

# Sistemas Hardware-Software

Aula 3 – Dados na memória RAM e código executável

2020 – Engenharia

Igor Montagner  
Fábio Ayres



# Atividade prática

## Experimentos (20 minutos)

1. Compilar e executar experimentos0-4.c
2. Anotar resultados para discussão

# Representação de dados em RAM

- Endianness
- Arrays e matrizes
- Strings
- Código

# Little endian versus big endian

```
int i = 0x11223344;
```

## Little Endian

		0x100	0x101	0x102	0x103		
		44	33	22	11		

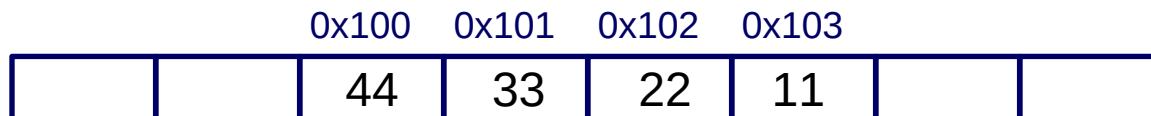
## Big Endian

		0x100	0x101	0x102	0x103		
		11	22	33	44		

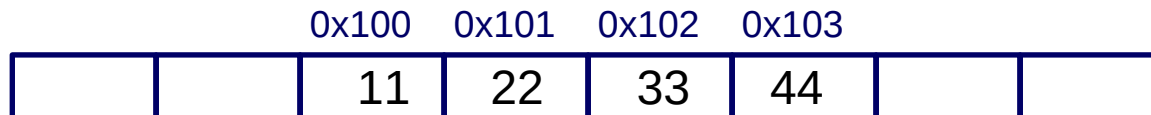
# Little endian versus big endian

```
int i = 0x11223344;
```

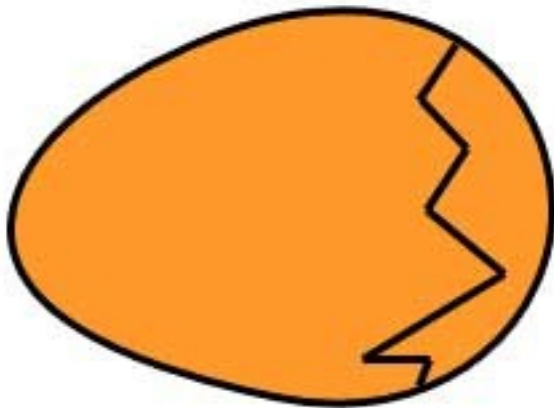
**Little Endian** → Byte **menos** significativo primeiro



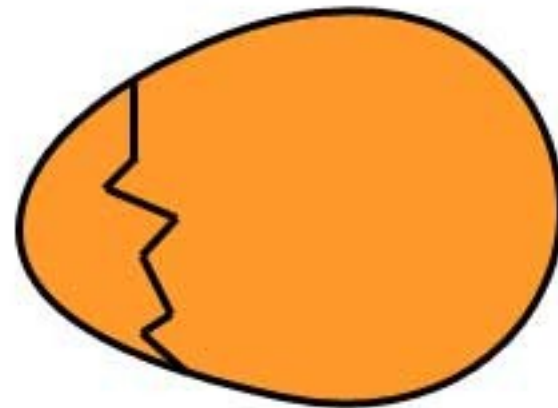
**Big Endian** → Byte **mais** significativo primeiro



# Little endian versus big endian



**BIG ENDIAN** - The way people always broke their eggs in the Lilliput land



**LITTLE ENDIAN** - The way the king then ordered the people to break their eggs

# Little endian versus big endian

- Unidade de trabalho é o byte!
- CPUs Intel/AMD (x64) são little endian
- ARM pode ser little/big endian
- Vale para todos os tipos de dados nativos (inteiros, ponteiros e fracionários)

