Árvore Binária Com Leitura De Arquivo

O código é um sistema de gerenciamento de dados de alunos em uma árvore binária de busca, proporcionando funcionalidades para adicionar, remover, buscar, imprimir e salvar informações dos alunos em um arquivo. É uma aplicação simples, mas ilustra conceitos importantes de estruturas de dados e manipulação de árvores binárias.

O código

main.c: Este arquivo contém a função main(), que é o ponto de entrada do programa. Ele configura a árvore binária, exibe um menu interativo para interação com o usuário e gerencia as ações escolhidas no menu.

```c
#include "tree.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {
  TreeNode *raiz;
  iniciaArvore(&raiz);
  menu(&raiz);

  return 0;
}
```

tree.c: Este arquivo contém a implementação das funções relacionadas à manipulação da árvore binária de busca. As operações incluem inicialização da árvore, criação de nós, inserção, remoção, busca e impressão em várias ordens. Além disso, também permite carregar e salvar dados em um arquivo.

iniciaArvore(TreeNode **arvore): Inicializa a árvore definindo a raiz como NULL, e prepara a árvore para receber dados.

```c
void iniciaArvore(TreeNode **arvore) { *arvore = NULL; }
```

criar_no(long long matricula, const char *nome): Aloca memória para criar um novo nó na árvore com informações de aluno - matrícula, nome e ponteiros para subárvores esquerda, direita e pai.

```c
TreeNode *criar_no(long long matricula, const char *nome) {
```

```
  TreeNode *novoNO = (TreeNode *)malloc(sizeof(TreeNode));
  strcpy(novoNO->nome, nome);
  novoNO->matricula = matricula;
  novoNO->esq = NULL;
  novoNO->dir = NULL;
  novoNO->pai = NULL;

  return novoNO;
}
```

entradaPerguntaUsuario(): Solicita ao usuário que insira uma matrícula.

```
long long entradaPerguntaUsuario() {
  long long matricula;

  printf("Informe a matricula: ");
  scanf("%lld", &matricula);

  return matricula;
}
```

inserir(TreeNode **arvore, long long matricula, const char *nome): Insere um novo aluno na árvore de forma ordenada com base na matrícula.

```
TreeNode *inserir(TreeNode **arvore, long long matricula, const
char *nome) {
  if ((*arvore) == NULL) {
    (*arvore) = criar_no(matricula, nome);
  } else if (matricula < (*arvore)->matricula) {
    (*arvore)->esq = inserir(&(*arvore)->esq, matricula, nome);
    (*arvore)->esq->pai = (*arvore); // Define o pai do nó à
esquerda
  } else {
    (*arvore)->dir = inserir(&(*arvore)->dir, matricula, nome);
    (*arvore)->dir->pai = (*arvore); // Define o pai do nó à
direita
  }
  return (*arvore);
}
```

carregarDadosDoArquivo(TreeNode **arvore): Lê dados de um arquivo e carrega informações dos alunos de um arquivo para a árvore

```
void carregarDadosDoArquivo(TreeNode **arvore) {
  long long mat;
  char nome[100];

  FILE *dado;
  dado = fopen("Lista_Aluno_Matricula_Atual.txt", "r");
  if (!dado) {
    printf("\nErro na abertura do arquivo
'Lista_Aluno_Matricula_Atual'\n");
    return;
  } else {
    while (fscanf(dado, "%lld ", &mat) == 1) {
      // Ler o nome até encontrar uma nova linha ou EOF
      if (fgets(nome, sizeof(nome), dado) != NULL) {
        // Remover o caractere de nova linha, se presente
        size_t len = strlen(nome);
        if (len > 0 && nome[len - 1] == '\n') {
          nome[len - 1] = '\0'; // Substitui a nova linha por um
terminador nulo
        }
        // Converte o nome para uma string
        char *nome_str = strdup(nome);
        // Inserir na árvore usando o método inserir personalizado
        inserir(arvore, mat, nome_str);
      }
    } // while
    fclose(dado);
  } // else
}
```

buscar(TreeNode *arvore, long long matricula): Localiza um nó com a matrícula especificada na árvore.

```
TreeNode *buscar(TreeNode *arvore, long long matricula) {
  TreeNode *no = arvore;

  while (no) {
    if (matricula == no->matricula) {
      return no;
```

```
    } else if (matricula < no->matricula) {
      no = no->esq;
    } else {
      no = no->dir;
    }
  }
  return NULL; // Retorna NULL se a matrícula não for encontrada.
}
```

encontrar_minimo(TreeNode *no): Encontra o nó com o valor mínimo (menor matrícula) na subárvore à esquerda de um nó específico na árvore binária de busca. Usado em operações de remoção na árvore, quando é necessário encontrar o nó com o valor mínimo na subárvore à direita de um nó que está sendo removido, para substituí-lo pelo nó com o valor mínimo. Essa operação é necessária para manter a propriedade da árvore binária de busca, onde os valores menores estão à esquerda e os valores maiores estão à direita.

```
TreeNode *encontrar_minimo(TreeNode *no) {
  while (no->esq) {
    no = no->esq;
  }
  return no;
}
```

