Lista 8 de Laboratório de Programação II
Nome : Messias Feres Curi Melo
Matrícula : 2022003764

Problema 1
1.1

main.c

```c
#include <stdio.h>
#include "ABP.h"

int main(){
    ABP* A;
    int escolha, elemento, nivel, resultado;

    printf("-=x Menu x=-\n");
    printf("1 - Criar ABP\n");
    printf("2 - Inserir um elemento\n");
    printf("3 - Buscar um elemento\n");
    printf("4 - Remover um elemento\n");
    printf("5 - Imprimir a ABP em ordem\n");
    printf("6 - Imprimir a ABP em pré-ordem\n");
    printf("7 - Imprimir a ABP em pós-ordem\n");
    printf("8 - Mostrar a quantidade de nós na ABP\n");
    printf("9 - Destruir a ABP\n");
    printf("10 - Sair\n");

    do{
        printf("\nInsira sua escolha: ");
        scanf("%d", &escolha);
        if(escolha == 1){
            A = criaABP();
            printf("-> ABP criada com sucesso!");
        }else if(escolha == 2){
            printf("- Insira o elemento a ser adicionado: ");
            scanf("%d", &elemento);
            resultado = insereElem(A, elemento);
            if(resultado){
                printf("-> Elemento adicionado com sucesso!");
            }else{
                printf("-> Elemento não foi adicionado!");
            }
        }else if(escolha == 3){
            printf("- Insira o elemento a ser buscado: ");
            scanf("%d", &elemento);
            resultado = pesquisa(A, elemento);
            if(resultado){
                printf("-> Elemento encontrado!");
            }else{
                printf("-> Elemento não encontrado!");
            }
        }else if(escolha == 4){
            printf("- Insira o elemento a ser removido: ");
            scanf("%d", &elemento);
            resultado = removeElem(A, elemento);
             if(resultado){
```

```c
                printf("-> Elemento removido com sucesso!!");
            }else{
                printf("-> Elemento não foi removido!");
            }
        }else if(escolha == 5){
            printf("- Insira o nível: ");
            scanf("%d", &nivel);
            em_ordem(*A, nivel);
        }else if(escolha == 6){
            printf("- Insira o nível: ");
            scanf("%d", &nivel);
            pre_ordem(*A, nivel);
        }else if(escolha == 7){
            printf("- Insira o nível: ");
            scanf("%d", &nivel);
            pos_ordem(*A, nivel);
        }else if(escolha == 8){
            resultado = contaNos(A);
            printf("-> Quantidade de nós: %d", resultado);
        }else if(escolha == 9){
            destroiABP(A);
            printf("-> ABP destruida com sucesso!");
        }
    }while (escolha != 10);

    return 0;
}
```

