

# Sistemas Hardware-Software

Aula 02 – Dados na memória RAM e código executável

**Engenharia**  
Fabio Lubacheski  
Maciel C. Vidal  
Igor Montagner  
Fábio Ayres

# Aula de hoje !

## Representação de dados em RAM

- Endianness – ordem dos bytes em um inteiro na memória.
- Arrays, strings e ponteiros
- Structs
- Representação de Código executável



# Atividade prática

## Experimentos (15 minutos)

1. Clone o repositório da disciplina

<https://github.com/Insper/SistemasHardwareSoftware.git>

2. Acesse os arquivos da

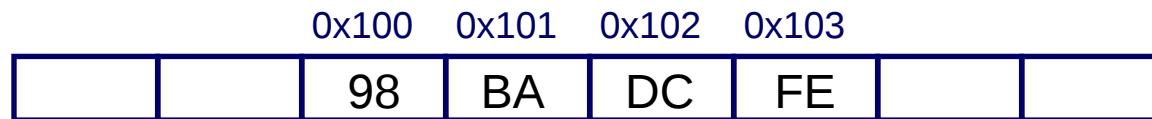
pasta **../SistemasHardwareSoftware/content/aulas/02-ram/**

3. Siga as orientações do handout **EXPERIMENTOS** da aula **02-ram**.

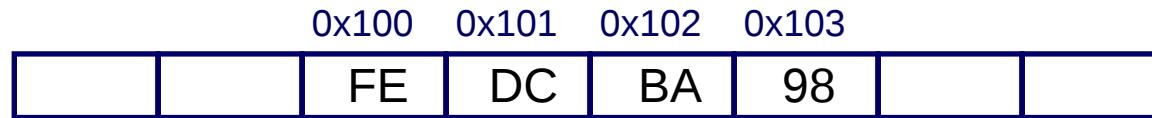
# Little endian versus big endian

```
int i = 0xFEDCBA98;
```

**LittleEndian** → Byte **menos** significativo primeiro  
**LSB** (Least Significant Byte)



**BigEndian** → Byte **mais** significativo primeiro  
**MSB** (Most Significant Byte)



# Little endian versus big endian

- Unidade de trabalho é o byte!
- CPUs Intel/AMD (**x64**) são **little endian**
- CPUs SPARC são big endian
- ARM/PowerPC pode ser little/big endian
- Vale para todos os tipos de dados nativos (inteiros, ponteiros e fracionários)
- **Agora conseguimos entender o `experimento1.c`**

# Arrays na RAM – experimento2.c

```
short arr[] = {1, 2, 3, 4, 5};  
show_bytes((unsigned char *) &arr, sizeof(short) * 5);
```

Qual a saída do código acima?

# Arrays na RAM – experimento2.c

```
short arr[] = {1, 2, 3, 4, 5};  
show_bytes((unsigned char *) &arr, sizeof(short) * 5);
```

Qual a saída do código acima?

01 00 02 00 03 00 04 00 05 00

# Strings na RAM – experimento3.c

```
|char *string = "Oi C :- )";  
show_bytes((unsigned char *) string, strlen(string) + 1);
```

Saída do código acima

String:  
Oi C :- )

Valor guardado no array:  
'O' (4f) | 'i' (69) | ' ' (20) | 'C' (43)\_| ' ' (20) | ':' (3a)

# Ponteiros na RAM – experimento4.c

```
int a = 10;
int *ap = &a;
printf("Endereço de a\t: %p\nPróximo int\t: %p\n", ap, ap+1);

long l = 10;
long *lp = &l;
printf("Endereço de l\t: %p\nPróximo long\t: %p\n", lp, lp+1);
```

