Ponteiros e Alocação Dinâmica de Memória

Manassés Ribeiro
manasses.ribeiro@ifc.edu.br

Agenda

- Ponteiros
- Operadores
- Alocação Dinâmica de Memória
- Funcionamento dos vetores em C

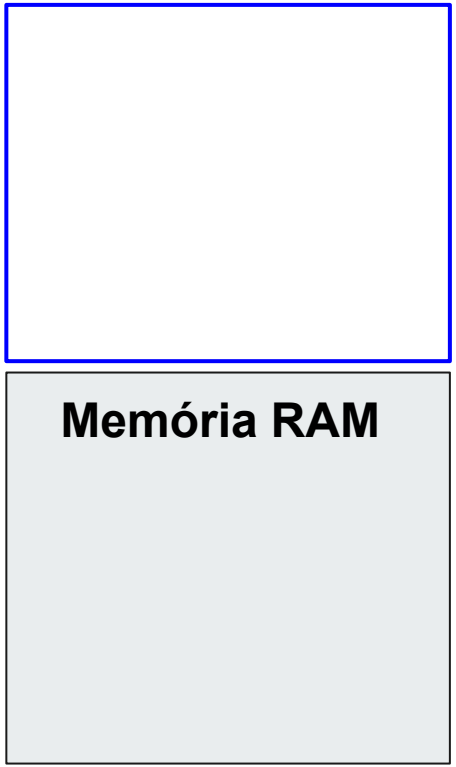
Ponteiros

- Ponteiros são “variáveis” que guardam o endereço de memória (localização) de alguma outra coisa.
  - São declarados com um * antes do nome da variável. Exemplo:

```
int *i;
float *f;
char *c;
```

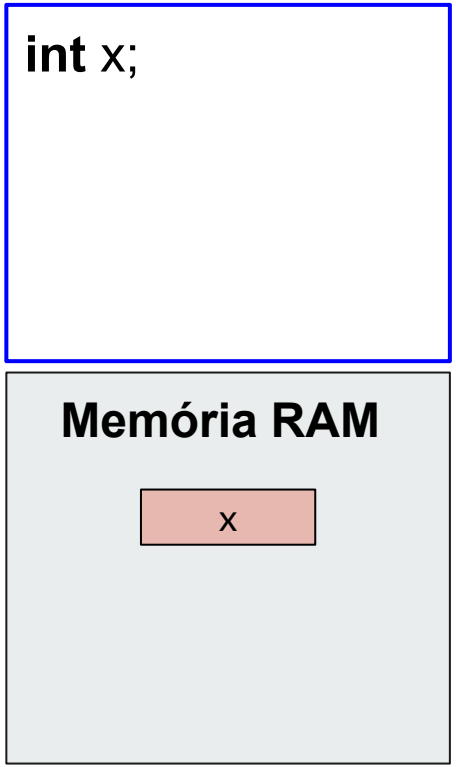
Operador &

- O operador **&** obtém o **endereço** de uma variável:
  - como ponteiros são variáveis, também ocupam memória e pode-se obter o endereço do ponteiro e ter ponteiros para ponteiros (múltiplos *).



Operador &

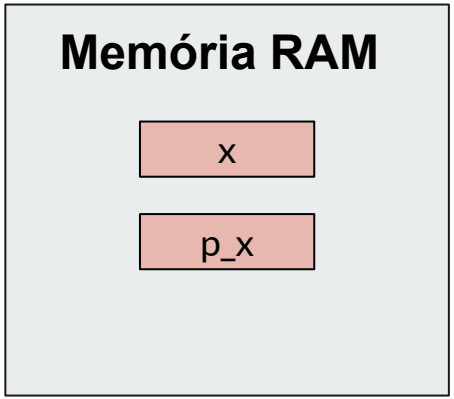
- O operador **&** obtém o **endereço** de uma variável:
  - como ponteiros são variáveis, também ocupam memória e pode-se obter o endereço do ponteiro e ter ponteiros para ponteiros (múltiplos *).



Operador &

- O operador **&** obtém o **endereço** de uma variável:
  - como ponteiros são variáveis, também ocupam memória e pode-se obter o endereço do ponteiro e ter ponteiros para ponteiros (múltiplos *).

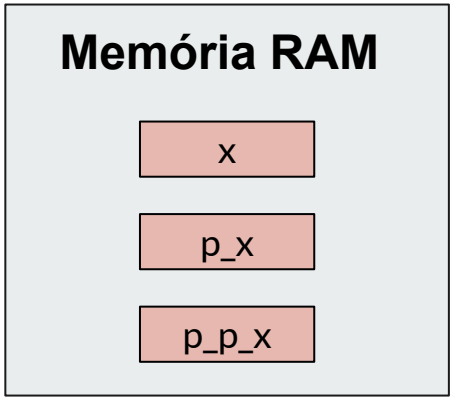
```
int x;
int *p_x;


p_x = &x;
p_p_x = &p_x;
```



Operador &

- O operador **&** obtém o **endereço** de uma variável:
  - como ponteiros são variáveis, também ocupam memória e pode-se obter o endereço do ponteiro e ter ponteiros para ponteiros (múltiplos *).

