

# Estrutura de Dados 2

## Aula 2

### Tipos Abstratos de Dados (TADs)



# Tópicos da aula

---

- **Tipos abstratos de dados (TAD):**

- Elementos de um TAD:
  - Atributos;
  - Operações.

- **Modularização baseada em TAD:**

- Exemplo de projeto.

# Introdução

- **Definição:**

Um tipo abstrato de dados (TAD) é estrutura de dados que define um conjunto de operações possíveis sobre os dados, sem expor os detalhes internos de como essas operações são implementadas.

- **Vantagens do TAD:**

- Meio eficaz de organizar e estruturar o código, promovendo:
  - Encapsulamento;
  - Modularidade;
  - Reutilização;
  - Manutenção eficiente.

# Introdução

- **Exemplo de TAD (dicionário de dados):**
  - Um dicionários é capaz de armazenar e organizar informações em pares chave-valor:
    - Armazena pares (chaves, valor) ou (K: *Key*, V: *Value*);
    - Pares chave-valor podem ser de qualquer tipo;
    - Pares-chave e valor podem ser inclusive iguais;
    - Itens com a mesma chave ficam na mesma posição do dicionário.

# Introdução

- **Exemplo de TAD (dicionário de dados):**

- Estrutura:

Chave	Valor
$K_0$	$V_0$
$K_1$	$V_1$
...	
$K_n$	$V_n$

Algumas aplicações

- Sistema bancário:

- Recuperação dos dados de uma conta;
- Chave de busca: número da conta;
- Valor: informações da conta.

- Sistema acadêmico:

- Recuperar os dados de um aluno;
- Chave de busca: registro acadêmico;
- Valor: histórico, notas e horários.

# Introdução

- **Exemplo de TAD (dicionário de dados):**
  - Interface:
    - **Busca:** percorre o dicionário pela chave.
    - **Inclusão:** inclui uma chave no dicionário.
    - **Remoção:** remove o conteúdo e a chave informada (se existir).
  - Implementação:
    - Árvore de busca binária (BST);
    - Árvore de busca balanceada;
    - Tabela de espalhamento;
    - ...

# **Tipos abstratos de dados (TAD)**

# Tipos abstratos de dados (TAD)

---

- **Definições:**

- Um TAD será composto por atributos e operações (métodos).
- Todos os acessos a um tipo de dado abstrato devem ser feito por meio de suas operações:
  - Contudo, seremos flexíveis nessa questão;



# Tipos abstratos de dados (TAD)

---

- **Passos para criar um TAD:**

- 1) Definir o tipo;
- 2) Definir um construtor para o tipo (aloca memória);
- 3) Definir um destrutor para o tipo (libera memória);
- 4) Definir as operações que serão feitas sobre o tipo de dado;
- 5) Implementar as operações do tipo de dado (protótipos);
- 6) Implementar operações internas (não visíveis ao usuário do TAD).

# Tipos abstratos de dados (TAD)

- **Convenções:**

- Iremos adotar os seguintes padrões de codificação:
  - Nome do TAD sempre deve ter a primeira letra maiúscula;
  - Nomes das operações devem ter primeira letra minúscula;
  - Usaremos o estilo CamelCase para nomes compostos:
    - Exemplo 1: notas\_aluno → notasAluno.

# Tipos abstratos de dados (TAD)

- **Convenções:**

- Normalmente, em C, um TAD é implementado por 2 arquivos:
  - Arquivo de cabeçalho (.h):
    - Especifica as declarações do tipo e de suas operações (interface).
  - Arquivo de implementação (.c):
    - Define as implementações das operações do tipo.

**Dica:** defina o mesmo nome para os arquivos .h e .c

# Utilizando TADs

# Utilizando TADs

## Exercício de sala de aula:

Escreva um programa para cadastramento e pesquisa de alunos. Um aluno possui duas informações, RA e nota. O usuário irá cadastrar alunos até que informe o RA zero. Nesse caso, a entrada de dados deverá ser finalizada. Para armazenar os dados dos alunos você deverá implementar um array dinâmico. Assim, sempre que o array atingir sua capacidade máxima devemos dobrar seu tamanho. Dentre as operações que devem ser implementadas estão:

- Inserir aluno;
- Buscar aluno;
- Exibir lista de alunos cadastrados;
- Exibir nota de um aluno;
- Exibir RA de um aluno.

# Utilizando TADs

---

- **Arquivos utilizados no projeto:**
  - Aluno:
    - Aluno.h
    - Aluno.c
    - DynamicArray.h
    - DynamicArray.c
    - main

# Utilizando TADs

- **Arquivos utilizados no projeto:**

- Aluno:

- Aluno.h
      - Aluno.c
    - DynamicArray.h
    - DynamicArray.c
    - main
- } Durante a aula vamos analisar somente o TAD Aluno.

