# Alocação Dinâmica

UFOP
Universidade Federal de Ouro Preto

CSI030-PROGRAMAÇÃO DE COMPUTADORES I

DECSI

# Links para os conteúdos

- Videoaula: **https://youtu.be/HcubsGgTdAY**

- Códigos da videoaula e outros exemplos: **https://github.com/fboliveira/CSI030-Prog-Comp-01/tree/master/Codes/10-alocacao**

# Alocação dinâmica de memória

- Pode-se alocar dinamicamente (quando o programa está em execução) uma quantidade de memória contígua e associá-la a um ponteiro.

- Isto permite criar programas sem saber, em tempo de codificação, qual o tamanho dos dados a serem armazenados (vetores, matrizes, etc).

- Assim, não é necessário armazenar mais memória do que de fato se deseja usar.

# Alocação dinâmica de memória

- A biblioteca **stdlib.h** possui duas funções para fazer alocação de memória:
  - **void* calloc(int blocos, int tamanho)**: recebe o número de blocos de memória a serem alocados e o tamanho de cada bloco. Os bits da memória alocada são zerados.
  - **void* malloc(int qtde_bytes)**: recebe a quantidade de bytes a serem alocados na memória. Não zera os bits alocados.
- Se não for necessário zerar os bits da memória alocada, a função **malloc** é preferível por ser mais rápida.

# Alocação dinâmica de memória

- A biblioteca **stdlib.h** possui a seguinte função para liberar memória:

  - **free(void* ponteiro)**: recebe um ponteiro com o endereço da memória a ser desalocada. Como ela pode receber um ponteiro de qualquer tipo, o tipo do parâmetro deve ser **void** *.

- Toda memória alocada com **calloc**() ou **malloc**() deve ser liberada com **free**() após seu uso!

# Exemplos de alocação e liberação de memória

- O código abaixo aloca **100** inteiros para o ponteiro **p** e outros **100** inteiros para o ponteiro **q**. Equivale a declararmos 2 vetores de 100 posições! A memória é liberada no final.

```
int *p=NULL, *q=NULL;
p = (int*) calloc(5, sizeof(int));
q = (int*) malloc(5 * sizeof(int));

for (i = 0; i < 5; i++){
    p[ i ] = 1;
    q[ i ] = 2;
}

free(p);
free (q);
```

# Ponteiros e tipos na memória

- Como o computador sabe onde começa e onde termina a região de memória para **p[3]**, por exemplo?

  - O compilador sabe que **p** é um ponteiro para inteiros.
  - Ele também sabe que **p** aponta para um endereço de memória em que são armazenados inteiros.
  - Para encontrar o quarto inteiro (**p[3]**), o compilador gera código para que o ponteiro **p[3]** aponte para 3 blocos de memória (cada um do tamanho de um inteiro) depois do endereço de **p**.

# Vetores Unidimensionais Dinâmicos na Memória

```
int aI [3];
char aC[4];

int *pi;
char *pc;

pi = (int*)malloc(3*sizeof(int));
pc = (char*)malloc(4*sizeof(char));
```

| Endereço | | | | |
|---|---|---|---|---|
| 700 | pi[0] | pi[1] | pi[2] | |
| 600 | pc[0] | pc[1] | pc[2] | pc[3] |
| 500 | | | | |
| 400 | | | | |
| 300 | *pc= 600 | | *pi= 700 | |
| 200 | aC[0] | aC[1] | aC[2] | aC[3] |
| 100 | aI[0] | aI[1] | aI[2] | |

# Ponteiros para ponteiros

- Como visto, uma variável ponteiro está alocada na memória como qualquer outra variável.

- Pode-se, então, criar um segundo ponteiro que possua o endereço de memória do primeiro ponteiro.
- Isto se chama ponteiro para ponteiro e pode ser declarado assim:
  - **tipo **variavel;**
- Exemplos:
  - **int ** ppi;**
  - **char **ppc;**

# Exemplo de ponteiro para ponteiro

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

| variável | a | b | c |
|---|---|---|---|
| conteúdo | 5 | 100 | 200 |
| endereço | 100 | 200 | 300 |

Saída:
5
5
5

Resoluções:
a = 5
* b = *(100)
** c = *(*(200)) = *(&100)

# Alocação Dinâmica de Matrizes

- Esta é a forma de se criar matrizes dinamicamente:
  - Crie um ponteiro para ponteiro.
  - int **a
  - Associe um vetor de ponteiros dinamicamente com este ponteiro de ponteiro. O tamanho deste vetor é o número de linhas da matriz.
  - a = (int**) malloc(n * sizeof(int *));

# Alocação Dinâmica de Matrizes

- Esta é a forma de se criar matrizes dinamicamente:
  - 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).

```
for (i = 0; i < n; i++)
a[ i ] = (int*) malloc(m * sizeof(int));
```

  - Deve-se liberar toda a memória alocada após o uso!

# Vetores Multidimensionais Dinâmicos na Memória

```
int **a, n=2, m=3, i;

a = (int**) malloc(n * sizeof(int *));

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

for(i = 0; i < n; i++)
     free(a[ i ]);

free(a);
```

| Endereço | | | |
|---|---|---|---|
| 210 | a[1][0] | a[1][1] | a[1][2] |
| 190 | a[0][0] | a[0][1] | a[0][2] |
| 160 | | | |
| 130 | | a[0] = 190 | a[1] = 210 |
| 100 | a = 140 | | |

UFOP
Universidade Federal de Ouro Preto

# Exercícios

DECSI

# Exercício 1

- Crie um vetor dinâmico de tamanho informado pelo usuário.
- Solicite os valores do vetor ao usuário.
- Em seguida, imprima o conteúdo do vetor.

# Exercício 2

- Converta o código do exercício anterior de modo que sejam criados:
  - Procedimento para preencher o vetor, solicitando os valores ao usuário.
  - Procedimento para imprimir o vetor.
  - Função para efetuar a alocação dinâmica do vetor, retornando o endereço com a alocação.

# Exercício 3

- Crie uma matriz dinâmica de tamanho L linhas e C colunas.
- Solicite o preenchimento do seu conteúdo pelo usuário.
- Em seguida, imprima a matriz.
- Dica: utilize L=2 e C=3 para testar o seu programa!

# Exercício 4

- Converta o código do exercício anterior de modo que sejam criados:
  - Procedimento para preencher a matriz, solicitando os valores ao usuário.
  - Procedimento para imprimir a matriz.
  - Função para efetuar a alocação dinâmica da matriz, retornando o endereço com a alocação.

# Referências


- DEITEL, P; DEITEL, H. C How to Program. 6a Ed. Pearson, 2010.

- Material de aula do Prof. Ricardo Anido, da UNICAMP: http://www.ic.unicamp.br/~ranido/mc102/

- Material de aula da Profa. Virgínia F. Mota: https://sites.google.com/site/virginiaferm/home/disciplinas