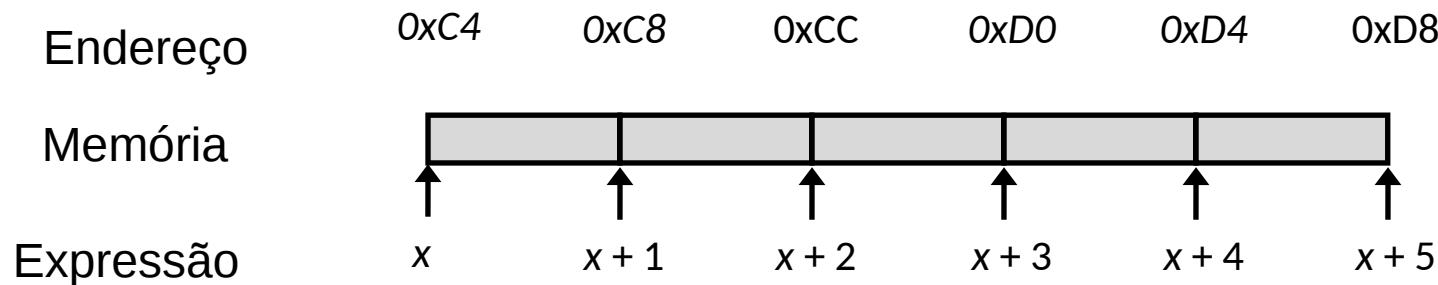
Saída do código acima

Endereço de a	:	0x7ffc14ab5a1c
Próximo int	:	0x7ffc14ab5a20
Endereço de l	:	0x7ffc14ab5a20
Próximo long	:	0x7ffc14ab5a28

# Ponteiros na RAM – experimento4.c

Ponteiro representa um endereço. Podemos fazer aritmética !

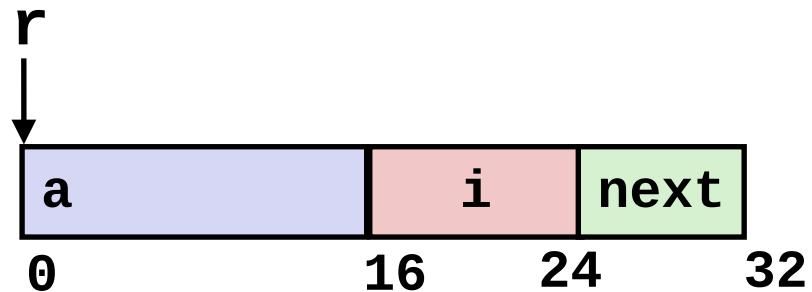
```
int *x; //0xC4
```



$$*(x+i) \leftrightarrow x[i]$$

# Structs na RAM

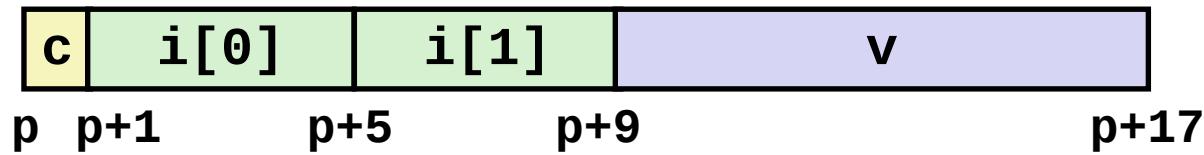
```
struct rec {  
    int a[4];  
    long i;  
    struct rec *next;  
};
```



- Bloco contíguo de memória
- Campos armazenados na ordem dada na declaração
  - Compilador não muda ordem dos campos
- **Tamanho e offset** exato dos campos fica a **cargo do compilador**
- Código de máquina não conhece structs
  - Quem organiza o código é o compilador

# Structs na RAM - alinhamento

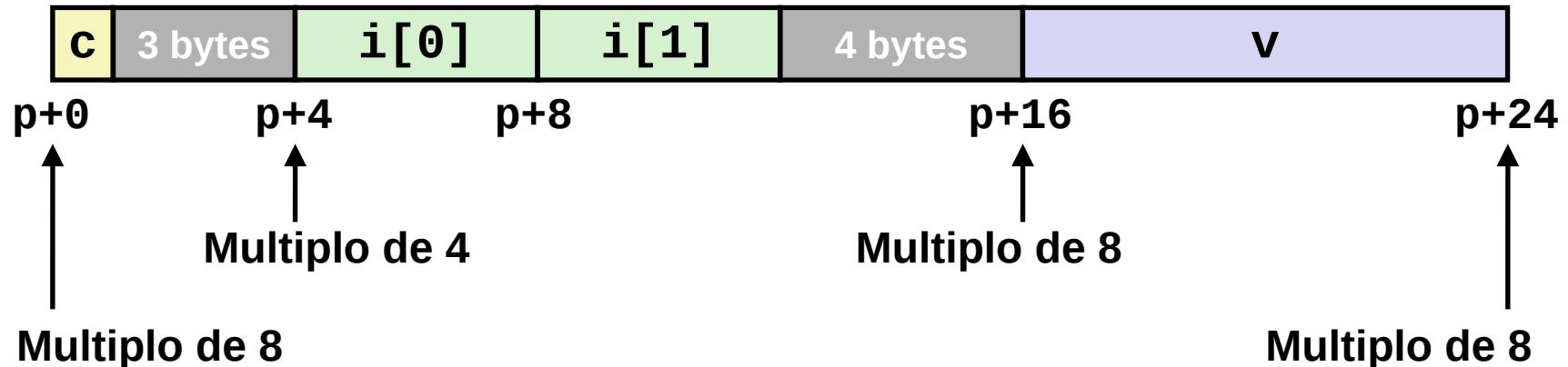
Dados desalinhados



```
struct S1 {  
    char c;  
    int i[2];  
    double v;  
} *p;
```