# Utilizando TADs

- **Arquivos utilizados no projeto:**

- Aluno:

- Aluno.h

- Aluno.c

- DynamicArray.h

- DynamicArray.c

- main

} A implementação do array dinâmico deve ser realizada como exercício extraclasse.



# Utilizando TADs

- **TAD: Aluno (arquivo.h):**

```
#ifndef ALUNO_H_INCLUDED
#define ALUNO_H_INCLUDED

typedef struct{
    int ra;
    float nota;
}Aluno;

Aluno * criarAluno();
void destruirAluno(Aluno *aluno);
void definirRa(Aluno *aluno, int ra);
void definirNota(Aluno *aluno, float nota);
void exibirRa(Aluno *aluno);
void exibirNota(Aluno *aluno);

#endif // ALUNO_H_INCLUDED
```

# Utilizando TADs

- **TAD: Aluno (arquivo.h):**

```
#ifndef ALUNO_H_INCLUDED  
#define ALUNO_H_INCLUDED
```

```
typedef struct{  
    int ra;  
    float nota;  
}Aluno;
```

Iremos declarar a struct no arquivo.h. Dessa forma, permitimos que a struct seja acessada por quem utilizar o TAD

```
Aluno * criarAluno();  
void destruirAluno(Aluno *aluno);  
void definirRa(Aluno *aluno, int ra);  
void definirNota(Aluno *aluno, float nota);  
void exibirRa(Aluno *aluno);  
void exibirNota(Aluno *aluno);  
  
#endif // ALUNO_H_INCLUDED
```

# Utilizando TADs

- **TAD: Aluno (arquivo.h):**

```
#ifndef ALUNO_H_INCLUDED
#define ALUNO_H_INCLUDED

typedef struct{
    int ra;
    float nota;
}Aluno;
```

Especificação das operações realizadas sobre o TAD.

```
Aluno * criarAluno();
void destruirAluno(Aluno *aluno);
void definirRa(Aluno *aluno, int ra);
void definirNota(Aluno *aluno, float nota);
void exibirRa(Aluno *aluno);
void exibirNota(Aluno *aluno);
```

```
#endif // ALUNO_H_INCLUDED
```

# Utilizando TADs

- **TAD: Aluno (arquivo.c):**

```
#include <stdlib.h>
#include <stdio.h>
#include "Aluno.h"

Aluno * criarAluno(){
    Aluno *aluno = (Aluno *) malloc(sizeof(Aluno));

    if(aluno != NULL){
        aluno->ra = 0;
        aluno->nota = 0.0;
    }
    return aluno;
}

void destruirAluno(Aluno *aluno){
    free(aluno);
}
```

# Utilizando TADs

- **TAD: Aluno (arquivo.c):**

```
#include <stdlib.h>
#include <stdio.h>
#include "Aluno.h"
```

Inclusão do header Aluno.h

```
Aluno * criarAluno(){
    Aluno *aluno = (Aluno *) malloc(sizeof(Aluno));

    if(aluno != NULL){
        aluno->ra = 0;
        aluno->nota = 0.0;
    }
    return aluno;
}

void destruirAluno(Aluno *aluno){
    free(aluno);
}
```

# Utilizando TADs

- **TAD: Aluno (arquivo.c):**

```
#include <stdlib.h>
#include <stdio.h>
#include "Aluno.h"
```

```
Aluno * criarAluno(){
    Aluno *aluno = (Aluno *) malloc(sizeof(Aluno));

    if(aluno != NULL){
        aluno->ra = 0;
        aluno->nota = 0.0;
    }
    return aluno;
}
```

```
void destruirAluno(Aluno *aluno){
    free(aluno);
}
```

**Construtor:** alocamos dinâmica um Aluno

# Utilizando TADs

- **TAD: Aluno (arquivo.c):**

```
#include <stdlib.h>
#include <stdio.h>
#include "Aluno.h"

Aluno * criarAluno(){
    Aluno *aluno = (Aluno *) malloc(sizeof(Aluno));

    if(aluno != NULL){
        aluno->ra = 0;
        aluno->nota = 0.0;
    }
    return aluno;
}

void destruirAluno(Aluno *aluno){
    free(aluno);
}
```

**Destrutor:** desaloca o espaço de memória reservado para o Aluno

# Utilizando TADs

- **TAD: Aluno** ([arquivo.c](#)):

```
void definirRa(Aluno *aluno, int ra){
    aluno->ra = ra;
}

void definirNota(Aluno *aluno, float nota){
    aluno->nota = nota;
}

void exibirRa(Aluno *aluno){
    printf("ra: %d\n", aluno->ra);
}

void exibirNota(Aluno *aluno){
    printf("nota: %.2f\n", aluno->nota);
}
```

