# LINGUAGEM DE PROGRAMAÇÃO II

# 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 váriavel 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.

# Estruturas Heterogêneas de Dados

- É possível atribuir o conteúdo de uma estrutura a outra estrutura do mesmo tipo, não sendo necessário atribuir elemento por elemento da estrutura. Exemplo:

```
#include<stdio.h>
void main () {
    struct empregado {
        char nome[40];
        float salario;
    } temp, emp1;
    puts("Entre com nome."); gets(emp1.nome);
    puts("Qual o salario?"); scanf("%f", &emp1.salario);
     temp=emp1;
    printf("O salario de %s e %.2f\n", temp.nome,
                                        temp.salario);
}
```

# 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.

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

float comp(struct ponto p1, struct ponto p2){
      return sqrt (pow(p2.x-p1.x,2)+pow(p2.y-p1.y,2));
}

void main () {
     struct ponto P1, P2;
     puts ("Coordenadas do ponto 1");
     printf("x1 = ? "); scanf("%f", &P1.x);
     printf("y1 = ? "); scanf("%f", &P1.y);
     puts ("Coordenadas do ponto 2");
     printf("x1 = ? "); scanf("%f", &P2.x);
     printf("y1 = ? "); scanf("%f", &P2.y);
     printf("\nComprimento da reta = %.2f", comp(P1,P2));
}
```

# Ponteiros para Estruturas

- Para definir ponteiros para estruturas a declaração é similar a declaração de um ponteiro normal, exemplo:

```
struct aluno {
    char nome[40];
    int ano_entrada;
    float n1, n2, media;
} *maria;
```

- 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);
```

# Ponteiros para Estruturas

- Quando trabalhamos com variáveis de um tipo básico, podemos associar seu endereço a um ponteiro diretamente.

- Porém, ao fazermos essa associação entre um ponteiro e o endereço de uma estrutura precisamos alocar um espaço de memória, de forma explícita, para armazenar esse tipo de ponteiro.

- Isto ocorre porque o tamanho total para armazenar uma estrutura na memória corresponde à soma dos tamanhos dos seu componentes, então precisaremos realizar essa conta, usando as funções malloc() e sizeof().

# Ponteiros para Estruturas

- Sintaxe:

```
var_pt = (struct var_pt*) malloc(sizeof(struct var_pt));
```

Onde o trecho (struct var_pt*) é usado para realizar uma conversão de tipo (cast).

- Exemplo:

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

struct data * cria_data (int d, int m, int a) {
    struct data *dt;
    dt = (struct data*) malloc(sizeof(struct data));
    dt->dia = d;
    dt->mes = m;
    dt->ano = a;
    return dt;
}
```

# 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;

     dt = cria_data(25,12,2016);

     imprime_data(dt);
     imprime_data2(*dt);
     imprime_data3(dt);

     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 ;

tpData * cria_data (int d, int m, int a) {
    tpData *dt;
    dt = (tpData*) malloc(sizeof(tpData));
    dt->dia = d;
    dt->mes = m;
    dt->ano = a;
    return dt;
}
```

# Declarando um Tipo Struct

```
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 mes = %d\n ano = %d",d1.dia,
       d1.mes, d1.ano);
}

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

# Declarando um Tipo Struct

```
main () {
    tpData * dt;

    dt = cria_data(25,12,2016);

    imprime_data(dt);
    imprime_data2(*dt);
    imprime_data3(dt);

    getch();
}
```