## Vantagens:

- Cast simples
- Operações com inteiros enormes

# Endianness importa para arrays?

```
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 em RAM

```
igor@igor-elementary:~/Dropbox/INSPER/2020/sistemas-hardware-software/aulas/03-ram/src$ ./e3  
Valor guardado o array : '0' (4f) | 'i' (69) | ' ' (20) | 'C' (43)  
| ' ' (20) | ':' (3a) | ')' (29) | '' (00) |
```

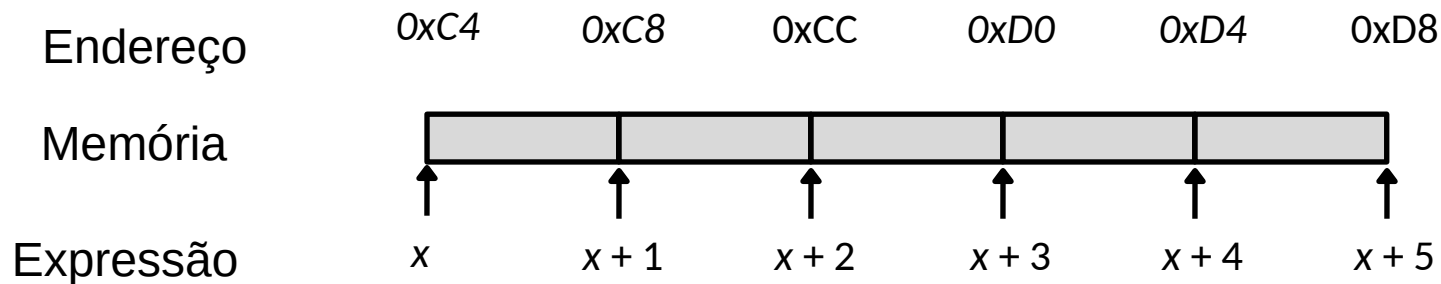
# Ponteiros em RAM

