


# Tipos abstratos de dados

Prof. Henrique Y. Shishido

Universidade Tecnológica Federal do Paraná

*shishido@utfpr.edu.br*

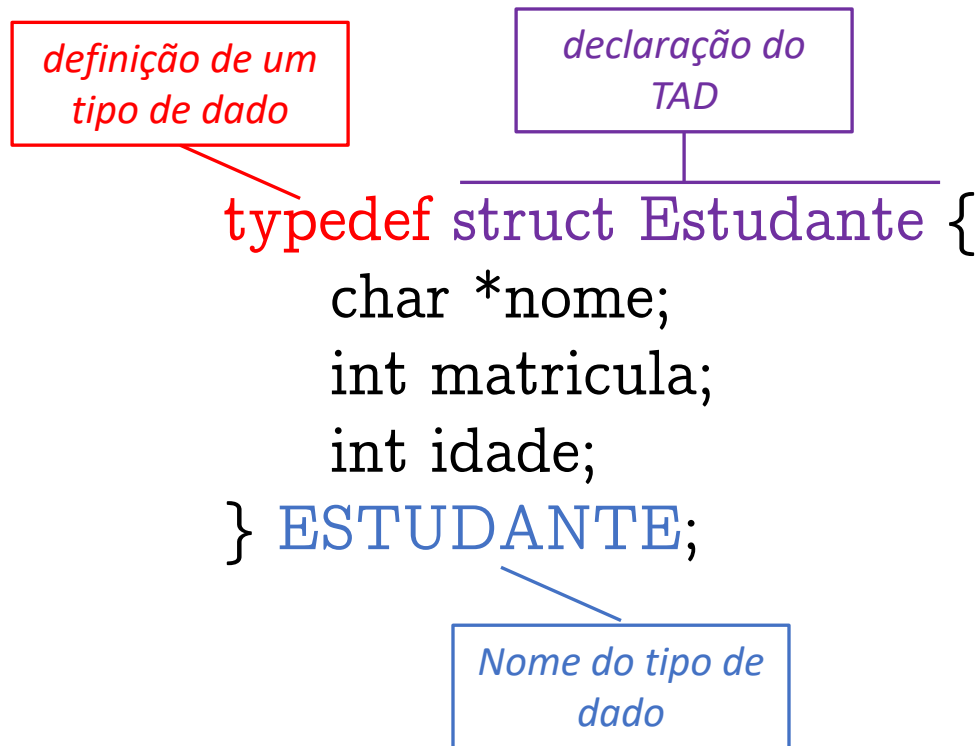
# Tipos abstratos de dados

- Um **Tipo Abstrato de Dados (TAD)** representa um **conjunto de dados** que permite **reduzir a complexidade** de manipulação de dados através da abstração das variáveis envolvidas em uma **única entidade** fechada.
- Por exemplo, ao considerar os dados de um estudante, sem o uso da TAD, essa entidade seria representada por variáveis soltas:
  - `char *nome;`
  - `int idade;`
  - `int matricula;`
  - `char *estado;`
- Se considerar um programa que manipule 10 estudantes, sem o uso de TAD, provavelmente seria necessário declarar vetores de cada uma das variáveis:
  - `char *nome[10];`
  - `int idade[10];`
  - `int matricula[10];`
  - `char *estado[10];`

Dificuldade na manipulação e entendimento de código

# Tipo abstrato de dados

- Um Tipo Abstrato de Dados é implementado por meio de uma estrutura (**struct**) com as variáveis pertencentes ao TAD (nome, idade, matricula, estado):



## Exemplo

```
#include <stdio.h>  
  
typedef struct Estudante {  
    char *nome;  
    int matricula;  
    int idade;  
} ESTUDANTE;  
  
int main() {  
  
    ESTUDANTE *est;  
  
    return 0;  
}
```

# Tipo abstrato de dados (source: exemplo\_struct.c)

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

```
typedef struct Estudante {  
    char *nome;  
    int matricula;  
    int idade;  
} ESTUDANTE;
```

