

LINGUAGEM DE PROGRAMAÇÃO II

© 2022, Gizelle Kupac Vianna (DECOMP/UFRRJ)

REGISTROS OU STRUCTS

Aula 6

Estruturas Heterogêneas de Dados

- Um registro, ou estrutura (struct), é um conjunto de uma ou mais variáveis, que podem ser de **tipos diferentes**, agrupadas sob um **único nome**.
- Agrupar dados diversos em uma única estrutura e referenciar esse conjunto por um único nome facilita a manipulação dos dados armazenados.
- Uma boa analogia com esse tipo de estrutura seria uma ficha catalográfica, como uma ficha médica, por exemplo.

Ficha do Aluno

DADOS PESSOAIS

ENDEREÇO

DADOS FAMILIARES

DOCUMENTOS

DADOS ESCOLARES

Dados Pessoais

Matrícula



Nome

Data de nascimento

dd/mm/aaaa

Nacionalidade do aluno

☐ Brasileira ☐ Brasileira Nascido no Exterior ☐ Naturalizado ☐ Estrangeira

UF de nascimento

Selecione... ▼

Estado civil

Selecione... ▼

Profissão

Selecione... ▼

Religião

Selecione... ▼

Recebe escolarização em
outro espaço

Selecione... ▼

Aluno com deficiência, transtorno global do desenvolvimento ou altas habilidades/Superdotação ☐ Sim ☐ Não

Recomendações médicas

▲

▼

Escola Básica de _____

Ano Letivo _____ / _____

Quem sou eu?



Eu

Idade: ____ anos Data de nascimento: ____ / ____ / ____ Sexo: M ☐ F ☐

Filiação:

Naturalidade: _____ Nacionalidade: _____

Freguesia: _____ Concelho: _____ Distrito: _____

Residência:

Telefone: _____ Telemóvel: _____

FICHA DE DADOS PESSOAIS

IDENTIFICAÇÃO

Nome: _____

Filiação:

Nome do

Pai: _____

Nome da Mãe:

Sexo: ☐ Fem ☐ Masc Data Nascimento: ____ / ____ / ____ Nacionalidade : _____

Naturalidade: _____ UF: ____ Se Estrangeiro data de chegada ao Brasil: ____ / ____ / ____

Estado Civil: ☐ Solteiro ☐ Casado ☐ Separado Judicialmente ☐ Divorciado ☐ Viúvo

Se casado informar nome do

cônjuge: _____

Cor: ☐ Branca ☐ Preta ☐ Amarela ☐ Parda ☐ Indígena

Grupo Sangüíneo / Fator RH: _____ Deficiência: ☐ Física ☐ Sensorial e Auditiva ☐ Mental ☐ Múltiplas

Nível de Escolaridade : _____ Habilitação: _____

Pós-Graduação : _____ Área: _____

DOCUMENTAÇÃO

CPF : _____ Pis/Pasep : _____ Identidade : _____

Orgão Emissor: _____ UF: ____ Data Emissão: ____ / ____ / ____

Título Eleitor: _____ Zona: _____ Seção: _____ Data Emissão: ____ / ____ / ____

Certificado de Reservista Nº: _____ Série: _____ Órgão Expedidor: _____

Data do Primeiro Emprego com registro : ____ / ____ / ____

ENDEREÇO

Rua:

Número:

Complemento:

Estruturas Heterogêneas de Dados

- Podemos perceber pelos exemplos que, apesar dos dados serem diferentes, todos pertencem a uma única pessoa. Logo, formam uma única ficha.
- As estruturas facilitam manipular agrupamentos complexos de dados.
- Em um procedimento como ordenar as informações sobre os alunos de uma universidade, por exemplo. A ordenação pode ser efetuada como se todos os dados que compõem a estrutura fossem uma entidade única, utilizando um dos campos com chave de ordenação.

