

CET635 – Linguagem de Programação II

Aula 3. *Tipos de Dados Definidos*

Estruturas

Profa. Susana M Iglesias

Roteiro

- Motivação
- Introdução
- Definindo um tipo estrutura
- Declarando variáveis de tipo estrutura
- Operações com variáveis de tipo estrutura
- Inicialização de variáveis
- Acesso a membros de uma estrutura
- Estruturas na memória
- Criação de sinônimos
- Exemplos
- Estruturas autoreferenciadas
- Alocação dinâmica de estruturas

Motivação

- Existem numerosas entidades “complexas” que não podem ser representadas usando os tipos básicos de dados.
- Exemplos:



Fabricante	String
Modelo	String
Ano	int
Combustível	char
Torque	float

Motivação

- Exemplos ...

Aluno	Nome	String
	Idade	int
	Sexo	char
	Matricula	long int
	email	String
	CR	double
Turma	Disciplina	String
	Período	String
	Código	int
	<u>Alunos</u>	<u>vetor de alunos</u>

Introdução

- Para resolver o problema anterior a linguagem C permite criar tipos definidos pelo usuário,
- Representamos uma entidade “complexa” criando um tipo agregado ou estrutura,
- Estruturas são grupos de variáveis relacionadas entre si que são agrupadas sob um mesmo nome,
- As estruturas podem conter variáveis de muitos tipos diferentes de dados relacionadas entre si,

Introdução

- A estrutura é um tipo de dado definido pelo usuário,
- A estrutura é um tipo de dado derivado, elas são construídas utilizando tipos de dados básicos,
- As variáveis que compõem uma estrutura são chamados de membros, elementos ou campos da estrutura.

DEFININDO UMA ESTRUTURA

- Sintaxe:

```
struct nome_da_estrutura{  
    tipo_membro1 nome_membro1;  
    tipo_membro2 nome_membro2;  
    ...  
    tipo_membroN nome_membroN;  
};
```

- Estrutura aluno:

```
struct Taluno{  
    int matricula;  
    double CR;  
    char sexo;  
};
```

- Estrutura turma:

```
struct Tturma{  
    char *disciplina;  
    int cod;  
    struct Taluno Al[25];  
};
```

DEFININDO UMA ESTRUTURA

- Definir uma estrutura, cria um novo tipo de dados,
- O fato de definir uma estrutura não cria nenhuma variável e nenhum espaço é reservado na memória,
- Geralmente as estruturas são definidas fora de todas as funções, no começo do programa, e utilizadas ao longo de todo o código,
- Uma estrutura é formada pelo rotulo ou nome da estrutura e seus membros.

DEFININDO UMA ESTRUTURA

- Um membro de uma estrutura pode ser um tipo estrutura criado anteriormente,
- Um membro de uma estrutura **NÃO** pode ser uma instancia da própria estrutura,
- Um membro de uma estrutura pode ser um ponteiro para o uma variável da mesma estrutura (estruturas autoreferenciadas).

DECLARANDO VARIÁVEIS DE TIPO ESTRUTURA

- Sintaxe:

struct nome_estrutura variavel1, variavel2;

- Exemplo:

```
struct Taluno{
    int matricula;
    double CR;
    char sexo;
};

int main(){
    struct Taluno aluno1, alunos[10];
    ...
    return 0;
}
```

DECLARANDO VARIÁVEIS DE TIPO ESTRUTURA

- Outras alternativas:

```
struct Taluno{  
    int matricula;  
    double CR;  
    char sexo;  
}aluno1, alunos[10];
```

```
struct{  
    int matricula;  
    double CR;  
    char sexo;  
}aluno1, alunos[10];
```

- **Cuidado!!!**, ao declarar variáveis na própria definição da estrutura poderíamos estar criando variáveis globais.