Declaração de um  
**TAD -> Estudante**

```
int main() {
```

Declaração de uma variável  
usando o tipo ESTUDANTE

```
    ESTUDANTE est;
```

```
    est.nome = "Henrique";  
    est.matricula = 11111;  
    est.idade = 35;
```

Atribuição de valor

Leitura de valor

```
    printf("\nNome: %s\nIdade: %d\nMatricula: %d\n", est.nome, est.matricula, est.idade);
```

```
    return 0;
```

```
}
```

**Saída**

```
Nome: Henrique  
Idade: 11111  
Matricula: 35
```

# Ponteiros e estruturas (source: struct\_pointer.c)

- Como em qualquer outro tipo de dado, é possível usar ponteiros para TADs

```
typedef struct Estudante {  
    char *nome;  
    int matricula;  
    int idade;  
} ESTUDANTE;  
  
int main() {  
  
    ESTUDANTE est;  
    ESTUDANTE *p_est;  
  
    p_est=&est;  
  
    est.nome = "Henrique";  
    est.matricula = 12345;  
    est.idade = 35;  
  
    printf("\nNome: %s\nMatricula: %d\nIdade: %d\n",  
          est.nome, est.matricula, est.idade);  
    printf("\nNome: %s\nMatricula: %d\nIdade: %d\n\n",  
          p_est->nome, p_est->matricula, p_est->idade);  
  
    return 0;  
}
```

## Saída

```
Nome: Henrique  
Matricula: 12345  
Idade: 35
```

```
Nome: Henrique  
Matricula: 12345  
Idade: 35
```

# Ponteiros e estruturas (source: struct\_pointer02.c)

```
typedef struct Estudante {  
    char *nome;  
    int matricula;  
    int idade;  
} ESTUDANTE;
```

```
int main() {
```

```
    ESTUDANTE *est;
```

```
    est->nome = "Henrique";  
    est->matricula = 11111;  
    est->idade = 35;
```

```
    printf("\nNome: %s\nIdade: %d\nMatricula: %d\n",  
           est->nome, est->matricula, est->idade);
```

```
    (*est).nome = "Pedro";  
    (*est).matricula = 12345;  
    (*est).idade = 20;
```

```
    printf("\nNome: %s\nIdade: %d\nMatricula: %d\n",  
           (*est).nome, (*est).matricula, (*est).idade);
```

```
    return 0;
```

- Uma anotação do tipo `(*p_est).nome` é confusa. Assim, a linguagem C define um operador adicional `->` para acessar membros de estruturas através de ponteiros;
- O operador `->` substitui o operador `.` No caso da utilização de um ponteiro para uma *struct*.

## Saída

```
Nome: Henrique  
Idade: 11111  
Matricula: 35
```

```
Nome: Pedro  
Idade: 12345  
Matricula: 20
```

# Tipo abstrato de dados (source: vetor\_struct.c)

```
typedef struct Estudante {  
    char *nome;  
    int matricula;  
    int idade;  
} ESTUDANTE;  
  
int main() {
```

```
    ESTUDANTE *v_est;
```

```
    v_est=(ESTUDANTE*) malloc(sizeof(ESTUDANTE)*2);
```

Declaração vetor  
usando ***malloc()***

```
    v_est[0].nome = "Henrique";  
    v_est[0].matricula = 12345;  
    v_est[0].idade=35;
```

```
    v_est[1].nome = "Pedro";  
    v_est[1].matricula = 11111;  
    v_est[1].idade=20;
```

Atribuição de valor a  
cada item do vetor

```
    for(int i=0; i<2; i++) {  
        printf("\nNome: %s\nMatricula: %d\nIdade: %d\n",  
            v_est[i].nome, v_est[i].matricula, v_est[i].idade);  
    }  
    return 0;
```

Leitura de valor a  
cada item do vetor

```
}
```

# Alocação e liberação de memória (source: struct\_malloc\_free.c)

- Função **free()** libera o espaço de memória ocupado pelo ponteiro;
- Por motivo de otimização do uso de recursos, a função **free()** deve ser invocada assim que a estrutura não for mais usada

```
typedef struct Estudante {  
    char *nome;  
    int matricula;  
    int idade;  
} ESTUDANTE;  
  
int main() {  
  
    ESTUDANTE *est;  
  
    est=(ESTUDANTE*) malloc(sizeof(ESTUDANTE));  
  
    est->nome = "Henrique";  
    est->matricula = 11111;  
    est->idade = 35;  
  
    printf("\nNome: %s\nIdade: %d\nMatricula: %d\n",  
        est->nome, est->matricula, est->idade);  
  
    free(est); Liberação de memória  
    return 0;  
}
```



