Estruturas de Dados Heterogêneas

Structs são coleções de dados heterogêneos agrupados em um mesmo elemento de dados**. T**ambém conhecidas como **Registros**, definem tipos de dados que agrupam variáveis sob um mesmo tipo de dado. Aos dados agrupados em uma **struct** dá-se o nome de **campos**(*fields*).

Ex: armazenar os dados de uma pessoa Nome, Idade, CPF, Salário.

```
typedef struct //Cria uma STRUCT para armazenar os dados de uma pessoa
{
    float Peso;   // define o campo Peso
    int Idade;    // define o campo Idade
    float Altura; // define o campo Altura
} Pessoa; // Define o nome do novo tipo criado
```

Após a criação do tipo, podemos declarar variáveis do tipo **Pessoa**:

```
Pessoa Paulo, P0, P1, P2;
Pessoa Pessoas[10];  // cria um vetor de 10 pessoas.
```

Para acessar os campos de uma **struct**, `NomeDaVariavel.NomeDoCampo`

Exemplo:

```
P1.Idade = 20;
P1.Peso = 65.5;
P1.Altura = 1.65;

Pessoas[0].Idade = 21;
Pessoas[0].Peso = 75.0;
Pessoas[4].Altura = 1.75;
```

Outra vantagem de utilizar **struct** é a possibilidade de atribuir os dados de uma **struct** para outra, com apenas um comando de atribuição, como neste exemplo:

```
P2 = Pessoas[0];
```

Passagem de Structs por Parâmetro

Para passar uma struct **por valor** basta declará-la como um dos parâmetros, como no exemplo a seguir

```
#include <stdio.h>

typedef struct //Cria uma STRUCT para armazenar os dados de uma pessoa
{
    float Peso;   // define o campo Peso
    int Idade;    // define o campo Idade
    float Altura; // define o campo Altura
} Pessoa; // Define o nome do novo tipo criado
```

```
void ImprimePessoa(Pessoa P) // declara o parâmetro como uma struct
{
  printf("Idade: %d  Peso: %f Altura: %f\n", P.Idade, P.Peso, P.Altura);
}

int main(){
    Pessoa Aluno1, Aluno2, P1, P2;
    Pessoa Pessoas[10];

    Aluno1.Idade = 20;
    Aluno1.Peso = 55.5;
    Aluno1.Altura = 1.55;

    Pessoas[4].Idade = 21;
    Pessoas[4].Peso = 75.0;
    Pessoas[4].Altura = 1.75;

    P2 = Pessoas[4];
    P2.Idade++;

    // chama a função que recebe a struct como parâmetro
    ImprimePessoa(Joao);
    ImprimePessoa(Pessoas[4]);
    ImprimePessoa(P2);

   return 0;
}
```

Retorno de Structs em Funções

Como uma struct define um tipo de dado, este tipo pode ser retornado em uma função, da mesma forma que ocorre com qualquer outro tipo de dado.
No exemplo a seguir, a função retorna uma struct que conterá, em seus campos, os dados que foram recebidos por parâmetro.

```
#include <stdio.h>

typedef struct //Cria uma STRUCT para armazenar os dados de uma pessoa
{
    float Peso;   // define o campo Peso
    int Idade;    // define o campo Idade
    float Altura; // define o campo Altura
} Pessoa; // Define o nome do novo tipo criado

Pessoa SetPessoa(int idade, float peso, float altura){
    Pessoa P;
    P.Idade = idade;
    P.Peso = peso;
    P.Altura = altura;
    return P; //retorna a struct contendo os dados passados por parâmetro
}
```

```
void ImprimePessoa(Pessoa P){ // declara o parâmetro como uma struct
  printf("Idade: %d  Peso: %f Altura: %f\n", P.Idade, P.Peso, P.Altura);
}

int main()
{
    Pessoa Joao;

//atribui o retorno da função a uma variável struct
    Joao = SetPessoa(15,60.5,1.75);
    ImprimePessoa(Joao);
   return 0;
}
```

Passagem de Structs por Referência

Para passar uma struct por referência, deve-se passar um ponteiro para a struct, como no exemplo a seguir.

```
#include <stdio.h>

typedef struct //Cria uma STRUCT para armazenar os dados de uma pessoa
{
    float Peso;   // define o campo Peso
    int Idade;    // define o campo Idade
    float Altura; // define o campo Altura
} Pessoa; //Define o nome do novo tipo criado

void ImprimePessoa(Pessoa P){
  printf("Idade: %d  Peso: %f Altura: %f\n", P.Idade, P.Peso, P.Altura);
}

void SetPessoa(Pessoa *P, int idade, float peso, float altura)
{ // Nesta função o parâmetro P é um ponteiro para uma struct
    (*P).Idade = idade;  // o campo pode ser acessado desta forma
    P->Peso = peso;      // ou desta
    P->Altura = altura;
}

int main() {
    Pessoa Joao, P1;
    SetPessoa(&Joao, 15, 70.5, 1.75);
    ImprimePessoa(Joao);
    SetPessoa(&P1, 25, 60.5, 1.65);
    ImprimePessoa(P1);

    return 0;
}
```

```
#define TAMANHO 50
struct Endereco {
char rua[TAMANHO];
char cidade_estado_cep[TAMANHO];
};
struct Endereco endereco1, endereco2;

/*
typedef struct {
char rua[TAMANHO];
char cidade_estado_cep[TAMANHO];
}Endereco;
Endereco endereco1, endereco2;
*/

int main(){
    printf("\n Entre rua: ");
    gets(endereco1.rua);
    printf("\n Entre cidade/estado/cep: ");
    gets(endereco1.cidade_estado_cep);
    printf("\t %s\n", endereco1.rua);
    printf("\t %s\n", endereco1.cidade_estado_cep);

    return 0;
}
```