

# MC-102 — Aula 22

## Ponteiros e Alocação Dinâmica

Instituto de Computação – Unicamp

10 de Outubro de 2012

# Roteiro

- 1 Ponteiros e Alocação Dinâmica
- 2 Alocação Dinâmica de Matrizes
- 3 Exercício

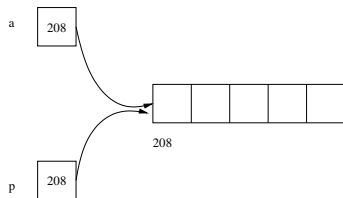
# Ponteiros e Alocação Dinâmica

- Lembre-se que uma variável vetor possui um endereço, que podemos atribuí-la para uma variável ponteiro:

```
int a[] = {1, 2, 3, 4, 5};  
int *p;  
p = a;
```

- E podemos então usar **p** como se fosse um vetor:

```
for(i = 0; i<5; i++)  
    p[i] = i*i;
```



# Ponteiros e Alocação Dinâmica

- Podemos alocar dinamicamente uma quantidade de memória contígua e associá-la com um ponteiro.
- Desta forma podemos criar programas sem saber a priori o número de dados a ser armazenado.

- ▶ Em aulas anteriores, ao trabalhar com matrizes por exemplo, assumíamos que estas tinham dimensões máximas.

```
#define MAX 100
```

```
.  
. .  
. . .
```

```
int m[MAX][MAX];
```

- ▶ Mas o que fazer se o usuário precisar trabalhar com matrizes maiores? Mudar o valor de MAX e recompilar o programa?

## Ponteiros e Alocação Dinâmica

Na biblioteca **stdlib.h** existem duas funções para fazer alocação de memória.

- **calloc** : Nesta função são passados como parâmetro o número de blocos de memória a serem alocados e o tamanho em bytes de cada bloco.

- ▶ Exemplo: alocar 100 inteiros:

```
int *p;  
p = calloc(100, sizeof(int));
```

- **malloc** : Nesta função é passado um único argumento, o número de bytes a serem alocados.

- ▶ Exemplo: alocar 100 inteiros:

```
int *p;  
p = malloc(100*sizeof(int));
```

- **Diferenças:** O *calloc* *zera* todos os bits da memória alocada enquanto que o *malloc* não. Logo se não for necessário uma inicialização (com zero) da memória alocada, o *malloc* é preferível por ser um pouco mais rápido.

# Ponteiros e Alocação Dinâmica

Juntamente com estas funções, está definida a função **free** na biblioteca **stdlib.h**.

- **free** : Esta função recebe como parâmetro um ponteiro, e libera a memória previamente alocada e apontada pelo ponteiro.

▶ Exemplo:

```
int *p;  
p = calloc(100, sizeof(int));  
....  
free(p);
```

- **Regra para uso correto de alocação dinâmica:** Toda memória alocada durante a execução de um programa deve ser desalocada (com o **free**) quando não for mais utilizada!

## Exemplo: Produto interno de 2 vetores

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    double *v1, *v2, prodInt;
    int i, n;

    printf("Digite a dimensão do vetor: ");
    scanf("%d", &n);
    v1 = malloc(n * sizeof(double));
    v2 = malloc(n * sizeof(double));

    printf("Entre com dados de v1:");
    for(i=0; i<n; i++)
        scanf("%lf", &v1[i]);
    printf("Entre com dados de v2:");
    for(i=0; i<n; i++)
        scanf("%lf", &v2[i]);

    prodInt = 0;
    for(i=0; i<n; i++)
        prodInt = prodInt + (v1[i]*v2[i]);
    printf("Produto Interno: %lf\n", prodInt);
    free(v1);
    free(v2);
}
```

# Ponteiros e Alocação Dinâmica

- Você pode fazer ponteiros distintos apontarem para uma mesma região de memória.
  - ▶ Tome cuidado para não utilizar um ponteiro se a região de memória apontada foi desalocada!

```
double *v1, *v2;

v1 = malloc(100 * sizeof(double));
v2 = v1;
free(v1);

for(i=0; i<n; i++)
    v2[i] = i*i;
```

O código acima está errado e pode causar erros durante a execução já que v2 está acessando posições de memória que não pertencem mais ao programa!



## Ponteiros e Alocação Dinâmica

O programa abaixo imprime resultados diferentes dependendo se comentamos ou não o comando **free(v1)**. Por que?