Estruturas Heterogêneas de Dados

- Uma estrutura, então, é apenas uma coleção de variáveis agrupadas sob um único nome.
- Os elementos da estrutura tem alguma relação semântica, por exemplo: alunos de uma universidade, discos de uma coleção, elementos de uma figura geométrica, etc.
- Como exemplo, criamos uma estrutura para representar um aluno e armazenar o seu nome, registro, ano de entrada e curso:

```
struct aluno {  
    char nome[40];  
    int registro;  
    int ano_entrada;  
    char curso[20];  
};
```


Estruturas Heterogêneas de Dados

- Sintaxe:

- Inicialmente precisamos criar a forma da estrutura.
- A palavra chave *struct* inicia essa declaração e, em seguida, pode aparecer um identificador, que poderá ser usado quando formos declarar variáveis que usam a mesma estrutura de dados.
- A declaração continua com a lista de declarações **entre chaves** e termina com um ponto-e-vírgula (;).
- Um item da estrutura e uma variável não membro da estrutura podem ter o mesmo nome, já que é possível distingui-las por contexto.

Estruturas Heterogêneas de Dados

- A declaração mostrada não alocou espaço de memória, já que nenhuma variável foi realmente definida.
- Esta declaração inicial cria apenas um modelo de como estruturas do tipo `aluno` devem ser construídas.
- Para criar variáveis do tipo `aluno` usamos a seguinte declaração:

```
struct aluno paulo, carlos, ana;
```
- Na declaração acima, três estruturas do tipo `aluno` foram criadas e foi alocado um espaço para armazenar os dados de cada um dos três alunos.

Estruturas Heterogêneas de Dados

- A declaração acima é idêntica, na forma, a declaração de variáveis de um tipo pré-definido, como por exemplo:

```
int a, b, c;
```

- A diferença é que o identificador de tipo agora é composto e formado pela palavra chave *struct*, seguida do nome usado na criação do modelo:

```
struct aluno paulo, carlos, ana;
```

Estruturas Heterogêneas de Dados

- É possível declarar ao mesmo tempo o modelo da estrutura e as variáveis do programa. Por exemplo,

```
struct aluno {  
    char nome[40];  
    int registro;  
    int ano_entrada;  
    char curso[20];  
} paulo, carlos, ana;
```

- Para referenciar um elemento da estrutura usa-se o nome da variável do tipo da estrutura seguida de um ponto e do nome do elemento. Por exemplo,

```
paulo.ano_entrada = 1999;
```

- Para ler o nome do curso que paulo cursa pode-se usar o comando:

```
gets(paulo.curso);
```

Estruturas Heterogêneas de Dados

- Estruturas podem conter outras estruturas como membros. Por exemplo:

```
struct data {  
    int dia, mes, ano;  
}  
struct aluno {  
    char nome[40];  
    int registro;  
    int ano_entrada;  
    char curso[20];  
    struct data data_nascimento;  
};
```

- Para se referir ao mês de nascimento da variável *Paulo*, usamos a sintaxe `paulo.data_nascimento.mes` onde o operador ponto(.) associa os campos hierarquicamente, da esquerda para a direita.

Matrizes de Estruturas

- Estruturas aparecem frequentemente na forma de matrizes, por exemplo:

```
struct aluno turma[100];
```

- Se quisermos imprimir o nome do terceiro aluno, por exemplo, devemos usar a sintaxe:

```
printf("%s\n", turma[2].nome);
```

```
#define MAX 4

void main () {
    struct aluno{
        char nome[40];
        float n1, n2, media;
    } turma[MAX], turma2[MAX];
    int i, j, pos;

    puts("Lendo dados da turma");
    for (i=0; i<MAX; i++) {
        printf("Dados do aluno %d\n", i);
        puts("Nome?"); gets(turma[i].nome);
        puts("Primeira nota?"); scanf("%f", &turma[i].n1);
        puts("Segunda nota?"); scanf("%f", &turma[i].n2);
        turma[i].media=(turma[i].n1+turma[i].n2)/2.0;
    }
}
```


Passagem de Estruturas para Funções

- Para passar elementos da estrutura para funções, usamos a maneira normal, por exemplo:

```
struct ponto {  
    float x, y;  
} p1, p2;
```

- Para passar a coordenada x do ponto $p1$ para a função *positivo* usamos a seguinte declaração:

```
if ( positivo(p1.x) == 0 )  
    puts("Eixo y");  
else if ( positivo(p1.x>0) )  
    puts("Eixo positivo dos x");  
else  
    puts("Eixo negativo dos x");
```

Estruturas Heterogêneas de Dados

- Caso seja necessário passar o endereço de um dos elementos basta colocar o operador & antes do nome da **estrutura**, por exemplo:

```
troca_x (&p1.x, &p2.x);
```

- Também é possível passar uma estrutura inteira para uma função e, ao fazer isso, estaremos passando todos os valores armazenados nos membros da estrutura.
- Como este tipo de passagem é feito por valor, alterações nos membros da estrutura não modificam os valores da estrutura na função que chamou.