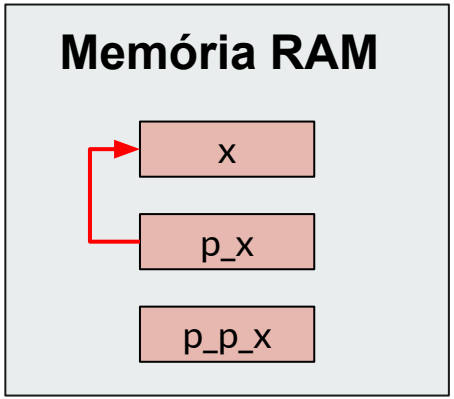
```
int x;
int *p_x;
int *p_p_x;
```



Operador &

- O operador **&** obtém o **endereço** de uma variável:
  - como ponteiros são variáveis, também ocupam memória e pode-se obter o endereço do ponteiro e ter ponteiros para ponteiros (múltiplos *).

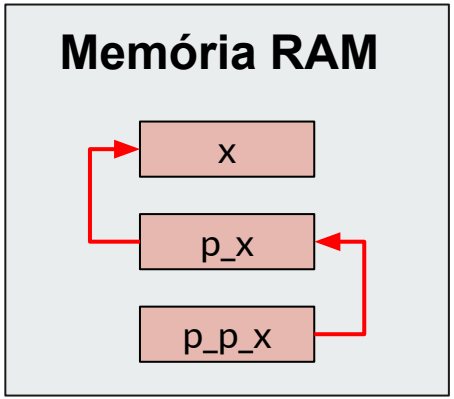
```
int x;
int *p_x;
int *p_p_x;

p_x = &x;
```



Operador &

- O operador **&** obtém o **endereço** de uma variável:
  - como ponteiros são variáveis, também ocupam memória e pode-se obter o endereço do ponteiro e ter ponteiros para ponteiros (múltiplos *).

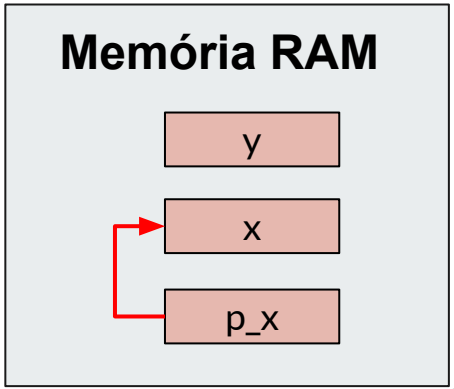
```
int x;
int *p_x;
int *p_p_x;

p_x = &x;
p_p_x = &p_x;
```



Operador *

- O operador * unário faz o contrário do **&**:
  - dado um ponteiro, acessa o conteúdo apontado por ele

```
int x, y, *p_x;
p_x = &x;
```

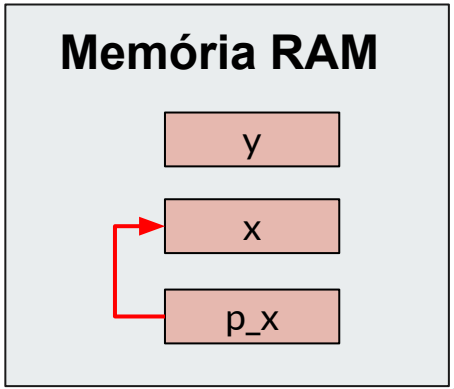


Operador *

- O operador * unário faz o contrário do **&**:
  - dado um ponteiro, acessa o conteúdo apontado por ele

Atribui 5 na posição apontada por **p_x**

```
int x, y, *p_x;

p_x = &x;
p_x = 5;
```

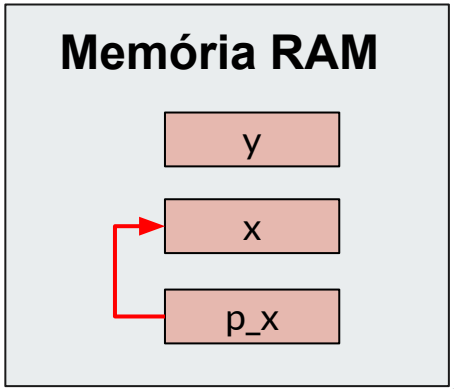


Operador *

- O operador * unário faz o contrário do **&**:
  - dado um ponteiro, acessa o conteúdo apontado por ele

Lê o conteúdo da memória apontada por **p_x** e atribui em **y**

```
int x, y, *p_x;

p_x = &x;
p_x = 5;

y = *p_x;
```



Alocação Dinâmica de Memória

- O C possui 3 **regiões** de memória, que são: a pilha, a memória estática, e o **heap**.
  - Na pilha são criadas as variáveis locais.
  - Na memória estática são criadas as variáveis globais e locais estáticas.
- As variáveis da pilha e da memória estática precisam ter tamanho conhecido antes do programa ser compilado.
- **Como obter memória de forma dinâmica ?**

