ESTRUTURA DE DADOS

AULA 03 – ESTRUTURA DE DADOS HETEROGÊNEA

(STRUCT)

STRUCT>> CONTEXTO

✓**Ao manusear dados no mundo real muitas vezes precisamos representar as informações que não são fáceis de armazenar em variáveis de um único tipo, pois são na verdade um conjunto de elementos com características diferentes, ou seja, que possuem tipo de dados diferentes.**

✓**Ex1.:** ***o conjunto de informações que caracterizam um aluno: Nome, CPF, RG, data de nascimento, coeficiente de rendimento, etc.. O aluno seria a estrutura.***

✓**Ex2.:** ***o funcionamento de uma ficha pessoal que tenha nome, telefone, endereço, data de nascimento etc. A ficha seria uma estrutura.***

VARIÁVEIS DE TIPOS DIFERENTES

```
char nome[30];
int CPF, RG;
char data_nasc[10];
float coeficiente_rend;
```

Solução simples.

E se forem 50 alunos?

SE FOREM 50 ALUNOS: SOLUÇÕES...

- **Utilizar apenas uma variável de cada tipo e sobrescreve-las 50 vezes**

```
char nome[30];
int CPF, RG;
char data_nasc[10];
float coeficiente_rend;
```

- **Declarar 250 variáveis (sendo 50 de cada tipo) e cada uma delas com nomes diferentes.**

```
char nome1[30], nome2[30], /*...*/ nome50[30];
int CPF1,CPF2, /*...*/ CPF50;  RG1, RG2,/*...*/, RG50;
char data_nasc1[10], data_nasc2[10], /*...*/ data_nasc50[10];
float coeficiente_rend1,coeficiente_rend2, /*...*/ coeficiente_rend50;
```

- **Declarar cinco vetores (um para cada tipo).**

```
char nome[50][30];
int CPF[50], RG[50];
char datas_nasc[50][10];
float coeficiente_rend[50];
```

STRUCT: CARACTERÍSTICAS

- Conhecida como registro (*record*);
- Serve para agrupar um conjunto de dados não similares;
- Esses tipos de dados podem ser primitivos ou derivados;
- Permite a criação de novos tipos de dados (tipo de dados derivados), ao agrupar em um só elemento os diferentes tipos;
- Definições:
  - É um conjunto de dados formado por tipos de dados diferentes (mas não só), chamados campos do registro, em um mesmo elemento.
  - É uma coleção de uma ou mais variáveis, que podem ser de tipos diferentes, agrupadas sob um único nome/identificador.

STRUCT: CRIAÇÃO

- As *structs* permitem uma organização dos dados divididos em membros (campos) em uma estrutura de registros.
- As *structs* podem ser definidas:
  - dentro de funções, neste caso, serão apenas internas a estas funções, ou seja, são locais;
  - Fora do *main()*, ou seja, globalmente;

```
struct <etiqueta> {
    <tipo> campo_1;
    <tipo> campo_2;
    ...
    <tipo> campo_n;
} <variáveis>;
```

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

STRUCT: SINTAXE DA DEFINIÇÃO/CRIAÇÃO

```
struct nome_do_tipo_da_estrutura
        {
        tipo_1 nome_1;
        tipo_2 nome_2;
        ...
        tipo_n nome_n;
        } ;
```

```
struct data
{
        int dia;
        int mes;
        int ano;
};
...
struct data x;
```

Dois comandos

- **define a estrutura como um novo tipo;**
- **declara uma variável do novo tipo definido.**

```
struct nome_do_tipo_da_estrutura
        {
        tipo_1 nome_1;
        tipo_2 nome_2;
        ...
        tipo_n nome_n;
        } variáveis_estrutura;
```

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

Um comando

- **define a estrutura e declara uma variável do novo tipo definido.**

STRUCT: CARACTERÍSTICAS

- As variáveis contidas em uma estrutura denomina-se campos (ou membros).
- Campos de uma mesma estrutura devem ter nomes diferentes.
- Estruturas diferentes podem conter campos com o mesmo nome.
- A definição de uma estrutura:
  - não reserva qualquer espaço em memória;
  - cria um novo tipo de dados, que pode ser usado para declarar variáveis.
  - pode ser feita dentro da função principal (*main*) ou fora dela.
- Os campos de uma estrutura podem ser de qualquer tipo, inclusive uma estrutura previamente definida ou da própria estrutura.

