ESTRUTURAS / REGISTROS (STRUCT)

Estruturas/Registros – é o tipo de dados definido pelo usuário que permite agregar várias 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).

CRIAÇÃO DE UMA ESTRUTURA

Para criar uma 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 função específica para processar as sequências de 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 **s2** em string **s1**

Exemplo 1: 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
```

CRIAÇÃO DE UM VETOR DE ESTRUTURAS

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 específico de uma estrutura que faz parte de um vetor será feito dessa forma:

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

ESTRUTURAS E COMO ARGUMENTOS DE UMA FUNÇÃO

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

Exemplo 2: Criação de um vetor de estruturas **Books.** Estrutura como argumento de uma função

```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;
};
```

ALOCAÇÃO DINÂMICA DE ESTRUTURAS

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

Operador seta

Em C e C++ operador seta permite acesso para campos de estrutura a partir de uma variável do tipo ponteiro.

Operador seta é composto por símbolo de “menos” (-) e símbolo “maior” (>) e tem a forma de (**->**)

Sintaxe do operador seta

(ponteiro para estrutura) -> (campo da estrutura)

Exemplo 3: Alocação dinâmica de estrutura e uso de variável seta

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

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

//=====================================================================
int main()
{
  struct Books *aPtr; // Ponteiro para Livro 1
  struct Books *bPtr; // Ponteiro para Livro 2

  // Alocação dinamica para  Livro 1
  aPtr = (struct Books *)malloc( sizeof(struct Books));

  // Alocação dinamica para  Livro 2
  bPtr = (struct Books *)malloc( sizeof(struct Books));

 //---------------------------------------------------------------------
  printf("\n Entrada de dados para Livro 1  \n");

  printf(" Identificador: ");
  scanf("%d", &aPtr->book_id);

  printf(" Titulo do livro: ");
  scanf(" %[^\n]%*c", aPtr->title) ;

  printf(" Autor do livro: ");
  scanf(" %[^\n]%*c", aPtr->author);

```

```c
//-------------------------------------------------------------------
 printf("\n Entrada de dados para Livro 2 \n");

 printf(" Identificador: ");
 scanf("%d", &bPtr->book_id);

 printf(" Titulo do livro: ");
 scanf(" %[^\n]%*c", bPtr->title) ;

 printf(" Autor do livro: ");
 scanf(" %[^\n]%*c", bPtr->author);

//-------------------------------------------------------------------
 printf("\n ================================== \n ");
 printf(" Livros cadastrados:\n");

 printf( "\n_________ Livro 1 _________\n");
 printf( "Codigo (book_id) : %d\n", aPtr->book_id);
 printf( "Titulo (title)   : %s\n", aPtr->title);
 printf( "Autor  (author)  : %s\n", aPtr->author);

 printf( "\n_________ Livro 2 _________\n");
 printf( "Codigo (book_id) : %d\n", bPtr->book_id);
 printf( "Titulo (title)   : %s\n", bPtr->title);
 printf( "Autor  (author)  : %s\n", bPtr->author);


 free(aPtr);
 free(bPtr);

 return 0;
}
```

Observação sobre formatação de entrada de dados

Leitura de dados em formato texto requer cuidados específicos e varia dependendo do formato de dados de entrada.

Existem várias formas de fazer a leitura de dados textuais e outras funções além da **scanf()** podem ser utilizadas dependendo do caso.

Por exemplo, quando se pretende fazer a leitura de uma sequência de caracteres que contém espaços em branco no meio tem que tomar cuidado para limpar o “fluxo” de dados de entrada e especificar que espaço em branco não significa o fim de entrada de dados da seguinte forma:

```c
printf(" Titulo do livro: ");
scanf(" %[^\n]%*c", (bPtr+i)->title) ;
```

É possível criar de forma dinâmica um vetor de estruturas com a quantidade de elementos defendia ao longo da execução do programa

Exemplo 4: Um vetor dinâmico de estruturas

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

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

//=====================================================================
int main()
{
  struct Books *bPtr;
  int noOfBooks;
  int i;

  printf("Quantos livros vai ter no cadastro: ");
  scanf("%d", &noOfBooks);

  // Alocação dinamica para noOfBooks estructras
  bPtr = (struct Books *)malloc(noOfBooks * sizeof(struct Books));

  for (i = 0; i < noOfBooks; i++ )
  {

    printf("\n Entrada de dados para livro %i: \n", i);

    printf(" Identificador: ");
    scanf("%d", &(bPtr+i)->book_id);

    printf(" Titulo do livro: ");
    scanf(" %[^\n]%*c", (bPtr+i)->title) ;

    printf(" Autor do livro: ");
    scanf(" %[^\n]%*c", (bPtr+i)->author);
  }
  printf("\n ================================== \n ");
  printf(" Livros cadastrados:\n");

  for (i = 0; i < noOfBooks; i++ )
  {
    printf( "\n__________ Livro %d __________\n", i);
    printf( "Codigo (book_id) : %d\n", (bPtr + i)->book_id);
    printf( "Titulo (title)   : %s\n", (bPtr + i)->title);
    printf( "Autor  (author)  : %s\n", (bPtr + i)->author);
  }

  free(bPtr);

  return 0;
}
```