

# Memória do Computador

- Quando criamos em nosso programa uma variável de um determinado tipo, o Sistema Operacional, ao executar o nosso programa reserva uma posição de memória.
- Mas ao invés de utilizarmos o endereço da variável como referência, fazemos uso do identificador (nome) que atribuímos a variável.

int × = 10:

Ponteiros	
<ul> <li>Definição:         <ul> <li>Ponteiros ou Apontadores são tipos de variáveis que armazenam endereços de memória, possivelmente, endereços de memória de outras variáveis.</li> </ul> </li> </ul>	
■ Em Linguagem <i>C</i> :  □ tipo *identificador:	
Exemplos: int *nota; char *texto:	
float *peso; double *media;	

### **Ponteiros**

#### ■ Operadores de Ponteiros:

- $\hfill\Box$  Existem dois "novos" operadores que são utilizados com ponteiros em  ${\bf C}$  :
  - & Utilizado para obter o endereço de uma variável.
  - \* Pode se comportar de duas maneiras:
    - Quando usado na declaração indica que a variável será um ponteiro.
    - Quando usado na manipulação das variáveis retorna o conteúdo do ponteiro é utilizado para obter o conteúdo do endereço apontado.

#### □ Exemplos:

int \*ptr, nota = 10; ptr = &nota; nota = \*ptr;

# Ponteiros

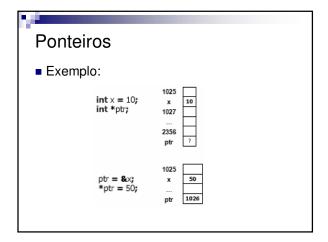
Declarações das variáveis

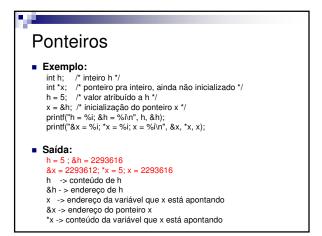
int m - m guarda um inteiro float n - n guarda um ponto flutuante

int \*x - x é um ponteiro para um inteiro

float \*y - y é um ponteiro para um ponto flutuante

 OBS.: Ter muito cuidado para só utilizar ponteiros depois de inicializá-los com um endereço conhecido. Usar sem a inicialização pode levar a travamentos do micro, etc...





# Ponteiros Observações: Se mudarmos o valor de h, então o valor de \*x mudará e vice-versa. Se temos 2 ponteiros p1 e p2 podemos realizar operações aritméticas com eles: p1 = p2, dessa forma p1 irá apontar para o mesmo endereço que p2 aponta. \*p1 = \*p2, dessa forma fazemos com que o conteúdo da variável apontada por p1 tenha o mesmo valor do conteúdo da variável que p2 aponta.

```
Ponteiros

■ Qual é a saída ?

int x = 10;

int *ptr;

ptr = &x;

*ptr = *ptr * x;

printf("%d", *ptr);

printf("%d", x);

printf("%d", ptr);
```

```
Ponteiros

• Qual é a saída ?

int x[3] = {1, 2, 3};

int *p;

p = x;

p[0] = p[1] = p[2];

printf("%d\n", p[1]);

printf("%d\n", x[1]);

printf("%d\n", p);
```

```
Ponteiros

Posições na Memória:

p++;
p--;
p = p + 15;
p1 > p2

Conteúdo do Ponteiro:

(*p)++;
(*p)--;
(*p + 15);
*p1 > *p2
```

#### Ponteiros e Vetores ■ Vetores e matrizes são ponteiros(vet[i] é igual a \*(vet + i)). Exemplo: Semelhanças: int i, $vet[] = \{1,2,3,4,5\};$ \*(x + 0) = \*x = x[0](x + 1) = x[1]int \*v; v = &vet[0];\*(x + 2) = x[2]for(i = 0; i < 5; i++) \*(x + n) = x[n]printf("%i ", v[i]); for(i = 0; i < 5; i++) printf(" %i", \*v);

}

Ponteiros para Ponteiros e Matrizes

Variável que tem um endereço para outro endereço.

Notação: int \*\*p;
Matriz NxN - 2D

É possível usar ponteiros para ponteiros para ponteiros.