TYPEDEF

- A linguagem C, permite, através do comando ***typedef*** **definir um novo nome para um determinado *tipo*.**
- **Sua forma geral é:**

```
typedef antigo_nome   novo_nome;
```

- **Ex:** **para dar um novo nome ao tipo inteiro, basta fazer:**

```
typedef int inteiro;
```

- **Deste ponto em diante, pode-se declarar qualquer variável como sendo tipo inteiro ao invés de *int*.**

TYPEDEF

- O comando *typedef* também pode ser utilizado para dar **nome a tipos complexos (apelidos)**, como as estruturas.

- **Sintaxe** do *typedef* para **renomear** ***struct >>>***

- **Ex.:** de utilização do *typedef* com ***struct >>>***

```
typedef struct nome_da_estrutura
{
    <tipo> campo_1;
    <tipo> campo_2;
    ...
    <tipo> campo_n;
} nome_do_tipo;
```

```
typedef struct data
{
    int dia;
    int mes;
    int ano;
} tipoData;
```

STRUCT

Qual a vantagem de fazer isto?

- **É que não será mais necessário usar a palavra chave *struct* para declarar variáveis do tipo ficha pessoal. Basta agora usar o novo tipo definido *TFicha*.**
- **No exemplo abaixo, para *TFicha* que é um novo tipo de dado. Para a criação deste novo tipo foi utilizado o *typedef*, pois não foi indicado o *token struct* na declaração da variável, por exemplo:**

```
main()
{
    TFicha  ex;
    ...
}
```

STRUCT + TYPEDEF>> CRIAÇÃO

- **O *typedef* com *struct* evita que a palavra-chave *struct* tenha que ser colocada toda vez que uma estrutura é definida.**
- **Ex.:**

```
typedef struct data
{
    int dia;
    int mes;
    int ano;
} tipoData;
```

```
tipoData dia_de_hoje;
```

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

```
struct data dia_de_hoje;
```

TYPEDEF E STRUCT

```
typedef struct {
        tipo_1 nome_1;
        tipo_2 nome_2;
        ...
        tipo_n nome_n;
        } nome_do_tipo_da_estrutura ;
```

Uma variação

- **Utilização do *typedef* para definir o nome do novo tipo.**

```
typedef struct
        {
                int dia;
                int mes;
                int ano;
        }tdata;
        ...
        tdata x;
```

STRUCT: ACESSANDO OS CAMPOS...

- Para acessar os campos de um estrutura individualmente ou realizar qualquer manipulação com o seu conteúdo é necessário entender que cada campo tem um identificador que deve estar associado à variável definida como estrutura.
- Sintaxe para acessar e manipular campos:

```
<nome_da_variável>.<campo>
```

- Ex:

```
printf ("Digite o nome do aluno: ");
scanf ("%s", &aluno.nome);

printf ("Digite a idade do aluno: ");
scanf ("%d", &aluno.idade);
```

Exemplo:

```
#include <stdio.h>
typedef struct tipo_endereco
{
        char rua [50];
        int numero;
        char bairro [20];
        char cidade [30];
        char sigla_estado [3];
        long int CEP;
} TEndereco;

typedef struct ficha_pessoal
{
        char nome [50];
        long int telefone;
        TEndereco endereco;
}TFicha;
```

