ESTRUTURA DE DADOS I

Aula 09

Prof. Sérgio Luis Antonello

"A mente que se abre a uma nova ideia jamais voltará ao seu tamanho original“ (Albert Einstein)

FHO - Fundação Hermínio Ometto

28/04/2025

Plano de Ensino

1. Unidade I – Métodos de ordenação em memória principal (objetivos d, e, f)
   1.1. Revisão de tipos de dados básicos em C, variáveis indexadas e recursividade
   1.2. Noções de complexidade computacional
   1.3. Conceitos e métodos de ordenação de dados
   1.4. Bubblesort, Insertsort e Selectsort
   1.5. Quicksort e Mergesort
   1.6. Shellsort e Radixsort
2. Unidade II – Métodos de pesquisa em memória principal (objetivos e, f)
   2.1. Pesquisa sequencial
   2.2. Pesquisa binária
   2.3. Hashing
3. Unidade III – Tipo abstrato de dados (TAD) (objetivo a)
   3.1. Revisão de registros, ponteiros e alocação dinâmica de memória
   3.2. Tipo abstrato de dados (TAD): conceitos e aplicações
4. Unidade IV – Estrutura de dados lineares (objetivos a, b, c)
   4.1. Lista Encadeada: conceitos e aplicações
   4.2. Pilha: conceitos e aplicações
   4.3. Fila: conceitos e aplicações

Cronograma do Plano de Ensino

- 28/04 - Devolutiva P1; Tipo Abstrato de Dados (TAD).
- 05/05 - Conceitos de estruturas lineares: Lista ligada; Pilha; Fila.
- 12/05 - Algoritmos para de Lista Simplesmente Encadeada.
- 19/05 - Algoritmos para de Lista Simplesmente Encadeada.
- 26/05 - Implementação de Pilha e de Fila.
- 02/06 - Semana Científica do Curso.
- 09/06 - Desenvolvimento do trabalho A2.
- **16/06 - Prova 2.**
- **23/06 - Prova SUB.**

Sumário

- Primeiro momento (revisão)
  - Devolutiva da P1
- Segundo momento
  - Registro (revisão)
  - Ponteiro (revisão)
  - Alocação dinâmica de memória (revisão)
  - Tipo Abstrato de Dados (TAD)
- Terceiro momento (síntese)
  - Retome pontos importantes da aula

1. Primeiro momento: revisão

- Devolutiva da Prova 1

2. Segundo momento

- Registro
- Ponteiro
- Alocação dinâmica de memória
- Tipo Abstrato de Dados (TAD)

3. Alocação de memória (resgate de conteúdo)

- Vídeo 1
- Vídeo 2

3. Alocação de memória

- Alocação Estática
  - int i, cont, x;
  - char nome[40], status, opcao;
  - float salario, irpf;
- Alocação Dinâmica
  - ponteiro = malloc(40);
  - p = malloc(4);
  - p = malloc(sizeof(int));
  - p = malloc(10 * sizeof(int));
  - p = malloc(sizeof(struct aluno));
  - p = malloc(30 * sizeof(struct aluno));

4. Registro (resgate de conteúdo)

- Registro

Registro Funcionário	
Matricula	Tipo Inteiro
Nome	Tipo Cadeia de Caracteres
Dt Nascimento	Tipo Data
Cargo	Tipo Cadeia de Caracteres
Salário	Tipo Real

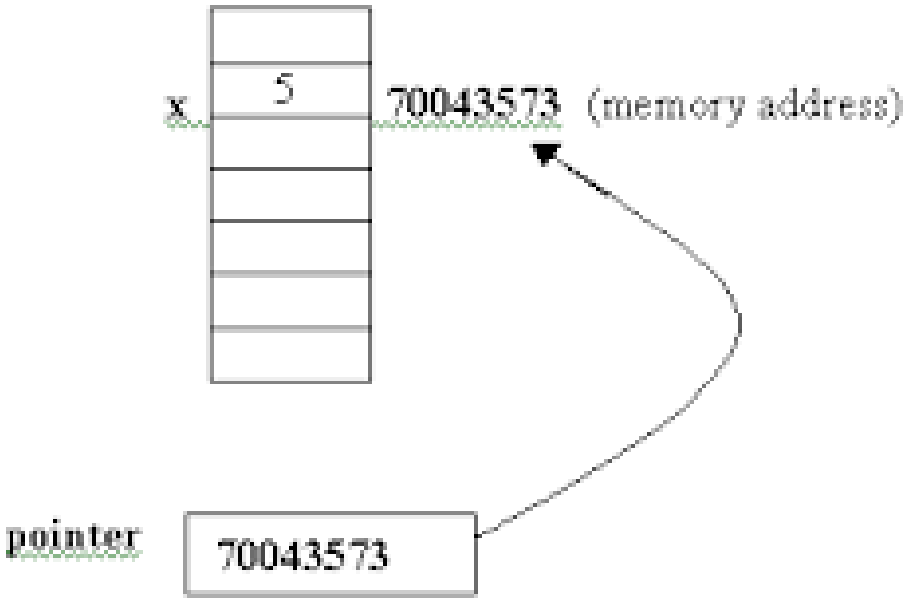
4. Registro: typedef

```
typedef struct dadosPessoal {
    int   matricula;
    char  nome[50];
    char  datanasc[8];
    char  cargo[40];
    float salario;
} cadastroPessoal;

int main () {
    // declaração da variável do tipo registro
    cadastroPessoal funcionario;
    ...
    // trabalhando com atributos da estrutura
    funcionario.matricula = 721;
    funcionario.salario = 8750,00;
    strcpy(funcionario.cargo, "Desenvolvedor PL");
}
```

