Autores: Lucas Garcia e Luis Augusto

Arquivo `.c` (implementação)

1. Função `abrirArquivo()`

- **Objetivo**: Abrir um arquivo para leitura ou escrita. Se o arquivo não puder ser aberto, o programa exibe uma mensagem de erro e é finalizado.

```c
FILE *abrirArquivo(char *nomeArq, char *modo) {
    FILE *arq = fopen(nomeArq, modo);
    if (arq == NULL) {
        printf("ERRO ao abrir o arquivo.\n");
        exit(-1);
    }
    printf("INFO: Arquivo Aberto! Bom uso.\n");
    return arq;
}
```

- **Explicação**: Esta função utiliza `fopen` para abrir um arquivo no modo especificado (leitura ou escrita). Caso o arquivo não seja encontrado, o programa imprime uma mensagem de erro e encerra.

2. Função `calcularTempo()`

- **Objetivo**: Calcular o tempo de execução de uma operação usando a função `clock_t` para medir o tempo.

```c
void calcularTempo(double ini, double fim) {
    double tempoDecorrido = (double)(fim - ini) / CLOCKS_PER_SEC;
    printf("Tempo de execucao: %f segundos\n", tempoDecorrido);
}
```

- **Explicação**: O tempo de execução de uma operação é calculado e impresso em segundos. É útil para medir o desempenho do programa.

3. Função `salvarDadosNoArquivo()`

- **Objetivo**: Salvar os dados da árvore binária em um arquivo de texto.

```c
void salvarDadosNoArquivo(NoArvore *no, FILE *arquivoLista) {
    if (no == NULL) return;

    salvarDadosNoArquivo(no->esquerdo, arquivoLista);
    fprintf(arquivoLista, "%s\n%lld\n", no->nome, no->matricula);
    salvarDadosNoArquivo(no->direito, arquivoLista);
}
```

- **Explicação**: A função percorre todos os nós da árvore e, para os nós ocupados, grava o nome e a matrícula no arquivo.

4. Função `inicializarArvore()`

- **Objetivo**: Inicializar uma árvore binária com uma capacidade baseada no número de matrículas esperado, multiplicado por um fator de segurança.

```c
*/
void inicializarArvore(ArvoreBinaria *arvore) {
    arvore->raiz = NULL;
    arvore->tamanho = 0;
}
```

- **Explicação**: A função aloca dinamicamente memória para a árvore, inicializando todos os nós como vazios. Usa um fator de segurança para alocar mais memória do que o necessário, prevenindo futuras alocações.

6. Função `caminhamento_Em_Ordem()`

- **Objetivo**: Exibir a árvore binária em ordem (ordem crescente de matrículas).

```c
void caminhamento_Em_Ordem(NoArvore *R){
    if(R != NULL){
        caminhamento_Em_Ordem(R->esquerdo);
        printf("Matrícula: %lld, Nome: %s\n", R->matricula, R->nome);
        caminhamento_Em_Ordem(R->direito);
    }//if
}
```

- **Explicação**: A árvore é percorrida recursivamente em ordem, começando pelo filho esquerdo, depois o nó raiz, e por último o filho direito, imprimindo os nós ocupados.

7. **Função** `caminhamento_Pre_Ordem()`

- **Objetivo**: Exibir a árvore binária em ordem (ordem crescente de matrículas).

```c
void caminhamento_Pre_Ordem(NoArvore *R){
    if(R != NULL){
        printf("Matrícula: %lld, Nome: %s\n", R->matricula, R->nome);
        caminhamento_Pre_Ordem(R->esquerdo);
        caminhamento_Pre_Ordem(R->direito);
    }//if
}
```

- **Explicação**: O caminhamento em pré-ordem é uma forma de percorrer uma árvore binária, que consiste em visitar cada nó antes de seus subnós, primeiro visita a raiz, depois percorre a subárvore esquerda em pré-ordem logo em seguida percorre a subárvore direita em pré-ordem e repete esse passo.