Exemplo

```
#include <stdio.h>
#include <string.h>
struct tipo_endereco
{
        char rua [50];
        int numero;
        char bairro [20];
        char cidade [30];
        char sigla_estado [3];
        long int CEP;
};

struct ficha_pessoal
{
        char nome [50];
        long int telefone;
        struct tipo_endereco endereco;
};

main ()
{
        struct ficha_pessoal ficha;
        strcpy (ficha.nome,"Luiz Osvaldo Silva");
        ficha.telefone=4921234;
        strcpy (ficha.endereco.rua,"Rua das Flores");
        ficha.endereco.numero=10;
        strcpy (ficha.endereco.bairro,"Cidade Velha");
        strcpy (ficha.endereco.cidade,"Belo Horizonte");
        strcpy (ficha.endereco.sigla_estado,"MG");
        ficha.endereco.CEP=31340230;
}
```

Atribuição de structs do mesmo tipo

Se duas variáveis forem declaradas como estruturas do mesmo tipo, o C irá copiar uma estrutura, campo por campo, na outra, no caso de uma atribuição.

```
struct est1
{
    int i;
    float f;
};
void main()
{
    struct est1 primeira, segunda;
    primeira.i = 10;
    primeira.f = 3.1415;
    segunda = primeira;
   printf(" Os valores armazenados na segunda struct sao :  %d  e  %f
", segunda.i , segunda.f);
}
```

ANINHAMENTO DE STRUCTS

- **É possível também aninhar uma *struct* dentro de outra, isto é uma *struct* pode conter entre suas variáveis outra estrutura.**
- **O padrão ANSI C especifica que as estruturas podem ser aninhadas até 15 níveis, mas a maioria dos compiladores permite mais.**

PRIMEIROS PASSOS

1º :

Criar uma struct de 2 campos, um inteiro e um real;

2º :

Armazenar os valores que serão fornecidos pelo usuário;

3º :

Somar os valores e exibi-lo;

4º :

Exibir o maior número.

5º:

Verificar se o número 1 está presente na struct.

VETOR DE STRUCT

- A *struct* é um elemento versátil e pode ser combinada com outras estruturas, como vetores e matrizes.
- Ao combinar *structs* com *arrays* amplia-se a possibilidade de manipular um conjunto de elementos com tipos de dados diferentes.
- Uma variável que assume o tipo da *struct* pode ser também um vetor ou matriz.
- Neste caso, é necessário:
  - na declaração indicar o tamanho do *array* e associar a variável a um tipo *struct*;
  - na manipulação de os índices também deverão aparecer junto à variável indicando qual a posição do *array* que se deseja acessar naquele momento.

COMBINANDO STRUCT E ARRAY>> DECLARAÇÃO

- **A sintaxe da declaração é similar ao que já foi apresentado anteriormente.**
  - **Vetor de *struct*:**

    struct <nome_struct> <nome_variavel> [tamanho];

  - **Matriz de *struct*:**

    struct <nome_struct> <nome_variavel> [linha][coluna];

Atenção!

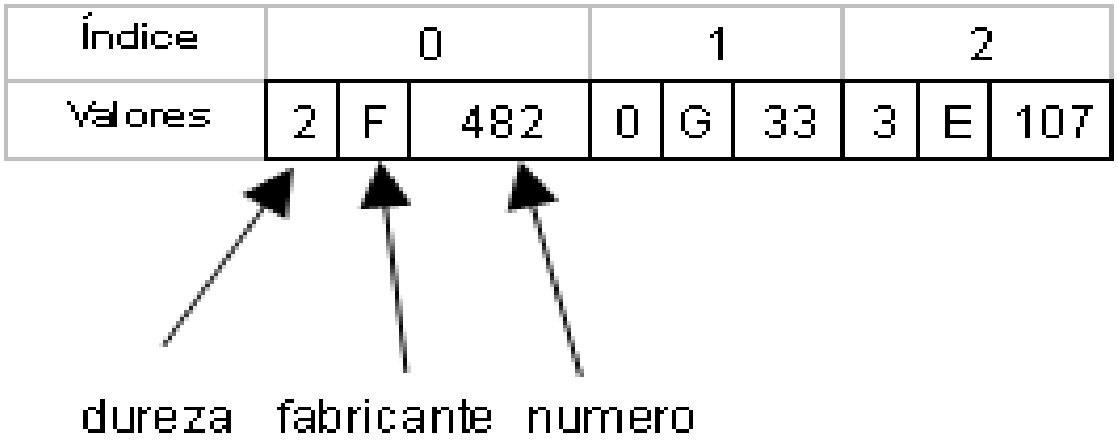
Lembre-se que a *struct* deve ser criada antes e que é um tipo de dado e não uma variável!