5. Ponteiro (resgate de conteúdo)

- Ponteiro



5. Ponteiro

- É utilizado para armazenar um endereço de memória.
- O ponteiro deve ser do mesmo tipo do conteúdo armazenado neste endereço.
- O que caracteriza a declaração de um ponteiro é o uso de um asterisco (*) na frente do nome do mesmo.
- int *p;

5. Ponteiro

- Para atribuir um valor a um ponteiro deve-se copiar para ele um endereço de memória.
- Se o endereço desejado for de uma variável declarada no programa, basta utilizar o & (e-comercial) antes do nome da variável, que seu endereço estará sendo utilizado.
- Exemplos:

```
int var;
int *p;
var = 10;
p = &var;
```

5. Aritmética de ponteiro

- Deslocamentos na memória podem ser feitos a partir do endereço de memória armazenado no ponteiro.
- O deslocamento (quantidade de bytes na memória) é proporcional ao tipo que o ponteiro foi declarado.
  - Exemplo: suponha um ponteiro p
    - p++; avança para a próxima posição de memória a partir da posição que p aponta.
    - p+=2; avança duas posições de memória a partir da posição que p aponta.
    - p--; retroage uma posição de memória a partir da posição que p aponta.

6. Alocação dinâmica de registros

```
typedef struct dadosPessoal {
    int   matricula;
    char  nome[50];
    char  datanasc[8];
    char  cargo[40];
    float salario;
} cadastroPessoal;

int main () {
    // declaração da variável do tipo registro
    cadastroPessoal *func;

   // alocação do vetor (de 10 posições)
  func = (cadastroPessoal*) malloc(10*sizeof(cadastroPessoal));

}
```

6. Alocação dinâmica de registros

```
typedef struct dadosPessoal {
    int   matricula;
    char  nome[50];
    char  datanasc[8];
    char  cargo[40];
    float salario;
} cadastroPessoal;

int main () {
    // declaração da variável do tipo registro
    cadastroPessoal *func;

    // Para acessar os atributos do registro
    // usa-se o ponteiro
    (*func).matricula = 31;
    (*func).salario = 5460.00;
}
```

6. Alocação dinâmica de registros

```
typedef struct dadosPessoal {
    int   matricula;
    char  nome[50];
    char  datanasc[8];
    char  cargo[40];
    float salario;
} cadastroPessoal;

int main () {
    // declaração da variável do tipo registro
    cadastroPessoal *func;

    // sintaxe alternativa para acessar os atributos
    // do registro por meio do ponteiro
    func->matricula = 31;
    func->salario = 5460.00;
}
```

6. Alocação dinâmica de registros

- Preenchendo um vetor de registros

```
int main() {
    // declaração da variável do tipo registro
    cadastroPessoal *func;
    int i;

    // alocação do vetor (de 10 posições)
    func = (cadastroPessoal*) malloc(10*sizeof(cadastroPessoal));

    for (i = 0; i < 10; i++) {
        // entrada de dados
        scanf("%d", &func[i].matricula);

        // ou por atribuiçao
        func[i].salário = 7240.00;
    }
}
```

6. Alocação dinâmica de registros

- Atribuindo o conteúdo de uma variável a um elemento do vetor

```
int main() {
    // declaração da variável do tipo registro
    cadastroPessoal *func;
    cadastroPessoal aux;
    int i;


    // alocação do vetor (de 10 posições)
    func = (cadastroPessoal*) malloc(10*sizeof(cadastroPessoal));

    for (i = 0; i < 10; i++) {
        aux.matricula = 427;
        aux.salário = 7240.00;

        func[i] = aux;
    }
}
```