8. **Função** `caminhamento_Pos_Ordem()`

- **Objetivo**: Exibir a árvore binária em ordem (ordem crescente de matrículas).

```c
void caminhamento_Pos_Ordem(NoArvore *R){
    if(R != NULL){
        caminhamento_Pos_Ordem(R->esquerdo);
        caminhamento_Pos_Ordem(R->direito);
        printf("Matrícula: %lld, Nome: %s\n", R->matricula, R->nome);
    }//if
}
```

- **Explicação**: O caminhamento em pos-ordem é uma forma de percorrer uma árvore binária, que consiste em visitar cada sub nó antes de seu nó, primeiro Caminhar na subárvore à esquerda, depois caminhar na subárvore à direita logo em seguida visita a raiz e repete esse passo.

Função `inserirNo()`

- **Objetivo**: Inserir um aluno na árvore binária de acordo com a matrícula.

```c
NoArvore* inserirNo(NoArvore *no, long long int matricula, char *nome) {
    if (no == NULL) {
        no = (NoArvore *)malloc(sizeof(NoArvore));
        if (no == NULL) {
            printf("Erro de alocação de memória.\n");
            exit(1);
        }
        no->matricula = matricula;
        strcpy(no->nome, nome);
        no->esquerdo = no->direito = NULL;
    } else if (matricula < no->matricula) {
        no->esquerdo = inserirNo(no->esquerdo, matricula, nome);
    } else if (matricula > no->matricula) {
        no->direito = inserirNo(no->direito, matricula, nome);
    } else {
        printf("Matrícula já existente.\n");
    }
    return no;
}
```

- **Explicação**: A função insere um novo aluno na árvore. Se a árvore estiver cheia, é redimensionada. A inserção ocorre buscando a posição correta de acordo com a matrícula.

8. Função `buscarNo()`

- **Objetivo**: Buscar uma matrícula na árvore usando a matrícula.

```c
NoArvore* buscarNo(NoArvore *no, long long int matricula) {
    if (no == NULL) {
        printf("Aluno não encontrado.\n");
        return NULL;
    }
    if (matricula < no->matricula) {
        return buscarNo(no->esquerdo, matricula);
    } else if (matricula > no->matricula) {
        return buscarNo(no->direito, matricula);
    } else {
        printf("Aluno encontrado: %s (Matrícula: %lld)\n", no->nome, no->matricula);
        return no;
    }
}
```

- **Explicação**: A função percorre a árvore de forma recursiva, buscando a matrícula. Se for encontrada, imprime as informações do aluno.

9. Função `removerNo()`

- **Objetivo**: Remover um aluno da árvore binária com base na matrícula.

```c
NoArvore* removerNo(NoArvore *no, long long int matricula) {
    if (no == NULL) {
        printf("Matrícula não encontrada.\n");
        return NULL;
    }

    if (matricula < no->matricula) {
        no->esquerdo = removerNo(no->esquerdo, matricula);
    } else if (matricula > no->matricula) {
        no->direito = removerNo(no->direito, matricula);
    } else {
        if (no->esquerdo == NULL) {
            NoArvore *temp = no->direito;
            free(no);
            return temp;
        } else if (no->direito == NULL) {
            NoArvore *temp = no->esquerdo;
            free(no);
            return temp;
        }

        NoArvore *temp = encontrarMinimo(no->direito);
        no->matricula = temp->matricula;
        strcpy(no->nome, temp->nome);
        no->direito = removerNo(no->direito, temp->matricula);
    }
    return no;
}
```

- **Explicação**: A função busca o nó correspondente à matrícula e o marca como vazio, efetivamente removendo o aluno da árvore.

10. Função `pedirOpcao()`

- **Objetivo**: Exibir o menu principal e solicitar a opção escolhida pelo usuário.

```
long long int pedirOpcao() {
    int op;
    printf("\n--- Menu Principal ---\n");
    do {
        printf("1 - Inserir na Arvore\n");
        printf("2 - Excluir da Arvore\n");
        printf("3 - Pesquisar na Arvore\n");
        printf("4 - Total de Matriculas\n");
        printf("5 - Imprimir em Pre-ordem\n");
        printf("6 - Imprimir em Em-ordem\n");
        printf("7 - Imprimir em Pos-ordem\n");
        printf("0 - Sair\n");
        printf("Digite a opção: ");
        op = input();
    } while ((op < 0) || (op > 7));
    return op;
}
```

- **Explicação**: Esta função exibe um menu de opções para o usuário e solicita uma escolha. O loop `do-while` garante que o programa continue solicitando uma opção válida até que o usuário insira um número entre 1 e 7.