```
igor@igor-elementary:~/Dropbox/INSPER/2  
are/aulas/03-ram/src$ ./e4  
Endereço de a: 0x7ffeb7d0323c  
Próximo int: 0x7ffeb7d03240  
Endereço de l: 0x7ffeb7d03240  
Próximo long: 0x7ffeb7d03248
```

# Ponteiros em RAM

Ponteiro representa um endereço. Podemos fazer aritmética !

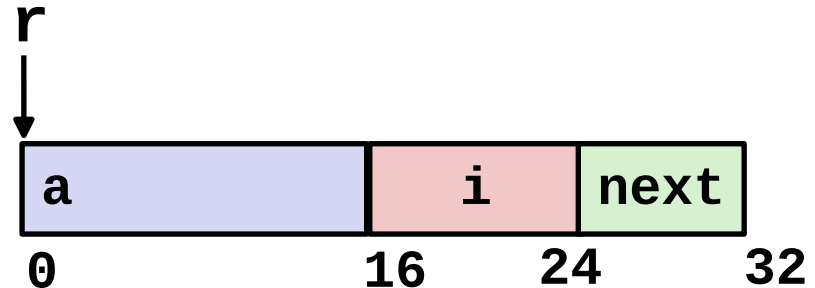
```
int *x; //0xC4
```



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

# Structs em RAM

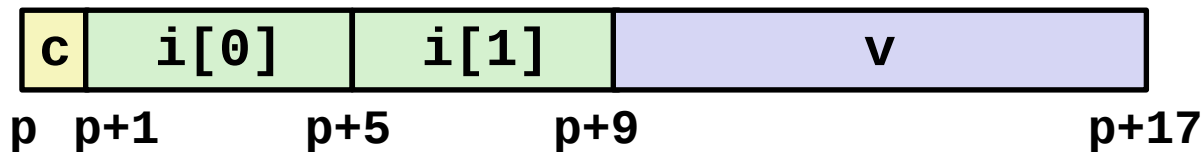
```
struct rec {  
    int a[4];  
    size_t i;  
    struct rec *next;  
};
```



- 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 em RAM - alinhamento

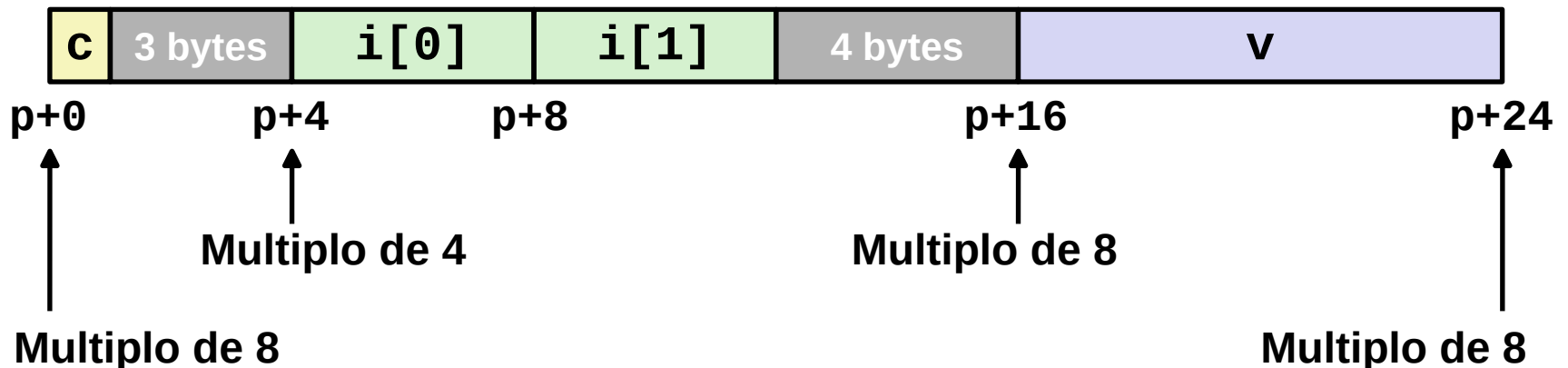
Dados desalinhados



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

Dados alinhados:

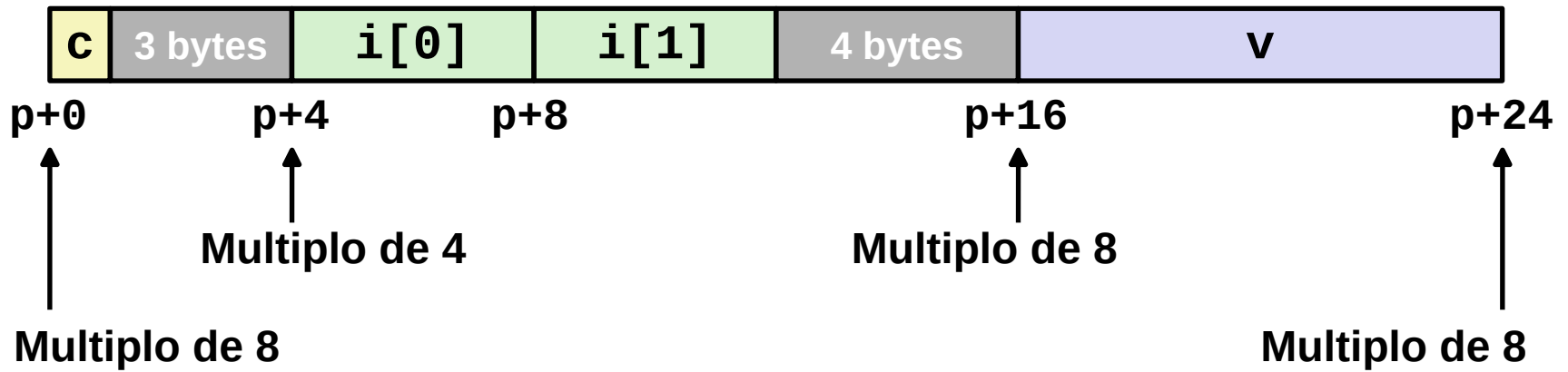
- Se o item requer K bytes...
- ... Então o endereço deve ser múltiplo de K.



# Structs em 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 em RAM - alinhamento

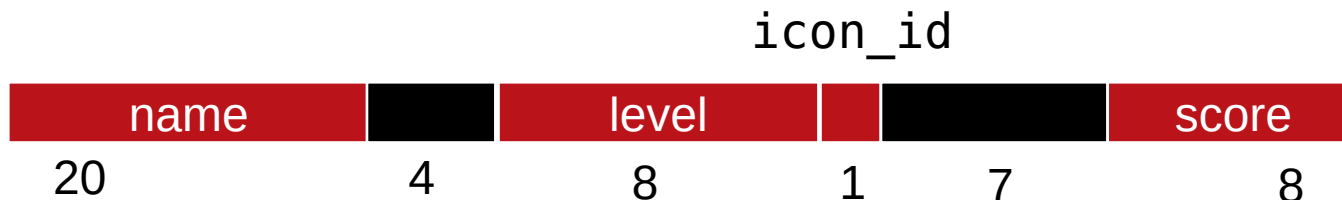
```
struct player {  
    char name[20];  