OPERAÇÕES -VARIÁVEIS ESTRUTURA

- As operações validas com variáveis estruturas são:
 - Atribuição
 - Obter o endereço (&)
 - Usar o operador `sizeof()` para determinar seu tamanho
- Atribuição de estruturas

```
int main(){
    struct Taluno aluno1, aluno2;
    ...
    aluno1 = aluno2;
    ...
    return 0;
}
```

OPERAÇÕES -VARIÁVEIS ESTRUTURA

- Obtendo endereço

```
int main(){
    struct Taluno aluno1, *ptrA;
    ...
    ptrA = &aluno1;
    ...
    return 0;
}
```

- Usando sizeof()

```
int main(){
    struct Taluno aluno1;
    ...
    printf("Tamanho %d\n", sizeof(aluno1);
    ...
    return 0;
}
```

INICIALIZANDO ESTRUTURAS

- As estruturas podem ser inicializadas utilizando uma lista de inicializadores,
- Exemplo:

```
struct Taluno alunos[3]={ {200501317,8.3, 'M' } ,  
                           {200501234,7.1, 'F' } ,  
                           {200501453,9.0, 'M' } } ;
```

ACESSO A MEMBROS DE ESTRUTURAS

- Podemos acessar os membros de uma estrutura utilizando os operadores:
 - (.) Operador de ponto ou operador membro de estrutura
 - (->) Operador de seta ou operador ponteiro de estrutura
- Usamos o operador (.) quando acessamos o membro da estrutura por meio do nome da variável.

ACESSO A MEMBROS DE ESTRUTURAS

- Usamos o operador (->) quando acessamos ao membro da estrutura por meio de um ponteiro à variável estrutura.

- Sintaxe:

nome_estrutura.nome_membro;

ponteiro_estrutura->nome_membro;

ACESSO A MEMBROS DE ESTRUTURAS

- Exemplo:

```
/* Acesso a membros de estrutura */  
struct Taluno{  
    int matricula;  
    double CR;  
    char sexo;  
};  
  
int main(){  
    struct Taluno *Ptr, mAluno = {20060908, 9.0, 'M'};
```

ACESSO A MEMBROS DE ESTRUTURAS

- Exemplo ...

```
/* Acesso a membros de estrutura (cont.) */

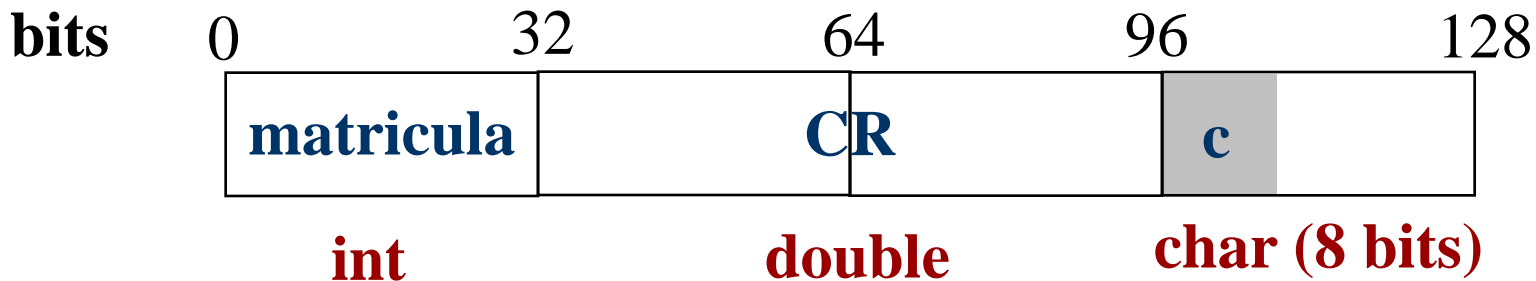
//Usando operador ponto
printf("Matricula: %d\n", mAluno.matricula);
printf("CR: %f\n", mAluno.CR);
printf("Sexo: %c\n", mAluno.sexo);

Ptr = &mAluno;
//Usando operador seta
printf("Matricula: %d\n", Ptr->matricula);
printf("CR: %f\n", Ptr->CR);
printf("Sexo: %c\n", Ptr->sexo);

return 0;
}
```

ESTRUTURAS NA MEMÓRIA

- Representação na memória da estrutura `aluno`.



