
  - ✓ Produz o mesmo resultado independentemente do número de vezes e de quando é chamado
  - ✓ É código que pode ser executado concorrentemente consigo próprio, ou seja, reentrar aquando da sua execução
- Algumas das condições:
  - ✓ Não pode alterar o seu próprio código
  - ✓ Não pode aceder variáveis globais não constantes
  - ✓ Não pode chamar código não reentrante
- Vantagens:
  - ✓ Pode ser chamado recursivamente
  - ✓ Pode ser partilhado por vários processos do sist. operativo