Tipos de dados - C

- **simples**
- **estruturados**

- ✓ char
- ✓ int
- ✓ float
- ✓ void
- ✓ ponteiros

- ✓ arranjos
- ✓ estruturas
- ✓ arquivos
- ✓ enumerações

Diferença entre arranjo e estrutura

Arranjo (homogêneo)
- Todos os elementos do mesmo tipo.
- Um só nome, com índice para identificar cada elemento.

Estrutura (heterogêneos)
- Elementos podem ser de tipos diferentes
- Cada elemento é identificado através de um nome único - campo.

Exercício

A partir da estrutura definida no item 1, implemente as funções ler_bilhete e mostrar_bilhete, que permitem, respectivamente, ler e mostrar todos os dados relativos a um determinado bilhete.

Observe que as funções devem receber a estrutura bilhete como parâmetro, o que implica o uso de ponteiros.


```
struct data
   {
     int dia, mes, ano;
   };
struct horario
   {
     int hora, min;
   };
struct est_bilhete
   {
     int cod;
     char nome_p[20];
     char origem[20], destin
     struct data data_b;
     struct horario horario_
     int assento;
     float valor;
   };
```

```
// função que efetua a leitura dos dados de uma passag
  void ler_bilhete (struct est_bilhete *bilhete ) // pont
 {
  printf ("Cod. da Empresa: ");
  scanf ("%d", &bilhete->cod);
  fflush(stdin);
  printf ("Nome do Passageiro: ");
  gets ((*bilhete).nome_p);
  printf ("Origem: ");
  gets (bilhete->origem);
  printf ("Destino: ");
  gets (bilhete->destino);
  printf ("Data do Bilhete (dd mm aa): ");
  scanf ("%d%d%d", &bilhete->data_b.dia, &bilhete->data_b.mes,
                                          &bilhete->data_b.ano);
  printf ("Horario do Bilhete (hh mm): ");
  scanf ("%d%d", &(*bilhete).horario_b.hora, &bilhete->horario_b.min);
  printf ("Assento: ");
  scanf ("%d", &bilhete->assento);
  printf ("Valor do Bilhete: ");
  scanf ("%f", &bilhete->valor);
 }
```



```
struct data
   {
     int dia, mes, ano;
   };
struct horario
   {
     int hora, min;
   };
struct est_bilhete
   {
     int cod;
     char nome_p[20];
     char origem[20], destino[20];
     struct data data_b;
     struct horario horario_b;
     int assento;
     float valor;
   };
```

```
// função que efetua a impressão dos dados de uma passagem
  void mostra_bilhete ( struct est_bilhete *bilhete )
  {
    printf ("\nCod. da Empresa: %d", bilhete->cod);
    printf ("\nNome do Passageiro: %s",bilhete->nome_p);
    printf ("\nOrigem: %s ",bilhete->origem);
    printf ("\nDestino: %s ",bilhete->destino);
    printf ("\nData do Bilhete: %d/%d/%d", bilhete->data_b.dia,
                                          bilhete->data_b.mes, bilhete->data_b.ano);
    printf ("\nHorario do Bilhete: %d:%d hs.", bilhete->horario_b.hora,
                                          bilhete->horario_b.min);
    printf ("\nAssento: %d", bilhete->assento);
    printf ("\nValor do Bilhete: R$ %.2f \n\n", bilhete->valor);
  }
```



```
// programa principal:
 int main( )
{
 struct est_bilhete passagem;
 system("color f1");
 ler_bilhete(&passagem);
 mostra_bilhete(&passagem);
 system("pause");
 return 0;
}
```

```
/*  Name: viajebem.cpp        Author: Cida Souto.
    Date: 20/05/08 00:07      Description: definir tipo
    bilhete de passagem de onibus.  */
#include <stdio.h>
#include <stdlib.h>
// struct como parâmetro: precisa ser global!!!
struct data
   {
     int dia, mes, ano;
   };
struct horario
   {
     int hora, min;
   };
struct est_bilhete
   {
     int cod;
     char nome_p[20];
     char origem[20], destino[20];
     struct data data_b;
     struct horario horario_b;
     int assento;
     float valor;
   };
void ler_bilhete (struct est_bilhete *bilhete )
.....
void mostra_bilhete ( struct est_bilhete *bilhete )
.....

int main ( )
{

} // fim de main
```