- Em dependência do sistema de memória (tamanho da palavra, mecanismo de endereçamento) e os tipos membros de nossa estrutura podem existir “buracos” na memória.

ESTRUTURAS NA MEMÓRIA

- As variáveis estrutura não podem ser comparadas utilizando operadores relacionais. Porque?
- Ao comparar uma estrutura devemos utilizar os membros da estrutura. Isto é, duas estruturas são iguais se cada um de seus membros são iguais.

ESTRUTURAS NA MEMÓRIA

- Comparando estruturas:

```
int main() {  
    struct Taluno aluno1, alunos2;  
    ...  
    if (aluno1 == aluno2) { ... };  
    ...  
    return 0;  
}
```

- A comparação anterior é **incorreta**.

ESTRUTURAS NA MEMÓRIA

- Comparando estruturas ...

```
int main() {
    struct Taluno aluno1, aluno2;
    ...
    if ((aluno1.matricula == aluno2.matricula)
        && (aluno1.CR == aluno2.CR)
        && (aluno1.sexo == aluno2.sexo))
    { ... };
    ...
    return 0;
}
```

- A comparação anterior é **correta**.

CRIAÇÃO DE SINONIMOS (`typedef`)

- A palavra-chave `typedef` fornece um mecanismo para a criação de sinônimos ou alias para tipos de dados definidos previamente,
- Ao utilizarmos `typedef` não criamos um novo tipo de dados, pelo contrário definimos um novo nome para um tipo já existente,
- O uso de `typedef`, facilita a compreensão dos tipos de dados criados pelo programador,
- As estruturas são freqüentemente definidas usando `typedef` para criar nomes mais curtos e intuitivos,

CRIAÇÃO DE SINONIMOS (typedef)

- Sintaxe: `typedef Tipo_de_Dado Alias;`

- Exemplo:

```
struct aluno{
    int matricula;
    double CR;
    char sexo;
};

typedef struct aluno Taluno;

int main(){
    Taluno aluno1, alunos[10];
    ...
    return 0;}

```


CRIAÇÃO DE SINONIMOS (typedef)

- Geralmente ao criarmos estruturas usando `typedef` o nome da estrutura é omitido,
- Exemplo:

```
typedef struct {  
    int matricula;  
    double CR;  
    char sexo;  
} Taluno;  
  
int main() {  
    Taluno aluno1, alunos[10];  
    ...  
    return 0;}
```

EXEMPLO 1

- Que faz o seguinte programa?

```
#define N 3
```

```
typedef struct{  
    int matricula;  
    double CR;  
} Taluno;
```

```
Aluno: 0 Matricula: 200501317 Rendimento: 8.33  
Aluno: 1 Matricula: 200501234 Rendimento: 7.10  
Aluno: 2 Matricula: 200501453 Rendimento: 9.00
```

```
int main()  
{  
    Taluno *Ptr, alunos[N]={ {200501317,8.3},  
                               {200501234,7.1},  
                               {200501453, 9}};  
  
    int i;  
    Ptr = alunos;  
    for(i=0;i<N;i++,Ptr++)  
        printf("Aluno: %d Matricula: %10d Rendimento: %3.2f\n",  
               i,  
               Ptr->matricula,  
               Ptr->CR) ;  
  
    system("PAUSE");  
    return 0;  
}
```

EXEMPLO 2

- Que faz o seguinte programa?

```
typedef struct{
    char  matricula[10];
    float CR;
    char  sexo;
} Taluno;

int main()
{
    Taluno aluno;

    printf("Digite matricula: ");
    gets(aluno.matricula);
    printf("Digite sexo (M ou F): ");
    scanf("%c", &aluno.sexo);
    printf("Digite CR: ");
    scanf("%f", &aluno.CR);
    printf("\nDados\nMatricula: %s\nRendimento: %3.2f\nSexo: %c\n",
        aluno.matricula,
        aluno.CR,
        aluno.sexo);

    system("PAUSE");
    return 0;
}
```

