MC202 - Estruturas de Dados

Alexandre Xavier Falcão

Instituto de Computação - UNICAMP

afalcao@ic.unicamp.br

Vetores, matrizes e alocação de memória

- Vetores e matrizes são variáveis compostas homogêneas, pois consistem de múltiplas variáveis simples do mesmo tipo.

Vetores, matrizes e alocação de memória

- Vetores e matrizes são variáveis compostas homogêneas, pois consistem de múltiplas variáveis simples do mesmo tipo.

- Elas podem ocupar espaço na pilha (*stack*), se declaradas assim

  int $x[10]$; vetor de variáveis inteiras $x[0], x[1], \ldots, x[9]$.

Vetores, matrizes e alocação de memória

- O ideal, porém, é aproveitar o espaço muito maior de memória do *heap*. Neste caso, a declaração

  int *x; ponteiro para variável local inteira.

  ocupa apenas 4 bytes na pilha.

Vetores, matrizes e alocação de memória

- O ideal, porém, é aproveitar o espaço muito maior de memória do *heap*. Neste caso, a declaração

  int *x; ponteiro para variável local inteira.

  ocupa apenas 4 bytes na pilha.

- A alocação de memória dinâmica

  $x =$ (int *)calloc(10,sizeof(int));

  aloca 10 variáveis inteiras $x[0], x[1], \ldots, x[9]$ no *heap*, retornando o endereço de $x[0]$ para a variável x.

Agenda

- Vetores.
- Matrizes.
- Ponteiros de ponteiros de ponteiros ....

Vetores

Podemos declarar vetores de qualquer tipo de variável simples e, como veremos mais adiante, vetores de variáveis compostas.

```
float *CriaVetorFloat(int n)
{
  float *v = (float *)calloc(n,sizeof(float));
  return(v);
}
```

```
void DestroiVetorFloat(float **v)
{
  if ((*v) != NULL){
    free(*v);
    *v = NULL;
  }
}
```

Vetores

Podemos declarar vetores de qualquer tipo de variável simples e, como veremos mais adiante, vetores de variáveis compostas.

```
float *CriaVetorFloat(int n)
{
  float *v = (float *)calloc(n,sizeof(float));
  return(v);
}
```

```
void DestroiVetorFloat(float **v)
{
  if ((*v) != NULL){
    free(*v);
    *v = NULL;
  }
}
```

float *v = CriaVetorFloat(n);
DestroiVetorFloat(&v);

criam e destroem o vetor no *heap*. Sendo v um ponteiro para $v[0]$, seu endereço &v deve ser copiado em uma variável do tipo ponteiro para ponteio, cuja declaração é **v na função DestroiVetorFloat.

Memória durante a execução do código

Para entender o fluxo de informações ao alocar memória para vetores e manipular seus elementos, vamos usar `www.pythontutor.com` e o código vetores.c.

Matrizes

Podemos declarar matrizes de qualquer tipo de variável simples ou composta.

```
int **CriaMatrizInt(int nlin, int ncol)
{
  int **m = (int **)calloc(nlin,sizeof(int *));

  for (int l=0; l < nlin; l++)
    m[l] = (int *)calloc(ncol,sizeof(int));
  return(m);
}
```

```
void DestroiMatrizInt(int ***m, int nlin)
{
  if ((*m) != NULL){
    for (int l=0; l < nlin; l++)
      free((*m)[l]);
    free(*m);
    *m = NULL;
  }
}
```

Matrizes

Podemos declarar matrizes de qualquer tipo de variável simples ou composta.

```
int **CriaMatrizInt(int nlin, int ncol)
{
  int **m = (int **)calloc(nlin,sizeof(int *));

  for (int l=0; l < nlin; l++)
    m[l] = (int *)calloc(ncol,sizeof(int));
  return(m);
}
```

```
void DestroiMatrizInt(int ***m, int nlin)
{
  if ((*m) != NULL){
    for (int l=0; l < nlin; l++)
      free((*m)[l]);
    free(*m);
    *m = NULL;
  }
}
```

float $**m$ = CriaMatrizInt(*nlin*, *lcol*);
DestroiMatrizInt(&*m*, *nlin*);

criam e destroem a matriz no *heap*. Sendo *m* um ponteiro duplo para *m*[0][0], seu endereço &*m* deve ser copiado em uma variável do tipo ponteiro triplo, cuja declaração é $***m$ na função DestroiMatrizInt.

Memória durante a execução do código

Para entender o fluxo de informações ao alocar memória para matriz e manipular seus elementos, vamos usar `www.pythontutor.com` e o código matrizes.c.