Impressão: não precisa receber ponteiro!!!!

```
// função que efetua a impressão dos dados de uma passagem, sem ponteiro:
 void mostra_bilhete ( struct est_bilhete bilhete ) // recebe o conteúdo
 {
  printf ("\nCod. da Empresa: %d", bilhete.cod);
  printf ("\nNome do Passageiro: %s",bilhete.nome_p);
  printf ("\nOrigem: %s ",bilhete.origem);
  printf ("\nDestino: %s ",bilhete.destino);
  printf ("\nData do Bilhete: %d/%d/%d", bilhete.data_b.dia,
                                         bilhete.data_b.mes, bilhete.data_b.ano);
  printf ("\nHorario do Bilhete: %d:%d hs.", bilhete.horario_b.hora,
                                             bilhete.horario_b.min);
  printf ("\nAssento: %d", bilhete.assento);
  printf ("\nValor do Bilhete: R$ %.2f \n\n", bilhete.valor);
 }

 // programa principal:
  int main( )
  {
   struct est_bilhete passagem;
   system("color f1");
   ler_bilhete(&passagem);       // passa endereço
   mostra_bilhete(passagem);     // passa conteúdo
   system("pause");
   return 0;
  }
```


Declaração de Estruturas

Forma 1- só o tipo:

```
struct funcionario
    {
       int   cod;
       char  nome[30];
       int   depto;
       float salario;
    };
```

Forma 2 – associado a variável:

```
struct
    {
      int   cod;
      char  nome[30];
      int   depto;
      float salario;
    } func1, func2 ;
```

Forma 3 – com tipo e **struct** com variável:

```
struct funcionario
      {
        int   cod;
        char nome[30];
        int   depto;
        float salario;
      } ;
struct funcionario func1, func2;
```

Forma 4 – omitindo **struct na** variável:

```
struct funcionario
      {
          int   cod;
        char nome[30];
        int   depto;
        float salario;
      } ;
funcionario func1, func2;
```


Definição de tipos do usuário - typedef

- Variáveis do tipo estruturas devem ser precedidas da definição e do termo ***struct***.
- Para simplificar, pode-se usar sinônimos para designar tipos, através do recurso ***typedef***.

Exemplo:

```
typedef  struct  pessoa  //minúsculas
 {
        int idade;
        char est_civil, sexo;
        char nome[60];
 } PESSOA  // em maiúsculas
```

- Podem ser utilizadas as duas formas diferentes:

```
struct pessoa  Julio, Joao;
PESSOA julio joao ;
```


typedef – onde utilizar

- Sempre visíveis para todo o programa, isto é, devem ser globais.
- Estruturas globais: definidas no início do programa, antes de qualquer função.

```
/*Descrição do programa: */
# include <  ...  >
# include   <  ...  >

struct data {.......}
typedef  .... ;
/*protótipos das funções */
...
/*  funções    */
funcao1 ( ) {  ... }
funcao2 ( ) {  ... }
/* programa principal */
int main ()      {....}
```


typedef – Exemplos

```
/* Estruturas embutidas */
# include  <  ...  >
// só nome do typedef
typedef  struct
   {
        int dia;
        char mes [3+1] ;
        int ano;
   }  DATA;
// nome de struct e typedef
typedef  struct  s_pessoa
   {
        char nome [60] ;
        float salario;
        DATA dt_nasc;
   }  PESSOA
```

```
/* Vetor de Estruturas */
struct s_pessoa homem, mulher[3];
   ou
PESSOA homem, mulher [3];
```

```
/* Inicializando na declaração */
 PESSOA homem =
    {"Julio", 1230.00, {12, "Mai", 1985} },
 mulher [3] =
    {
        {"Ana", 5400.00,  {2, "Jan", 1980} },
        {"Joana", 400.00, {28, "Dez", 1989}
},
        {"Laura", 2100.00, {2, "Jul", 1979}
},
```


typedef – Exemplos

```
/* Inicializando na declaração */
 PESSOA homem =
    {"Julio", 1230.00, {12, "Mai", 1985} },
 mulher [3] =
    {
        {"Ana", 5400.00,  {2, "Jan", 1980} },
        {"Joana", 400.00, {28, "Dez", 1989} },
        {"Laura", 2100.00, {2, "Jul", 1979} },
    }
```

mulher: vetor de 3 elementos do tipo estrutura

mulher[i] : i-ésima estrutura

mulher[i].dt_nasc: estrutura com 3 campos de data

mulher[i].dt_nasc.mes : *string* do mês da data de nascimento

mulher[i].dt_nasc.mes[0] : primeiro caractere do *string* do mês da data de nascimento

