

MODO DE USAR

A **EdEstude** foi desenvolvida com o objetivo de fornecer implementações didáticas e modulares de diversas estruturas de dados na linguagem C, permitindo que estudantes e desenvolvedores possam integrá-la facilmente em seus projetos. O código-fonte está organizado em arquivos de cabeçalho (`.h`) e implementação (`.c`), estruturados de forma independente para facilitar a reutilização e manutenção do código.

Os arquivos da biblioteca estão disponíveis gratuitamente no repositório oficial do projeto:

<https://github.com/joaovgfreitas/EdEstude>

Após o download, basta extrair os arquivos para o diretório do seu projeto ou em um diretório destinado a bibliotecas externas.

Para utilizar a biblioteca em seu código, inclua o arquivo de cabeçalho correspondente ao módulo desejado no programa principal, por meio da diretiva `#include`. Por exemplo:

```
#include "pilha.h"
```

Em seguida, o programa pode ser compilado junto com os arquivos da biblioteca utilizando um compilador C compatível, como o GCC (GNU Compiler Collection) ou o próprio visual studio code, que atualmente conta com extensões para a compilação (Lembrando que é possível utilizar os códigos de teste e exemplos disponibilizados pela biblioteca para verificar se a inserção da mesma foi bem sucedida).

Após a compilação e execução bem sucedidas, o usuário poderá acessar todas as funcionalidades implementadas na **EdEstude**.

Por fim, é importante ressaltar que a **EdEstude** foi desenvolvida por um único estudante com o apoio do professor orientador, e é disponibilizada como software livre e de código aberto. É incentivada a colaboração e contribuição de qualquer pessoa para o projeto, podendo modificar e adicionar funções e estruturas livremente.

DOCUMENTAÇÃO

A biblioteca proposta tem como objetivo disponibilizar implementações didáticas e comentadas de estruturas de dados em linguagem C, visando facilitar o aprendizado e a prática de programação por parte dos estudantes. Seu código-fonte está organizado em módulos independentes, distribuídos em arquivos .h e .c, onde os arquivos .h contêm as declarações de funções, tipos e constantes utilizadas em outros módulos, tendo como prefixo o nome da estrutura correspondente. Essa divisão modular tem como propósito garantir melhor organização, reutilização e manutenção do código, permitindo que cada estrutura de dados seja desenvolvida e estudada de forma isolada e clara. O nome escolhido para a biblioteca foi **EdEstude**.

Estrutura dos Módulos

Os módulos foram organizados de acordo com as principais estruturas abordadas pela disciplina de estrutura de dados na maioria das universidades, sendo elas: Pilha, Fila, Lista encadeada, Árvore Binária de Busca, Árvore n-aria e Árvore AVL. Cada módulo refere-se a uma dessas estruturas, conforme pode ser observado no Quadro 1.

Quadro 1 - Módulos do EdEstude

Módulo	Descrição	Arquivo
Pilha	Implementa uma pilha sobre a estrutura LIFO (Last In First Out), o último a entrar será o primeiro a sair, com funções de inserção, remoção, exibir e destruir.	pilha.h
Fila	Implementa uma fila sobre a estrutura Estrutura FIFO (First In First Out), primeiro a entrar é o primeiro a sair, com funções de inserção, remoção, exibir e destruir.	fila.h
Lista Encadeada Dinâmica	Implementa uma lista encadeada, com inserção no início e fim da lista, remoção, exibir e liberar a lista.	listae.h

Continuação do Quadro 1

Módulo	Descrição	Arquivo
Árvore Binária de Busca	Implementa uma árvore binária de busca, com funções para criação, inserção, remoção e percursos entre nós.	arvorebb.h
Heap Máximo	Implementa uma árvore binária heap, que a chave de cada nó é maior ou igual (no max heap) à chave de seus filhos.	heap.h
Árvore AVL	Implementa uma árvore AVL com funções para criação, inserção, remoção, busca e liberar memória.	arvoreavl.h

Fonte: o Autor (2025)

Funções Implementadas

Neste tópico, as funções implementadas por esta biblioteca serão expostas e descritas, mostrando seus parâmetros (caso existam) e o que ela retorna (se houver), de acordo com cada módulo exposto no tópico anterior.