Notação: int \*\*\*p;
Matriz NxNxN - 3D

Alocação Dinâmica: facilita a manipulação de vetores e matrizes.

Alocação Dinâmica

Alocar o tamanho de vetores ou matrizes dinamicamente (ou seja, no decorrer do programa)

Forma Não Dinâmica: int vet[10]

Forma Dinâmica: int \*vet

Biblioteca: <stdlib.h>

Funções:

void \* malloc(size\_t n \* size\_t size)
void \* calloc(size\_t n, size\_t size)
void \* realloc(void \* ptr, size\_t n)
void free(void \* ptr)

```
Alocação Dinâmica

Exemplo:
    main()
    {
        int *vet;
        O programa pede o tamanho do vetor, q é dado por N
        vet = (int *)malloc(N * sizeof(int));
        ou
        vet = (int*)calloc(N, sizeof(int));
        Depois o programa indica que mais posições no vetor serão criadas, e esse valor é dado por M
        vet = (int*)realloc(vet, M);
        No final a função free é usado para desalocar a memória free(vet);
    }

OBS.: O comando free NÃO APAGA O CONTEÚDO DO VETOR, mas as suas posições de memória poderão ser usadas para outras alocações.
```

Alocação Dinâmica

Com essas funções podemos criar uma matriz dinâmica, e como foi visto anteriormente uma matriz usa ponteiro para ponteiro:

Exemplo:
int \*\*mat, i, linhas, colunas;
mat = (int \*\*)malloc(linhas \* sizeof(int \*));
for(i = 0; i < linhas; i++)
mat[i] = (int \*)malloc(colunas \* sizeof(int));

Para desalocar a matriz da memória:
for(i = 0; i < linhas; i++)
free(mat[i]);
free(mat);

# Usando Ponteiros para Passagem por Referência em Funções

```
Passagem por
Referência (C++):
Passagem por
                           Passagem por
Referência (C):
                                                             void cubo(int &a)
void cubo(int a)
                            void cubo(int * a)
                                                                a = a * a * a;
   a = a * a * a;
                               (*a)=(*a)*(*a)*(*a);
                                                             int main()
int main()
                            int main()
                                                                int h = 5;
   int h = 5;
                               int h = 5:
                                                                cubo(h);
printf("%i", h);
   cubo(h);
printf("%i", h);
                               cubo(&h);
                               printf("%i", h);
                                                             Saída: 125
Saída: 5
                            Saída: 125
```

# Usados para Strings (Vetores de Caracteres)

■ Exemplo
char a[14] = "string literal";
char \*p = "string literal";

for(i = 0; i < 14; i++)
printf("%c", a[i]);

printf("\n%s", p);

## Exercício

- Escreva um programa em linguagem C que solicita ao usuário os pesos e alturas de no máximo 100 pessoas. Em seguida o seu programa deve solicitar ao usuário que escolha entre calcular o Desvio Padrão dos pesos ou das alturas. O seu programa deve imprimir o resultado do cálculo do desvio padrão. Usar ponteiros.
- P.S.: Seu programa deveria ter no máximo 4 laços (for, while, etc...)

## Dicas

- Desvio Padrão:
  - □ 1. Calcule a média dos valores.
  - 2. Calcule o quadrado da diferença de cada valor em relação a média.
  - $\square$  3. Some todos os quadrados.
  - 4. Divida o valor da soma pelo número de elementos menos um.
  - □ 5. Tire a raiz quadrada do resultado.

## Dicas

■ Média :

```
\begin{split} & \text{media} = 0; \\ & \text{for( } i = 0; i < n; i++) \\ & \text{media} += \text{vetor[i]}; \\ & \text{media} = \text{media / n;} \end{split}
```

# Dicas

■ Variância:

```
\begin{split} & \text{dif} = 0; \\ & \text{for}(\ i = 0; \ i < n; \ i++) \\ & \text{dif} \ += \ (\text{vetor}[i] - \text{media}) \ ^* \ (\text{vetor}[i] - \text{media}); \\ & \text{dif} \ = \ \text{dif} \ / \ (n-1); \end{split}
```

■ Desvio Padrão:

```
dif = sqrt(dif);
```