# Exercício (source: realloc.c)

Criar um vetor da *struct Item* (definido abaixo) usando números inteiros com tamanho igual a 5 usando a função ***malloc()***. Em seguida, aumentar a capacidade deste vetor para 6 posições. A nova posição deste vetor deverá receber a media dos 5 elementos existentes no vetor. Imprima o vetor resultante.

```
typedef struct Item {  
    int valor;  
} ITEM;
```

# Exercício (source: media\_aluno.c)

Elabore um programa que recebe os dados do aluno (nome e matrícula) e suas notas (P1, P2 e T) (usando a struct ALUNO). Em seguida, calcule a media aritmética das notas (P1, P2 e T), armazene na variável **media** e a exiba para o usuário.

```
typedef struct Aluno {  
    char nome[40];  
    int matricula;  
    float P1;  
    float P2;  
    float T;  
    float media;  
} ALUNO;
```

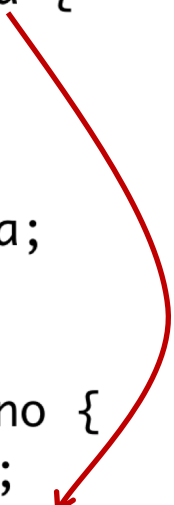
# Exercício (source: imprimeDados.c)

Implemente uma função **imprimeDados()** que receba o aluno criado e imprima os valores deste aluno dentro da função.

# Structs aninhadas

- Para criar estruturas mais complexas, é possível definir TADs aninhados, como no exemplo a seguir:

```
typedef struct Nota {  
    float P1;  
    float P2;  
    float T;  
    float media;  
} NOTA;  
  
typedef struct Aluno {  
    char *nome;  
    int matricula;  
    NOTA *nota; Struct aninhada  
} ALUNO;
```



# Structs aninhadas

- É importante ressaltar que é preciso inicializar os ponteiros das *structs* aninhadas;

```
int main() {  
  
    ALUNO *aluno1;  
  
    aluno1 = (ALUNO*) malloc(sizeof(ALUNO));  
    aluno1->nota = (NOTA*) malloc(sizeof(NOTA));  
  
    aluno1->nome = "Henrique";  
    aluno1->matricula = 12345;  
    aluno1->nota->P1 = 7; Acesso a um item de uma variável aninhada  
    aluno1->nota->P2 = 5;  
    aluno1->nota->T = 8;  
    aluno1->nota->media = (aluno1->nota->P1 +  
                           aluno1->nota->P2 +  
                           aluno1->nota->T) / 3;  
  
    printf("\nA media do aluno %s é: %.2f\n",  
           aluno1->nome, aluno1->nota->media);  
  
    free(aluno1->nota); A liberação de memória também deve ser  
    free(aluno1); realizada nos ponteiros internos (aluno1->nota)  
    return 0; da struct principal (aluno1).  
}
```

# Exercício (source: ex\_vetAlunos.c)

Usando a *struct* ALUNO, faça um programa que cadastre 2 alunos usando o conceito de vetor de *structs*. Para isso, deverão ser usadas as *structs* ALUNO e NOTA. O programa deverá imprimir os dados dos dois alunos cadastrados.

# Exercício (source: ex\_calcMediaImprimeDados.c)

Usando a estrutura `ALUNO`, faça um programa que realize o cadastro de 5 alunos usando o conceito de *struct*. Após isso, criar duas funções: **`float CalcMedia(ALUNO *a, int qtde)`** que recebe o vetor de alunos e retorna a media das notas P1 destes 5 alunos; e **`void imprime(ALUNO *a, int qtde)`** que recebe este vetor de alunos imprima os dados dos 5 alunos cadastrados.