Pilha

A. Nome da função: **criarPilha()**

Descrição: Cria uma pilha, alocando memória para a estrutura da pilha, inicializa o topo e define a capacidade da pilha.

Parâmetros: **capacidade** (o tamanho da pilha)

Retorno: Retorna a pilha criada.

B. Nome da função: **estaVazia()**

Descrição: Verifica se a pilha está vazia.

Parâmetros: **p** (pilha criada).

Retorno: 1 (true) se estiver vazia, 0 (false) caso contrário.

C. Nome da função: **estaCheia()**

Descrição: Verifica se a pilha está cheia.

Parâmetros: **p** (pilha criada).

Retorno: 1 (true) se estiver vazia, 0 (false) caso contrário.

D. Nome da função: **empilhar()**

Descrição: Insere um elemento no topo da pilha.

Parâmetros: **p** (pilha criada) e **valor**

Retorno: Vazio

E. Nome da função: **desempilhar()**

Descrição: Remove e retorna o elemento do topo.

Parâmetros: **p** (pilha criada).

Retorno: topo atual (após remoção).

F. Nome da função: **top()**

Descrição: Retorna o elemento do topo (sem remover).

Parâmetros: **p** (pilha criada).

Retorno: valor do topo.

G. Nome da função: **destruirPilha()**

Descrição: Libera a memória alocada

Parâmetros: **p** (pilha criada).

Retorno: Vazio

Fila

A. Nome da função: **criarFila()**

Descrição: Cria uma fila e Inicializa a fila vazia.

Parâmetros: **capacidade**

Retorno: **f** (fila criada).

B. Nome da função: **filaVazia()**

Descrição: Verifica se a fila está vazia.

Parâmetros: **f** (fila criada).

Retorno: 1 se vazia, 0 caso contrário.

C. Nome da função: **filaCheia()**

Descrição: Verifica se a fila está cheia.

Parâmetros: **f** (fila criada).

Retorno: 1 se cheia, 0 caso contrário

D. Nome da função: **enfileirar()**

Descrição: Insere um elemento no final da fila.

Parâmetros: **f** (fila criada) e **valor**.

Retorno: Vazio.

E. Nome da função: **desenfileirar()**

Descrição: Remove o elemento do início da fila.

Parâmetros: **f** (fila criada).

Retorno: elemento removido.

F. Nome da função: **frente()**.

Descrição: Retorna o elemento do início da fila sem removê-lo.

Parâmetros: **f** (fila criada).

Retorno: elemento do início.

G. Nome da função: **exibirFila()**.

Descrição: Exibe todos os elementos da fila.

Parâmetros: **f** (fila criada).

Retorno: Vazio.

H. Nome da função: **destruirFila()**.

Descrição: Libera a memória utilizada pela fila.

Parâmetros: **f** (fila criada).

Retorno: Vazio.

Lista Encadeada Dinâmica

A. Nome da função: **criarNo()**.

Descrição: Função para criar um novo nó.

Parâmetros: **valor**.

Retorno: novo nó criado.

B. Nome da função: **inserirInicio()**.

Descrição: Insere um elemento no início da lista.

Parâmetros: **inicio** (nó criado) e **valor**.

Retorno: Vazio.

C. Nome da função: **inserirFim()**.

Descrição: Insere um elemento no início da lista.

Parâmetros: **inicio** (nó criado) e **valor**.

Retorno: Vazio.

D. Nome da função: **removerElemento()**.

Descrição: Remove o nó que contém o valor especificado.

Parâmetros: **inicio** (nó criado) e **valor**.

Retorno: Vazio.

E. Nome da função: **exibirLista()**.

Descrição: Percorre e imprime os elementos da lista.

Parâmetros: **inicio** (nó criado).

Retorno: Vazio.

F. Nome da função: **liberarLista()**.

Descrição: Libera toda a memória ocupada pelos nós.

Parâmetros: **inicio** (nó criado).

Retorno: Vazio.

Árvore de Busca Binária

A. Nome da função: **criarNo()**.

Descrição: Cria dinamicamente um novo nó com o valor informado.