    long level;  
    char icon_id;  
    long score;  
};
```

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

# Structs em 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”



# Dados na memória



- Inteiros e float (endianness)
- 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?

# Representação de código

Como o código é transformado em executável?

Código C/C++            Assembly            Código de máquina

# Representação de código

Como o código é transformado em executável?

Código C/C++       $\longrightarrow$       Assembly       $\longrightarrow$       Código de máquina

Código de máquina vale para qualquer Sistema Operacional?

Vale para qualquer tipo de processador/CPU?

# 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

ELF header
Program header table (required for executables)
.init section
.text section
.rodata section
.data section
.bss section
.symtab
.debug
.line
.strtab
Section header table (required for relocatables)

# 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?

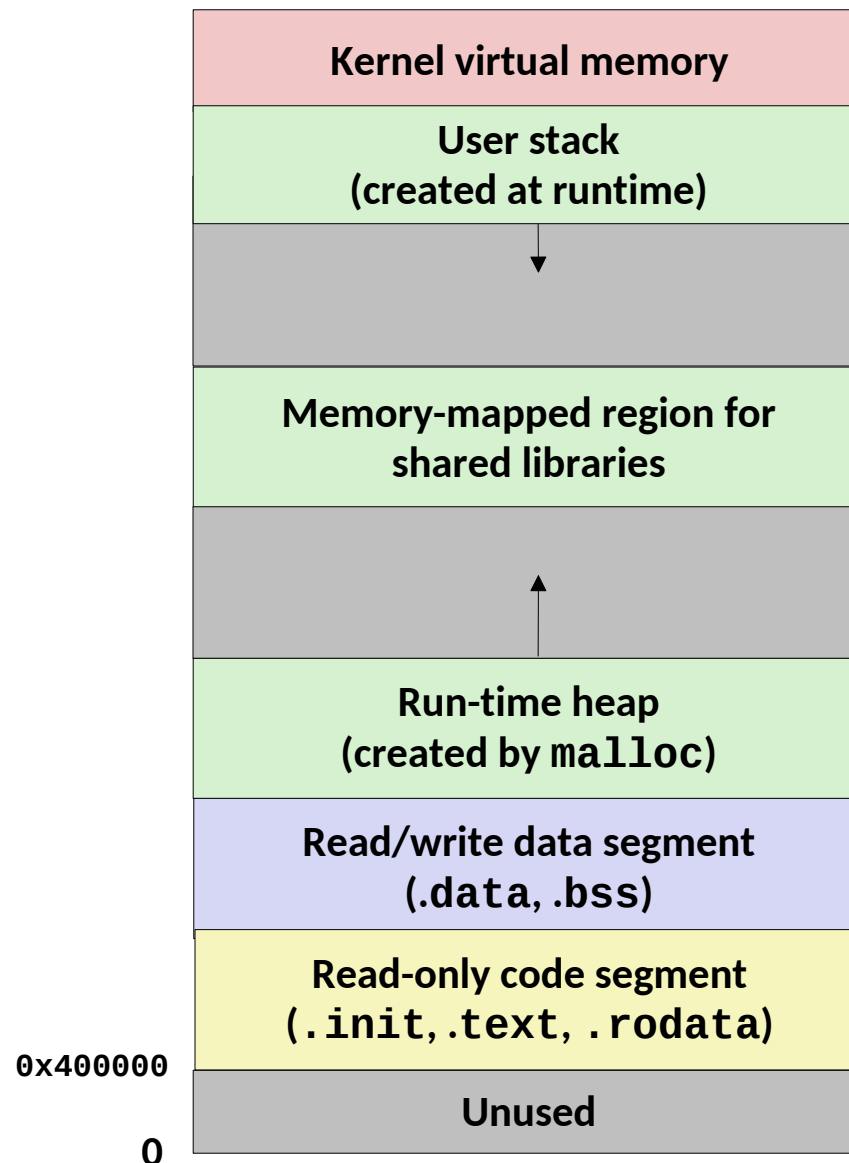