```
#include <stdio.h>
#include <stdlib.h>

int main(){
    double *v1, *v2, *v3;
    int i;

    v1 = malloc(100 * sizeof(double));
    v2 = v1;

    for(i=0; i<100; i++)
        v2[i] = i*i;
    free(v1);

    v3 = calloc(100,sizeof(double));

    for(i=0; i<100; i++)
        printf("%lf\n", v2[i]);

    free(v3);
}
```

# Alocação Dinâmica de Matrizes

- Em aplicações científicas e de engenharias, é muito comum a realização de diversas operações sobre matrizes.
- Como vimos, em situações reais o ideal é alocar memória suficiente para conter os dados a serem tratados. Não usar nem mais e nem menos!
- Como alocar vetores-multidimensionais dinamicamente?

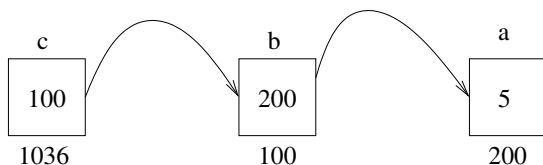
# Ponteiros para ponteiros

- Uma variável ponteiro está alocada na memória do computador como qualquer outra variável.
- Portanto podemos criar um ponteiro que contém o endereço de memória de um outro ponteiro.
- Para criar um ponteiro para ponteiro: **tipo \*\*nomePonteiro;**

```
▶ int main(){
    int a=5, *b, **c;
    b = &a;
    c = &b;
    printf("%d\n", a);
    printf("%d\n", *b);
    printf("%d\n", *(*c));
}
```

## Ponteiros para ponteiros

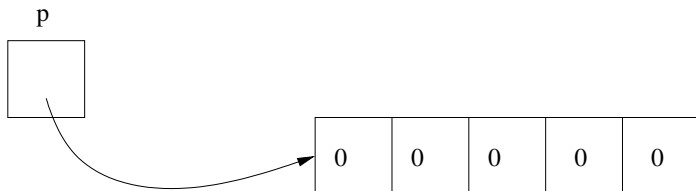
O programa imprime 5 três vezes, mostrando as três formas de acesso à variável **a**: **a**, **\*b**, **\*\*c**.



# Ponteiros para ponteiros

- Pela nossa discussão anterior sobre ponteiros, sabemos que um ponteiro pode ser usado para referenciar um vetor alocado dinamicamente.

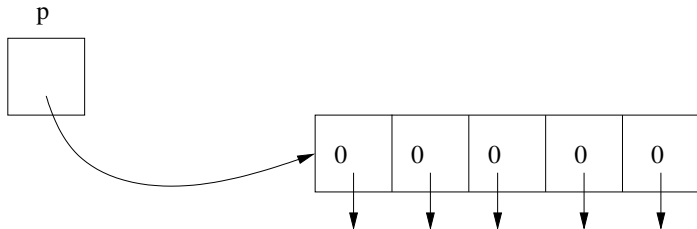
```
▶ int *p;  
  p = calloc(5, sizeof(int));
```



# Ponteiros para ponteiros

- A mesma coisa acontece com um ponteiro para ponteiro, só que neste caso o vetor alocado é de ponteiros.

```
▶ int **p;  
  p = calloc(5, sizeof(int *));
```

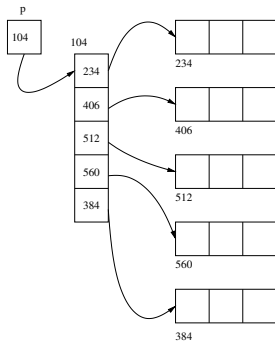


- ▶ Note que cada posição do vetor acima é do tipo **int \***, ou seja, um ponteiro para inteiro!

# Ponteiros para ponteiros

- Como cada posição do vetor é um ponteiro para inteiro, podemos associar cada posição dinamicamente com um vetor de inteiros!

```
▶ int **p;  
   int i;  
   p = calloc(5, sizeof(int *));  
  
   for(i=0; i<5; i++){  
       p[i] = calloc(3, sizeof(int));  
   }
```



# Alocação Dinâmica de Matrizes

Esta é a forma de se criar matrizes dinamicamente:

- Crie um ponteiro para ponteiro.
- Associe um vetor de ponteiros dinamicamente com este ponteiro de ponteiro. O tamanho deste vetor é o número de linhas da matriz.
- Cada posição do vetor será associado com um outro vetor do tipo a ser armazenado. Cada um destes vetores é uma linha da matriz (portanto possui tamanho igual ao número de colunas).

OBS: No final você deve desalocar toda a memória alocada!!



# Alocação Dinâmica de Matrizes

```
int main(){
    int **p, i, j;

    p = calloc(5, sizeof(int *));
    for(i=0; i<5; i++){
        p[i] = calloc(3, sizeof(int));
    }
    printf("Digite os valores da matriz\n");
    for(i = 0; i<5; i++){
        for(j=0; j<3; j++){
            scanf("%d", &p[i][j]);
        }
    }

    printf("Matriz lida\n");
    for(i = 0; i<5; i++){
        for(j=0; j<3; j++){
            printf("%d, ", p[i][j]);
        }
        printf("\n");
    }
    //desalocando memória usada
    for(i=0; i<5; i++){
        free(p[i]);
    }
    free(p);
}
```

## Exercício

Crie um programa que multiplica duas matrizes quadradas do tipo **double** lidas do teclado. Seu programa deve ler a dimensão  $n$  da matriz, em seguida alocar dinamicamente duas matrizes  $n \times n$ . Depois ler os dados das duas matrizes e imprimir a matriz resultante da multiplicação destas.