Estruturas, Uniões e Enumerações

Profª. Giorgia Mattos - giorgiamattos@gmail.com

Structs

- **Estruturas** são variáveis estruturadas que permitem que seus elementos sejam de tipos diferentes. Por isso, são conhecidas como tipos de dados heterogêneos
- Uma estrutura serve para conter dados de tipos diferentes relacionados entre si (por exemplo: nome da pessoa, data de nascimento, RG, CPF,...)
- Cada campo/membro da estrutura possui um nome e um tipo, relacionado ao tipo de dado que o campo irá armazenar
- A definição de uma estrutura em C pode ser feita usando (1) declaração de rótulos e (2) declaração de tipos

Structs

(1) Definição de estruturas usando declaração de rótulos

```
struct rótulo-da-estrutura {
        tipo1 campo1;
        tipo2 campo2;
        ...
        tipoN campoN;
} lista-de-variáveis1;

struct rótulo-da-estrutura lista-de-variáveis2;
...
struct rótulo-da-estrutura lista-de-variáveisN;
```

Structs

- **Exemplo:** variável chamada registroPessoa que contém o nome e a data de nascimento de uma pessoa

```
struct registro {
                char nome[50];
                short dia, mes, ano;
};

struct registro registroPessoa;
```

```
struct {
    char nome[50];
    short dia, mes, ano;
} registroPessoa;
```

Como a struct não tem rótulo, outras variáveis teriam que ser definidas no mesmo local onde a variável registroPessoa foi definida

Structs

(2) Definição de estruturas usando definição de tipos

- O uso de **typedef** é a melhor forma de definição de estruturas

```
typedef struct rótulo-da-estrutura {
            tipo1 campo1;
            tipo2 campo2;
            ...
            tipoN campoN;
} nome-do-tipo;

nome-do-tipo lista-de-nomes-de-variáveis;
```

Structs

- **Exemplo:** definição do tipo tRegistro e a declaração das variáveis

```
typedef   struct {
                        char nome[50];
                        short dia, mes, ano;
                } tRegistro;

tRegistro       Pessoa, OutraPessoa,
                *ptrRegistro;
```

Structs

Inicialização de estruturas

- Uma estrutura pode ser inicializada de modo similar a um vetor
- O número de valores de inicialização não deve exceder o número de campos e deve ser compatível com o respectivo campo

```
typedef     struct {
                        char nome[50];
                        short dia, mes, ano;
                    } tRegistro;

tRegistro Pessoa = {“Fulano de Tal”, 19, 12, 1963};
```

Structs

Atribuição entre estruturas

- Uma estrutura pode ser atribuída à outra desde que ambas sejam do mesmo tipo

```
typedef        struct {
                              char nome[50];
                              short dia, mes, ano;
                       } tRegistro;

tRegistro Pessoa = {“Fulano de Tal”, 19, 12, 1963}, OutraPessoa,
          *ptrRegistro;

OutraPessoa = Pessoa;
Pessoa = OutraPessoa;

ptrRegistro = &Pessoa;
OutraPessoa = *ptrRegistro;
```

Structs

Acesso aos campos da estrutura

- Utilizamos o operador . (ponto) entre o nome da variável que representa a estrutura e o nome do campo que se deseja acessar

```
typedef        struct {
                              char nome[50];
                              short dia, mes, ano;
                        } tRegistro;
tRegistro Pessoa = {"Fulano de Tal", 19, 12, 1963};

    Pessoa.dia = 6;
    scanf ("%d",&Pessoa.mes);
    printf ("Nome: %s", Pessoa.nome);
```

Structs

Acesso aos campos da estrutura

- Utilizamos o operador -> (seta) entre o nome do ponteiro que representa a estrutura e o nome do campo que se deseja acessar

```
typedef     struct {
                    char nome[50];
                    short dia, mes, ano;
                    } tRegistro;

tRegistro Pessoa = {“Fulano de Tal”, 19, 12, 1963};
tRegistro *ptrRegistro = &Pessoa;

Pessoa.dia = 6;
strcpy (Pessoa.nome, “Outro nome”);

ptrRegistro->ano = 1966;
(*ptrRegistro->ano) = 1966
```

