

# MC-102 — Aula 21

## Ponteiros III

Alexandre M. Ferreira

IC – Unicamp

19/05/2017

# Roteiro

- 1 Exemplo de Ponteiros e Alocação Dinâmica
- 2 Exercício
- 3 Informações Extras: Ponteiros para Ponteiros e Alocação Dinâmica de Matrizes

# Exemplo de Ponteiros e Alocação Dinâmica

Vamos criar uma aplicação que cria um vetor dinâmico com funções para implementar as seguintes operações:

- Inclusão de um elemento no final do vetor.
- Exclusão da primeira ocorrência de um elemento no vetor.
- Impressão do vetor.

# Exemplo de Ponteiros e Alocação Dinâmica

- O tamanho do vetor deve se ajustar automaticamente: se elementos são inseridos devemos “aumentar” o tamanho do vetor para inclusão de novos elementos, e se elementos forem removidos devemos “diminuir” o tamanho vetor.
- Temos duas variáveis associadas ao vetor:
  - ▶ **size**: denota quantos elementos estão armazenados no vetor.
  - ▶ **maxSize**: denota o tamanho alocado do vetor.

# Exemplo de Ponteiros e Alocação Dinâmica

Temos as seguintes regras para ajuste do tamanho alocado do vetor:

- O vetor deve ter tamanho alocado no mínimo igual a 4.
- Se o vetor ficar cheio, então devemos alocar um novo vetor com o dobro do tamanho atual.
- Se o número de elementos armazenados no vetor for menor do que  $1/4$  do tamanho alocado do vetor, então devemos alocar um novo vetor com metade do tamanho atual.

# Exemplo de Ponteiros e Alocação Dinâmica

Implementaremos as seguintes funções:

- `int * initVet(int *size, int *maxSize);`

Aloca um vetor inicial de tamanho 4, setando **size** com valor 0, **maxSize** com valor 4, e devolve o endereço do vetor alocado.

- `void printVet(int *v, int size, int maxSize);`

Imprime o conteúdo e tamanhos associados ao vetor **v**.

- `int * addVet(int *v, int *size, int *maxSize, int e);`

Adiciona o elemento **e** no final do vetor **v**. Caso não haja espaço, um novo vetor com o dobro do tamanho deve ser alocado. A função sempre retorna o endereço do vetor, sendo um novo alocado ou não. Além disso os valores de **size** e **maxSize** devem ser atualizados.

# Exemplo de Ponteiros e Alocação Dinâmica

Implementaremos as seguintes funções:

- `int find(int *v, int size, int e);`

Determina se o elemento **e** está presente ou não no vetor **v**. Caso esteja presente, retorna a posição da primeira ocorrência de **e** em **v**. Caso não esteja presente, retorna -1.

- `int * removeVet(int *v, int *size, int *maxSize, int e);`

Remove a primeira ocorrência do elemento **e** do vetor **v** caso este esteja presente. O valor de **size** deve ser decrementado de 1. Caso o número de elementos armazenados seja menor do que  $\frac{1}{4} \text{maxSize}$ , então um novo vetor de tamanho  $\frac{1}{2} \text{maxSize}$  deve ser alocado no lugar de **v**. A função sempre retorna o endereço inicial do vetor alocado, sendo um novo vetor alocado ou não.

# Exemplo de Ponteiros e Alocação Dinâmica

Funções **initVet** e **printVet**:

```
// Cria vetor com maxSize inicial 4.
// Devolve o endereço do vetor criado.
int * initVet(int *size, int *maxSize){
    int *v = malloc(4* sizeof(int));
    *size = 0;
    *maxSize = 4;
    return v;
}

// Imprime vetor.
void printVet(int *v, int size, int maxSize){
    int i;
    printf("Vetor de tamanho %d (max. alocado %d)\n", size, maxSize);
    for(i=0; i<size; i++){
        printf("%d, ", v[i]);
    }
    printf("\n");
}
```



