

Tópico 5

Structures e Unions

Introdução (1 de 4)

- **Estrutura** é tipo de dados **definido pelo usuário** que pode armazenar informações relacionadas (mesmo de tipos de dados diferentes) juntas.
- Uma **estrutura** é declarada usando a palavra-chave **struct** seguida por um nome que representa a estrutura.
- **Todas as variáveis** de uma estrutura são declaradas **dentro** dela.

Introdução (2 de 4)

- Uma **estrutura** é **definida** usando a seguinte **sintaxe**:

```
struct nome_estrutura
```

```
{  tipo_dado nome_variável1;
```

```
tipo_dado nome_variável2;
```

```
...
```

```
};
```

Introdução (3 de 4)

```
struct estudante  
{ int matricula;  
  char nome[20];  
};
```

- A definição da estrutura **não aloca memória**.
- Apenas fornece um modelo que transmite ao compilador como a estrutura é apresentada na memória e fornece detalhes dos membros.

Introdução (4 de 4)

- A **memória é alocada** para a estrutura quando **declaramos** uma **variável** de estrutura.
- Por exemplo, podemos definir uma variável de aluno escrevendo:
`struct estudante estud1;`

Declaração Typedef (1 de 2)

- Quando **precedemos** um nome de estrutura com a palavra-chave **typedef**, a estrutura se torna um **novo tipo**.
- Por exemplo, considere a seguinte declaração:

```
typedef struct estudante  
{  
    int matricula;  
    char nome[20];  
};
```

Declaração Typedef (2 de 2)

- Agora **podemos declarar** imediatamente as **variáveis** deste novo tipo de dados **conforme declaramos** as variáveis do tipo **int, float, char, double**, etc.
- Para declarar uma variável de estrutura aluno, vamos apenas escrever:

estudante estud1;

Inicializando Structures

- **Inicializar** uma **estrutura** significa **atribuir** algumas **constantes** aos membros da estrutura.
- Quando o **usuário não inicializa explicitamente a estrutura**, a linguagem **C** faz isso **automaticamente**.
- Para membros **int** e **float**, os valores são inicializados para **zero** e os membros **char** e **string** são inicializados para **'\0'** por padrão.

Inicializando Structures

- Os **inicializadores** são **colocados** entre **colchetes** e **separados** por **vírgulas**.
- Observe que os inicializadores devem corresponder a seus tipos correspondentes na definição da estrutura.

Inicializando Structures

- A sintaxe geral para inicializar uma variável de estrutura é fornecida da seguinte maneira:

```
struct nome_struct
```

```
{
```

```
    tipo_dado nome_var1;
```

```
    tipo_dado nome_var2;
```

```
    tipo_dado nome_var3;
```

```
    .....
```

```
} struct_var = {constante1, constante2, constante3, ....};
```

Acessando Membros de uma Structure

- Cada membro de uma estrutura pode ser usado como uma variável normal, mas seu nome será um pouco mais longo.
- Uma variável de membro de estrutura geralmente é acessada usando um '.' (Operador ponto).
- A sintaxe para acessar um membro de uma estrutura é:
`struct_var.nome_membro`

Acessando Membros de uma Structure

- Por exemplo, para atribuir valor aos membros de dados individuais da variável de estrutura stud1, podemos escrever:

```
stud1.matric = 01;
```

Acessando Membros de uma Structure

- Podemos atribuir uma estrutura a outra estrutura do mesmo tipo.
- Por exemplo, se tivermos **duas** variáveis de estrutura **stud1** e **stud2** do tipo struct estudante

```
struct estudante stud1 = {01, "Rahul", "BCA", 45000};
```

```
struct estudante stud2;
```

Acessando Membros de uma Structure

- Então, para atribuir uma variável de estrutura a outra, escreveremos:

```
stud2 = stud1;
```

Estruturas Aninhadas

- Uma estrutura pode ser colocada **dentro de outra** estrutura.
- Essa estrutura que contém outra estrutura como seu membro é chamada de **estrutura aninhada**.

```
typedef struct
{ char first_name[20];
  char mid_name[20];
  char last_name[20];
} NAME;
```

```
typedef struct
{ int dd;
  int mm;
  int yy;
}DATE;
```

```
typedef struct
{ int r_no;
  NAME name;
  DATE DOB;
}student;
```

Para atribuir valores aos campos da estrutura, escreveremos:

```
struct student stud1;
stud1.name.first_name = "Janak";
stud1.DOB.dd = 15;
stud1.DOB.mm = 03;
stud1.DOB.yy= 1990;
```

Arrays de Estruturas

- A sintaxe geral para declarar uma matriz de estrutura pode ser dado como:

```
struct struct_name struct_var [índice];
```

```
struct student stud [30];
```

- Agora, para atribuir valores ao iº aluno da classe, escreveremos:

```
stud [i] .r_no = 09;
```

```
stud [i] .name = "RASHI";
```

```
stud [i] .curso = "MCA";
```

```
stud [i] .fees = 60000;
```

Passando Membros Individuais da Estrutura para uma Função

- Para passar qualquer membro individual da estrutura para uma função, devemos usar o operador de seleção direta para nos referir aos membros individuais para os parâmetros reais.
- O programa chamado não sabe se as duas variáveis são variáveis comuns ou membros da estrutura.