Digite matricula: 200502415
Digite sexo (M ou F): M
Digite CR: 7.3

Dados
Matricula: 200502415
Rendimento: 7.30
Sexo: M

EXEMPLO 3

- *Crie um programa que receba os dados de 5 alunos (nro matricula, nome, idade, coeficiente de rendimento). Imprima o nome do melhor aluno e a matricula do aluno mais jovem.*

```
#define N 5

typedef struct{
    int matricula;
    char nome[15];
    int idade;
    float CR;
}Taluno;

int main(){
    Taluno alunos[N]; int melhor=0, jovem=0, i;

    for(i=0;i<N;i++){
        printf("\nDados do aluno %d\n", i+1);
        printf("Digite a matricula: ");
        scanf("%d", &alunos[i].matricula);
        fflush(stdin);
        printf("Digite o nome: ");
        gets(alunos[i].nome);
        printf("Digite a idade: ");
        scanf("%d", &alunos[i].idade);
        printf("Digite o coef. de rendimento: ");
        scanf("%f", &alunos[i].CR);
        if (alunos[melhor].CR<alunos[i].CR) melhor = i;
        if (alunos[jovem].idade>alunos[i].idade) jovem = i;
    }
```

```
printf("O melhor aluno e %s\n", alunos[melhor].nome);  
printf("A matricula do mais jovem e %d\n",  
      alunos[jovem].matricula);  
  
system("PAUSE");  
return 0;  
}
```

Dados do aluno 1

Digite a matricula: 20062504

Digite o nome: Jose

Digite a idade: 21

Digite o coef. de rendimento: 9.0

Dados do aluno 2

Digite a matricula: 20062505

Digite o nome: Pedro

Digite a idade: 22

Digite o coef. de rendimento: 8.7

Dados do aluno 3

Digite a matricula: 20062506

Digite o nome: Elisa

Digite a idade: 20

Digite o coef. de rendimento: 9.2

Dados do aluno 4

Digite a matricula: 20062507

Digite o nome: Arivaldo

Digite a idade: 25

Digite o coef. de rendimento: 7.5

Dados do aluno 5

Digite a matricula: 20062508

Digite o nome: Josinaldo

Digite a idade: 23

Digite o coef. de rendimento: 7.0

O melhor aluno e Elisa

A matricula do mais jovem e 20062506

EXEMPLO 4

- *Crie um programa que leia os resultados de uma corrida de fórmula 1 onde participaram 5 equipes. Para cada equipe será lido o nome da equipe e o tempo da corrida em segundos. Imprima uma lista com os nomes das equipes segundo a ordem de chegada.*

```
#define N 5
```

```
typedef struct{  
    char nome[10];  
    float tempo;  
}Tcarro;
```

```
int main()  
{  
    Tcarro carros[N];  
    int ordem[N], i, j, buff;  
  
    printf("Entrada de Dados\n");  
    for(i=0;i<N;i++){  
        ordem[i]=i;  
        fflush(stdin);  
        printf("Digite o nome da equipe: ");  
        gets(carros[i].nome);  
        printf("Digite o tempo de chegada: ");  
        scanf("%f", &carros[i].tempo);  
    }  
}
```

```

for(i=0;i<N;i++){
    for(j=0;j<N-1;j++){
        if (carros[ordem[j]].tempo>carros[ordem[j+1]].tempo){
            buff = ordem[j];
            ordem[j] = ordem[j+1];
            ordem[j+1] = buff;
        }
    }
    printf("\nSaida de dados\n");
    for(i=0;i<N;i++){
        printf("Equipe: %10s    Tempo: %3.1f\n",
            carros[ordem[i]].nome,
            carros[ordem[i]].tempo);
    }

    system("PAUSE");
    return 0;
}

```

Entrada de Dados

Digite o nome da equipe: Ferrari

Digite o tempo de chegada: 55

Digite o nome da equipe: Williams

Digite o tempo de chegada: 60

Digite o nome da equipe: BMW

Digite o tempo de chegada: 53

Digite o nome da equipe: Porsche

Digite o tempo de chegada: 49

Digite o nome da equipe: Toyota

Digite o tempo de chegada: 62

Saida de dados

Equipe: Porsche Tempo: 49.0

Equipe: BMW Tempo: 53.0

Equipe: Ferrari Tempo: 55.0

Equipe: Williams Tempo: 60.0

Equipe: Toyota Tempo: 62.0

Pressione qualquer tecla para continuar. . .