6. Alocação dinâmica de registros

- Recuperando os dados de um vetor de registros

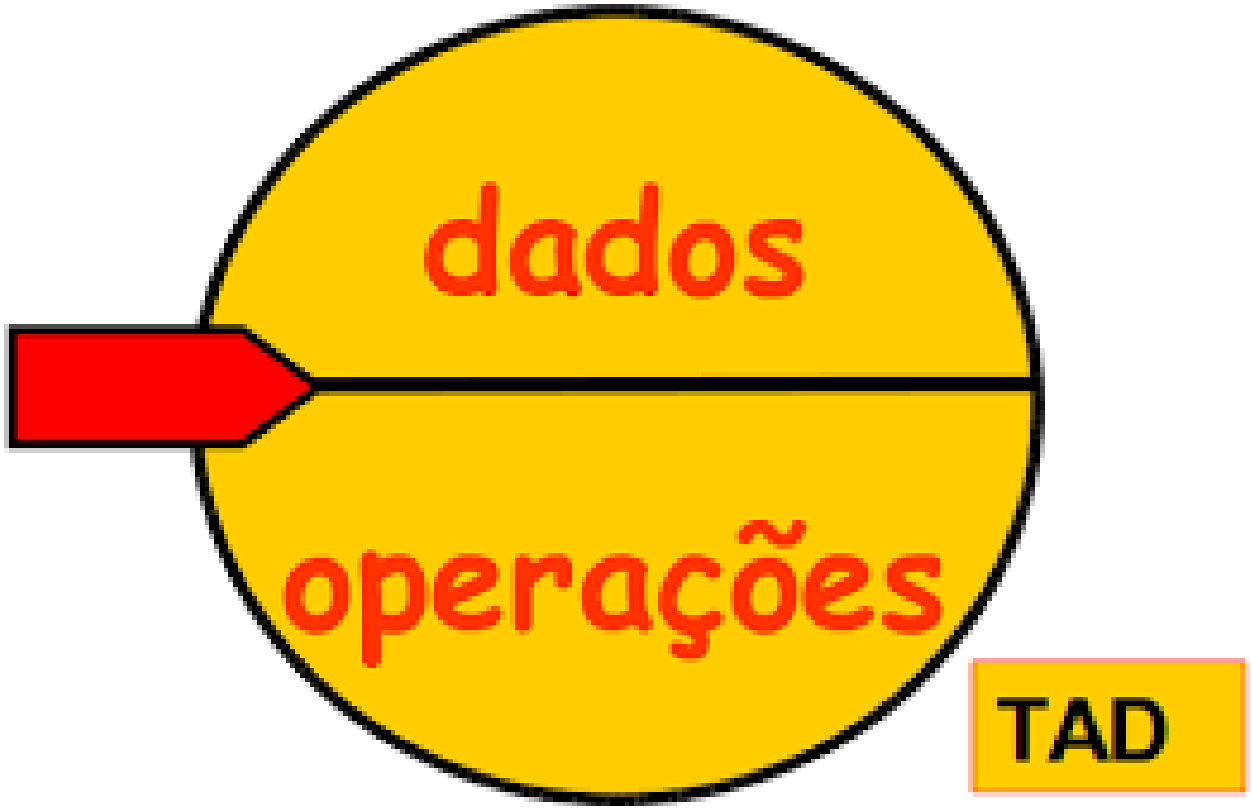
```
int main() {

    ...

    for (i = 0; i < 10; i++) {
        printf(“Matricula: %d\n", func[i].matricula);
        printf("Nome: %s\n", func[i].nome);
        printf("Idade: %d\n", func[i].salario);
        ...
    }
}
```

7. Tipo Abstrato de Dados

- Tipo Abstrato de Dados (TAD)



7. Tipo Abstrato de Dados (TAD)

- Tipos de dados podem ser vistos como métodos para interpretar o conteúdo da memória do computador
  - não pense em termos do que um computador pode fazer (interpretar os bits...), mas pense em termos do que os programadores desejam fazer (exemplo: somar dois inteiros).
  - O programador não se importa muito com a representação no hardware, mas sim com o conceito matemático de inteiro.
  - Um tipo inteiro ´suporta´ certas operações ...