Passando Membros Individuais da Estrutura para uma Função

```
typedef estrutura
```

```
{
```

```
    int x;
```

```
    int y;
```

```
} PONTO;
```

```
PONTO p1 = {2,3};
```

```
exibir (p1.x, p1.y); // passando membros de p1 para a função
```

```
exibir ()
```

Passando Estruturas por Porteiros

- C permite criar um **ponteiro** para uma **estrutura**.
- Como em outros casos, um ponteiro para uma estrutura **nunca é em si uma estrutura**, mas apenas uma **variável** que **contém** o **endereço** de uma **estrutura**.

Passando Estruturas por Porteiros

- A sintaxe para declarar um ponteiro para uma estrutura pode ser fornecida como:

```
struct struct_name  
{  
    tipo_dado member_name1;  
    tipo_dado member_name2;  
    .....  
} *ptr;  
// ou
```

```
struct struct_name *ptr;
```

Passando Estruturas por Porteiros

- Para nossa estrutura de aluno, podemos declarar uma variável de ponteiro escrevendo:

```
struct student *ptr_stud, stud;
```

- A próxima etapa é atribuir o endereço de stud ao ponteiro usando o operador de endereço (&). Então, para atribuir o endereço, escreveremos:

```
ptr_stud = & stud;
```

- Para acessar os membros da estrutura, uma maneira é escrever:

```
(* ptr_stud) .roll_no;
```

Passando Estruturas por Porteiros

- Uma alternativa para a declaração acima pode ser usada usando o operador '**apontando para**' (->):
`ptr_stud->roll_no = 01;`

Estruturas AutoReferenciais

- Estruturas autoreferenciais são aquelas estruturas que contêm uma referência a dados de seu mesmo tipo.
- Ou seja, uma estrutura autoreferencial contém um ponteiro para dados que são do mesmo tipo da estrutura.

Struct no

```
{  
    int val;  
    struct node * next;  
};
```

Estruturas AutoReferenciais

- No exemplo anterior, o nó da estrutura contém dois tipos de dados: um valor inteiro e o próximo que é um ponteiro para um nó.
- Você deve estar se perguntando por que precisamos de tal estrutura?
- Na verdade, a estrutura autoreferencial é a base de outras estruturas de dados.

Unions

- Semelhante a estruturas, uma union é uma coleção de variáveis de diferentes tipos de dados.
- A diferença entre uma estrutura e uma união é que em unions, as informações podem ser armazenadas em um campo a qualquer momento.
- As uniões são usadas para economizar memória. Eles são úteis para aplicativos que envolvem vários membros, onde os valores não precisam ser atribuído a todos os membros a qualquer momento.

Unions

- A sintaxe para declaração de união pode ser fornecida como:

```
union nome-union
```

```
{  
    tipo-dado var-name1;  
    tipo-dado var-name2;  
    .....  
};
```

Inicializando Unions

```
#include <stdio.h>
typedef union POINT2
{
    int x;
    int y;
};
int main()
{
    POINT2 P2;
    P2.x = 4;
    printf("\n A coordenada x de P2 é: %d", P2.x);
    P2.y = 5;
    printf("\n A coordenada y de P2 é %d", P2.y);
    return 0;
}
```

Arrays de Unions

```
#include <stdio.h>
union POINT
{
    int x, y;
};
int main()
{
    int i;
    union POINT points[3];
    points[0].x = 2;          points[0].y = 3;          points[1].x = 4;
    points[1].y = 5;          points[2].x = 6;          points[2].y = 7;
    for(i=0;i<3;i++)
        printf("\n Coordenadas do Ponto[%d] são %d e %d", i,
            points[i].x, points[i].y);
    return 0;
}
```

Unions dentro de Structures

```
#include <stdio.h>
struct student
{
    union
    {
        char name[20];
        int roll_no;
    };
    int marks;
};
/* in main() function*/

printf("\n Você pode inserir o nome ou número do rolo do aluno");
printf("\n Você quer inserir o nome? (Y or N): "); gets(choice);
if(choice=='y' || choice=='Y')
{
    printf("\n Entre com o nome: ");
    gets(stud.name);
}
else
{
    printf("\n Entre com o numero da lista: ");
    scanf("%d", &stud.roll_no);
}
```