Structs

Acesso aos campos da estrutura

- Usa-se o operador . (ponto) para variáveis
- Usa-se o operador -> (seta) para ponteiros

```
typedef     struct {
                    char nome[50];
                    short dia, mes, ano;
                    } tRegistro;

tRegistro Pessoa = {“Fulano de Tal”, 19, 12, 1963};
tRegistro *ptrRegistro = &Pessoa;
```

Campo	Acesso com Pessoa	Acesso com ptrRegistro
nome	Pessoa.nome	ptrRegistro->nome
dia	Pessoa.dia	ptrRegistro->dia
mes	Pessoa.mes	ptrRegistro->mes
ano	Pessoa.ano	ptrRegistro-ano

Structs

Estruturas aninhadas ou compostas

- Um campo de uma estrutura pode ser de qualquer tipo, inclusive de um tipo estrutura

```
typedef    struct {
                char nome[50];
                short dia, mes, ano;
           } tRegistro;
```

Pode ser reescrita como:

Structs

```
typedef     struct {
                short dia, mes, ano;
            } tData;

typedef     struct {
                char nome[50];
                tData data;
            } tRegistro;

tRegistro Pessoa = {“Fulano de Tal”, {19,12,1966}};
tRegistro *ptrRegistro = &Pessoa;
```

Pessoa.data.dia = 6;	ptrRegistro->data.dia = 6;
Pessoa.data.mes = 6;	ptrRegistro->data.mes = 6;

Structs

VETORES DE ESTRUTURAS

```
typedef     struct {
                        char nome[50];
                        short dia, mes, ano;
                    } tRegistro;
```

Pode-se declarar um vetor VPessoa como:

```
tRegistro Vpessoa[20];
```

- **Exemplo:** Calcular a média dos 50 alunos da turma de Introdução a Programação.

```
#include <stdio.h>

#define MAX 50

typedef struct {
                int num;
                char nome[30];
                float a, b, c;
        }Taluno;

int main ()
{
  Taluno aluno[MAX];
  int i;
  float m;
```

```
printf ("-Digite os dados do aluno-\n");

for (i=0; i<MAX; i++) {
    aluno[i].num = i+1;
    printf (“Aluno: %d\n",aluno[i].num);
    printf ("Nome: "); fgets(aluno[i].nome,30,stdin);
    printf ("Nota 1: "); scanf("%f", &aluno[i].a);
    printf ("Nota 2: "); scanf("%f", &aluno[i].b);
    printf ("Nota 3: "); scanf("%f", &aluno[i].c);

    m = (aluno[i].a+aluno[i].b+aluno[i].c)/3;
    printf ("\nMEDIA = %.2f\n\n",m);
}

return 0;

}
```

Structs e Funções

- **Estruturas como Parâmetros de Funções**
  - É possível passar a estrutura inteira, ou
  - Passar um ponteiro para estrutura

- **Estruturas como Retorno de Funções**
  - É possível retornar a estrutura inteira, ou
  - Retornar um ponteiro para estrutura

Structs e Funções

Estruturas como Parâmetros de Funções

```
void ExibeRegistro1 (tRegistro registro) {
        printf ("Nome: %s\n", registro.nome);
        printf ( "Nascimento: %d/%d/%d\n",
                registro.dia, registro.mes, registro.ano );
}


void ExibeRegistro2 (const tRegistro *ptrRegistro) {
        printf("Nome: %s\n", ptrRegistro->nome);
        printf( "Nascimento: %d/%d/%d\n",
                        ptrRegistro->dia, ptrRegistro->mes,
                        ptrRegistro->ano );
}
```

```c
void ExibeRegistro1 (tRegistro registro) {
        printf ("Nome: %s\n", registro.nome);
        printf ( "Nascimento: %d/%d/%d\n",
                registro.dia, registro.mes, registro.ano );
}


void ExibeRegistro2 (const tRegistro *ptrRegistro) {
        printf("Nome: %s\n", ptrRegistro->nome);
        printf( "Nascimento: %d/%d/%d\n",
                        ptrRegistro->dia, ptrRegistro->mes, ptrRegistro->ano );
}
```

```c
tRegistro Pessoa = {“Fulano de Tal", 19, 12, 1963};
tRegistro *ptrPessoa = &Pessoa;

ExibeRegistro1(Pessoa);
ExibeRegistro2(&Pessoa);
ExibeRegistro1(*ptrPessoa);
ExibeRegistro2(ptrPessoa);
```

