

# ESTRUTURA DE DADOS

Prof. Nilton

# Aula de hoje

- Introdução a tipos abstratos de dados
- Introdução à Registros (Structs)
- Exemplos
- Exercícios

# Tipos abstratos de dados

Existe uma importante técnica de programação baseada na definição de **tipos estruturados**, conhecida como **tipos abstratos de dados (TAD)**, cujo a ideia central é encapsular (esconder) de quem usa um determinado tipo a forma concreta com que ele foi implementado. Por exemplo, se criarmos um **tipo** para representar um ponto no espaço, um cliente que utiliza esse **tipo** o faz de forma abstrata, com base apenas nas funcionalidades oferecidas sem ter acesso aos detalhes de implementação. Isso permite desacoplar a utilização da implementação, facilitando a manutenção e reutilização do **tipo**.

Um **TAD** é descrito pela finalidade do tipo e de suas operações, e não pela forma como está implementado.

# Tipos estruturados

Quando estudamos vetores, lidávamos com conjuntos de dados sempre do mesmo tipo. Ou seja, até agora todos os elementos de um vetor são inteiros, caracteres, reais, etc., mas sempre são do mesmo tipo.

Para desenvolver programas mais complexos precisamos trabalhar de uma maneira mais abstrata para representar os dados. Tomemos como exemplo uma aplicação que deve representar o cadastro dos alunos matriculados em uma determinada disciplina. Os dados associados a cada aluno são vários: nome, número de matrícula, notas, etc. e seus tipos são diferentes.

A linguagem C oferece mecanismos para estruturar dados complexos, nos quais as informações são compostas por diversos campos.

# Tipos estruturados – Registros e structs

Um registro (= record) é um pacote de variáveis, possivelmente de tipos diferentes. Cada variável é um campo do registro. Na linguagem C, registros são conhecidos como structs (o nome é uma abreviatura de structure). Exemplo:

```
struct etiqueta{
```

```
    tipo var1;
```

```
    tipo var2;
```

```
    ...
```

```
    tipo varN;
```

```
};
```

```
struct etiqueta var1, var2, ..., varN;
```

# Tipos estruturados – Registros e structs

Uma estrutura pode ser definida de formas diferentes. No corpo da estrutura encontram-se os membros, ou seja, as variáveis de diversos tipos que compõem esse tipo de dado heterogêneo definido pelo usuário. Depois de definida uma estrutura, **uma (ou mais) variável do tipo estrutura deve ser definida, para permitir a manipulação dos membros da estrutura.**

# Registros e structs - Sintaxe

Neste exemplo o nome da estrutura é colocado logo em seguida da palavra-chave Struct. A Etiqueta da estrutura é nomeada como ALUNO. Ao final da estrutura, são declaradas várias variáveis do tipo da estrutura, isto é, aluno\_especial, aluno\_regular e aluno\_ouvinte, são variáveis do tipo aluno.

```
struct aluno {  
    int codigo;  
    char nome[200];  
    float nota;  
};  
struct aluno aluno_especial, aluno_regular, aluno_ouvinte;
```

# Registros e structs - Sintaxe

Neste exemplo o nome da estrutura é colocado ao final, logo em seguida ao fechamento da estrutura, isto é, a Etiqueta da estrutura é nomeada depois. Após a declaração da estrutura, são definidas as várias variáveis do tipo da estrutura.

```
struct {  
    int codigo;  
    char nome[200];  
    float nota;  
} aluno, aluno_especial, aluno_regular, aluno_ouvinte;
```



# Registros e structs - Sintaxe

Neste exemplo, usei a palavra-chave **Typedef** para definir uma estrutura, antes da palavra-chave **Struct**. O nome da estrutura é colocado ao final, logo em seguida ao fechamento da estrutura. Após a declaração são definidas várias variáveis do tipo da estrutura.

```
typedef struct {  
    int codigo;  
    char nome[200];  
    float nota;  
} aluno;  
  
aluno aluno_especial, aluno_regular, aluno_ouvinte;
```

