BCC 201 - Introdução à Programação I

# Estruturas II

Guillermo Cámara-Chávez
UFOP

# Conceito de *struct* I

- Vetores e matrizes
  - Estruturas de dados **homegêneas**
  - Armazenam vários valores, mas todos de um mesmo tipo (todos *int*, todos *double*, todos *float*

# Conceito de *struct* II

- Problemas reais
  - Temos coleções de dados que são de **tipos diferentes**
  - Exemplo: ficha de um cadastro de cliente
    - Nome: *string*
    - Endereço: *string*
    - Telefone: *string*
    - Salário: *float*
    - Idade: *int*

# Conceito de *struct* III

- Registro (ou *struct*)
  - Tipo de dado estruturado heterogêneo
    - Coleção de variáveis referenciadas sobre um mesmo nome
  - Permite agrupar dados de diferentes tipos numa mesma estrutura (ao contrário de matrizes que possuem elementos de um mesmo tipo)
    - Cada componente de um registro pode ser de um tipo diferente
    - Estes componentes são referenciados por um nome

# Conceito de *struct* IV

- Os elementos do registro
  - São chamados de campos ou membros da *struct*
- É utilizado para armazenar informações de um mesmo objeto
- Exemplos:
  - carro $\rightarrow$ cor, marca, ano, placa
  - pessoa $\rightarrow$ nome, idade, endereço

# Conceito de *struct* V

- Campo (*Field*)
  - Conjunto de caracteres com o mesmo significado
  - Exemplo: nome
- Registro (*struct* ou *record*)
  - Conjunto de campos relacionados
  - Exemplo: nome, endereço, telefone, salários e idade de uma pessoa

# Sintaxe na Linguagem C/C++ I

- A palavra reservada *struct* indica ao compilador que está sendo criada uma estrutura
- Uma estrutura deve ser declarada após incluir as bibliotecas e antes do *main*

```
struct <identicador_struct>
{
    tipo <nome_variável_campo1>;
    tipo <nome_variável_campo2>;
    ...
}<variáveis_estrutura>;

struct <identificador_struct> <var1>, <var2>;
```

# Sintaxe na Linguagem C/C++ II

- Se o compilador C for compatível com o padrão C ANSI
  - Informação contida em uma *struct* pode ser atribuída a outra *struct* do mesmo tipo
  - Não é necessário atribuir os valores de todos os elementos/campos separadamente
  - Por exemplo: `<var1> = <var2>;`
    - Todos os campos de `<var1>` receberão os valores correspondentes dos campos de `<var2>`

# Sintaxe na Linguagem C/C++ III

- Para acessar os campos da *struct*
  - Utiliza-se o nome da variável *struct*, seguido de ponto, seguido do nome do campo
  - Por exemplo: `<var1>.<nome_campo>`

# Sintaxe na Linguagem C/C++ IV

- Por exemplo um *struct* endereço que guarda os elementos nome, rua, cidade, estado e cep

```
struct endereco
{
    string nome;
    string rua;
    ...
    long int cep;
};
```

  - Foi feita apenas a declaração da *struct*, ainda não foi criada nenhuma variável da *struct* endereço
  - o comando para declarar uma variável com esta *struct* é: `struct endereco info_end;`

# Sintaxe na Linguagem C/C++ V

- Já vimos que para acessar os membros de uma *struct* deve-se usar `nome_variável.nome_membro`
- Portanto, considerando o exemplo anterior
  - Para inicializar o cep da variável `info_end` que é uma variável da *struct* endereço se faria:
    `info_end.cep = 123456;`
  - Para obter o nome da pessoa e colocar na *string* nome da *struct* se poderia utilizar:
    `getline(cin, info_end.nome);`
  - Para imprimir o endereço seria:
    `cout << info_end.rua;`

# Vetor de *struct* I

- O uso mais comum de *struct* é em vetores
- Para declarar um vetor de *struct*
  - Define-se a *struct*
  - Define-se o vetor do tipo *struct* criado
- Exemplo:

```
struct aluno Turma31[28];
struct endereco vetorEndAmigos[100];
```

# Vetor de *struct* II

Crie um programa que permita armazenar o nome e da data de nascimento de até 10 pessoas. Cada pessoa deve ser representada por uma *struct* dentro de um vetor. A data de nascimento também deve ser uma *struct*. O nome de cada pessoa deve ser informado pelo teclado.
A geração da data de nascimento deve ser feita aleatoriamente através da seguinte função:

```
TData CriaData()
{
    TData D;
    D.Mes = 1 + (rand() % 12);
    D.Ano = 1950 + (rand() % 49);
    D.Dia = 1 + (rand() % 30);
    return D;
}
```

# Vetor de *struct* III

Criar as seguintes funções:

1. inserir dados;
2. listar todos os nomes e respectivas idades;
3. listar os nomes das pessoas mais velhas do que uma certa idade fornecida.

