Universidade Federal Fluminense

Disciplina: Programação de Computadores II

Professor: Alessandro Copetti

Maio/2017

Estruturas (struct)

Como armazenar uma sequência de estruturas em memória, a qual corresponderá a uma tabela ou a um cadastro?

Tendo:

```
#define MAX_ALUNOS 50

typedef struct aluno {
    char nome[100];
    int idade;
    // ...
} Aluno;
```

Existem três formas/opções de armazenamento em memória:

1) Vetor de estruturas

Declaração:

```
Aluno vetAluno[MAX_ALUNOS];
```

2) Vetor de Ponteiros - com tamanho pré-determinado

- Declaração:

```
Aluno *vetAluno[MAX_ALUNOS];
```

- Procedimento para alocar:

```
for (i=0; i< MAX_ALUNOS; i++)
    vetAluno[i] = (Aluno *) malloc(sizeof(Aluno));
```

3) Vetor de Ponteiros - com tamanho determinado em tempo de execução

- Declaração:

```
Aluno **vetAluno;
```

- Procedimento para alocar:

```
vetAluno = (Aluno **) malloc(MAX_ALUNOS * sizeof(Aluno *);
for (i=0; i< MAX_ALUNOS; i++)
    vetAluno[i] = (Aluno *) malloc(sizeof(Aluno));
```

Comparação entre as três opções:

A primeira comparação diz respeito ao uso da memória. Vamos supor que a estrutura Aluno ocupe 500 bytes e pretendemos ter no máximo 50 alunos. Na primeira opção, **vetor de estruturas**, o espaço total ocupado na declaração é de 50alunos * 500bytes = 25.000 bytes. Após isso, o vetor já pode ser preenchido. Na segunda opção, **vetor de ponteiros com tamanho pré-determinado**, a declaração ocupa 50alunos * 4bytes = 200 bytes, pois cada endereço contido dentro de cada elemento do vetor ocupa 4 bytes. Se todos os 50 alunos forem criados, terá uma adição de 25.000 bytes. No entanto, essa criação pode ser feita sob demanda, e o desperdício se restringirá apenas nos elementos do vetor que não estão apontando para estruturas. Na terceira opção, **Vetor de Ponteiros - com tamanho determinado em tempo de execução**, a declaração ocupa somente 4 bytes. Após fazer a alocação do vetor e dos 50 alunos, totalizará 25.200 bytes. No entanto, o tamanho do vetor pode ser determinado em tempo de execução e não haverá desperdício de memória.

A segunda comparação é no tocante a programação. Na primeira opção, após ser declarado o vetor de estruturas, o uso do vetor é no formato: *vetAluno[i].campo*. Nas opções dois e três, após a inicialização, o uso é no formato: *vetAluno[i]->campo*.

Quando chamamos uma função passando o vetor, para todas as três opções passamos somente o nome do vetor e o tamanho: *funcao(vetAluno, MAX_ALUNOS);* No recebimento dos parâmetros na função, na primeira opção, recebemos com *Aluno *ptr* e acessamos os elementos do vetor de estruturas com *ptr[i].campo*. Já nas opções dois e três, o recebimento é feito com *Aluno **ptr*, e o acesso é com *ptr[i]->campo*.

O código abaixo ilustra as três opções.

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

#define MAX_ALUNOS 3

typedef struct aluno {
   char nome[100];
   int idade;
} Aluno;


void mostraVetorPtrPtr(Aluno **ptr, int n) {
   int i;
   for (i=0; i<n; i++)
      printf("Elemento %d: nome=%s, idade=%d\n", i, ptr[i]->nome, ptr[i]->idade);
}

void mostraVetor(Aluno *ptr, int n) {
   int i;
   for (i=0; i<n; i++)
       printf("Elemento %d: nome=%s, idade=%d\n", i, ptr[i].nome, ptr[i].idade);
}

void inicializa1(Aluno *ptr, int n) {
    int i=0;
    for (;i < n; i++) {
```

```
        strcpy(ptr[i].nome, "");
        ptr[i].idade=0;
    }
}

void inicializa2(Aluno **aluno, int n) {
    int i=0;
    for (;i < n; i++) {
        aluno[i] = (Aluno *) malloc(sizeof(Aluno));
        strcpy(aluno[i]->nome, "");
        aluno[i]->idade=0;
    }
}

Aluno **inicializa3(int n) {
    int i=0;
    Aluno **aluno = (Aluno **) malloc(n*sizeof(Aluno *));
    for (;i < n; i++) {
        aluno[i] = (Aluno *) malloc(sizeof(Aluno));
        strcpy(aluno[i]->nome, "");
        aluno[i]->idade=0;
    }
    return aluno;
}

void main (void) {
   Aluno vetAluno1[MAX_ALUNOS];
   inicializa1(vetAluno1, MAX_ALUNOS);
   mostraVetor(vetAluno1, MAX_ALUNOS);

   Aluno *vetAluno2[MAX_ALUNOS];
   inicializa2(vetAluno2, MAX_ALUNOS);
   mostraVetorPtrPtr(vetAluno2, MAX_ALUNOS);

   Aluno **vetAluno3;
   vetAluno3 = inicializa3(MAX_ALUNOS);
   mostraVetorPtrPtr(vetAluno3, MAX_ALUNOS);
}
```