Estrutura de Dados e Algoritmos

Estruturas

Estruturas

- ***struct*** é uma abreviação para structure.
- É uma **estrutura de dados composta que define fisicamente uma lista de variáveis agrupadas sob um nome em um bloco de memória.**

Estruturas

```
struct [tag da estrutura] {
   tipo_da_variavel nome_da_variavel;
   tipo_da_variavel nome_da_variavel;
   ...
   tipo_da_variavel nome_da_variavel;
} [uma ou mais variáveis da estrutura];
```

Estruturas

```
struct Livros {
   char  titulo[50];
   char  autor[50];
   char  assunto[100];
   int   id_livro;
} livro;
```

Estruturas

```
struct Livros {
   char  titulo[50];
   char  autor[50];
   char  assunto[100];
   int   id_livro;
};
/* Declarando Livro1 e Livro2 do tipo Livros */
struct Livros Livro1, Livro2;
```

Estruturas

```
struct Livros {
   char  titulo[50];
   char  autor[50];
   char  assunto[100];
   int   id_livro;
} Livro1, Livro2;
```

// declarando as variáveis na própria estrutura, não é muito recomendável

Estruturas

```
struct Livros {
   char  titulo[50];
   char  autor[50];
   char  assunto[100];
   int   id_livro;
};

// inicializando
struct Livros Livro1 = { "Título genérico", "Blog Trybe",
"Um livro bem genérico", 6495407 };
```

Estruturas

```
struct Livros {
   char  titulo[50];
   char  autor[50];
   char  assunto[100];
   int   id_livro;
};
/* Declarando Livro1 do tipo Livro */
struct Livros Livro1;
strcpy( Livro1.titulo, "Título genérico"); // inicializando os valores para cada variável separada
strcpy( Livro1.autor, "Blog Trybe");
strcpy( Livro1.assunto, "Um livro bem genérico");
Livro1.id_livro = 6495407;
```

Estruturas

```
/* mostrando as informações do livro 1 */
  printf( "Livro 1 titulo : %s\n", Livro1.titulo);
  printf( "Livro 1 autor : %s\n", Livro1.autor);
  printf( "Livro 1 assunto : %s\n", Livro1.assunto);
  printf( "Livro 1 id_livro : %d\n", Livro1.id_livro);

  /* mostrando as informações do livro 2 */
  printf( "Livro 2 titulo : %s\n", Livro2.titulo);
  printf( "Livro 2 autor : %s\n", Livro2.autor);
  printf( "Livro 2 assunto : %s\n", Livro2.assunto);
  printf( "Livro 2 id_livro : %d\n", Livro2.id_livro);
```

typedef

typedef: é usado para renomear um tipo de dado.

```
// C program to demonstrate typedef
#include <stdio.h>

// After this line BYTE can be used
// in place of unsigned char
typedef unsigned char BYTE;

int main()
{
    BYTE b1, b2;
    b1 = 'c';
    printf("%c ", b1);
    return 0;
}
```

Outro exemplo:

declare uma variável ponto (que é um par ordenado de inteiros). Preceda a declaração de um typedef:

```
typedef struct {
   int x;
   int y;
} ponto;
```

Agora ponto passa a ser o nome de um novo tipo (idêntico a par ordenado de inteiros). Esse tipo pode ser usado para declarar novos pontos:

```
ponto a, b;
```

Alocação Dinâmica

```
 typedef struct
{
        char nome[100];
        int idade;
} pessoa;
main()
{
// alocando uma estrutura
        pessoa *p = (pessoa*) malloc(sizeof(pessoa));
        if (p)
        {
                p->idade = 3;
                printf("%d\n", p->idade);
                free(p);
        }
}
```

Conjuntos de Estruturas

```
typedef struct
{
   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, é possivel declarar variáveis do tipo Pessoa, desta forma:

```
Pessoa Joao, P1, P2;
Pessoa Povo[10];  // cria um vetor de 10 pessoas.
```

Conjuntos de Estruturas

Para acessar os campos de uma struct, usa-se a sintaxe **NomeDaVariavel.NomeDoCampo**, conforme o exemplo a seguir.

Joao.Idade = 15;
Joao.Peso = 60.5;
Joao.Altura = 1.75;

Povo[4].Idade = 23;
Povo[4].Peso = 75.3;
Povo[4].Altura = 1.89;

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:

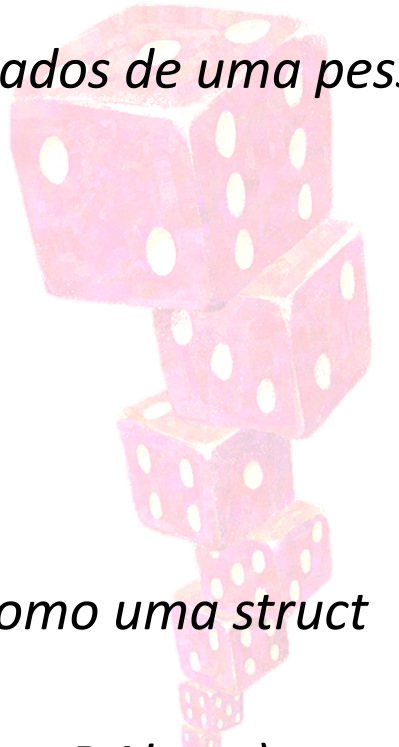
P2 = Povo[4];

Estruturas como Parâmetros

```
#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);
}
```



Estruturas como Parâmetros

```
int main()
{
   Pessoa Joao, P2;
   Pessoa Povo[10];

   Joao.Idade = 15;
   Joao.Peso = 60.5;
   Joao.Altura = 1.75;

   Povo[4].Idade = 23;
   Povo[4].Peso = 75.3;
   Povo[4].Altura = 1.89;

   P2 = Povo[4];
        P2.Idade++;
   // chama a função que recebe a struct como parâmetro
   ImprimePessoa(Joao);
   ImprimePessoa(Povo[4]);
   ImprimePessoa(P2);
  return 0;
}
```

Retorno de Structs em Funções

```
#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
}
```

Retorno de Structs em Funções

```
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;
}
```

Structs como parâmetro por referência

```
#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);
}
```

Structs como parâmetro por referência

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


```
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;
   SetPessoa(&Joao, 15, 70.5, 1.75);
   ImprimePessoa(Joao);
   return 0;
}
```

Referências

https://www.embarcados.com.br/struct-registros-em-linguagem-c/

https://www.inf.pucrs.br/~pinho/LaproI/Structs/Structs.htm

http://linguagemc.com.br/alocacao-dinamica-de-memoria-em-c/

http://www.univasf.edu.br/~criston.souza/algoritmos/arquivos/aula14.pdf

https://www.ime.usp.br/~pf/algoritmos/aulas/footnotes/typedef.html

https://www.geeksforgeeks.org/typedef-versus-define-c/