# Vetor de *struct* IV

```
typedef struct Data{
    int Dia, Mes, Ano;
}TData;

typedef struct Aluno{
    string nome;
    TData nasc;
}TAluno;

TData CriaData();
void InserirDados(TAluno V[], int n);
void ImprimeIdades(TAluno V[], int n);
void ImprimeMaisVelhos(TAluno V[], int n, int idade);
```

# Vetor de *struct* V

```
int main(){
    TAluno A[4];
    InserirDados(A, 4);
    ImprimeDados(A, 4);
    return 0;
}

TData CriaData(){
    TData D;
    D.Mes = 1 + (rand() % 12);
    D.Ano = 1950 + (rand() % 49);
    D.Dia = 1 + (rand() % 30);
    return D;
}
```

# Vetor de *struct* VI

```
void InserirDados(TAluno V[], int n){
    int i;
    for (i = 0; i < n; i++){
        cout << "\n Inserir Nome: ";
        getline(cin, V[i].nome);
        V[i].nasc = CriaData();
    }
}

void ImprimeIdades(TAluno V[], int n){
    int i;
    for (i = 0; i < n; i++)
        cout << V[i].nome << " "
             << 2015-V[i].nasc.Ano);
}
```

# Vetor de *struct* VII

```
void ImprimeMaisVelhos(TAluno V[], int n, int idade)
{
    int i;
    for (i = 0; i < n; i++)
        if ( (2015-V[i].nasc.Ano) > idade)
            cout << V[i].nome << endl;
}
```

# Vetor de *struct* VIII

Seja uma estrutura para descrever os carros de uma determinada revendedora, contendo os seguintes campos:

- marca: string
- ano: inteiro
- cor: string
- preço: real

# Vetor de *struct* IX

Criar as seguintes funções:

1. Definir um procedimentos para ler o vetor `vetcarros`.
2. Definir um procedimento que receba um preço e imprima os carros (marca, cor e ano) que tenham preço igual ou menor ao preço recebido.
3. Defina um procedimento que leia a marca de um carro e imprima as informações de todos os carros dessa marca (preço, ano e cor).
4. Defina um procedimentos que leia uma marca, ano e cor e informe se existe ou não um carro com essas características. Se existir, informar o preço.

# Vetor de *struct* X

```
typedef struct Carro{
    string marca;
    int ano;
    string cor;
    double preco;
}TCarro;

void Insere(TCarro V[], int n);
void MostraPreco(TCarro V[], int n, double valor);
void MostraMarca(TCarro V[], int n, string marca);
void MostraVarios(TCarro V[], int n, string marca,
                  int ano, string cor);
```

# Vetor de *struct* XI

```
int main()
{
    TCarro V[4];
    Insere(V, 4);
    cout << "\n Mostrar por preco: \n";
    MostraPreco(V, 4, 20000);
    cout << "\n Mostrar por marca: \n";
    MostraMarca(V, 4, "fiat");
    cout << "\n Mostrar varios: \n";
    MostraVarios(V, 4, "fiat", 2005, "prata");
    return 0;
}
```

# Vetor de *struct* XII

```
void Insere(TCarro V[], int n){
    int i;
    for (i = 0; i < n; i++){
        cout << "\n Digite Marca Ano Cor Preco";
        cin >> V[i].marca >> V[i].ano
            >> V[i].cor >> V[i].preco;
    }
}

void MostraPreco(TCarro V[], int n, double valor){
    int i;
    for (i = 0; i < n; i++){
        if (V[i].preco <= valor)
            cout << V[i].marca << " "
                 << V[i].cor << " " << V[i].ano;
    }
}
```

# Vetor de *struct* XIII

```
void MostraMarca(TCarro V[], int n, string marca){
    int i;
    for (i = 0; i < n; i++){
        if ( V[i].marca == marca )
            cout << V[i].preco << V[i].ano
                 << V[i].cor;
    }
}

void MostraVarios(TCarro V[], int n, string marca,
                  int ano, string cor)
{
    int i;
    for (i = 0; i < n; i++){
        if ( V[i].marca == marca &&
             V[i].ano == ano &&
             V[i].cor ==  cor )
            cout << V[i].preco;
    }
}
```