Outras operações que podem ser realizadas sobre o TAD aluno



# Utilizando TADs

- **TAD: Aluno (arquivo.c):**

```
#include <stdlib.h>
#include <stdio.h>
#include "Aluno.h"

Aluno * criarAluno(){
    Aluno *aluno = (Aluno *) malloc(sizeof(Aluno));

    if(aluno != NULL){
        aluno->ra = 0;
        aluno->nota = 0.0;
    }
    return aluno;
}

void destruirAluno(Aluno *aluno){
    free(aluno);
}
```

**Destrutor:** desaloca o espaço de memória reservado para o Aluno

# Utilizando TADs

- **TAD: main:**

```
#include <stdio.h>
#include <stdlib.h>
#include "Aluno.h"

int main()
{
    //criando alunos:
    Aluno *a1 = criarAluno();
    Aluno *a2 = criarAluno();

    if(a1 != NULL){
        printf("exibindo dados do aluno 1:\n");

        definirNota(a1, 8.7);
        definirRa(a1, 1204251);

        exibirNota(a1);
        exibirRa(a1);

        destruirAluno(a1);
    }
    else{
        printf("Erro: memória insuficiente\n");
        exit(1);
    }

    if(a2 != NULL){
        printf("\nexibindo dados do aluno 2:\n");

        definirNota(a2, 6);
        definirRa(a2, 2260000);

        exibirNota(a2);
        exibirRa(a2);

        destruirAluno(a2);
    }
    else {
        printf("Erro: memória insuficiente\n");
        exit(1);
    }

    return 0;
}
```

# Utilizando TADs

- **TAD: main:**

```
#include <stdio.h>
#include <stdlib.h>
#include "Aluno.h"
```

Incluindo o TAD Aluno

```
int main()
{
    //criando alunos:
    Aluno *a1 = criarAluno();
    Aluno *a2 = criarAluno();

    if(a1 != NULL){
        printf("exibindo dados do aluno 1:\n");

        definirNota(a1, 8.7);
        definirRa(a1, 1204251);

        exibirNota(a1);
        exibirRa(a1);

        destruirAluno(a1);
    }
    else{
        printf("Erro: memória insuficiente\n");
        exit(1);
    }

    if(a2 != NULL){
        printf("\nexibindo dados do aluno 2:\n");

        definirNota(a2, 6);
        definirRa(a2, 2260000);

        exibirNota(a2);
        exibirRa(a2);

        destruirAluno(a2);
    }
    else {
        printf("Erro: memória insuficiente\n");
        exit(1);
    }

    return 0;
}
```

# Utilizando TADs

- **TAD: DynamicArray:**

- Implemente o array dinâmico para cadastrar novos alunos tal como especificado no enunciado do problema. Sua implementação pode ser construída com base na seguinte struct:

```
#ifndef DYNAMICARRAY_H_INCLUDED
#define DYNAMICARRAY_H_INCLUDED

#include "Aluno.h"

typedef struct{
    Aluno *alunos;
    int capacidade;
    int i;
}DynamicArray;

#endif // DYNAMICARRAY_H_INCLUDED
```

# Utilizando TADs

- **TAD: DynamicArray:**

- Implemente o array dinâmico para cadastrar novos alunos tal como especificado no enunciado do problema. Sua implementação pode ser construída com base na seguinte struct:

```
#ifndef DYNAMICARRAY_H_INCLUDED
#define DYNAMICARRAY_H_INCLUDED
```

```
#include "Aluno.h"
```

```
typedef struct{
    Aluno *alunos;
    int capacidade;
    int i;
}DynamicArray;
```



*alunos*: ponteiro para um vetor de Alunos;  
*capacidade*: tamanho atual do vetor de Alunos;  
*i*: posição (índice) do último aluno inserido.

```
#endif // DYNAMICARRAY_H_INCLUDED
```

# Utilizando TADs

- **TAD: DynamicArray:**

- Implemente o array dinâmico para cadastrar novos alunos tal como especificado no enunciado do problema. Sua implementação pode ser construída com base na seguinte struct:

```
#ifndef DYNAMICARRAY_H_INCLUDED
#define DYNAMICARRAY_H_INCLUDED
```

```
#include "Aluno.h"
```

```
typedef struct{
    Aluno *alunos;
    int capacidade;
    int i;
}DynamicArray;
```



*alunos*: ponteiro para um vetor de Alunos;  
*capacidade*: tamanho atual do vetor de Alunos;  
*i*: posição (índice) do último aluno inserido.

```
#endif // DYNAMICARRAY_H_INCLUDED
```

**Dica:** dobre o tamanho do vetor sempre que ele atingir sua capacidade (realocação)

# Utilizando TADs

- **Compilação via terminal:**

- Como o programa está dividido em vários arquivos precisamos considerar cada arquivo durante a compilação.
- Para compilar o projeto desenvolvido neste aula utilize o seguinte comando em seu terminal (dentro do diretório dos arquivos):
  - `gcc main.c Aluno.c -o exe`
- Para executar o programa resultante (exe), use o comando:
  - `./exe`

**Dúvidas?**

---