ABP.h

C ABP.h > ...

```c
/*---------------- File: ABP.h  ------------------------+
|Arvore Binaria de Pesquisa (ABP)                       |
|                                                       |
|                                                       |
| Implementado por Guilherme C. Pena em 12/10/2023      |
+-------------------------------------------------------+ */

#ifndef ABP_H
#define ABP_H

#include <stdio.h>
#include <stdlib.h>

typedef struct NO{
    int info;
    struct NO* esq;
    struct NO* dir;
}NO;

typedef struct NO* ABP;

NO* alocarNO(){
    return (NO*) malloc (sizeof(NO));
}

void liberarNO(NO* q){
    free(q);
}

ABP* criaABP(){
    ABP* raiz = (ABP*) malloc (sizeof(ABP));
    if(raiz != NULL)
        *raiz = NULL;
    return raiz;
}

void destroiRec(NO* no){
    if(no == NULL) return;
    destroiRec(no->esq);
    destroiRec(no->dir);
    liberarNO(no);
    no = NULL;
}

void destroiABP(ABP* raiz){
    if(raiz != NULL){
        destroiRec(*raiz);
        free(raiz);
```

```c
    }
}

int estaVazia(ABP* raiz){
    if(raiz == NULL) return 0;
    return (*raiz == NULL);
}


int insereRec(NO** raiz, int elem){
    if(*raiz == NULL){
        NO* novo = alocarNO();
        if(novo == NULL) return 0;
        novo->info = elem;
        novo->esq = NULL; novo->dir = NULL;
        *raiz = novo;
    }else{
        if((*raiz)->info == elem){
            printf("-> Elemento Existente!\n");
            return 0;
        }
        if(elem < (*raiz)->info)
            return insereRec(&(*raiz)->esq, elem);
        else if(elem > (*raiz)->info)
            return insereRec(&(*raiz)->dir, elem);
    }
    return 1;
}

int insereIte(NO** raiz, int elem){
    NO *aux = *raiz, *ant = NULL;
    while (aux != NULL){
        ant = aux;
        if(aux->info == elem){
            printf("-> Elemento Existente!\n");
            return 0;
        }
        if(elem < aux->info) aux = aux->esq;
        else aux = aux->dir;
    }
    NO* novo = alocarNO();
    if(novo == NULL) return 0;
    novo->info = elem;
    novo->esq = NULL; novo->dir = NULL;
    if(ant == NULL){
        *raiz = novo;
    }else{
```

```
96          if(elem < ant->info) ant->esq = novo;
97          else ant->dir = novo;
98      }
99      return 1;
100 }
101
102 int insereElem(ABP* raiz, int elem){
103     if(raiz == NULL) return 0;
104     return insereRec(raiz, elem);
105 }
106
107 int pesquisaRec(NO** raiz, int elem){
108     if(*raiz == NULL) return 0;
109     if((*raiz)->info == elem) return 1;
110     if(elem < (*raiz)->info)
111         return pesquisaRec(&(*raiz)->esq, elem);
112     else
113         return pesquisaRec(&(*raiz)->dir, elem);
114 }
115
116 int pesquisaIte(NO** raiz, int elem){
117     NO* aux = *raiz;
118     while(aux != NULL){
119         if(aux->info == elem) return 1;
120         if(elem < aux->info)
121             aux = aux->esq;
122         else
123             aux = aux->dir;
124     }
125     return 0;
126 }
127
128 int pesquisa(ABP* raiz, int elem){
129     if(raiz == NULL) return 0;
130     if(estaVazia(raiz)) return 0;
131     return pesquisaRec(raiz, elem);
132 }
133
134 int removeRec(NO** raiz, int elem){
135     if(*raiz == NULL) return 0;
136     if((*raiz)->info == elem){
137         NO* aux;
138         if((*raiz)->esq == NULL && (*raiz)->dir == NULL){
139             printf("-> Caso 1: Liberando %d..\n", (*raiz)->info);
140             liberarNO(*raiz);
141             *raiz = NULL;
142         }else if((*raiz)->esq == NULL){
```

```
143             printf("-> Caso 2.1: Liberando %d..\n", (*raiz)->info);
144             aux = *raiz;
145             *raiz = (*raiz)->dir;
146             liberarNO(aux);
147         }else if((*raiz)->dir == NULL){
148             printf("-> Caso 2.2: Liberando %d..\n", (*raiz)->info);
149             aux = *raiz;
150             *raiz = (*raiz)->esq;
151             liberarNO(aux);
152         }else{
153             printf("-> Caso 3: Liberando %d..\n", (*raiz)->info);
154             NO* Filho = (*raiz)->esq;
155             while(Filho->dir != NULL)
156                 Filho = Filho->dir;
157             (*raiz)->info = Filho->info;
158             Filho->info = elem;
159             return removeRec(&(*raiz)->esq, elem);
160         }
161         return 1;
162     }else if(elem < (*raiz)->info)
163         return removeRec(&(*raiz)->esq, elem);
164     else
165         return removeRec(&(*raiz)->dir, elem);
166 }
167 
168 NO* removeAtual(NO* atual){
169     NO* no1, *no2;
170     if(atual->esq == NULL){
171         no2 = atual->dir;
172         liberarNO(atual);
173         return no2;
174     }
175 
176     no1 = atual;
177     no2 = atual->esq;
178     while(no2->dir != NULL){
179         no1 = no2;
180         no2 = no2->dir;
181     }
182     if(no1 != atual){
183         no1->dir = no2->esq;
184         no2->esq = atual->esq;
185     }
186     no2->dir = atual->dir;
187     liberarNO(atual);
188     return no2;
189 }
```

```c

int removeIte(NO** raiz, int elem){
    if(*raiz == NULL) return 0;
    NO* atual = *raiz, *ant = NULL;
    while(atual != NULL){
        if(elem == atual->info){
            if(atual == *raiz)
                *raiz = removeAtual(atual);
            else{
                if(ant->dir == atual)
                    ant->dir = removeAtual(atual);
                else
                    ant->esq = removeAtual(atual);
            }
            return 1;
        }
        ant = atual;
        if(elem < atual->info)
            atual = atual->esq;
        else
            atual = atual->dir;
    }
    return 0;
}

int removeElem(ABP* raiz, int elem){
    if(pesquisa(raiz, elem) == 0){
        printf("-> Elemento inexistente!\n");
        return 0;
    }
    return removeIte(raiz, elem);
}

void em_ordem(NO* raiz, int nivel){
    if(raiz != NULL){
        em_ordem(raiz->esq, nivel+1);
        printf("-> [%d, %d] ", raiz->info,nivel);
        em_ordem(raiz->dir, nivel+1);
    }
}

void pre_ordem(NO* raiz, int nivel){
    if(raiz != NULL){
        printf("-> [%d, %d] ", raiz->info,nivel);
        pre_ordem(raiz->esq, nivel+1);
        pre_ordem(raiz->dir, nivel+1);
```

