ESTRUTURAS / REGISTROS (STRUCT)
(Lista 8)

Estruturas/Registros – é o tipo de dados definido pelo usuário que permite agregar varias vaiáveis de tipos de dados diferentes.

Cada unidade de informação contida em uma estrutura é chamada de campo.

Os campos podem ser de diferentes tipos primitivos, ou ainda, podem representar outras estruturas.

Desse modo, as estruturas (registros) são conhecidos como variáveis compostas heterogêneas.

Estruturas são usadas para armazenar um conjunto de informações sobre uma entidade (objeto).

Declaração de uma estrutura

Para criar um estrutura deve-se colocar uma palavra-chave **struct**.
Essa palavra-chave define um novo tipo de dados com mais de um campo.

Exemplo:

Formato geral	Exemplo

struct Nome
{
  member definition;
  member definition;
  ...
  member definition;
}
	struct Books
	{
	  char title[50];
	  char author[50];
	  int book_id;
	}

Nesse exemplo é criada uma estrutura chamada **Books** com 3 campos:

- title – é um vetor de 50 caracteres
- autor – é um vetor de 50 caracteres
- book_id – é um número inteiro

	Books
title	
author	
book_id	

Declaração de uma variável do tipo estrutura

A declaração de variáveis desse tipo pode ser feita logo depois da declaração de uma estrutura:

Exemplo:

```
struct Books
{
   char  title[50];
   char  author[50];
   int   book_id;
} book;
```

Nesse exemplo a declaração de uma variável **book** do tipo **Books** é feita logo depois da declaração da própria estrutura.

As variáveis também poderão ser declarados ao longo do programa usando a palavra-chave **struct** junto com o **nome** da estrutura previamente criada pelo usuário.

Exemplo:

```
struct Books Book1;
struct Books Book2;
```

Nesse exemplo são criadas (declaradas) duas variáveis **Book1** e **Book2** do tipo **Books**.

Acesso aos campos de uma estrutura

Para acessar os campos (ou membros) de uma estrutura é usado operador (**.**) que é colocado entre o nome da variável e nome do campo:

Exemplo:

```
Book1.book_id = 123;
```

Nesse exemplo é atribuído o valor 123 para campo **book_id** da variável **Book1**.

No próximo exemplo será usada uma funções especifica para processar as sequencias dos caracteres. Essas funções fazem parte da biblioteca **string.h** e algumas delas estão apresentados na tabela a seguir:

strcpy(s1, s2);	copia string **s2** em **s1**
strcat(s1, s2);	concatena (adiciona) string **s2** no final do string **s1**
strlen(s1);	retorna o tamanho do string s1
strcmp(s1, s2);	retorna 0 se **s1** é igual a **s2**; menor que 0 se **s1<s2**; maior que 0, se **s1>s2**
strchr(s1, ch);	retorna o ponteiro para primeira ocorrência do caractere **ch** em string **s1**
strstr(s1, s2);	retorna o ponteiro para primeira ocorrência da string **s1** em string **s1**

Exemplo 1 (1p): Criação de uma estrutura **Books**

```c
#include <stdio.h>
#include <string.h>

struct Books
{
  int book_id;
  char title[50];
  char author[50];
};
//=========================================================
int main( )
{
  struct Books Book1; /* Declare Book1 of type Book */
  struct Books Book2; /* Declare Book2 of type Book */

/* book 1 specification */
  strcpy( Book1.title, "Logica de Programação");
  strcpy( Book1.author, "FORBELLONE");
  Book1.book_id = 101;

/* book 2 specification */
  strcpy( Book2.title, "Treinamento em Linguagem C");
  strcpy( Book2.author, "MIZRAHI");
  Book2.book_id = 102;

/* print Book1 info */
  printf( "\n\n__________ Livro 1 (Book1) __________\n");
  printf( "Codigo (book_id) : %d\n", Book1.book_id);
  printf( "Titulo (title)   : %s\n", Book1.title);
  printf( "Autor  (author)  : %s\n", Book1.author);

/* print Book2 info */
  printf( "\n\n__________ Livro 2 (Book2) __________\n");
  printf( "Codigo (book_id) : %d\n", Book2.book_id);
  printf( "Titulo (title)   : %s\n", Book2.title);
  printf( "Autor  (author)  : %s\n", Book2.author);

  return 0;
}
```

```
__________ Livro 1 (Book1) __________
Codigo (book_id) : 101
Titulo (title)   : Logica de Programação
Autor  (author)  : FORBELLONE


__________ Livro 2 (Book2) __________
Codigo (book_id) : 102
Titulo (title)   : Treinamento em Linguagem C
Autor  (author)  : MIZRAHI
```

Declaração de um vetor

Em C/C++ é possível criar um vetor de estruturas. Neste caso cada elemento do vetor será uma estrutura.