Alocação Dinâmica de Memória

- Com o uso de **ponteiros** e funções do SO, pode-se obter memória à medida da necessidade.
- A **biblioteca <stdlib.h>** do C possui funções para alocação dinâmica de memória:
  - malloc
  - free
  - realoc

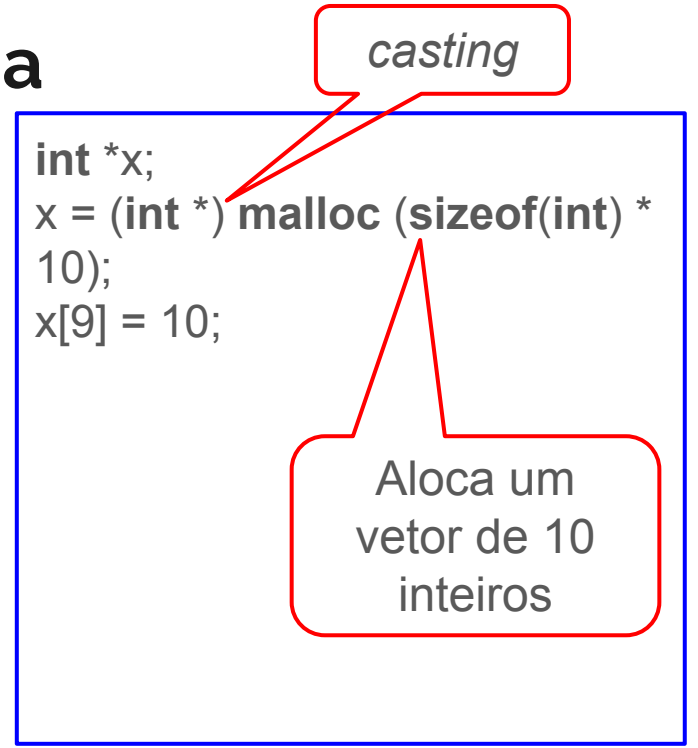
Alocação Dinâmica de Memória

- void *malloc (**int** tamanho);
  - **Aloca um bloco de memória no heap com tamanho bytes e retorna um ponteiro**. O ponteiro deve ser "casted" para o tipo de ponteiro a ser usado.

```
int *x;
x = (int *) malloc (sizeof(int) * 10);
x[9] = 10;
```

Alocação Dinâmica de Memória

- void *malloc (**int** tamanho);
  - **Aloca um bloco de memória no heap com tamanho bytes e retorna um ponteiro**. O ponteiro deve ser "casted" para o tipo de ponteiro a ser usado.



Alocação Dinâmica de Memória

- **void free(void *ponteiro);**
  - **Desaloca um bloco de memória alocado com malloc.** A memória alocada só poderá ser utilizada após ser liberada!
  - Só poderá ser desalocada uma memória que foi alocada anteriormente.

```
void main() {
        int *x, i;

        x = (int *)
        malloc(sizeof(int) * 10);

        for(i = 0; i < 10; i++){
                x[i] = rand() % 100;
        }

        free(x);
}
```

Alocação Dinâmica de Memória

- **void free(void *ponteiro);**
  - **Desaloca um bloco de memória alocado com malloc.** A memória alocada só poderá ser utilizada após ser liberada!
  - Só poderá ser desalocada uma memória que foi alocada anteriormente.

```
void main() {
        int *x, i;

        x = (int *)
        malloc(sizeof(int) * 10);

        for(i = 0; i < 10; i++){
                x[i] = rand() % 100;
        }

        free(x);
}
```

Desaloca a memória de **x**

Alocação Dinâmica de Memória

- **void * realloc(void *ponteiro, int novo_tamanho);**
  - A função realloc faz um bloco já alocado crescer ou diminuir, preservando o conteúdo já existente.

```
void main() {
        int *x, i;

        x = (int *)
        malloc(sizeof(int) * 10);

        x = (int *) realloc(x, 20 *
        sizeof(int));
        x = (int *) realloc(x, 30 *
        sizeof(int));

        free(x);
}
```

Exercícios

1) Faça um programa modularizado em C que crie um vetor dinâmico de 10 posições. Crie procedimentos para carregar e escrever o vetor dinâmico;
2) Faça um programa modularizado em C que crie um vetor dinâmico de 10 posições para armazenar os dados de pessoas (nome, cpf, idade e salário). Crie procedimentos para carregar e escrever o vetor dinâmico.

Funcionamento dos vetores em C

- Para o C, um vetor é um ponteiro para a sua primeira posição (índice 0);
  - ou seja, para um **vetor v**, **v** ou **&v[0]** direcionam para o primeiro elemento do vetor;
- Em C, **int v[ ]** e **int *v** são sinônimos;

Resumo

- Ponteiros
- Operadores
- Alocação Dinâmica de Memória
- Funcionamento dos vetores em C

Ponteiros e Alocação Dinâmica de Memória

Manassés Ribeiro
manasses.ribeiro@ifc.edu.br