Programação Imperativa e Funcional

Structs, Unions e Enums

Diego Bezerra

dfb2@cesar.school

C.E.S.A.R school

Na aula anterior...

- Discussão sobre funções e procedimentos em C
- **Exercícios avaliativos usando o Beecrowd**

Objetivos da aula

- Structs
- Unions
- Enums

Introdução

- Juntamente com vetores, as **structs** permitem implementar diversos outros tipos agregados mais complexos
- Permite **definir tipos de dados que agrupam variáveis sob um mesmo tipo**
- Imagine que temos que apresentar uma solução em que necessitamos **agrupar dados de um cachorro em um único tipo de dados** (cachorro que possui tamanho, raça e cor)
    - **Structs** em C tem um importante papel nesta tarefa

Struct

- **Structs** são coleções arbitrárias de variáveis logicamente relacionadas
- Como no vetor, essas variáveis compartilham o mesmo nome e ocupam posições consecutivas de memória
- As variáveis que fazem parte de uma **struct** são denominadas **membros** e são identificadas por **nomes**

Struct

```
int tamanho;
char raca[40];
char cor[40];
```

Abstração

Tamanho	Raça	Cor
40 cm	Pug	Fulvo

Formalização

Struct

```
struct Cachorro //nome
{
     int tamanho; //membro
     char raca[40]; //membro
     char cor[40];//membro
};
```

Abstração

Tamanho	Raça	Cor
40 cm	Pug	Fulvo

Formalização

Struct

```
struct Cachorro //nome
{
     int tamanho; //membro
     char raca[40]; //membro
     char cor[40];//membro
};
```

- Representa uma abstração do mundo real
- Utiliza tipos básicos para composição
- Permite definir um tipo de dado
- Em outras linguagens **structs** podem ser chamadas de **registros**
- **Membros** também podem ser chamados de **campos**

```
struct <identificador>
{
     <tipo><membro>;
};
```

Struct anônima

- Neste exemplo, temos uma **struct** anônima, não podendo referenciar em outras partes do programa pelo nome **Cachorro**
- Teriamos que usar o termo **struct Cachorro**

```
int main() {
     struct Cachorro //tipo de dado
     {
          int tamanho;
          char raca[40];
          char cor[40];
     };

     struct Cachorro doguinho;
     return 0;
}
```

Struct rotulada

- Para criarmos uma estrutura rotulada, devemos usar a palavra reservada **typedef e o nome a ser usado como rotulo**

```
int main() {
    struct Cachorro //tipo de dado rotulado
    {
        int tamanho;
        char raca[40];
        char cor[40];
    } typedef Cachorro;

    Cachorro doguinho;
    return 0;
}
```

Struct: declaração e inicialização

- Assim como vetores, podemos inicializar structs na declaração

```
int main() {
    struct Cachorro //tipo de dado rotulado
    {
        int tamanho;
        char raca[40];
        char cor[40];
    } typedef Cachorro;

    Cachorro doguinho = {50, “Pastor alemão”,
“Cinza”};
    return 0;
}
```

Struct: declaração e inicialização

- Podemos declarar e inicializar depois

```
int main() {
    struct Cachorro //tipo de dado
    {
        int tamanho;
        char raca[40];
        char cor[40];
    } typedef Cachorro;

    Cachorro doguinho;
    scanf("%d %s %s", &doguinho.tamanho, doguinho.raca, doguinho.cor);
    printf("%d", doguinho.tamanho);
    return 0;
}
```

Struct: vetores de structs

- Podemos combinar **vetores** e **structs** de muitas maneiras
  - Podemos ter um **struct** que tem como membro um vetor ou criar um vetor com elementos do tipo **struct**

```
int main() {
    struct Cachorro //tipo de dado
    {
        int tamanho;
        char raca[40]; // Membro do tipo vetor de char (string)
        char cor[40]; // Membro do tipo vetor de char (string)
    } typedef Cachorro;
    Cachorro canil[10]; // Vetor com elemento do tipo estruturado
    return 0;
}
```

Struct: membros do tipo struct

- Podemos definir membros também com o tipo estruturado (aninhados)

```
int main() {
    struct Tutor {char nome[100];} typedef Tutor;
    struct Cachorro {
        int tamanho;
        char raca[40]; // Membro do tipo vetor de char (string)
        char cor[40]; // Membro do tipo vetor de char (string)
        Tutor pessoa; // Membro do tipo estruturado
    } typedef Cachorro;

    return 0;
}
```