remover(TreeNode **arvore, long long matricula): Remove um nó com a matrícula especificada, mantendo a estrutura da árvore.

```
void remover(TreeNode **arvore, long long matricula) {
TreeNode *no = buscar(*arvore, matricula);
  if (!no) {
    printf("Matrícula %lld não encontrada na árvore.\n",
matricula);
    return;
  }
  if (!no->esq) {
    transplantar(arvore, no, no->dir);
  } else if (!no->dir) {
    transplantar(arvore, no, no->esq);
  } else {
    TreeNode *sucessor = encontrar_minimo(no->dir);
    if (sucessor->pai != no) {
```

```
      transplantar(arvore, sucessor, sucessor->dir);
      sucessor->dir = no->dir;
      sucessor->dir->pai = sucessor;
    }
    transplantar(arvore, no, sucessor);
    sucessor->esq = no->esq;
    sucessor->esq->pai = sucessor;
  }
  free(no);
}
```

transplantar(TreeNode **arvore, TreeNode *alvo, TreeNode *novo_no): Atualiza os ponteiros de pai e filho para manter a estrutura da árvore após a remoção de um nó.

```
void transplantar(TreeNode **arvore, TreeNode *alvo, TreeNode
*novo_no) {
  if (!alvo->pai) {
    (*arvore) = novo_no;
  } else if (alvo == alvo->pai->esq) {
    alvo->pai->esq = novo_no;
  } else {
    alvo->pai->dir = novo_no;
  }
  if (novo_no) {
    novo_no->pai = alvo->pai;
  }
}
```

validarArvore(TreeNode *no): Verifica se a árvore está vazia e imprime uma mensagem correspondente.

```
void validarArvore(TreeNode *no) {
  if (no == NULL) {
    printf("\nArvore vazia.\n");
    return;
  }
}
```

imprimir_em_ordem(TreeNode *no): Imprime os elementos da árvore em ordem.

```
void imprimir_em_ordem(TreeNode *no) {
  if (no) {
    imprimir_em_ordem(no->esq);
    printf("\t Matrícula: (%lld) -- Nome: %s\n", no->matricula,
no->nome);
    imprimir_em_ordem(no->dir);
  }
}
```

imprimir_pre_ordem(TreeNode *no): Imprime os elementos da árvore em pré-ordem.

```
void imprimir_pre_ordem(TreeNode *no) {
  if (no) {
    printf("\t Matrícula: (%lld) -- Nome: %s\n", no->matricula,
no->nome);
    imprimir_pre_ordem(no->esq);
    imprimir_pre_ordem(no->dir);
  }
}
```

imprimir_pos_ordem(TreeNode *no): Imprime os elementos da árvore em pós-ordem.

```
void imprimir_pos_ordem(TreeNode *no) {
  if (no) {
    imprimir_pos_ordem(no->esq);
    imprimir_pos_ordem(no->dir);
    printf("\t Matrícula: (%lld) -- Nome: %s\n", no->matricula,
no->nome);
  }
}
```

percorreArv(TreeNode *no, FILE *dado): Percorre a árvore e escreve os dados dos alunos em um arquivo.

```
void percorreArv(TreeNode *no, FILE *dado) {
  if (no != NULL) {
    fprintf(dado, "%lld\n", no->matricula); // Escreve a matrícula
em uma linha
    fprintf(dado, "%s\n", no->nome); // Escreve o nome em uma
linha separada
    percorreArv(no->esq, dado);
    percorreArv(no->dir, dado);
  }
}
```

salvarArq(TreeNode *arvore): Abre um arquivo e chama `percorreArv` para salvar os dados dos alunos.

```
void salvarArq(TreeNode *arvore) {
  FILE *dado;
  dado = fopen("Lista_Aluno_Matricula_Atual.txt", "w");
  if (!dado) {
    printf("\nErro na abertura do arquivo
'Lista_Aluno_Matricula_Atual'\n");
  } else {
    percorreArv(arvore, dado);
    fclose(dado);
  }
}
```