# Exemplo de Ponteiros e Alocação Dinâmica

A função **addVet**:

```
int * addVet(int *v, int *size , int *maxSize , int e){
    if(*size < *maxSize){ //Tem espaço para o elemento.
        v[*size] = e;
        (*size)++;
        return v;
    }else{ //Precisamos alocar um espaço maior.
        ...
    }
}
```

# Exemplo de Ponteiros e Alocação Dinâmica

A função **addVet**:

```
int * addVet(int *v, int *size, int *maxSize, int e){
    if(*size < *maxSize){ //Tem espaço para o elemento.
        ...
    }else{ //Precisamos alocar um espaço maior.
        int *vaux = malloc(2*(*maxSize)*(sizeof(int)));
        int i;
        for(i=0; i < *size; i++) //Salva dados de v em vaux.
            vaux[i] = v[i];
        vaux[*size] = e; //Adiciona elemento no fim.

        (*size)++;
        *maxSize = 2*(*maxSize); //Atualiza dados de tamanho.

        free(v); //Libera memória não mais necessária.
        return vaux;
    }
}
```

# Exemplo de Ponteiros e Alocação Dinâmica

A função **addVet** completa:

```
int * addVet(int *v, int *size, int *maxSize, int e){
    if(*size < *maxSize){ //Tem espaço para o elemento.
        v[*size] = e;
        (*size)++;
        return v;
    }else{ //Precisamos alocar um espaço maior.
        int *vaux = malloc(2*(*maxSize)*(sizeof(int)));
        int i;
        for(i=0; i < *size; i++) //Salva dados de v em vaux.
            vaux[i] = v[i];
        vaux[*size] = e; //Adiciona elemento no fim.

        (*size)++;
        *maxSize = 2*(*maxSize); //Atualiza dados de tamanho.

        free(v); //Libera memória não mais necessária.
        return vaux;
    }
}
```

# Exemplo de Ponteiros e Alocação Dinâmica

A função **find** é:

```
//Retorna posição da primeira ocorrência de e
//ou -1 caso e não seja encontrado
int find(int *v, int size, int e){
    int i;
    for(i=0; i<size; i++)
        if(v[i] == e)
            return i;

    return -1;
}
```

# Exemplo de Ponteiros e Alocação Dinâmica

A função **removeVet** é:

```
int * removeVet(int *v, int *size , int *maxSize, int e){
    int i;
    i = find(v, *size , e);
    if(i != -1){ //Se elemento está em v.
        //Copia dados a partir da posição i+1 uma posição para trás.
        for(; i < (*size)-1 ; i++){
            v[i] = v[i+1];
        }
        (*size)--;

        //Se tamanho do vetor for > 4 e estiver menos de 1/4 ocupado
        //devemos diminuir tamanho do vetor pela metade.
        if( *size < (0.25 * (*maxSize)) && *maxSize > 4 ){
            ...
            Exercício.
        }
    }
    return v;
}
```

# Exemplo de Ponteiros e Alocação Dinâmica

Com as funções implementadas podemos executar o exemplo:

```
int main(){
    int *vet, size, maxSize;
    int i;

    vet = initVet(&size, &maxSize);

    for(i=0; i<20; i++){
        vet = addVet(vet, &size, &maxSize, i);
    }
    printVet(vet, size, maxSize);

    vet = removeVet(vet, &size, &maxSize, 14);
    printVet(vet, size, maxSize);

    for(i=5; i<15; i++){
        vet = removeVet(vet, &size, &maxSize, i);
    }
    printVet(vet, size, maxSize);

    for(i=0; i<20; i++){
        vet = removeVet(vet, &size, &maxSize, i);
    }
    printVet(vet, size, maxSize);

    free(vet);
}
```

## Exercício

Implemente a função de remoção de um elemento do vetor de tal forma que se o número de elementos armazenados (**size**) for menor do que  $\frac{1}{4}\mathbf{maxSize}$ , então o tamanho do vetor alocado deve ter tamanho igual a metade do anterior. A função deve devolver o endereço do início do vetor, sendo um novo alocado ou não. Além disso a função deve atualizar os valores **size** e **maxSize** caso necessário.

