Aula 9 – Estruturas de Dados

1. Introdução

As variáveis estudadas até o momento podem armazenar somente dados simples. Muitas vezes precisamos armazenar dados compostos, ou seja, várias variáveis que se referem ao mesmo conceito do mundo real. Exemplos:

- Um programa que processa matrículas de alunos em classes. Alunos possuem nome, número de matrícula e curso, classes possuem nome, professor designado, pré-requisitos, carga horária, etc.
- Um programa que gerencia uma biblioteca. Livros possuem título, gênero, autor(es), ISBN, etc.

2. Estruturas de Dados

- Relembrar o exercício em que um professor estabelece o peso das 3 provas parciais e depois calcula a média ponderada de cada aluno. Se formos ler e armazenar nome e notas parciais de cada aluno para depois calcular suas notas finais, poderíamos fazer:

```
char nomes[50][30];
float notas1[50], notas2[50], notas3[50];
```

- Seria melhor ter um "vetor de alunos" ao invés de quatro vetores separados. Podemos fazer isso com estruturas de dados. Se usa a palavra-chave `struct` para definir uma nova estrutura:

```
struct Aluno {
     char nome[30];
     float nota1, nota2, nota3;
} aluno;

main() {
     struct Aluno alunos[50];

     // ...
}
```

- A declaração de uma variável do tipo `struct Aluno` é opcional. Após declarada uma variável daquele tipo, podemos acessar os seus elementos internos usando o operador de seleção: `.`

```
strcpy(aluno.nome, "Vitor Souza");
aluno.nota1 = 10;
printf("%s: %.1f\n", aluno.nome, aluno.nota1);
```

- O mesmo poderia ter sido feito a um dos alunos do vetor de alunos. Por exemplo: `alunos[0].nota1 = 10;`
- Um tipo composto pode ter como membros variáveis de outros tipos compostos. Neste caso, o acesso a um determinado membro pode ser feito usando vários operadores de seleção em sequência:

```
struct Endereco {
     char rua[50], cidade[30], estado[3];
};

struct Aluno {
```

```
    /* ... */
    struct Endereco endereco;
} aluno;

main() {
    /* ... */

    strcpy(aluno.endereco.rua, "Av. Fernando Ferrari, 514 Ufes CT7");
    strcpy(aluno.endereco.cidade, "Vitória");
    strcpy(aluno.endereco.estado, "ES");
    printf("%s\n%s, %s\n", aluno.endereco.rua, aluno.endereco.cidade,
aluno.endereco.estado);
}
```

- Uma atribuição entre tipos compostos fará com que todos os membros de um tipo sejam atribuídos a todos os membros do outro. A cópia é recursiva, caso um dos membros seja também um tipo composto:

```
alunos[0] = aluno;
printf("%s, %.1f, %s\n", alunos[0].nome, alunos[0].nota1,
alunos[0].endereco.cidade);
```

- Devemos somente ter cuidado com cópia de ponteiros: se for copiado o endereço de memória de um membro de uma estrutura a outras, ambos apontam para o mesmo espaço de memória e, portanto, modificação em um reflete em um outro (não ocorre com o vetor de nome acima);
- Dado que atribuições são possíveis, podemos passar estruturas também como parâmetro para funções. No entanto, lembre-se que a passagem é por valor e, portanto, mudanças no parâmetro dentro da função não refletem na variável fora da mesma (exceto no caso de ponteiros, como explicado acima);

2.1. Ponteiros para estruturas

- O uso de ponteiros para estruturas traz duas vantagens:
    - Maior eficiência: menos memória gasta e menos tempo copiando membros de uma estrutura para outra nas atribuições;
    - Funcionamento mais intuitivo: quando um ponteiro para estrutura é passado para uma função, alterações na estrutura se refletem fora da função.
- Exemplo:

```
struct Aluno {
    char nome[30];
    float nota1, nota2, nota3;
};

void preencherAluno(struct Aluno *a) {
    printf("Digite nome e 3 notas:\n");
    gets(a->nome);
    scanf("%f %f %f", &a->nota1, &a->nota2, &a->nota3);
}

main() {
```

```
    struct Aluno *aluno;
    aluno = (struct Aluno *)malloc(sizeof(struct Aluno));
    preencherAluno(aluno);
    printf("%s: %.1f, %.1f, %.1f\n", aluno->nome, aluno->nota1,
aluno->nota2, aluno->nota3);
}
```

- Note um novo operador `->` ao invés de `.`: como agora estamos trabalhando com ponteiros, precisaríamos derreferenciar o ponteiro antes de acessar o membro da estrutura: `(*a).nome`. O operador `->` é um atalho para esta operação;

2.2. Definição de nomes para tipos compostos: typedef

- Declarar sempre uma variável do tipo ponteiro para estrutura utilizando os operadores `struct` e `*` pode se tornar tedioso. Ao invés disso, podemos definir um nome para o novo tipo:

```
struct TAluno {
    char nome[30];
    float nota1, nota2, nota3;
};

typedef struct TAluno *Aluno;
```

- Ou, resumidamente:

```
typedef struct TAluno {
    char nome[30];
    float nota1, nota2, nota3;
} *Aluno;
```

- Em seguida, podemos substituir `struct Aluno *` por `Aluno`. O termo `struct TAluno` só vai precisar aparecer no comando `sizeof`.

2.3. Listas encadeadas

- Imagine que um programa deva ler um número indefinido de dados (ex.: sobre alunos) e armazená-los em memória para, no final, efetuar algum cálculo. Como não sabemos de quantos alunos serão informados dados, não podemos criar um vetor de alunos;
- Precisamos, então, montar uma lista encadeada: cada estrutura de dados que representa um aluno terá um ponteiro para uma outra estrutura idêntica. Assim, encadeamos o 1º aluno no 2º, o 2º no 3º e assim por diante. O último possui o ponteiro nulo:

```
typedef struct TAluno {
    char nome[30];
    float nota1, nota2, nota3;
    struct TAluno *proximo;
} *Aluno;
```

3. Outras estruturas de dados

- Não serão vistas em detalhes neste curso;
- Union: permite que múltiplas variáveis ocupem uma mesma posição de memória;
- Enum: permite enumerar os valores possíveis de um tipo de dado.

Universidade Federal do Espírito Santo
Centro Tecnológico
Departamento de Informática

Exercícios – Estruturas de Dados

1) Refaça o exercício da média dos alunos citado no início da aula utilizando estruturas de dados compostas, sendo que não é possível saber de quantos alunos serão informados dados.

2) Reescreva o exercício 7 da lista 1 utilizando estruturas de dados:

"O Sr. Bondoso, dono da empresa SerFeliz, quer dar um abono de Natal no valor de 10% do salário para os funcionários que ganham até 5 salários mínimos. Faça um programa para ler o número de funcionários, o salário de cada funcionário, calcular o custo total dessa concessão de abono e imprimi-lo. Considere o salário mínimo como sendo R$ 600,00."