Parâmetros: **valor**.

Retorno: novo nó criado.

B. Nome da função: **inserir()**.

Descrição: Insere o valor na árvore respeitando a propriedade da árvore binária de busca.

Parâmetros: **raiz** (do tipo nó) e **valor**.

Retorno: raiz (sem modificá-la).

C. Nome da função: **encontrarMinimo()**.

Descrição: Função auxiliar para encontrar o menor valor de uma subárvore, necessária para a função de remoção.

Parâmetros: **raiz** (do tipo nó).

Retorno: raiz.

D. Nome da função: **remove()**.

Descrição: Remover um valor da árvore.

Parâmetros: **raiz** (do tipo nó) e **valor**.

Retorno: raiz.

E. Nome da função: **buscar()**.

Descrição: Procura um valor na árvore recursivamente.

Parâmetros: **raiz** (do tipo nó) e **valor**.

Retorno: nó se encontrado ou NULL se não encontrado

F. Nome da função: **emOrdem()**.

Descrição: Percorre a árvore em ordem crescente.

Parâmetros: **raiz** (do tipo nó).

Retorno: Vazio.

G. Nome da função: **preOrdem()**.

Descrição: Mostra a raiz antes dos filhos.

Parâmetros: **raiz** (do tipo nó).

Retorno: Vazio.

H. Nome da função: **posOrdem()**.

Descrição: Mostra a raiz após os filhos.

Parâmetros: **raiz** (do tipo nó).

Retorno: Vazio.

- I. Nome da função: **liberarArvore()**.
Descrição: Libera toda a memória da árvore.
Parâmetros: **raiz** (do tipo nó).
Retorno: Vazio.

Heap Máximo

- A. Nome da função: **criarHeap()**.
Descrição: Cria um novo heap com uma capacidade dada.
Parâmetros: **capacidade**.
Retorno: **heap**.
- B. Nome da função: **trocar()**.
Descrição: Função para trocar dois elementos no vetor do heap.
Parâmetros: **a** e **b** (números quaisquer).
Retorno: Vazio.
- C. Nome da função: **heapifyCima()**.
Descrição: Função auxiliar para manter a propriedade do heap, “para cima”.
Parâmetros: **heap** e **index** (índice especificado).
Retorno: Vazio.
- D. Nome da função: **heapifyBaixo()**.
Descrição: Função auxiliar para manter a propriedade do heap, “para baixo”.
Parâmetros: **heap** e **index** (índice especificado).
Retorno: Vazio.
- E. Nome da função: **inserirHeap()**.
Descrição: Insere um novo elemento no heap.
Parâmetros: **heap** e **valor**.
Retorno: Vazio.
- F. Nome da função: **extrairMax()**.
Descrição: Remove e retorna o elemento máximo do heap.
Parâmetros: **heap**.
Retorno: valor máximo removido.

G. Nome da função: **obterMax()**.

Descrição: Obtém o elemento máximo sem removê-lo.

Parâmetros: **heap**.

Retorno: valor máximo.

H. Nome da função: **estaVazio()**.

Descrição: Verifica se o heap está vazio.

Parâmetros: **heap**.

Retorno: verdadeiro ou falso.

I. Nome da função: **obterTamanho()**.

Descrição: Verifica o tamanho atual do heap.

Parâmetros: **heap**.

Retorno: tamanho do heap.

J. Nome da função: **liberarHeap()**.

Descrição: Libera a memória alocada para o heap.

Parâmetros: **heap**.

Retorno: Vazio.

Árvore AVL

A. Nome da função: **altura()**.

Descrição: Obtém a altura de um nó.

Parâmetros: **n** (do tipo nó).

Retorno: altura.

B. Nome da função: **maior()**.

Descrição: Retorna o maior entre dois números

Parâmetros: **valor**.

Retorno: maior número.

C. Nome da função: **criarNo()**.

Descrição: Cria um novo nó a partir de um valor informado.

Parâmetros: **valor**.

Retorno: novo nó criado.

D. Nome da função: **fatorBalanceamento()**.

Descrição: Calcula a diferença entre altura da subárvore esquerda e direita.

