Registros ou Estruturas (structs)

- Coleção de uma ou mais variáveis agrupadas com um único nome para facilitar manipulação.
- Variáveis do registro, ao contrário de um vetor, podem ser de tipos diferentes, inclusive vetores.
- Cada variável dentro de um registro é chamada de membro do registro.

Definindo e declarando Registros

- *Sintaxe de definição de registros:*

```
struct <nome do registro> {
   <tipo> <nome da variável 1 do registro>;
   <tipo> <nome da variável 2 do registro>;
   ...
   <tipo> <nome da variável n do registro>;
};
```

- Exemplo: uma pessoa possui vários atributos, como nome, idade, sexo, etc. A declaração destes atributos a seguir é feita usando variáveis que não possuem a menor ligação entre si. Poderiam se tratar de pessoas diferentes.

```
char nome[80];
char sexo;
int idade;
```

- O uso de um registro permite englobar estes atributos com um único nome:

```
struct pessoa {
   char nome[80];
   char sexo;
   int idade;
};
```

- O código anterior define um tipo de registro chamado `pessoa` que contém um inteiro, um caractere e uma cadeia de caracteres. No entanto, não cria nenhuma *instância* com este registro.
- Existem duas formas de se declarar instâncias de registros:
  - *Na definição* – seguir a definição do registro com uma lista de nomes de variáveis deste registro.
    - Sintaxe:

```
struct <nome do registro> {
   <variáveis do registro>
} <nome instância 1>, ..., <nome instância n>;
```

    - Exemplo:

```
struct pessoa {
   char nome[80];
   char sexo;
   int idade;
} joao, maria;
```

- *Fora da definição* – declarar variáveis do registro em um local do código diferente da definição.
  - Sintaxe:

```
struct <nome do registro> {
   <variáveis do registro>
}
// pode haver código adicional aqui
struct <nome do registro> <nome instância 1>, ..., <nome instância n>;
```

  - Exemplo:

```
struct pessoa {
   char nome[80];
   char sexo;
   int idade;
};

struct pessoa joao, maria;
```

Acessando Membros de Registros

- Acesso através do uso do operador de membro de registro ("`.`"), também chamado *operador ponto*, entre o nome do registro e o nome do membro.
- Exemplo:

```
strcpy(joao.nome, "João");
joao.sexo = 'M';
joao.idade = 20;
```

- Uma das vantagens do uso de registros é a cópia de informações entre registros com uma única instrução.
- No exemplo a seguir, tem-se duas instâncias (`p1` e `p2`) de um registro de pontos com três dimensões. Supondo que ambos os pontos possuem as mesmas informações, a atribuição entre instâncias de registros realiza a ***cópia*** de todas as variáveis de uma instância para a outra.

```
struct ponto3d {
   int x, y, z;
} p1, p2;

p1.x = 0;
p1.y = 3;
p1.z = -1;

p2 = p1;
```

iguais

```
struct ponto3d {
   int x, y, z;
} p1, p2;

p1.x = 0;
p1.y = 3;
p1.z = -1;

p2.x = 0;
p2.y = 3;
p2.z = -1;
```

- Uma instância de registro pode ser inicializada logo no momento da declaração como a seguir:

```
struct pessoa {
   char nome[80];
   char sexo;
   int idade;

} joao = { "João", 'M', 20 }
```

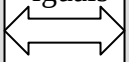
Registros de Registros

- Um registro pode conter outros registros como membros.
  Exemplo: registro `retangulo` que contém registros `coord` (coordenadas de pontos do retângulo).

```
#include <stdio.h>

int main() {
   int largura, altura;
   long area, perimetro;
   struct coord {
      int x;
      int y;
   };
   struct retangulo {
      struct coord sup_esq;
      struct coord inf_dir;
   } meu_retangulo;

   /* atualizando coordenadas do ponto superior esquerdo do retangulo */
   scanf("%d", &meu_retangulo.sup_esq.x);
   scanf("%d", &meu_retangulo.sup_esq.y);
   /* atualizando coordenadas do ponto inferior direito do retangulo */
   scanf("%d", &meu_retangulo.inf_dir.x);
   scanf("%d", &meu_retangulo.inf_dir.y);

   largura = meu_retangulo.inf_dir.x - meu_retangulo.sup_esq.x;
   altura  = meu_retangulo.sup_esq.y - meu_retangulo.inf_dir.y;

   area = largura * altura;
   perimetro = (2 * largura) + (2 * altura);

   printf("\nArea.....: %ld", area);
   printf("\nPerimetro: %ld", perimetro);
   return 0;
}
```

- Registros podem ser referenciados por ponteiros. Neste caso, o operador "`->`" pode ser usado no acesso aos membros do registro como a seguir:

```
struct coord *p;
p = &meu_retangulo.sup_esq;
(*p).x = 0;
(*p).y = 0;
```




```
struct coord *p;
p = &meu_retangulo.sup_esq;
p->x = 0;
p->y = 0;
```

Vetor de Registros

- .O programa a seguir cria um registro `cd` que contém um vetor de registros com as músicas do CD:

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

struct musica {
   char nome[50];
   char autor[50];
};
struct cd {
   char titulo[50];
   int numMusicas;
   struct musica musicas[30];
};

void main() {
   struct cd meuCD;

   strcpy(meuCD.titulo, "Abbey Road");
   meuCD.numMusicas = 2;
   strcpy(meuCD.musicas[0].nome, "Come Together");
   strcpy(meuCD.musicas[0].autor, "Lennon / MacCartney");
   strcpy(meuCD.musicas[1].nome, "Something");
   strcpy(meuCD.musicas[1].autor, "Harrison");
}
```

Funções que Recebem e Retornam Registros

- O programa a seguir cria dois registros com as coordenadas de dois pontos *a* (1, 2) e *b* (3, 0) e chama a função `maisADireita` para retornar qual deles está mais à direita no plano:

```
#include <stdio.h>
struct coord {
   int x;
   int y;
};
struct coord maisADireita(struct coord *a1, struct coord *b1) {
   if (a1->x > b1->x)
      return *a1;
   else
      return *b1;
}
void main() {
   struct coord a, b, coordDireita;
   a.x = 1;
   a.y = 2;
   b.x = 3;
   b.y = 0;

   coordDireita = maisADireita(&a, &b);
   printf("%d %d", coordDireita.x, coordDireita.y); // imprime "3 0"
}
```