Estruturas Autoreferenciadas

- Um membro de uma estrutura não pode ser uma instancia da própria estrutura,
- Entretanto um membro de uma estrutura pode ser um ponteiro a uma instancia da própria estrutura,
- Em tais casos temos estruturas autoreferenciadas que são a base de criação de estruturas de dados complexas como pilhas, filas e listas.

Estruturas Autorefenciadas

- Sintaxe:

```
struct nome_estrutura{  
    tipo_membro1 nome_membro1;  
    tipo_membro2 nome_membro2;  
    ...  
    struct nome_estrutura *Ptr;  
};
```

- Exemplo:

```
typedef struct carro{  
    char nome[10] ;  
    float tempo;  
    struct carro *proximo;  
}Tcarro;
```

Estruturas – Alocação Dinâmica

- Ao igual que os tipos de dados básicos variáveis estrutura podem ser alocadas dinamicamente utilizando `malloc()`, `sizeof()` e `free()`.
- Exemplo:

```
TCarro *ptr;  
  
ptr = malloc (sizeof(TCarro)) ;  
if (ptr == NULL) {//Erro}  
...  
ptr->tempo = 100;  
...  
free(ptr)
```

EXEMPLO 5

- O seguinte programa ilustra os conceitos de estruturas autoreferenciadas e alocação dinâmica de estruturas.

```
typedef struct carro{ //Definição da estrutura
    char nome[10];
    float tempo;
    struct carro *proximo;
}Tcarro;

int main(){
    Tcarro *ptr; //Declarando um ponteiro a estrutura
    ptr = malloc(sizeof(Tcarro)); //Alocação de memoria
    if (ptr == NULL){
        printf("Erro: Memoria insuficiente.\n");
        return -1;
    }
```



```

//Entrada de dados
printf("Digite o nome da equipe: ");
gets(ptr->nome);
printf("Digite o tempo de chegada: ");
scanf("%f", &(ptr->tempo));

//Saida de dados
printf("\nSaida de dados\n");
printf("Equipe: %10s    Tempo: %3.1f\n",
        ptr->nome,
        ptr->tempo);

//Liberando memoria
free(ptr);
system("PAUSE");
return 0;
}

```

```

Digite o nome da equipe: Ferrari
Digite o tempo de chegada: 75.3

```

```

Saida de dados
Equipe:      Ferrari    Tempo: 75.3
Pressione qualquer tecla para continuar. .

```

EXEMPLO 6

- Que faz o seguinte programa:

```
typedef struct carro{
    char nome[10];
    float tempo;
    struct carro *proximo;
}Tcarro;

int main()
{
    Tcarro *raiz = NULL, *ultimo;
    char nome[10];

    printf("Entrada de Dados\n");
    printf("Digite o nome da equipe (0 para finalizar):");
    gets(nome);
```

```
while (nome[0] != '0') {
    if (raiz == NULL) {
        raiz = malloc(sizeof(Tcarro));
        ultimo = raiz;
    }
    else{
        ultimo->proximo = malloc(sizeof(Tcarro));
        ultimo = ultimo->proximo;
    }
    if (ultimo == NULL) {
        printf("Erro: Memoria insuficiente.\n");
        return -1;
    }
    ultimo->proximo = NULL;
    strcpy(ultimo->nome, nome);
    printf("Digite o tempo de chegada: ");
    scanf("%f", &(ultimo->tempo));
    fflush(stdin);
    printf("Digite o nome da equipe (0 para finalizar):");
    gets(nome);
}
```

```
if (raiz != NULL) {  
    printf("\nSaida de dados\n");  
    while(raiz != NULL) {  
        ultimo = raiz;  
        printf("Equipe: %10s    Tempo: %3.1f\n",  
                ultimo->nome,  
                ultimo->tempo);  
        raiz = ultimo->proximo;  
        free(ultimo);  
    }  
}  
  
system("PAUSE");  
return 0;  
}
```