Parâmetros: **n** (nó criado).

Retorno: diferença entre alturas das subárvores esquerda e direita.

E. Nome da função: **rotacaoEsquerda()**

Descrição: Reorganiza nós para a direita para corrigir desbalanceamentos.

Parâmetros: **y** (do tipo nó).

Retorno: nova raiz reorganizada para a esquerda

F. Nome da função: **rotacaoDireita()**.

Descrição: Reorganiza nós para a direita para corrigir desbalanceamentos.

Parâmetros: **x** (do tipo nó).

Retorno: nova raiz reorganizada para a direita.

G. Nome da função: **inserir()**

Descrição: Insere o valor e aplica rotações se a árvore ficar desbalanceada.

Parâmetros: **raiz** (do tipo nó) e **valor**.

Retorno: raiz balanceada

H. Nome da função: **menorNo()**.

Descrição: Função auxiliar para encontrar o menor nó (sucessor) de uma subárvore, utilizada para a remoção.

Parâmetros: **raiz** (do tipo nó).

Retorno: raiz atual.

I. Nome da função: **remove()**.

Descrição: Remove um valor da árvore AVL, considerando os casos de remoção.

Parâmetros: **raiz** (do tipo nó) e **valor**.

Retorno: raiz.

J. Nome da função: **emOrdem()**.

Descrição: Exibe os elementos em ordem crescente.

Parâmetros: **raiz** (do tipo nó).

Retorno: Vazio.

K. Nome da função: **liberarArvore()**.

Descrição: Libera a memória alocada pela árvore.

Parâmetros: **raiz** (do tipo nó).

Retorno: Vazio.

Exemplos de Utilização

Para uma melhor compreensão de como utilizar as funções expostas acima, será necessário uma exemplificação de cada módulo. Esta exemplificação, via código C, busca demonstrar como utilizar as funções, podendo servir também como um teste (executando esses códigos em sua máquina) para verificar se a biblioteca foi implementada corretamente.

O quadro 2 representa um exemplo da utilização do EdEstude para criar e usar uma pilha.

Quadro 2 - Exemplo de Pilha

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

int main() {
    // 5 é a capacidade da pilha (quantidade de itens)
    Pilha *p = criarPilha(5);

    empilhar(p, 10);
    empilhar(p, 20);
    empilhar(p, 30);

    //Verifica o topo atual da pilha
    printf("Topo atual da pilha: %d\n", top(p));

    exibirPilha(p); // Exibe todos os elementos da pilha

    printf("\nRemovendo: %d", desempilhar(p));
    printf("\nRemovendo: %d", desempilhar(p));

    printf("\nTopo final da pilha, apos remocao: %d", top(p));

    destruirPilha(p);
    return 0;
}
```

Fonte: o Autor (2025).

O quadro 3 representa um exemplo da utilização do EdEstude para criar e usar uma fila.

Quadro 3 - Exemplo de Fila

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

int main() {
    Fila *f = criarFila(5);

    enfileirar(f, 10);
    enfileirar(f, 20);
    enfileirar(f, 30);

    exibirFila(f);
    int removido = desenfileirar(f);
    printf("Elemento removido: %d\n", removido);
    exibirFila(f);

    printf("Elemento no inicio da fila: %d\n", frente(f));
    printf("Esvaziando a fila:\n");
    while (!filaVazia(f)) {
        printf("Removido: %d\n", desenfileirar(f));
    }
    exibirFila(f);

    destruirFila(f); // libera memória

    return 0;
}
```

Fonte: o Autor (2025).

O quadro 4 representa um exemplo da utilização do EdEstude para criar e usar uma Lista Encadeada.