Exemplo:

```
struct endereco{
    char nome[30];
    char rua[40];
    char cidade[20];
    char estado[3];
    long int cep;
};
main()
{
struct endereco info_end[100];
:
// Imprime todos os nomes do vetor
for(int i = 0; i < 100; i++)
        printf("%s", info_end[i].nome);
}
```

```
#include  <stdio.h>;
#include <stdlib.h>

struct lapis{
int dureza;
char fabricante;
int numero;
};
int main(){
int i;
struct lapis p[3];
p[0].dureza=2;
p[0].fabricante='F';
p[0].numero=482;
p[1].dureza=0;
p[1].fabricante='G';
p[1].numero=33;
p[2].dureza=3;
p[2].fabricante='E';
p[2].numero=107;
printf("Dureza Fabricante Numero\n");
for(i=0; i<3; i++)
      printf("%d\t%c\t\t%d\n",p[i].dureza,p[i].fabricante,p[i].numero);
return 0;
system ("pause");
}
```




```
//declaração de variável p,
// como estrutura, sendo p um vetor
```

Exemplo

PRIMEIROS PASSOS

1º :

Criar um vetor de 10 posições do tipo struct que tem 2 campos, um inteiro e um real;

2º :

Armazenar os valores que serão fornecidos pelo usuário;

3º :

Somar os valores inteiros, somar os valores reais e exibi-los;

4º :

Exibir o maior número inteiro e o maior número real.

DESAFIOS...

1) Crie um programa para cadastrar o código de 20 municípios, sua população e sua área. Todos os dados devem ser informados pelo usuário. O programa deve:

a) Mostrar o maior município (considere que nenhum município tem área igual a outro);

b) Mostrar o código dos municípios que tem a maior população.

c) Calcular a média de todas populações cadastradas.

2) Faça um programa para cadastrar 100 produtos de uma loja com os seguintes dados: código do produto, quantidade mínima em estoque, quantidade atual em estoque e o preço. O programa deve:

DESAFIOS...

a) mostrar todos os dados dos produtos que contenham o estoque atual menor que o estoque mínimo para efetuar compra;

b) calcular o valor total dos produtos em estoque.

3) Faça um programa para um concurso de beleza onde precisa-se armazenar os dados das 30 candidatas, que são: número da inscrição, altura e peso. O programa deve:

a) Calcular e exibir a quantidade de candidatas que estão entre 1.70m e 1.80;

b) Mostrar a candidata mais magra e a mais gorda;

c) Calcular e exibir o peso médio das candidatas.

DESAFIOS...

4) Foi realizada uma pesquisa com 20 pessoas a respeito de salário, idade, número de filhos e sexo (M ou F). Faça um programa que:

a) receba os dados coletados na pesquisa;

b) mostra a média salarial de todos os pesquisados;

c) mostra a média das idades dos entrevistados;

d) mostra o número de mulheres cujo salário é maior R$ 4.500,00.

APÊNDICE: BIBLIOTECA <*string.h*>: algumas funções

- ***strcpy()*** **- copia a string-origem para a string-destino. Forma geral: *strcpy (string_destino, string_origem);***
- ***strcat()*** **- a string de origem permanecerá inalterada e será anexada ao fim da string de destino. Forma geral: *strcat (string_destino,string_origem);***
- ***strlen()*** **- retorna o comprimento da string fornecida. O terminador nulo não é contado. Isto quer dizer que, de fato, o comprimento do vetor da string deve ser um a mais que o inteiro retornado por *strlen()*. Formato geral: *strlen (string);***
- ***strcmp()*** **- compara a string 1 com a string 2. Se as duas forem idênticas a função retorna zero. Se elas forem diferentes a função retorna não-zero. Formato geral: *strcmp (string1,string2)*;**