Ponteiros para Estruturas

- Se for necessário passar toda a estrutura por referência, precisamos tomar cuidado com o tipo do dado que receberá o endereço passado => precisa ser um ponteiro para a estrutura, como no exemplo:

```
void imprime_data(struct data * d1)
```

- Onde a chamada da função teria a sintaxe:

```
imprime_data(&dt);
```

- Para acessar elementos da estrutura apontada por um ponteiro usa-se o chamado operador seta (->). Por exemplo para imprimir a média da aluna maria usaríamos o comando:

```
printf ("A media vale %.1f", maria->media);
```

Exemplos de Ponteiros para Estruturas

```
void imprime_data(struct data * d1) {  
    printf("\ndia = %d\n mes = %d\n ano = %d", d1->dia,  
        d1->mes, d1->ano);  
}
```

```
void imprime_data2(struct data d1) {  
    printf("\n\ndia = %d\n mes = %d\n ano = %d", d1.dia,  
        d1.mes, d1.ano);  
}
```

```
void imprime_data3(struct data * d1) {  
    printf("\n\ndia = %d\n mes = %d\n ano = %d",  
        (*d1).dia, (*d1).mes, (*d1).ano);  
}
```

Ponteiros para Estruturas

```
main () {  
    struct data *dt, dt2;  
  
    ...  
  
    imprime_data(dt);  
    imprime_data2(*dt);  
    imprime_data3(dt);  
  
    imprime_data(&dt2);  
    imprime_data2(dt2);  
    imprime_data3(&dt2);  
    getch();  
}
```

Ponteiros para Estruturas

- Para acessar os elementos de uma estrutura apontada por um ponteiro usamos o operador seta (->). Do exemplo anterior:

```
void imprime_data(struct data * d1) {  
    printf("\ndia = %d\n mes = %d\n ano = %d", d1->dia,  
        d1->mes, d1->ano);  
}
```

- Alternativamente, poderíamos usar os operadores * e ponto:

```
void imprime_data3(struct data * d1) {  
    printf("\n\ndia = %d\n mes = %d\n ano = %d",  
        (*d1).dia, (*d1).mes, (*d1).ano);  
}
```

Declarando um Tipo Struct

- Podemos criar novos tipos de dados usando a diretiva typedef, usando a sintaxe:

```
typedef tipo var;
```

- Essa declaração faz com que var passe a ser o nome de um novo tipo. Exemplo:

```
typedef struct {  
    int x;  
    int y;  
} ponto;
```

- Essa declaração cria o novo tipo, chamado ponto, que pode ser usado para declarar novos pontos, por exemplo:

```
ponto a, b;
```

Declarando um Tipo Struct

- Se usarmos typedef no exemplo anteriormente mostrado, teremos de:

```
struct data {  
    int dia, mes, ano;  
};
```

- Para:

```
typedef struct {  
    int dia, mes, ano;  
} tpData ;
```

- Com essa alteração, deixamos o código mais legível, como mostrado a seguir.

Declarando um Tipo Struct

```
typedef struct {  
    int dia, mes, ano;  
} tpData ;
```

```
void imprime_data(tpData * d1) {  
    printf("\ndia = %d\n mes = %d\n ano = %d", d1->dia,  
        d1->mes, d1->ano);  
}
```

```
void imprime_data2(tpData d1) {  
    printf("\n\ndia = %d\n mês = %d\n ano = %d",  
        d1.dia, d1.mes, d1.ano);  
}
```

Declarando um Tipo Struct

```
main () {  
    tpData *dt, dt2;  
  
    ...  
  
    imprime_data(dt);  
    imprime_data2(*dt);  
    imprime_data3(dt);  
  
    imprime_data(&dt2);  
    imprime_data2(dt2);  
    imprime_data3(&dt2);  
    getch();  
}
```