11. Função `pedirNum()`

- **Objetivo**: Solicitar um número (matrícula) ao usuário, usado tanto para inserção quanto para exclusão.

```c
long long int pedirNum(int caminhoASerEscolhido) {
    long long int num;
    if (caminhoASerEscolhido == 0) {
        printf("Digite um numero para ser inserido: ");
        scanf("%lld", &num);
    } else {
        printf("Digite um numero para ser excluido: ");
        scanf("%lld", &num);
    }
    return num;
}
```

- **Explicação**: Dependendo do valor do parâmetro `caminhoASerEscolhido`, a função solicita um número ao usuário para inserção (caminho `0`) ou para exclusão (caminho `1`). O número é interpretado como a matrícula de um aluno.

13. Função `menuPrincipal()`

- **Objetivo**: Gerenciar as operações de inserção, exclusão, pesquisa e exibição da árvore binária.

```
void menuPrincipal(ArvoreBinaria *arvore) {
    long long int op, matricula;
    string nome;
    int repete = 0;
    do {
        op = pedirOpcao();
        switch (op) {
            case 0:
                repete = 1;
                break;
            case 1:
                printf("Digite o numero da matrícula: ");
                matricula = inputDLLD();
                printf("Digite o nome: ");
                inputS(nome);
                arvore->raiz = inserirNo(arvore->raiz, matricula, nome);
                arvore->tamanho++;
                break;
            case 2:
                printf("Digite o numero da matrícula para remover: ");
                matricula = inputDLLD();
                arvore->raiz = removerNo(arvore->raiz, matricula);
                arvore->tamanho--;
                break;
```

```
            case 3:
                printf("Digite o numero da matrícula para buscar: ");
                matricula = inputDLLD();
                buscarNo(arvore->raiz, matricula);
                break;
            case 4:
                printf("O total de matrículas na árvore é: %d\n", arvore->tamanho);
                break;
            case 5:
                printf("\n\n===| Exibição da Árvore Binária (Pre Ordem) |===\n\n");
                caminhamento_Pre_Ordem(arvore->raiz);
                break;
            case 6:
                printf("\n\n===| Exibição da Árvore Binária (Em Ordem) |===\n\n");
                caminhamento_Em_Ordem(arvore->raiz);
                break;
            case 7:
                printf("\n\n===| Exibição da Árvore Binária (Pos Ordem) |===\n\n");
                caminhamento_Pos_Ordem(arvore->raiz);
                break;
            default:
                printf("Opção inválida. Tente novamente.\n");
                break;
        }
    } while (repete == 0);
}
```

- **Explicação**: Esta função atua como o ponto central do programa, permitindo que o usuário escolha entre diversas operações na árvore binária, como inserção, exclusão, exibição e pesquisa.

14. Função `contarMatriculas()`

- **Objetivo**: Contar o número de matrículas presentes em um arquivo de texto.

```
int contarMatriculas(FILE *arquivoLista) {
    char linha[100];
    int totalMatriculas = 0;
    while (! feof(arquivoLista)) {
        fscanf(arquivoLista, " %99[^\n]s" , linha);
        totalMatriculas++;
    }
    return totalMatriculas / 2;
}
```

- **Explicação**: A função conta o número de linhas em um arquivo e retorna o número de matrículas, assumindo que cada matrícula ocupa duas linhas (uma para o nome e outra para o número).

15. Função `lerEInserirMatriculas()`

- **Objetivo**: Ler as matrículas de um arquivo e inseri-las na árvore binária.

```
void lerEInserirMatriculas(ArvoreBinaria *arvore, FILE *arquivoLista) {
    rewind(arquivoLista);  // Reposicionar para o início do arquivo
    long long int matricula;
    string nome;

    while (! feof(arquivoLista)) {
        // Ler o nome
        fscanf(arquivoLista, " %99[^\n]s" , nome);
        // nome[strcspn(nome, "\n")] = 0;  // Remover o '\n' do nome
        if (! feof(arquivoLista)) {
            // Ler a matrícula
            fscanf(arquivoLista, "%lld" , &matricula);    unknown conversion type character 'l' in fo
            arvore->raiz = inserirNo(arvore->raiz, matricula, nome);  // Inserir na árvore binária
            arvore->tamanho++;
        }
    }
}
```

- **Explicação**: A função lê o nome e a matrícula do arquivo e insere esses dados na árvore binária. Ela percorre o arquivo até o fim, inserindo cada registro.