Quadro 4 - Exemplo de Lista Encadeada

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

int main() {
    No* lista = NULL; // lista inicialmente vazia

    inserirInicio(&lista, 10);
    inserirFim(&lista, 20);
    inserirFim(&lista, 30);
}
```

Continuação do Quadro 4 - Exemplo de Lista Encadeada

```
    inserirInicio(&lista, 5);

    exibirLista(lista);

    removerElemento(&lista, 20);
    exibirLista(lista);

    liberarLista(&lista);
    return 0;
}
```

Fonte: o Autor (2025).

O quadro 5 representa um exemplo da utilização do EdEstude para criar e usar uma Árvore binária de busca.

Quadro 5 - Exemplo de Árvore Binária de Busca

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

int main() {
    No* raiz = NULL; // raiz da árvore

    raiz = inserir(raiz, 50);
    raiz = inserir(raiz, 30);
    raiz = inserir(raiz, 70);
    raiz = inserir(raiz, 20);
    raiz = inserir(raiz, 40);
    raiz = inserir(raiz, 60);
    raiz = inserir(raiz, 80);

    printf("Percurso em ordem crescente: ");
    emOrdem(raiz);
    printf("\n");

    printf("Percurso pre-ordem: ");
    preOrdem(raiz);
    printf("\n");

    printf("Percurso pos-ordem: ");
    posOrdem(raiz);
    printf("\n");

    printf("Removendo o valor 50\n");
}
```

Continuação do Quadro 5 - Exemplo de Árvore Binária de Busca

```
    raiz = remover(raiz, 50);

    printf("Buscando o valor 40\n");
    int valorBusca = 40;
    No* encontrado = buscar(raiz, valorBusca);
    if (encontrado)
        printf("Valor %d encontrado na arvore!\n", valorBusca);
    else
        printf("Valor %d nao encontrado.\n", valorBusca);

    liberarArvore(raiz);
    return 0;
}
```

Fonte: o Autor (2025).

O quadro 6 representa um exemplo da utilização do EdEstude para criar e usar um Heap Máximo.

Quadro 6 - Exemplo de Heap Máximo

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

int main() {
    Heap* heap = criarHeap(10);
    if (heap == NULL) {
        printf("Falha ao criar o heap!\n");
        return 1;
    }

    printf("Heap esta vazio? %s\n", estaVazio(heap) ? "Sim" :
"Nao");
    printf("Tamanho inicial do heap: %d\n", obterTamanho(heap));

    printf("Inserindo elementos: 10, 20, 15, 30, 25\n");
    inserirHeap(heap, 10);
    inserirHeap(heap, 20);
    inserirHeap(heap, 15);
    inserirHeap(heap, 30);
    inserirHeap(heap, 25);
    printf("Tamanho apos insercoes: %d\n", obterTamanho(heap));

    int max = obterMax(heap);
    printf("Elemento maximo: %d\n", max);

    printf("Extraindo elementos maximos:\n");
}
```

Continuação do Quadro 6 - Exemplo de Heap Máximo

```
while (!estaVazio(heap)) {
    int elemento = extrairMax(heap);
    if (elemento != -1) {
        printf("Elemento extraido: %d\n", elemento);
    }
}

printf("Tamanho apos extracoes: %d\n", obterTamanho(heap));
printf("Heap esta vazio? %s\n", estaVazio(heap) ? "Sim" :
"Nao");

printf("Inserindo elemento 50 apos extracoes\n");
inserirHeap(heap, 50);
printf("Elemento maximo apos nova insercao: %d\n",
obterMax(heap));
printf("Tamanho do heap: %d\n", obterTamanho(heap));

LiberarHeap(heap);
printf("Heap Liberado.\n");

return 0;
}
```

Fonte: o Autor (2025).

O quadro 7 representa um exemplo da utilização do EdEstude para criar e usar uma árvore AVL.

Quadro 7 - Exemplo de árvore AVL

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

int main() {
    No* raiz = NULL;

    raiz = inserir(raiz, 30);
    raiz = inserir(raiz, 20);
    raiz = inserir(raiz, 40);
    raiz = inserir(raiz, 10);
    raiz = inserir(raiz, 25);
    raiz = inserir(raiz, 50);

    printf("\nRemovendo 20...\n");
    raiz = remover(raiz, 20);
    printf("Percurso ordenado: ");
}
```

Continuação do Quadro 7 - Exemplo de árvore AVL

```
    emOrdem(raiz);  
    printf("\n");  
  
    LiberarArvore(raiz);  
    return 0;  
}
```

Fonte: o Autor (2025).