Console

```
root@DESKTOP-CJCQ0HI:/mnt/c/Users/MessiasFCM/Desktop/Lista 7/atv1.1# gcc main.c ABP.h -o tp1
root@DESKTOP-CJCQ0HI:/mnt/c/Users/MessiasFCM/Desktop/Lista 7/atv1.1# ./tp1
-=x Menu x=-
1 - Criar ABP
2 - Inserir um elemento
3 - Buscar um elemento
4 - Remover um elemento
5 - Imprimir a ABP em ordem
6 - Imprimir a ABP em pré-ordem
7 - Imprimir a ABP em pós-ordem
8 - Mostrar a quantidade de nós na ABP
9 - Destruir a ABP
10 - Sair

Insira sua escolha: 1
-> ABP criada com sucesso!
Insira sua escolha: 2
- Insira o elemento a ser adicionado: 10
-> Elemento adicionado com sucesso!
Insira sua escolha: 2
- Insira o elemento a ser adicionado: 5
-> Elemento adicionado com sucesso!
Insira sua escolha: 2
- Insira o elemento a ser adicionado: 7
-> Elemento adicionado com sucesso!
Insira sua escolha: 2
- Insira o elemento a ser adicionado: 13
-> Elemento adicionado com sucesso!
Insira sua escolha: 5
- Insira o nível: 1
-> [5, 2] -> [7, 3] -> [10, 1] -> [13, 2]
Insira sua escolha: 5
- Insira o nível: 2
-> [5, 3] -> [7, 4] -> [10, 2] -> [13, 3]
Insira sua escolha: 5
- Insira o nível: 0
-> [5, 1] -> [7, 2] -> [10, 0] -> [13, 1]
Insira sua escolha: 6
- Insira o nível: 0
-> [10, 0] -> [5, 1] -> [7, 2] -> [13, 1]
Insira sua escolha: 7
- Insira o nível: 0
-> [7, 2] -> [5, 1] -> [13, 1] -> [10, 0]
Insira sua escolha: 8
-> Quantidade de nós: 4
Insira sua escolha: 9
-> ABP destruida com sucesso!
Insira sua escolha: 10
root@DESKTOP-CJCQ0HI:/mnt/c/Users/MessiasFCM/Desktop/Lista 7/atv1.1#
```