Dados alinhados:

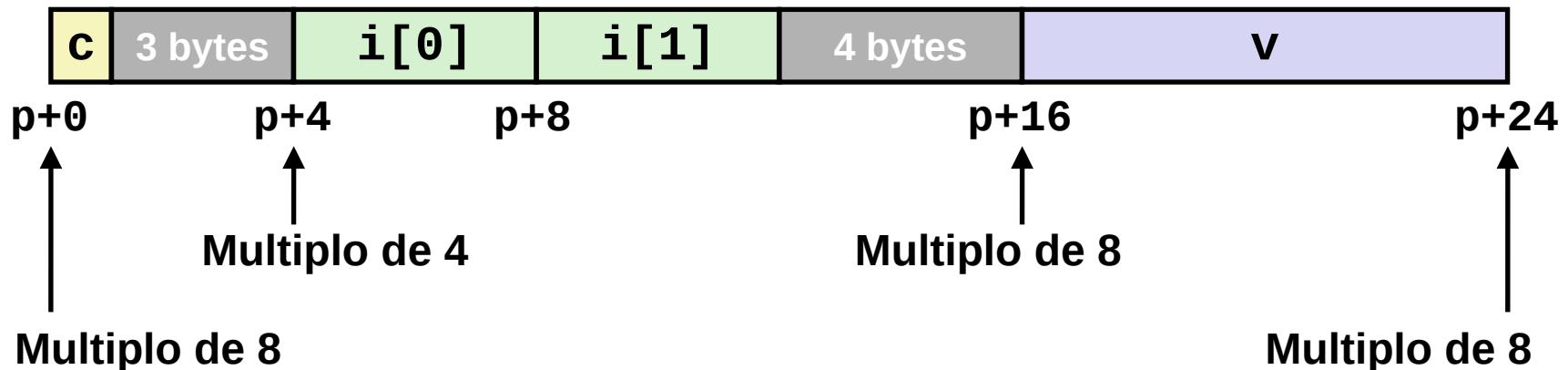
- Se o item requer K bytes...
- ... Então o endereço deve ser múltiplo de K.



# Structs na RAM - alinhamento

- Motivo: Memória é acessada em blocos alinhados de 8 bytes
  - Simplicidade de design de hardware
  - x86-64 funciona mesmo sem alinhamento, mas implica em perda de performance
- Alinhamento da struct = maior alinhamento de seus membros.

```
struct S1 {  
    char c;  
    int i[2];  
    double v;  
} *p;
```



# Structs na RAM - alinhamento

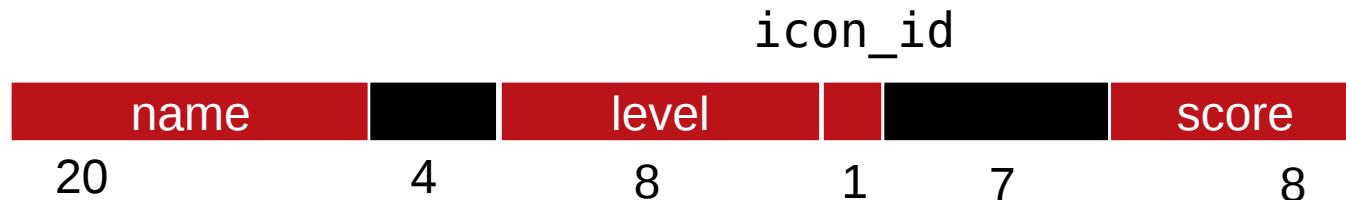
```
struct player {  
    char name[20];  
    long level;  
    char icon_id;  
    long score;  
};
```

Desenhe o layout de memória de player levando em conta alinhamento.