menu(TreeNode **arvore): Exibe um menu interativo para o usuário interagir com a árvore, escolhendo várias operações para manipular a árvore.

```
void menu(TreeNode **arvore) {
  long long matriculaInfo;
  int opcao;
  char nomeArquivo[] = "Lista_Aluno_Matricula_Atual.txt";

  carregarDadosDoArquivo(arvore);

  do {
    imprimir_menu();
    scanf("%d", &opcao);
    switch (opcao) {
    case 1:
```

```
    printf("\nDigite a matrícula: ");
    long long matricula;
    scanf("%lld", &matricula);
    char nome[100];
    printf("Digite o nome: ");
    scanf("%s", nome);
    inserir(arvore, matricula, nome);
    break;
  case 2:
    matriculaInfo = entradaPerguntaUsuario();
    remover(arvore, matriculaInfo);
    break;
  case 3:
    matriculaInfo = entradaPerguntaUsuario();
    TreeNode *busca = buscar(*arvore, matriculaInfo);
    validarArvore(*arvore);
    imprimirResultadoBusca(busca, matriculaInfo);
    break;
  case 4:
    printf("\nSalvando dados no arquivo...\n");
    salvarArq(*arvore);
    printf("Base de dados salva com sucesso!\n");
    break;
  case 5:
    printf("\n\n---------- Árvore em ORDEM ----------\n\n");
    validarArvore(*arvore);
    imprimir_em_ordem(*arvore);
    printf("\n-----------------------------------------\n\n");
    break;
  case 6:
    printf("\n\n---------- Árvore em PRÉ-ORDEM ----------\n\n");
    validarArvore(*arvore);
    imprimir_pre_ordem(*arvore);
    printf("\n-----------------------------------------\n\n");
    break;
  case 7:
    printf("\n\n---------- Árvore em PÓS-ORDEM ----------\n\n");
    validarArvore(*arvore);
    imprimir_pos_ordem(*arvore);
    printf("\n-----------------------------------------\n\n");
    break;
  case 0:
```

```
      printf("Saindo do programa.\n");
      liberar_nos(*arvore);
      break;
    default:
      printf("Opção inválida. Tente novamente.\n");
      break;
    }
  } while (opcao != 0);
}
```

imprimir_menu():Imprime o menu com as opções disponíveis para o usuário.

```
void imprimir_menu() {
  printf("\n---------- * MENU * ----------\n");
  printf("\t1. Inserir\n");
  printf("\t2. Remover\n");
  printf("\t3. Buscar\n");
  printf("\t4. Salvar base de dados\n");
  printf("\t5. Imprimir em ordem\n");
  printf("\t6. Imprimir pré-ordem\n");
  printf("\t7. Imprimir pós-ordem\n");
  printf("\t0. Sair\n\n");
  printf("Escolha uma opção: ");
}
```

imprimirResultadoBusca(TreeNode *no, long long matricula_buscar): Exibe o resultado da busca por matrícula na árvore, além de informar se uma matrícula foi encontrada ou não na árvore.

```
void imprimirResultadoBusca(TreeNode *no, long long
matricula_buscar) {
  if (no) {
    printf("\n\tMatrícula ENCONTRADA\n");
    printf("\t\t * Matrícula: (%lld) - %s\n", no->matricula,
no->nome);
  } else {
    printf("\n\tMatrícula %lld - NAO ENCONTRADA\n",
matricula_buscar);
  }
}
```

liberar_nos(TreeNode *no): Libera a memória alocada para os nós da árvore.

```
void liberar_nos(TreeNode *no) {
  if (no) {
    liberar_nos(no->esq);
    liberar_nos(no->dir);
    free(no);
  }
}
```

tree.h: Contém a declaração das estruturas de dados e das assinaturas (protótipos) das funções utilizadas no código relacionado à manipulação de uma árvore binária de busca. Ele não contém a implementação real das funções, apenas fornece informações sobre como essas funções devem ser chamadas e quais tipos de argumentos elas aceitam.

```
#ifndef TREE_H
#define TREE_H

#include <stdio.h>

typedef struct tipoNo {
  long long matricula;
  char nome[100];
  struct tipoNo *pai;
  struct tipoNo *esq;
  struct tipoNo *dir;
} TreeNode;

void iniciaArvore(TreeNode **arvore);
TreeNode *criar_no(long long matricula, const char *nome);
long long entradaPerguntaUsuario();
TreeNode *inserir(TreeNode **arvore, long long matricula, const
char *nome);
void carregarDadosDoArquivo(TreeNode **arvore);
TreeNode *buscar(TreeNode *arvore, long long matricula);
TreeNode *encontrar_minimo(TreeNode *no);
void transplantar(TreeNode **arvore, TreeNode *alvo, TreeNode
*novo_no);
```

```
void validarArvore(TreeNode *no);
void imprimir_em_ordem(TreeNode *no);
void imprimir_pre_ordem(TreeNode *no);
void imprimir_pos_ordem(TreeNode *no);
void percorreArv(TreeNode *no, FILE *dado);
void salvarArq(TreeNode *arvore);
void menu(TreeNode **arvore);
void imprimir_menu();
void imprimirResultadoBusca(TreeNode *no, long long
matricula_buscar);
void liberar_nos(TreeNode *no);

#endif
```

A estrutura TreeNode serve para representar os nós individuais de uma árvore binária de busca. Ela contém os dados necessários para armazenar informações sobre os elementos (no caso, alunos) que estão sendo organizados na árvore. Essa estrutura é fundamental para a construção e manipulação da árvore binária de busca, pois cada nó contém referências aos seus filhos (nós à esquerda e à direita) e ao seu pai, permitindo a organização hierárquica dos elementos na árvore.