1.2

main.c

```c
#include <stdio.h>
#include "ABP.h"

int main(){
    ABP* A;
    char nome[100];
    int escolha, matricula, resultado;
    double nota;

    printf("-=x Menu x=-\n");
    printf("1 - Criar ABP\n");
    printf("2 - Inserir um Aluno\n");
    printf("3 - Buscar um Aluno pelo nome e imprimir suas informações\n");
    printf("4 - Remover um Aluno pelo nome\n");
    printf("5 - Imprimir a ABP em ordem\n");
    printf("6 - Imprimir as informações do aluno com a maior nota\n");
    printf("7 - Imprimir as informações do aluno com a menor nota\n");
    printf("8 - Destruir a ABP\n");
    printf("9 - Sair");

    do{
        printf("\n\nInsira sua escolha: ");
        scanf("%d", &escolha);
        if(escolha == 1){
            A = criarABP();
            printf("-> ABP criada com sucesso!");
        }else if(escolha == 2){
           printf("Insira o nome do aluno: ");
           scanf("%s", nome);
           printf("Insira a matrícula do aluno: ");
           scanf("%d", &matricula);
           printf("Insira a nota do aluno: ");
           scanf("%lf", &nota);
           resultado = inserirAluno(A, nome, matricula, nota);
           if(resultado){
               printf("-> Aluno adicionado com sucesso!");
            }else{
               printf("-> Aluno não foi adicionado!");
           }
        }else if(escolha == 3){
            printf("Insira o nome do aluno a ser buscado: ");
            scanf("%s", nome);
            resultado = buscarAluno(A, nome);
            if(resultado){
                printf("-> Aluno encontrado!");
            }else{
                printf("-> Aluno não encontrado!");
            }
        }else if(escolha == 4){
            printf("Insira o nome do aluno a ser removido: ");
            scanf("%s", nome);
            resultado = removerAluno(A, nome);
            if(resultado){
                printf("-> Aluno removido com sucesso!");
            }else{
                printf("-> Aluno não foi removido!");
            }
        }else if(escolha == 5){
            printf("- Árvore em Ordem:\n");
            imprime(A);
        }else if(escolha == 6){
            Aluno maiorNota = encontrarMaiorNota(A);
            printf("- Aluno com a maior nota:\n");
            printf("-> Nome: %s\n", maiorNota.nome);
            printf("-> Matrícula: %d\n", maiorNota.matricula);
            printf("-> Nota: %.2lf\n", maiorNota.nota);
        }else if(escolha == 7){
            Aluno menorNota = encontrarMenorNota(A);
            printf("- Aluno com a menor nota:\n");
            printf("-> Nome: %s\n", menorNota.nome);
            printf("-> Matrícula: %d\n", menorNota.matricula);
            printf("-> Nota: %.2lf\n", menorNota.nota);
        }else if(escolha == 8){
            destroirABP(A);
            printf("-> ABP destruida com sucesso!");
        }
    }while (escolha != 9);

    return 0;
}
```