Structs e Funções

Estruturas como Retorno de Funções

```
tRegistro LeRegistro (void){
        tRegistro registro;

        printf("\nNome (max = %d letras): ", 50);
        fgets (registro.nome, 50, stdin);
        printf("\nDia de nascimento: ");
        scanf (“%d", &registro.dia);
        printf("\nMes de nascimento: ");
         scanf (“%d", &registro.mes);
        printf("\nAno de nascimento: ");
         scanf (“%d", &registro.ano);

        return registro;
}
```

Structs e Funções

Estruturas como Retorno de Funções

```
tRegistro *ModificaData (tRegistro *ptrRegistro){

        ptrRegistro->dia = 6;
        ptrRegistro->mês = 6;
        ptrRegistro->ano = 1966;

        return ptrRegistro;
}
```

```
tRegistro LeRegistro (void){
        tRegistro registro;

        printf("\nNome (max = %d letras): ", 50);
        fgets (registro.nome, 50, stdin);
        printf("\nDia de nascimento: ");
        scanf (“%d", &registro.dia);
        printf("\nMes de nascimento: ");
         scanf (“%d", &registro.mes);
        printf("\nAno de nascimento: ");
         scanf (“%d", &registro.ano);

        return registro;
}
```

```
tRegistro *ModificaData (tRegistro *ptrRegistro){

        ptrRegistro->dia = 6;
        ptrRegistro->mês = 6;
        ptrRegistro->ano = 1966;

        return ptrRegistro;
}
```

```
tRegistro Pessoa;
tRegistro *ptrPessoa;

Pessoa = LeRegistro ();
ExibeRegistro1 (Pessoa);
ptrPessoa = &Pessoa;
ptrPessoa = ModificaData (ptrPessoa);
ExibeRegistro2 (ptrPessoa);
```

Uniões

- São tipos de dados similares às estruturas, com a diferença de que os campos de uma união compartilham a mesma área de memória
- Todos os campos da união começam no mesmo endereço de memória
- São utilizadas com o objetivo de economizar memória quando os campos da união não coexistem
- Obedecem às regras sintáticas semelhantes às estruturas
- Para declarar uma união pode-se usar um dos formatos apresentados nas estruturas, trocando a palavra struct por **union**

Uniões

- Por exemplo:

```
typedef     union{
                        char campo1;
                        double campo2;
                        int campo3;
                    } tUniao;
tUniao minhaU;
```

- O compilador sempre aloca espaço suficiente para conter o membro de maior tamanho da união e todos iniciam no mesmo endereço

Uniões

- Os campos de uma união são mutuamente exclusivos, apenas um deles é considerado válido num dado instante, por exemplo

  minhaU.campo1 = ‘a’;

  minhaU.campo2 = 3.14;

- Ao final da segunda atribuição, o valor ‘a’ será perdido e o acesso ao campo1 ou campo3 produzirá um resultado sem sentido

- A união pode ser inicializada através da atribuição de um valor inicial ao primeiro campo da variável

  tUniao minhaU = {‘a’};

Enumerações

- São úteis quando se deseja utilizar um conjunto determinado de valores constantes com alguma afinidade entre si que podem estar associados a uma variável
- A declaração de variável de um tipo enumeração se dá com a palavra **enum** seguida de uma lista de indicadores entre chaves e pelo nome da variável
- A palavra **enum** pode, opcionalmente, ser seguida por um identificador (rótulo) e neste caso não há a necessidade da variável1,..., variávelN

enum rótulo {lista-de-nomes-de-constantes} variável1, ..., variávelN;

Enumerações

- Por exemplo:

```
enum cores {AZUL, AMARELO, BRANCO, PRETO};

//variável do tipo enumeração representado pelo rótulo cores
enum cores cor1;
enum cores cor2;
```