Struct: funções e procedimentos

```
struct Tutor {
    int matricula;
    char nome[100];
} typedef Tutor;

int get_matricula_tutor(Tutor t) {
    return t.matricula;
}

int main() {
    Tutor p = {1000, “Gabriel”};
    int mat_cli = get_matricula_tutor(p);
    return 0;
}
```

Struct: inicializando membros do tipo struct

```
int main() {
    struct Tutor {int matricula, char nome[100];}  typedef Tutor;
    struct Cachorro {
        int tamanho;
        char raca[40]; // Membro do tipo vetor de char (string)
        char cor[40]; // Membro do tipo vetor de char (string)
        Tutor pessoa; // Membro do tipo estruturado
    } typedef Cachorro;

    Tutor p = {001, “Gabriel”};
    Cachorro c = {50, “Cane Corso”, “Preto”, p};  // ou c.pessoa = p;

    return 0;
}
```

Struct: exercicios

- Defina um tipo de estrutura para armazenar um **horário** composto de **hora**, **minutos** e **segundos**. Crie e inicialize uma variável desse tipo e, em seguida, mostre seu valor na tela usando o formato "99:99:99".
- Defina um tipo de estrutura para armazenar dados de um **vôo** como, por exemplo os nomes das cidades de **origem** e **destino**, **datas** e horários de de **partida** e **chegada**. Crie uma variável desse tipo e atribua valores aos seus membros usando notação de ponto e, depois, inicialização.

Unions

- Uma união é um tipo especial de estrutura capaz de **armazenar um único membro por vez**
- Uma união é um recurso que nos permite armazenar diferentes tipos de dados num mesmo local da memória
  - Ideal quando você sabe que somente uma variável será usada por vez

```
union Sensor //nome
{
    float temperatura; //membro
    float umidade; //membro
    char estado;//membro
};
```

Union: declaração e inicialização

- **Usando o mesmo espaço de memória, os dados são sobrescritos quando inicializamos cada membro**
- Em structs, cada membro ocupa um espaço diferente de memória

```
void main() {
    union Sensor //nome
    {
        float temperatura; //membro
        float umidade; //membro
        char estado;//membro
    };
    union Sensor sensor;
    sensor.temperatura = 25.5;
    sensor.umidade = 60;
    sensor.estado = 'o'
}
```

Union rotulada

- Podemos rotular uma estrutura Union usando **typedef**

```
void main() {
    union Sensor //nome
    {
        float temperatura; //membro
        float umidade; //membro
        char estado;//membro
    } typedef Sensor;
    Sensor sensor;
    sensor.temperatura = 25.5;
    sensor.umidade = 60;
    sensor.estado = 'o'
    printf("Temperatura: %.2f\n", sensor.temperatura); }
```

Union: exercicios

- Usando uma união rotulada, defina um tipo de dados para representar figuras geométricas, como retângulos e círculos, e crie uma função para calcular a área de uma figura.

Enums

- Conjunto de constantes inteiras que especifica todos os valores legais que uma variável deste tipo pode assumir
  - Permite definir os valores possíveis para uma variável (ex.: dias da semana, meses do ano etc.)
- De forma geral, temos:

```
enum <identificador>//nome
{
    <lista_enum>
} <lista_variaveis>;
```

Enum: declaração e inicialização

- Os valores que semana1 e semana2 podem assumir estão definidos pelo **enum** dias
- Quando assume seg, fica associado o valor 0
- Quando assume ter, fica associado o valor 1
- E assim sucessivamente...

```
void main() {
     enum dias { seg, ter, qua, qui, sex, sab, dom};
     enum dias semana1, semana2;
     semana1 = qua;
     semana2 = dom;
     printf("%d", semana1); // imprime 2
}
```

Enum: com rotulo

- Podemos usar o **typedef** também para rotular Enums

```
int main() {
     enum dias {
          seg, ter, qua, qui, sex, sab, dom
     } typedef dias;
     dias semana1, semana2;
     semana1 = qua;
     semana2 = dom;
     print("%d", semana1}; // imprime 2
}
```

Enum: Exemplo

- Podemos utilizar para definir comandos de escape do teclado

```
int main() {
    enum escapes {
        bell='\a', backspace='\b', tab='\t', newline='\n',
    vertical_tab='\v'} typedef escapes;

    printf("%d%c%d", 10, newline, 10);
}
```

Enum: exercicios

- Escreva um programa que dado um inteiro entre 1 e 12 informado pelo usuario, imprima na tela o mes correspondente (ex.: 1: janeiro, 2: fevereiro, ...). Utilize a estrutura Enum e a estrutura de decisão switch.

Referências

- Rangel Netto, J. L. M., Cerqueira, R. D. G., & Celes Filho, W. (2004). **Introdução a estrutura de dados: com técnicas de programação em C**.