# Registros e structs – acesso aos atributos

Para acessar os membros da estrutura, quando ela é diretamente referenciada, devemos utilizar o Ponto, que também é chamado de operador de seleção direta, veja:

`aluno_especial.codigo`

`aluno_especial.nome`

`aluno_especial.nota`

# Registros e structs – atribuindo valores

Você pode atribuir valores aos membros das estruturas diretamente, e em qualquer parte do programa, conforme a seguir:

```
aluno_especial.codigo = 10;
```

```
aluno_especial.nome = "Manoel";
```

```
aluno_especial.nota = 10.0;
```

# Registros e structs – visualizando valores

Você pode imprimir os membros da estrutura em qualquer parte do programa que desejar. Se for preciso imprimir todos os membros da Estrutura de uma única vez, então é melhor criar uma função para isto.

```
cout << aluno_especial.codigo
```

```
cout << aluno_especial.nome
```

```
cout << aluno_especial.nota
```

# Exemplo

Vamos desenvolver um programa usando uma Struct e o conceito de modularização através funções para poder reutilizar trechos de códigos em outras partes do programa.

```
typedef struct Aluno
```

```
{
```

```
    int codigo;
```

```
    char nome[200];
```

```
    float nota;
```

```
};
```

```
Aluno aluno_especial;
```

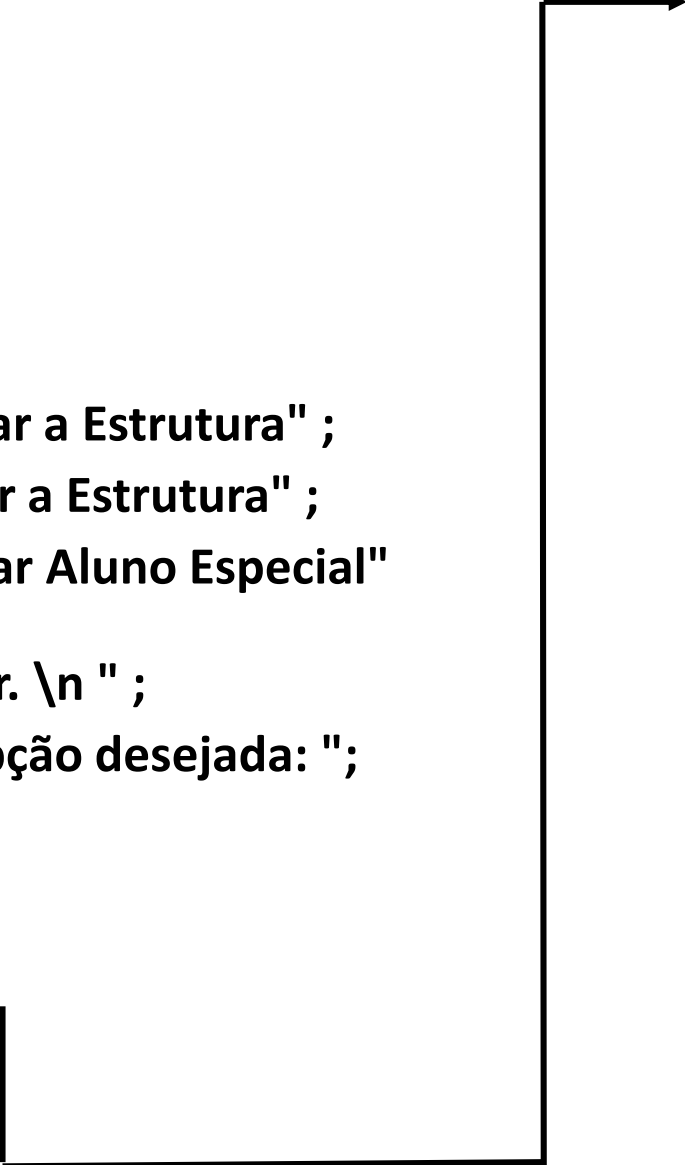
# Exemplo

```
void inicializar();  
void imprimir();  
void cadastrar();  
void menu();
```

```
int main()  
{  
    setlocale(LC_ALL, "portuguese");  
    menu();  
    return 0;  
}
```

```
void menu()
{
    int opcao;
    do
    {
        cout << " \n Opções: ";
        cout << " \n 1. Inicializar a Estrutura" ;
        cout << " \n 2. Imprimir a Estrutura" ;
        cout << " \n 3. Cadastrar Aluno Especial"
;

        cout << " \n 4. Para sair. \n " ;
        cout << " \n Digite a opção desejada: ";
        cin >> opcao;
    }
```



```
switch(opcao)
{
    case 1:
        inicializar();
        break;
    case 2:
        imprimir();
        break;
    case 3:
        cadastrar();
        break;
    default:
        cout << " \n Opção não existente. ";
        break;
}
while(opcao!=4);
}
```

```
void inicializar()
{
    aluno_especial.codigo = 0;
    strcpy(aluno_especial.nome, "NULL");
    aluno_especial.nota = 0.0;
}
```

```
void cadastrar()
{
    cout << " Digite o código do aluno especial: "
    cin >> aluno_especial.codigo;
    cout << " Digite o nome do aluno especial: ";
    scanf("%256[^\n]", &aluno_especial.nome);
    fflush(stdin);
    cout << " Digite a nota do aluno especial: ";
    cin >> aluno_especial.nota;
}
```



```
void imprimir()
{
    cout << " \n O código do aluno especial é:" << aluno_especial.codigo;
    cout << " \n O nome do aluno especial é: " << aluno_especial.nome;
    cout << " \n A nota do aluno especial é: " << aluno_especial.nota;
    cout << " \n \n";
}
```