7. Tipo Abstrato de Dados (TAD)

- Do ponto de vista do programador, muitas vezes é conveniente pensar nas estruturas de dados em termos das operações que elas suportam (interface), e não da maneira como elas são implementadas.
- Uma estrutura de dados definida dessa forma é chamada de um Tipo Abstrato de Dados (TAD)

7. Tipo Abstrato de Dados (TAD)

“Em computação, tipo abstrato de dado (TAD) é uma especificação de um conjunto de dados e operações que podem ser executadas sobre esses dados.

Além disso, é uma metodologia de programação que tem como proposta reduzir a informação necessária para a criação/programação de um algoritmo através da abstração das variáveis envolvidas em uma única entidade fechada, com operações próprias à sua natureza."

7. Tipo Abstrato de Dados (TAD)

- O TAD captura a essência (ideia) de um novo tipo de dado (inexistente na linguagem de programação utilizada), concebendo um conjunto de dados (variáveis) e um conjunto de operações (funções)

- TAD promove a separação entre conceito e implementação.
  - O TAD **encapsula** os dados, permitindo que usuários tenham acesso a apenas determinadas operações sobre esses dados.
  - Usuários só enxergam a **interface** e não a implementação!

7. Tipo Abstrato de Dados (TAD)

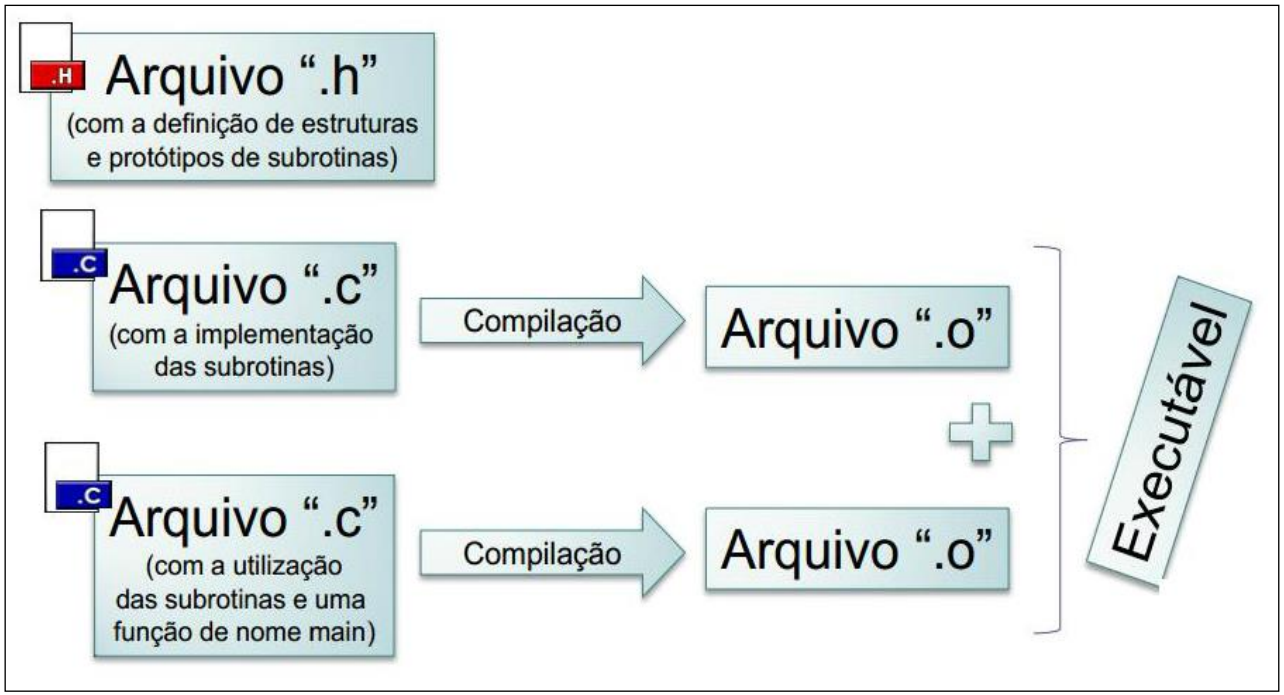
Exemplos de TAD

Mundo real	Dados	Operações
Estudante 	Nome Idade R.A. Notas Média	Matricular disciplina Gerar R.A. Atribuir nota Calcular média
Cartão de crédito 	Número Operadora Limite Saldo	Obter Cartão Vincular Compra Efetuar Pagamento Transação Fraudulenta?
Sacola 	Tamanho Máximo Número de Itens Itens	Adiciona Item Remove Item Sacola Vazia? Sacola Cheia?
TV 	Som Canal	Aumentar o volume Diminuir o volume Mudar de canal (para cima) Mudar de canal (para baixo)