# Structs na RAM - alinhamento

```
struct player {  
    char name[20];  
    long level;  
    char icon_id;  
    long score;  
};
```

Desenhe o layout de memória de player levando em conta alinhamento.



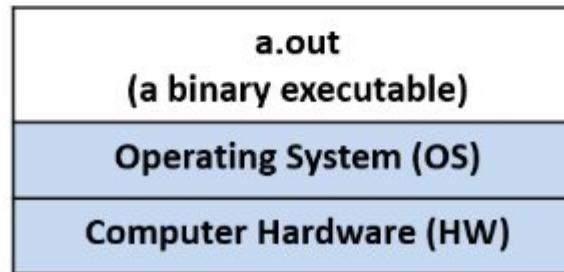
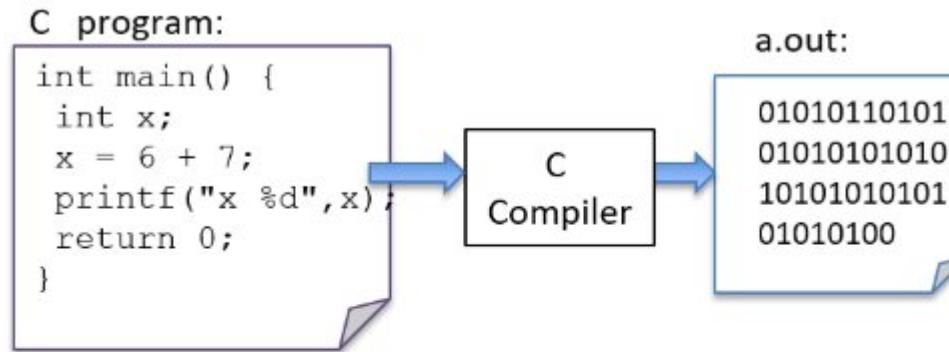
48 bytes  
11 bytes  
“desperdiçados”

# Resumo: dados na memória

- Inteiros (**little endian**)
- Arrays e matrizes (aritmética de endereços)
- Strings (array com char '\0' no fim)
- Struct (alinhamento; ponteiro para começo mais deslocamentos)

# Representação de código

Como o código é transformado em executável?



fonte:[https://diveintosystems.org/book/C1-C\\_intro/getting\\_started.html](https://diveintosystems.org/book/C1-C_intro/getting_started.html)

# Representação de código

Como o código é transformado em executável?

Código C/C++ → Assembly → Código de máquina

Para ver o Binário e Assembly use:

```
$ objdump -d <executavel>
```

# Estrutura dos arquivos executáveis

## *Executable and Linkable Format (ELF)*

- Formato de arquivo executável em máquinas x86-64 Linux

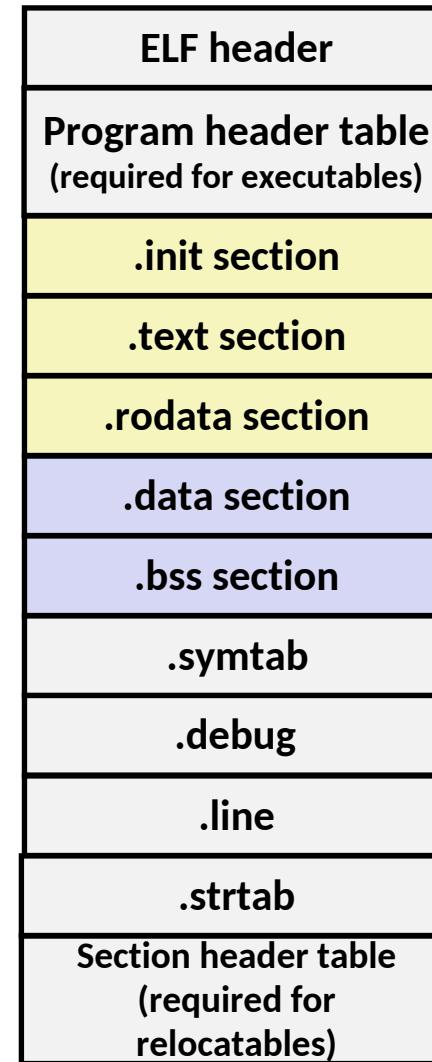
### Seções importantes

- **.text**: código executável
- **.rodata**: constantes
- **.data**: variáveis globais pré-inicializadas
- **.bss**: variáveis globais não-inicializadas