# Exercícios

- 1) Desenvolva um programa em C para vendas de passagens de ônibus. Defina uma STRUCT, crie o menu de opções e todas as funções necessárias para manipular esta STRUCT.

# Registros como campos de outros registros

Veremos situações em que os dados poderiam ser agregados de forma mais lógica, separada conforme sua natureza. Isso possibilita criarmos tipos genéricos que permitam a reutilização de informações. Exemplo: Endereço.

Esse tipo de estrutura fica mais limpa, pois deixa os tipos mais compactos e específicos e ainda facilita a leitura e interpretação de cada tipo.

# Registros como campos de outros registros

```
typedef struct Endereco{  
    char rua[40];  
    char numero[5];  
    char complemento[20];  
    char bairro[40];  
};  
Endereco endereco_pessoa;
```

# Registros como campos de outros registros

```
typedef struct Pessoa(  
    char nome[60];  
    Endereco endereco;  
    char sexo;  
    char telefone[20];  
    char email[256];  
);  
Pessoa pessoa;
```

# Exercícios

2) Imagine uma escola que quer fazer o cadastro de seus professores e funcionários. Crie um programa em C que permita cadastrar as informações básicas de cada tipo e use a generalização para separar as informações redundantes.

# Vetores como campo de registros

Podemos utilizar vetores para representar campos de um registro. Exemplo: armazenar as notas dos alunos.

```
typedef struct Aluno{
```

```
    int ra;
```

```
    string nome;
```

```
    float notas[10];
```

```
};
```

```
Aluno alunos;
```

```
alunos.notas[0] = 10;
```

# Vetores de registros

Os vetores de registros visam a armazenar conjuntos de elementos complexos. Por exemplo, em vez de armazenar apenas as notas dos alunos de uma turma, os vetores de registros podem armazenar as informações cadastrais de todos os alunos de uma turma.

```
Aluno vetAlunos[10];  
vetAlunos[0].ra = 1;  
vetAlunos[0].nome = "José";  
vetAlunos[0].notas[0] = 7;
```



Obrigado!!!