- Podemos também definir um tipo enumeração, usando o typedef:

```
typedef enum {AZUL, AMARELO, BRANCO, PRETO} tCores;
tCores cor1, cor2;
```

Exemplo

```
typedef enum {SOLTEIRO, CASADO, DIVORCIADO} tEstadoCivil;
```

```
typedef struct {
        char rua[30];
        char numero[5];
        char cidade[20];
        char uf[3];
        char cep [10];
    } tEndereco;
```

```
typedef struct {
        short dia, mes, ano;
    } tData;
```

```
typedef  struct {
        char nome[30];
        tEndereco endereco;
        tEstadoCivil estadoCivil;
        union {
            char nomeConjuge[30];
            short moraSozinho;
            tData dataDivorcio;
        } complemento;
    } tEmpregado;
```

```
tEmpregado empregado;
```

Exemplo

```
if (empregado.estadoCivil == CASADO){
        printf (“%s”,empregado.complemento.nomeConjuge);

} else   if (empregado.estadoCivil == SOLTEIRO) {
                    if (empregado.complemento.moraSozinho)
                            printf (“Mora sozinho”);
                    else
                            printf (“Não mora sozinho”);

        } else   if (empregado.estadoCivil == DIVORCIADO) {
                            printf(“%s”, empregado.complemento.dataDivorcio.dia);
                            printf(“%s”, empregado.complemento.dataDivorcio.mes);
                            printf(“%s”, empregado.complemento.dataDivorcio.ano);
                }
```

Exercícios

1. Criar uma estrutura para receber os nomes de clubes de futebol e seus respectivos pontos no campeonato. Ler os nomes e os pontos e mostrar qual equipe (nome e pontos) é a vencedora. Considerar 10 clubes no total.

2. Faça um programa que leia o código, a descrição, o valor unitário e a quantidade em estoque de 10 produtos comercializados em uma papelaria. Estas informações deverão ser armazenadas em um vetor de estruturas. Depois da leitura dos dados de entrada, o programa deverá:

- Realizar uma rotina que permita alterar a descrição, o valor unitário e a quantidade em estoque de determinado produto, que deverá ser localizado por meio do seu código;
- Realizar uma rotina que mostre todos os produtos cuja descrição comece com determinada letra (informada pelo usuário);
- Mostrar todos os produtos com quantidade em estoque inferior a 5 unidades.

Exercícios

3. Seja uma estrutura para descrever os carros de uma determinada revendedora, contendo os seguintes campos: marca, ano, cor e preço.

a) Escrever a definição da estrutura carro.

b) Declarar um vetor do tipo da estrutura definida acima, de tamanho 20.

Crie um menu para:

- Ler o vetor.
- Ler um preço e mostrar os carros (marca, cor e ano) que tenham preço igual ou menor ao preço recebido.
- Ler a marca de um carro e mostrar as informações de todos os carros dessa marca (preço, ano e cor).
- Ler a marca, ano e cor e informar se existe ou não um carro com essas características. Se existir, informar o preço.

Exercícios

4. Escreva um programa que auxilie no controle de uma fazenda que possui um total de 2000 cabeças de gado. A base de dados é formada por um conjunto de estruturas contendo os seguintes campos referente a cada cabeça de gado: código da cabeça de gado, número de litros de leite produzido por semana, quantidade de alimento ingerida por semana - em quilos, data de nascimento e abate ('N' (não) ou 'S' (sim)).

- Ler a base de dados armazenando em um vetor de estruturas.
- Preencher o campo abate, considerando que a cabeça de gado irá para o abate caso: tenha mais de 5 anos, ou produza menos de 40 litros de leite por semana, ou; produza entre 50 e 70 litros de leite por semana e ingira mais de 50 quilos de alimento por dia.

Crie o menu de opções para:

c) Calcular e mostrar a quantidade total de leite produzida por semana;

d) Calcular e mostrar a quantidade total de alimento consumido por semana;

e) Calcular e mostrar a quantidade total de leite a ser produzida por semana, após o abate;

f) Calcular e mostrar o total de alimento consumido por semana após o abate;

g) Calcular e mostrar a quantidade de cabeças de gado que irão para o abate.

h) Sair do programa.