16. Função `iniciarCodigo()`

- **Objetivo**: Controlar o fluxo do programa, desde a inicialização até o encerramento.

```c
void iniciarCodigo(FILE *arquivoLista, ArvoreBinaria *arvore) {
    arquivoLista = abrirArquivo("nomes_matriculas.txt", "r");
    int totalMatriculas = contarMatriculas(arquivoLista);
    printf("Total de matrículas no arquivo: %d\n", totalMatriculas);
    rewind(arquivoLista); // Volta ao início do arquivo

    inicializarArvore(arvore);
    double inicio = clock();
    // Ler as matrículas e inseri-las na árvore
    lerEInserirMatriculas(arvore, arquivoLista);
    double fim = clock();
    fclose(arquivoLista);
    calcularTempo(inicio, fim);
    // Menu para interação
    menuPrincipal(arvore);

    // Salvar os dados antes de encerrar
    arquivoLista = abrirArquivo("nomes_matriculas.txt", "w");
    salvarDadosNoArquivo(arvore->raiz, arquivoLista);
    fclose(arquivoLista);

    // Liberar a memória
    liberarArvore(arvore->raiz);
}
```

- **Explicação**: Esta função gerencia todo o processo de leitura dos dados, inicialização da árvore, inserção dos registros e o ciclo de interação com o usuário. Também cuida da gravação dos dados e da liberação da memória no final.

Arquivo `.h`

Este arquivo contém as definições de tipos, macros e as declarações das funções que serão implementadas no arquivo `.c`. Abaixo estão os detalhes de cada parte.

1. Bibliotecas Incluídas

- **Objetivo**: As bibliotecas incluídas fornecem as funcionalidades necessárias para a manipulação de strings, entrada e saída de dados, operações matemáticas, medição de tempo, entre outras.

```c
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <time.h>
```

- **math.h**: Fornece funções matemáticas, como a função `pow` para cálculos de potência.
- **stdlib.h**: Inclui funções como `malloc` para alocação de memória dinâmica e `exit` para encerrar o programa.
- **stdio.h**: Inclui funções de entrada e saída, como `printf`, `scanf`, `fopen`, `fclose`, entre outras.
- **ctype.h**: Fornece funções para manipulação de caracteres, como `isdigit`.
- **string.h**: Inclui funções para manipulação de strings, como `strcpy` e `strcmp`.
- **time.h**: Fornece funções para manipulação de tempo, como `clock_t` e `clock` para medir o tempo de execução.

2. Definição de Tipos e Macros

a) Definição do tipo `string`

- **Objetivo**: Definir um tipo de dado `string` como um array de caracteres de tamanho 101

```c
typedef char string[101];
```

Isso simplifica o uso de strings no programa, permitindo que seja referenciado como `string` ao invés de `char[101]`.

b) Definição do tipo `processTime`

- **Objetivo**: Definir um alias para o tipo `clock_t`, que é utilizado para medir o tempo de execução do programa.

```
typedef clock_t processTime;
```

3. Definição de Estruturas

a) Estrutura `NoArvore`

- **Objetivo**: Definir a estrutura de um nó na árvore binária. Cada nó armazena uma matrícula, o nome do aluno e ponteiros para seus filhos que podem ser esquerda ou direita.

```
typedef struct NoArvore {
    long long int matricula;
    string nome;
    struct NoArvore *direito;
    struct NoArvore *esquerdo;
} NoArvore;
```

- **`matricula`**: Um identificador único para o aluno.
- **`nome`**: Nome associado à matrícula do aluno.
- **`esquerdo`**: Ponteiro associado ao filho esquerdo da árvore.
- **`direito`**: Ponteiro associado ao filho direito da árvore.

b) Estrutura `ArvoreBinaria`

- **Objetivo**: Definir a estrutura da árvore binária, que é representada como um array de nós.

```
typedef struct {
    NoArvore *raiz;
    int tamanho;
} ArvoreBinaria;
```

- **`raiz`**: Ponteiro para um array de nós (`NoArvore`), que representa os elementos da árvore.
- **`tamanho`**:indica o tamanho da árvore.