## Informações Extras: Alocação Dinâmica de Matrizes

- Em aplicações científicas e de engenharias, é muito comum a realização de diversas operações sobre matrizes.
- 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?



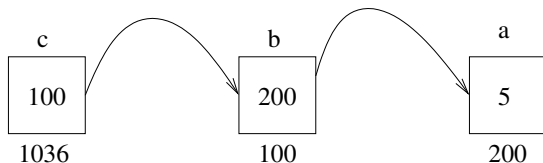
## Informações Extras: Ponteiros para ponteiros

- 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));
}
```

## Informações Extras: Ponteiros para ponteiros

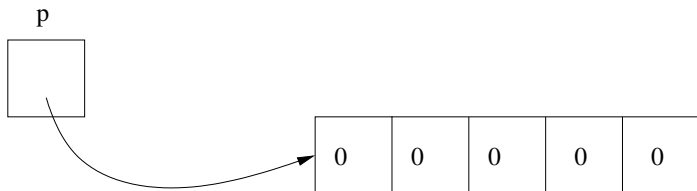
O programa imprime 5 três vezes, mostrando as três formas de acesso à variável **a**: **a**, **\*b**, **\*\*c**.



## Informações Extras: Ponteiros para ponteiros

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

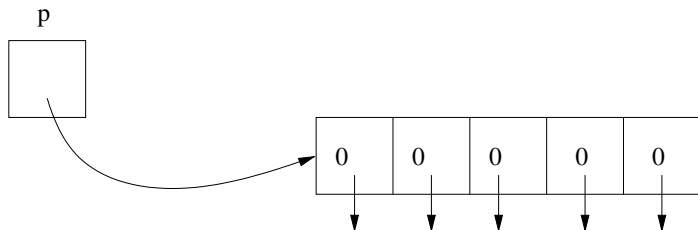
```
▶      int *p;  
      p = calloc(5, sizeof(int));
```



## Informações Extras: 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 *));
```

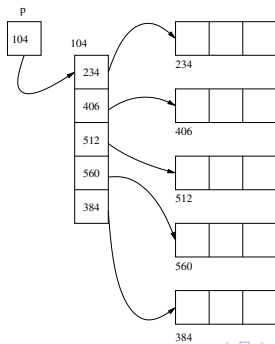


- ▶ Note que cada posição do vetor acima é do tipo `int *`, ou seja, um ponteiro para inteiro!

## Informações Extras: 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));  
}
```



## Informações Extras: 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 associada 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!!

## Informações Extras: 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));
    } //Alocou matriz 5x3

    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);
}
```



# Informações Extras: Alocação Dinâmica de Matrizes

Outro exemplo:

```
int main(){
    int **mat;          int i, j, n, m;

    printf("Numero_de_linhas:");
    scanf("%d", &n);
    printf("Numero_de_colunas:");
    scanf("%d", &m);

    mat = malloc(n * sizeof(int *));
    for(i=0; i<n; i++){
        mat[i] = malloc(m * sizeof(int));
    }

    for(i=0; i<n; i++){
        for(j=0; j<m; j++){
            mat[i][j] = i*j;
        }
    }
    .
    .
    .
}
```



# Informações Extras: Alocação Dinâmica de Matrizes

Outro exemplo:

```
.  
. .  
. . .  
for(i=0; i<n; i++){  
    for(j=0; j<m; j++){  
        mat[i][j] = i*j;  
    }  
}  
  
for(i=0; i<n; i++){  
    for(j=0; j<m; j++){  
        printf("%d, ", mat[i][j]);  
    }  
    printf("\n");  
}  
  
for(i=0; i<n; i++){  
    free(mat[i]);  
}  
free(mat);  
}
```

## Informações Extras: Alocação Dinâmica de Matrizes

Mas a forma mais eficiente de criar matrizes é:

- Para uma matriz de dimensões  $n \times m$ , crie um vetor unidimensional dinamicamente deste tamanho.
- Use linearização de índices para trabalhar com o vetor como se fosse uma matriz.
- Desta forma tem-se um melhor aproveitamento da cache pois a matriz inteira está sequencialmente em memória.

No final você deve desalocar toda a memória alocada!!