# Vetor de *struct* XIV

Seja um algoritmo para controlar os produtos do estoque de um supermercado. Para cada produto, tem-se os seguintes campos:

- nome: string
- setor: caracter
- quantidade: inteiro
- preço: real //preço por unidade do produto

# Vetor de *struct* XV

Crie um menu para:

1. Definir um bloco de instruções para inserir produtos
2. Definir um bloco de instruções para ler o vetor estoque.
3. Definir um bloco de instruções que receba um setor e devolva o número de diferentes produtos desse setor.
4. Definir um bloco de instruções que calcule e devolva o total de capital investido em produtos do supermercado.
5. Sair do Programa.

# Vetor de *struct* XVI

```
#define N 1000
typedef struct Produto
{
    string nome;
    char setor;
    int quantidade;
    double preco;
}TProduto;

int Menu();
int Inserir(TProduto* , int);
void LerEstoque(TProduto* , int);
int LerSetor(TProduto* , int);
double Capital(TProduto* , int);
```

# Vetor de *struct* XVII

```
int main()
{
    TProduto V[N]; int numprod = 0, tot;
    double valor; int op;
    do{
        op = Menu();
        switch (op){
        case 1: numprod = Inserir(V, numprod); break;
        case 2: LerEstoque(V, numprod); break;
        case 3: tot = LerSetor(V, numprod); break;
        case 4: valor = Capital(V, numprod);
            cout << "\n Total Capital:" << valor; break;
        case 5: cout << "\n Programa Terminado!"; break;
        default: cout << "Opcao invalida \n";
        }
    } while (op != 5);
    return 0;
}
```

# Vetor de *struct* XVIII

```
int Menu()
{
    int op;
    cout << "\n 1. Inserir dados";
    cout << "\n 2. Ler estoque ";
    cout << "\n 3. Produtos por setor ";
    cout << "\n 4. Total capital ";
    cout << "\n 5. Sair \n";
    cin >> op;
    cin.ignore();
    return op;
}
```

# Vetor de *struct* XIX

```
int Inserir(TProduto* V, int prodLocal)
{
    char resp;
    if (prodLocal < N){
        do{
            cout << "\n Inserir nome do produto: ";
            getline(cin, V[prodLocal].nome);
            cout << "\n Inserir setor, quant. e preco : \n";
            cin >> V[prodLocal].setor
                >> V[prodLocal].quantidade
                >> V[prodLocal].preco; cin.ignore();
            cout << "Continua Inserindo S/N \n";
            cin >> resp; cin.ignore();
            prodLocal++;
        } while (resp == 's' || resp == 'S');
    }
    else
        cout << "Nao eh possivel inserir mais produtos\n";
    return prodLocal;
}
```

# Vetor de *struct* XX

```
void LerEstoque(TProduto *V, int prodLocal)
{
    int i;
    if (prodLocal < N)
    {
        for (i = 0; i < prodLocal; i++)
            cout << V[i].nome << " "
            << V[i].setor << " "
            << V[i].quantidade << " "
            << V[i].preco << endl;
    }
}
```

# Vetor de *struct* XXI

```
int LerSetor(TProduto* V, int prodLocal)
{
    char setor;
    int i, cont = 0;
    if (prodLocal < N)
    {
        cout << "\n Inserir setor: ";
        cin >> setor;
        for (i = 0; i < prodLocal; i++)
            if (setor == V[i].setor)
                cont += V[i].quantidade;
        cout << "\n Produtos no setor "
             << setor << " = " << cont << endl;
    }
return cont;
}
```

# Vetor de *struct* XXII

```
double Capital(TProduto* V, int prodLocal)
{
    int i;
    double total = 0;
    if (prodLocal < N)
    {
        for (i = 0; i < prodLocal; i++)
            total += (V[i].preco*V[i].quantidade);
    }
    return total;
}
```

# Exercícios propostos I

Construa um programa para cadastro de veículos. Os dados que deverão ser armazenados sobre veículos são: marca, modelo, ano de fabricação, cor e placa. Use a estrutura conhecida como registro para compor os dados sobre o veículo. O programa deve ser capaz de armazenar os dados de 50 veículos. Para isto, use a etsrutura registro combinada com um vetor. O programa deverá permitir a entrada de quantos veículos o usuário quiser cadastrar. Crie uma opção para o usuário visualizar os veículos cadastrados.

FIM