## Executable Object File

ELF header
Program header table (required for executables)
.init section
.text section
.rodata section
.data section
.bss section
.symtab
.debug
.line
.strtab
Section header table (required for relocatables)

# Executável na memória

## Executable Object File

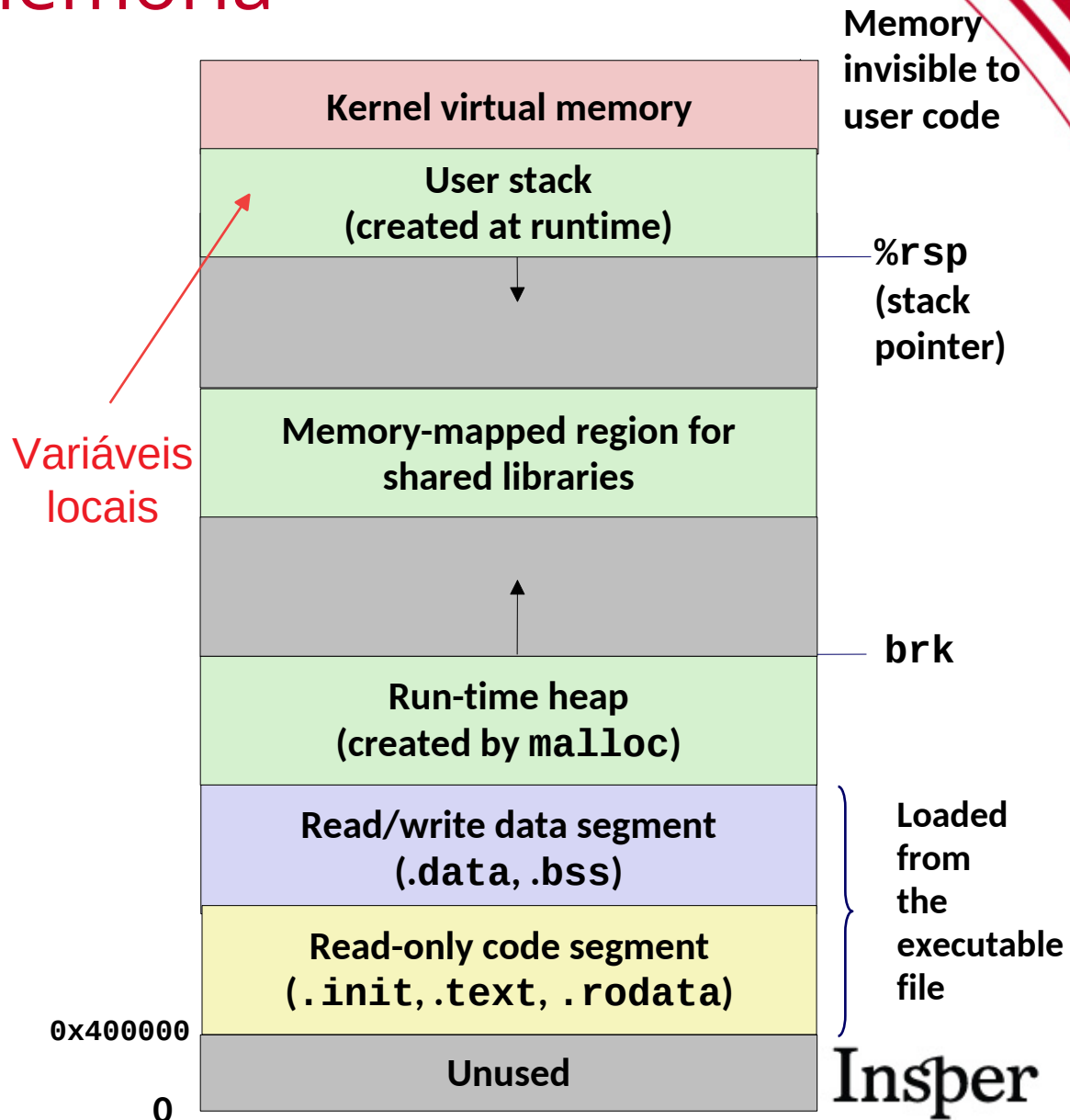
ELF header
Program header table (required for executables)
.init section
.text section
.rodata section
.data section
.bss section
.symtab
.debug
.line
.strtab
Section header table (required for relocatables)



# Executável na memória

## Executable Object File

ELF header
Program header table (required for executables)
.init section
.text section
.rodata section
.data section
.bss section
.symtab
.debug
.line
.strtab
Section header table (required for relocatables)





# 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

## Iniciando com GDB

1. abrir código executável em C
2. examinar seu conteúdo (globais e

# Insper

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