O exemplo a seguir cria um vetor **Lib**, que contem 5 elementos de estrutura **Books**

```
const int lib_size = 5;
struct Books Lib[lib_size];
```

Lib

	0	1	2	3	4
title					
author					
book_id					

Acesso a um campo especifico de uma estrutura que faz parte de um vetor será feito dessa forma:

```
Lib[0].book_id = 100;
```

Estruturas e vetores como argumentos de uma função

Existem varias formas de passagem de vetores para uma função com e sem a utilização dos ponteiros.

A utilização dos ponteiros será explicada em detalhes mais adiante, por enquanto vamos considerar as formas que não exigem uso dos ponteiros.

Para passar um vetor de 10 elementos como argumento para função podemos declarar uma função como:

```
void myFunction ( int param[ ] )
{
 …
}
```

Quando uma função recebe vetor como argumento ela tem acesso direto aos elementos daquele vetor: ela pode modificar os valores do vetor.

A principio a função “não sabe” quantos elementos tem um vetor que ela recebeu como argumento. Essa informação deve ser passada para função como argumento:

```
void myFunction ( int param[], int arraySize )
{
 …
}
```

Exemplo 2 (1p): Passagem de vetor como argumento para uma função

```c
#include <stdio.h>

void inputV(int v[], int vSize);

//=========================================================
int main( )
{
  int i;
  const int aSize = 5;
  int a[aSize];

  const int bSize = 10;
  int b[bSize];

  inputV(a, aSize);
  printf("\n Vetor a: \n");
  for(i = 0; i<aSize; i++)
    printf("%i  ", a[i]);

  inputV(b, bSize);
  printf("\n Vetor b: \n");
  for(i = 0; i<bSize; i++)
    printf("%i  ", b[i]);

  return 0;
}
//=========================================================
void inputV(int v[], int vSize)
{
  int i;

  for(i = 0; i< vSize; i++)
    v[i] = i;

  return;
}
```

Vetor a:
0 1 2 3 4
Vetor b:
0 1 2 3 4 5 6 7 8 9

Exemplo 3 (1p): Criação de um vetor de estruturas **Books**

```c
#include <stdio.h>
#include <string.h>

struct Books
{
  int  book_id;
  char title[50];
  char author[50];
};
//-----------------------------------------------------------
void printBook( struct Books book, int n  );
//===========================================================
int main( )
{
  const int lib_size = 5;
  struct Books Lib[lib_size];/* Array[]  Books */
  int i;

  for (i = 0; i < lib_size; i++)
  {
    printf("\n\n Entrada de dados para livro %i: \n", i);

    Lib[i].book_id = 100 + i;

    printf("\n Titulo do livro: ");
    scanf("%s", Lib[i].title) ;

    printf("\n Autor do livro: ");
    scanf("%s", Lib[i].author);
  }


  printf("\n_________ Conteudo do vetor Lib:__________ ");
  for (i = 0; i < lib_size; i++)
  {
    printBook( Lib[i], i );
  }

  printf("\n");
  return 0;
}
//===========================================================
void printBook( struct Books book, int n )
{
  printf( "\n\n__________ Livro %d __________\n", n);
  printf( "Codigo (book_id) : %d\n", book.book_id);
  printf( "Titulo (title)   : %s\n", book.title);
  printf( "Autor  (author)  : %s\n", book.author);
  return;
};
```

Exercícios:

Ex 4 (4 pontos):

Criar vetor **a** com 10 elementos e vetor **b** com 15 elementos.
Criar funções para:
- atribuir valores inciais aos elementos dos vetores.
- procurar pelo menor elemento em vetor.
- achar a soma dos elementos do vetor.
- achar o valor médio dos elementos do vetor.

Ex 5 (5 pontos):

Usando o conceito de estruturas(registros) escrever um programa para
1. Criar uma estrutura **Aluno**:
    - int Aluno_id
    - char Nome[50]
    - float Nota_p1
    - float Nota_p2
2. Criar um vetor para armazenar os dados sobre 5 alunos (**struct Aluno Turma[5]**),
3. Preencher o vetor **Turma** com dados, as notas são números no intervalo [0, 10].
4. Imprimir o conteúdo do vetor **Turma** em ordem inversa.
5. Procurar pelo aluno com maior e menor nota em cada uma das provas.