7. Tipo Abstrato de Dados (TAD)

Para implementar um TAD, é recomendável:

- Arquivo “.h” (estruturas e declarações das operações)
- Arquivo “.c” (funções com as operações)
- Arquivo “.c” (programa que faz uso do TAD)



7. Tipo Abstrato de Dados (TAD)

- Arquivo “alunos.h” - header

```
#ifndef LISTA_ALUNOS_H
#define LISTA_ALUNOS_H
// Definição da estrutura para representar um aluno
typedef struct {
    int ra;
    char nome[50];
} Aluno;
// Definição do tipo opaco para a lista de alunos
typedef struct ListaAlunos ListaAlunos;
// Exemplos de operações para manipular a lista de alunos
ListaAlunos* criarListaAlunos();
void destruirListaAlunos(ListaAlunos* lista);
int inserirAlunoLista(ListaAlunos* lista, const Aluno* aluno);
int removerAlunoLista(ListaAlunos* lista, int matricula);
void imprimirListaAlunos(const ListaAlunos* lista);
#endif
```

7. Tipo Abstrato de Dados (TAD)

- Arquivo “lista_alunos_operacoes.c” - operações

```
#include "lista_alunos.h"

// Implementação da função para imprimir os alunos da lista
void imprimirListaAlunos(ListaAlunos* lista) {
    int i;
    for (i = 0; i < lista->tamanho; i++) {
        printf("Nome: %s, Matrícula: %d\n", lista->alunos[i].nome, lista->alunos[i].matricula);
    }
    printf("\n");
}
. . . Outras operações
```

7. Tipo Abstrato de Dados (TAD)

- Arquivo “ista_alunos_main.c” – uso da TAD

```
#include "lista_alunos.h"
int main (void) {
    // Criando uma nova lista de alunos
    ListaAlunos* minhaListaDeAlunos = criarListaAlunos();

   // Criando um aluno
    Aluno aluno1 = {"Ana Souza", 101};

    // Inserindo aluno na lista
    inserirAlunoLista(minhaListaDeAlunos, &aluno1);

   // Imprimindo a lista de alunos
    printf("Alunos na lista:\n");
    imprimirListaAlunos(minhaListaDeAlunos);
}
```

8. TAD: compilação

Compilar no DevC++

- Criar projeto
  - ter um único arquivo com main;
  - incluir arquivo com as operações;
  - incluir header;
  - não esquecer de #include "lista_alunos.h“ nos arquivos de operações e main.

- Criar arquivo fonte
  - Deixar os 3 arquivos abertos (main, operações e header);
  - Compilar sempre a partir do arquivo que contém a main;

8. TAD: compilação

- ✓ Compilar no prompt de comando (código .c)

  gcc -c operacoes.c (arquivo de operações)

  gcc -c main.c (arquivo com main)

  gcc main.obj operacoes.obj -o alunos.exe

- ✓ Usar o compilador g++ para códigos .cpp

- ✓ Executar no prompt de comando

  exemplo.exe

9. Exercícios

Vamos Praticar!

9. Exercícios

1) Use TAD para resolver o problema BEE 1410 - Ele está impedido

   https://judge.beecrowd.com/pt/problems/view/1410

9. Exercícios

2) Codificar um programa em linguagem C que leia, a partir do teclado, os seguintes dados de um aluno:
   - ✓ R.A. (número inteiro de pelo menos 6 dígitos);
   - ✓ Nome (string de até 20 caracteres);
   - ✓ Idade (número inteiro);
   - ✓ Nota média geral (float).

Armazenar os dados em um registro chamado *aluno* e declarar uma variável do tipo *aluno *a*.

Como saída, para cada dado de aluno, imprimir seu conteúdo, seu endereço e seu tamanho na memória.

Imprimir, também, o endereço de memória do ponteiro *a* e o espaço total ocupado na memória por **a*.

9. Exercícios

3) Modifique o programa do exercício anterior e faça com que a variável “a” seja um vetor de 5 posições, criado para armazenar os dados de 5 alunos diferentes (alocar o vetor dinamicamente).

   Além disso, no mesmo programa, preencha os dados dos 5 alunos, armazene os dados de cada aluno no vetor *a* e, por último, imprima os dados armazenados.

10. Terceiro momento: síntese

- Para implementar TAD é recomendável dividir o código em arquivos distintos:
  - Um arquivo com estruturas e declarações das operações;
  - Um arquivo com as funções com as operações;
  - Um arquivo com o programa que faz uso do TAD.