### Outros formatos:

- *Portable Executable (PE)*: Windows
- *Mach-O*: Mac OS-X

## Executable Object File



# Estrutura dos arquivos executáveis

*Executable and Linkable Format (ELF)*

- Formato de arquivo executável em máquinas x86-64 Linux

Seções importantes

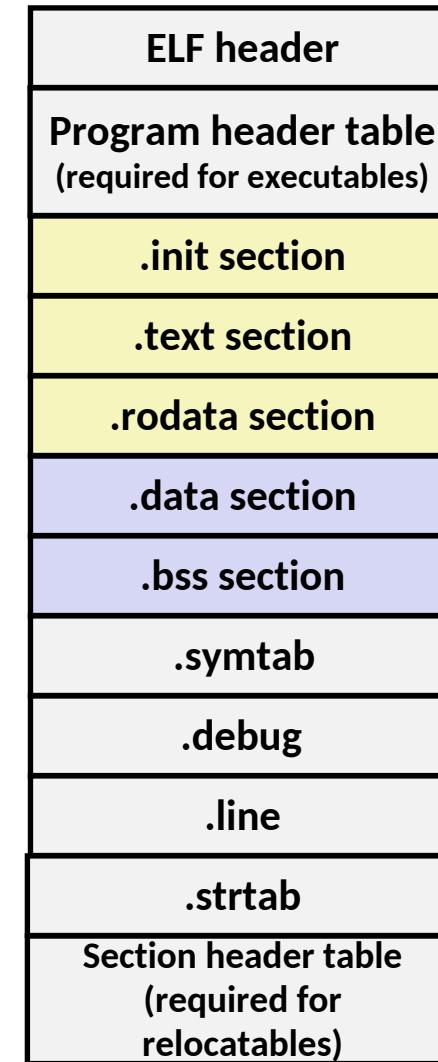
- .text**: código executável
- .rodata**: constantes
- .data**: variáveis globais pré-inicializadas
- .bss**: variáveis globais não-inicializadas

Outros formatos:

- Portable Executable (PE)*: Windows
- Mach-O*: Mac OS-X

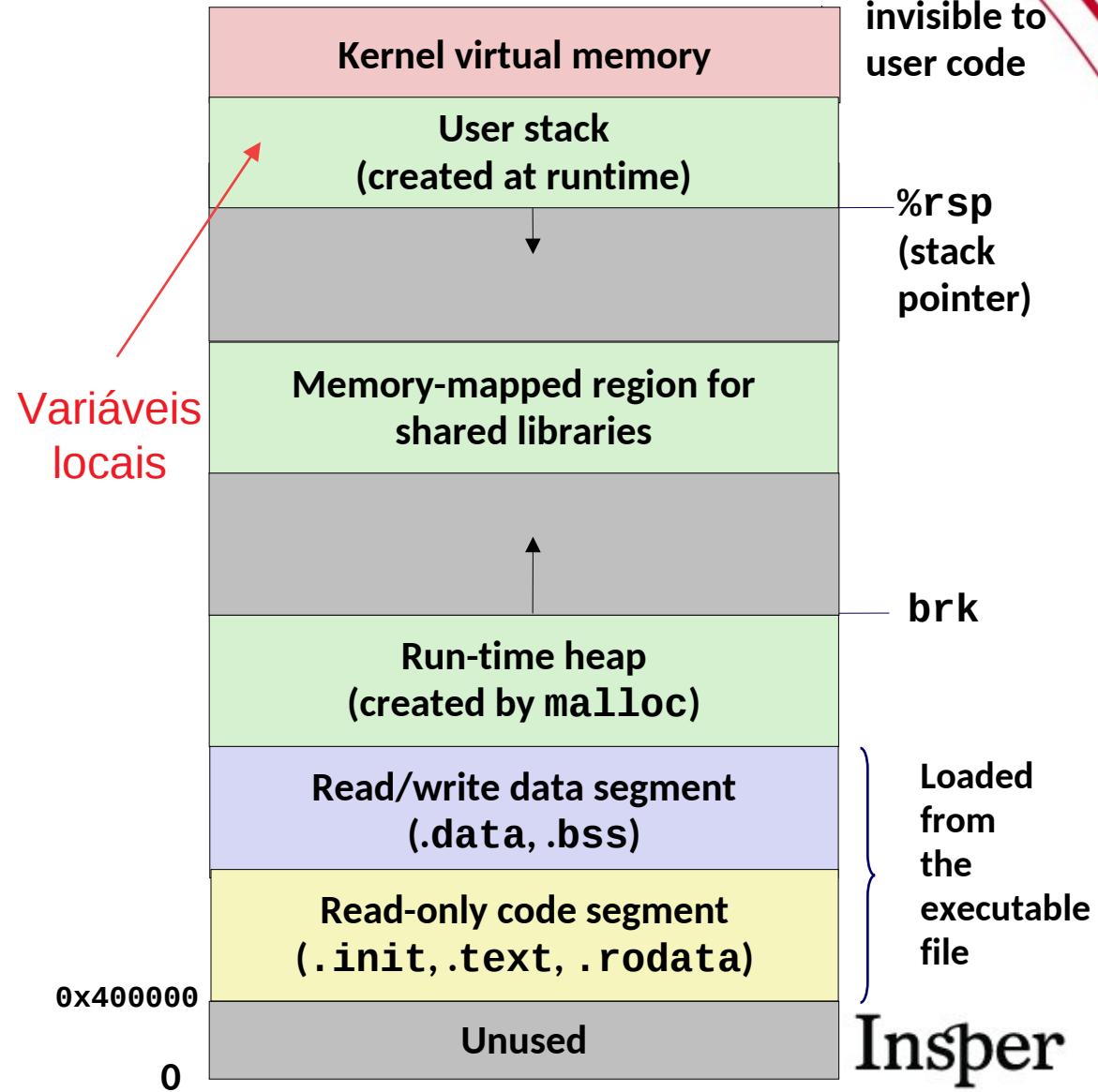
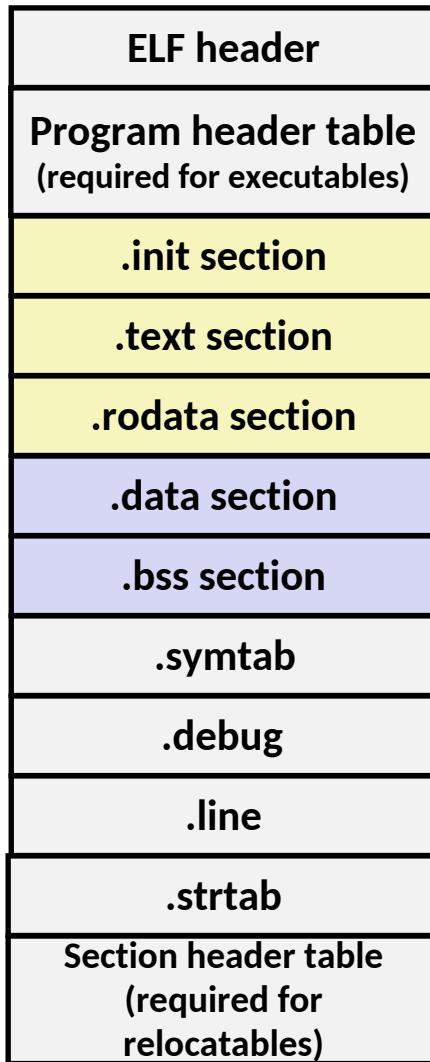
Cadê as variáveis locais?

Executable Object File



# Executável na memória

## Executable Object File



Insp<sup>er</sup>

# Representação de código

Um arquivo executável que contém dados globais e nosso código em instruções **x64**

- Executável tem várias seções
- **.text** guarda nosso código
- **.data** guarda globais inicializadas
- **.rodata** guarda constantes
- **.bss** reserva espaço para globais não inicializadas
- Variáveis locais só existem na execução do programa



# Atividade prática

## Examinando a execução de programas usando GDB

1. abrir código executável em C
2. examinar seu conteúdo (funções declaradas e valores de variáveis globais)

# Insper

[www.insper.edu.br](http://www.insper.edu.br)