TAD

Estrutura de Dados

TAD

- TAD – Tipo Abstrato de Dados
- A linguagem C possui diversos tipos de dados nativos como int, float, double, long, char, entre outros.
- Em algumas situações, porém, os tipos de dados nativos não representam adequadamente as informações que o programador precisa representar.
- Tipos Abstratos de dados são tipos de dados que podem ser criados pelo próprio programador C para conseguir representar informações de uma forma mais direta que utilizando apenas os tipos nativos da linguagem C.

Enumeracão

- Uma enumeração em C é um tipo de dado que suporta apenas um conjunto finito de valores.

```
enum diasemana {domingo, segunda, terca, quarta,
quinta, sexta,sabado};
enum mes {janeiro, fevereiro, marco, abril, maio,
junho, julho, agosto, setembro, outubro, novembro,
dezembro};
enum cor {branca, amarela, azul, verde, vermelha,
preta};
```

Enumeracão

- Neste exemplo, criou-se três enumerações diasemana, mes e cor. As enumerações criadas podem ser utilizadas para criar variaveis que somente aceitem como valores os elementos que foram listados entre chaves.
- No exemplo abaixo, utilizou-se uma enumeração para criar uma variavel e depois atribuir um determinado valor a esta variavel. Mais tarde o valor da variavel é testado para saber se vale sexta.
- Tanto sexta quanto segunda são valores validos pois foram listados entre chaves na de definição da enumeração. Internamente uma enumeração e representada por um numero. Ou seja, no exemplo da primeira enumeração a palavra domingo vale a
- mesma coisa que 0 (valor padrão), segunda vale 1, terça 2 e assim por diante. Quando atribuimos um valor Minha_Variavel = segunda na realidade ela esta recebendo o valor 1. Ja na comparação, estamos comparando com o valor 5.

Enumeracão

enum diasemana Minha_Variavel; // uso da enumeracao para criar uma variavel

```
Minha_Variavel = segunda;
if (Minha_Variavel==sexta){
}
else {
}
```

Para criar uma variavel, e necessario utilizar a palavra enum seguido do nome da enumeração criada e o nome da variavel. A forma de tornar o uso de uma enumeração semelhante ao de um tipo nativo da linguagem C, atraves do typedef.

Enumeracão

```
enum Diasemana {domingo, segunda, terca=5, quarta, quinta,
sexta=22,sabado}; // definicao da
enumeracao
enum Mes {janeiro, fevereiro, marco, abril, maio, junho,
julho, agosto, setembro, outubro, novembro,
dezembro};
enum Cor {branca, amarela, azul, verde, vermelha, preta};

typedef enum Diasemana diasemana;

diasemana Minha_Variavel;

Minha_Variavel = segunda;
```

Estruturas não homogeneas

- Usando struct, podemos criar variáveis utilizando posteriormente a palavra struct o nome da estrutura.

```
struct Registro {
    char nome[20];
    char endereco[20];
    int telefone;
};

struct Registro pessoa1;
```

Estruturas não homogeneas

- A forma de declarar variaveis que são uma estrutura semelhante ao enum no sentido de ser necessário escrever a palavra reservada novamente. Podemos adotar a mesma solução que antes e usar o **typedef** para criar um nome novo para o **struct Registro**. Agora, a forma de declarar variaveis ficara semelhante ao uso dos tipos nativos da linguagem C.

```
typedef struct Registro Registro_de_pessoa;
// tipo de dado nome da variavel
Registro_de_pessoa Variavel_V1;
```

Estruturas não homogeneas

```
struct Tipo_Telefone {
        int DDD;
        int numero;
}
typedef struct Tipo_Telefone Telefone;

struct Tipo_Endereco {
        char rua[20]
        int numero_da_casa;
        char cidade[20];
        char estado[3];
}
typedef struct Tipo_Endereco Endereco;
```

```
struct Registro_de_pessoa {
        char nome[20];
        Endereco endereco[20];
        Telefone telefone;
};
typedef struct Registro_de_pessoa
Pessoa;

// tipo de dado nome da variavel
Pessoa Variavel_V1;
```

Estruturas não homogeneas

```
typedef struct XXXX {
      char Nome[5];
      int Codigo;
      char Sexo;
      int c;
      char Opcao;
} Dados_Cliente;
```

```
void main (void)
{
      Dados_Cliente Cliente;

      Cliente.Codigo=12345; Cliente.Sexo= F ;
      printf("Codigo:%d\n",Cliente.Codigo);
      printf("Sexo :%c\n",Cliente.Sexo);
}
```

Inicialização estatica de estruturas

- E possível inicializar uma estrutura no mesmo momento no qual uma variável e criada.

```
typedef struct {
 char Nome[5];
 int Codigo;
}Dados_Cliente;

void main (void)
{
     Dados_Cliente Cliente;
     Dados_Cliente Pedro={"teste",123};
}
```

Modificando campos

```
#include <stdio.h>
struct Teste {
 int codigo;
 int CPF;
};

void muda_CPF ( struct Teste *xx) {
        xx->CPF= 100;
}

void main (void){
        struct Teste T;
        T.codigo=123;
        T.CPF=34;
        muda_CPF(&T);
}
```

Vetores de estruturas

```
struct Teste {
 int codigo;
 int CPF;
};
typedef struct Teste Teste;

Teste vetor[100];

void main (void) {
 vetor[0].CPF=100;
 vetor[0].codigo=1212;
 vetor[1].CPF=1233;
 vetor[1].codigo=345;
}
```

Vetores de estruturas

```
void insere_dados (Teste v[], int codigo, int CPF) {
     v[contador].codigo=codigo;
     v[contador].CPF=CPF ;
     contador++;
}

void mostra_todos (Teste v[]) {
 int x;
 for (x=0;x<contador;x++) {
     printf("%d %d\n",v[x].codigo, v[x].CPF);
 }
}
```

Vetores de estruturas

```
struct Teste {
 int codigo;
 int CPF;
};
typedef struct Teste Teste;
 struct Minha_Estrutura {
 Teste vetor[100];
  int contador;
};
typedef struct Minha_Estrutura Minha_Estrutura;
void insere_dados (Minha_Estrutura *v, int codigo, int CPF) {
                v->vetor[ v->contador ].codigo = codigo;
                v->vetor[ v->contador ].CPF = CPF;
                v->contador++;
}
```

Vetores de estruturas

```
void mostra_todos (Minha_Estrutura v) {
        int x; for (x=0;x<v.contador;x++) printf("%d %d \n",v.vetor[x].codigo,
v.vetor[x].CPF);
}
int main (void) {
        Minha_Estrutura M;
        M.contador = 0;
        insere_dados (&M, 100, 200);
        insere_dados (&M, 100, 300);
        mostra_todos (M);
}
```