Passagem de parâmetros do tipo estrutura: por valor

- A ***struct*** ou ***typedef*** correspondente deve ter sido declarada na área global do programa.
- Na declaração da função, o parâmetro deve ser declarado com a ***struct*** ou ***typedef*** correspondente.

```
typedef struct
{
    int dia, mes, ano; //aqui, data só de inteiros
} DATA;
typedef struct pessoa
{
    char nome[100];
    int idade;
    float salario;
    DATA  nasc; // usa outro typedef
} PESSOA;
void mostrar (struct pessoa x)   /* ou void mostrar (PESSOA x ) */
{
    printf ("Nome       : %s\n", x.nome);
    printf ("Idade      : %d\n\n", x.idade);
    printf ("Data Nasc : %d/%d/%d\n", x.nasc.dia, x.nasc.mes, x.nasc.ano);
}
int main ( )
{
   struct pessoa p = {"Joao",23, {23, 9, 1964}};
   mostrar (p); ....
}
```

Passagem por VALOR!

Passagem de parâmetros do tipo estrutura: por referência

- Para que a variável seja efetivamente alterada, o parâmetro precisa ser definido como ponteiro, através da passagem de endereço.

```
typedef struct
{
    int dia, mes, ano; //aqui, data só de inteiros
} DATA;
typedef struct pessoa
{
    char nome[100];
    int idade;
    float salario;
    DATA  nasc;
} PESSOA;
void ler (struct pessoa *x)   /* ou void mostrar (PESSOA *x ) */
{
   printf ("Nome       : %s\n", (*x). nome);
   printf ("Idade      : %d\n\n", (*x). idade);
   printf ("Data Nasc : %d/%d/%d\n", (*x).nasc.dia,(*x). nasc.mes, (*x). nasc.ano);
}
int main ( )
{
  struct pessoa p;
  ler(&p);
  mostrar (p); ....
}
```

Passagem por REFERÊNCIA!

Precedência de Operadores

```
void ler  (PESSOA x)
{ // leitura, por valor, :
   gets (x.nome);
```

```
void ler (PESSOA *x)
{ // leitura, por referência:
   gets ( (*x).nome);
```

- O parênteses é necessário porque o . (ponto) tem prioridade maior que o * (ponteiro).
- O compilador interpreta gets (*x.nome) como ponteiro para a estrutura x.nome : como não existe este tipo de estrutura, daria erro de compilação.
- Logo: gets (~~*x.~~nome) ERRADO!

Operador seta: ->

- Utilizado em funções com parâmetro do tipo estrutura, onde ponteiros devem ser utilizados.
- Indica o acesso a um campo de uma estrutura por meio de um ponteiro para esta estrutura.
- Escrever aluno-> media é o mesmo que escrever (*aluno).media.
- Atenção: a notação com o operador seta é a de uso mais freqüente.

Exercício

Fazer um programa que execute, seqüencialmente, as tarefas abaixo:

1. leia e armazene os dados das 4 equipes da *Gincana do Inverno* em um vetor equipes [4]. Para isso, utilize em cada elemento a estrutura estr_equipe, composta de:
   - nome da equipe (até 12 caracteres)
   - categoria (1 - Junior, 2 - Senior)
   - participantes (máximo 4) –número máximo permitido de participantes
2. baseado nas informações armazenadas no vetor de estruturas acima, leia e armazene os dados dos participantes em uma tabela participantes[10]. Para isso, utilize a estrutura estr_particip, contendo:
   - nome da equipe (até 12 caracteres) – consistir se existe esta equipe!!!
   - nome do participante (até 20 caracteres) – controlar número
3. liste cada equipe, com seus dados da tabela equipes, seguidos dos nomes dos membros desta equipe;

Obs: Na etapa de cadastramento de participante, o programa deve contar os cadastrados em cada equipe, garantindo que não ultrapassem o número máximo previsto.

Exercício

Utilize as definições abaixo:

```
// Gincana
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define NUMEQ 3 // número de equipes
#define NUMPART 12 // número máximo de participantes
struct est_equipe
 { /*declaração da estrutura */
     char nome_eq[13];
     int categ; // 1 ou 2
     int nro_partic; // até 4  ARRUMAR!!!!
 };
struct est_partic
 { /*declaração da estrutura utilizada*/
     char nome_eq[13];
     char nome_partic[21];
 };
int main()
{
   est_equipe equipe[NUMEQ]; // reserva área para NUMEQ equipes
   est_partic particip[NUMPART]; // reserva para até NUMPART
   participantes
   system("color f1");
   .....
```