ABP.h

C ABP.h > ...

```c
/*---------------- File: ABP.h  ------------------------+
|Arvore Binaria de Pesquisa (ABP)                       |
|                                                       |
|                                                       |
| Implementado por Guilherme C. Pena em 12/10/2023      |
+-------------------------------------------------------+ */

#ifndef ABP_H
#define ABP_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct Aluno {
    char nome[100];
    int matricula;
    double nota;
} Aluno;

typedef struct NO{
    Aluno aluno;
    struct NO* esq;
    struct NO* dir;
}NO;

typedef struct NO* ABP;

NO* alocarNO(char nome[], int matricula, double nota){
    NO* novoAluno = (NO*) malloc (sizeof(NO));
    if(novoAluno != NULL){
        strcpy(novoAluno->aluno.nome, nome);
        novoAluno->aluno.matricula = matricula;
        novoAluno->aluno.nota = nota;
        novoAluno->esq = NULL;
        novoAluno->dir = NULL;
    }
    return novoAluno;
}

void liberarNO(NO* q){
    free(q);
}

ABP* criarABP(){
    ABP* raiz = (ABP*) malloc (sizeof(ABP));
    if(raiz != NULL)
        *raiz = NULL;
```

```c
    return raiz;
}

void destroirRec(NO* no){
    if(no == NULL) return;
    destroirRec(no->esq);
    destroirRec(no->dir);
    liberarNO(no);
    no = NULL;
}

void destroirABP(ABP* raiz){
    if(raiz != NULL){
        destroirRec(*raiz);
        free(raiz);
    }
}

int estaVazia(ABP* raiz){
    if(raiz == NULL) return 0;
    return (*raiz == NULL);
}

int inserirRec(NO** raiz, char nome[], int matricula, double nota){
    if(*raiz == NULL){
        NO* novo = alocarNO(nome, matricula, nota);
        if(novo == NULL) return 0;
        *raiz = novo;
    }else{
        int comparacao = strcmp(nome, (*raiz)->aluno.nome);
        if(comparacao == 0){
            printf("-> Elemento Existente!\n");
            return 0;
        }
        if(comparacao < 0){
            return inserirRec(&(*raiz)->esq, nome, matricula, nota);
        }else if(comparacao > 0){
            return inserirRec(&(*raiz)->dir, nome, matricula, nota);
        }
    }
    return 1;
}

int inserirAluno(ABP* raiz, char nome[], int matricula, double nota){
    if(raiz == NULL) return 0;
    return inserirRec(raiz, nome, matricula, nota);
}
```

```
96
97  int pesquisarRec(NO** raiz, char nome[]){
98      if(*raiz == NULL) return 0;
99      int comparar = strcmp(nome, (*raiz)->aluno.nome);
100     if (comparar == 0){
101         return 1;
102     }else if (comparar < 0){
103         return pesquisarRec(&(*raiz)->esq, nome);
104     }else{
105         return pesquisarRec(&(*raiz)->dir, nome);
106     }
107 }
108
109 int buscarAluno(ABP* raiz, char nome[]){
110     if(raiz == NULL) return 0;
111     if(estaVazia(raiz)) return 0;
112     return pesquisarRec(raiz, nome);
113 }
114
115 NO* encontrarMaiorNotaRec(NO* raiz, double nota){
116     if (raiz == NULL){
117         return NULL;
118     }
119     NO* noMaiorNota = NULL;
120     if(raiz->aluno.nota <= nota){
121         noMaiorNota = encontrarMaiorNotaRec(raiz->dir, nota);
122     }else{
123         noMaiorNota = encontrarMaiorNotaRec(raiz->esq, nota);
124         if(noMaiorNota == NULL || noMaiorNota->aluno.nota < raiz->aluno.nota){
125             noMaiorNota = raiz;
126         }
127     }
128     return noMaiorNota;
129 }
130
131 Aluno encontrarMaiorNota(ABP* raiz){
132     NO* noMaiorNota = encontrarMaiorNotaRec(*raiz, 0);
133     return noMaiorNota->aluno;
134 }
135
136 NO* encontrarMenorNotaRec(NO* raiz, double nota){
137     if(raiz == NULL){
138         return NULL;
139     }
140     NO* noMenorNota = NULL;
141     if(raiz->aluno.nota >= nota){
142         noMenorNota = encontrarMenorNotaRec(raiz->esq, nota);
143     }else{
```

```
        noMenorNota = encontrarMenorNotaRec(raiz->dir, nota);
        if(noMenorNota == NULL || noMenorNota->aluno.nota > raiz->aluno.nota){
            noMenorNota = raiz;
        }
    }
    return noMenorNota;
}

Aluno encontrarMenorNota(ABP* raiz){
    NO* noMenorNota = encontrarMenorNotaRec(*raiz, 9999);
    return noMenorNota->aluno;
}

int removerRec(NO** raiz, char nome[]){
    if(*raiz == NULL) return 0;
    int comparar = strcmp(nome, (*raiz)->aluno.nome);
    if(comparar == 0){
        NO* aux;
        if((*raiz)->esq == NULL && (*raiz)->dir == NULL){
            liberarNO(*raiz);
            *raiz = NULL;
        }else if((*raiz)->esq == NULL){
            aux = *raiz;
            *raiz = (*raiz)->dir;
            liberarNO(aux);
        }else if((*raiz)->dir == NULL){
            aux = *raiz;
            *raiz = (*raiz)->esq;
            liberarNO(aux);
        }else{
            NO* Filho = (*raiz)->esq;
            while (Filho->dir != NULL){
                Filho = Filho->dir;
            }
            (*raiz)->aluno = Filho->aluno;
            return removerRec(&(*raiz)->esq, nome);
        }
        return 1;
    }else if(comparar < 0){
        return removerRec(&(*raiz)->esq, nome);
    }
    else{
        return removerRec(&(*raiz)->dir, nome);
    }
}

NO* removerAtual(NO* atual){
```

```
    NO* no1, *no2;
    if(atual->esq == NULL){
        no2 = atual->dir;
        liberarNO(atual);
        return no2;
    }

    no1 = atual;
    no2 = atual->esq;
    while(no2->dir != NULL){
        no1 = no2;
        no2 = no2->dir;
    }
    if(no1 != atual){
        no1->dir = no2->esq;
        no2->esq = atual->esq;
    }
    no2->dir = atual->dir;
    liberarNO(atual);
    return no2;
}

int removerIte(NO** raiz, char nome[]){
    if(*raiz == NULL) return 0;
    NO* atual = *raiz, *ant = NULL;
    while(atual != NULL){
        int comparar = strcmp(nome, atual->aluno.nome);
        if(comparar == 0){
            if(atual == *raiz){
                *raiz = removerAtual(atual);
            }else if(ant->dir == atual){
                ant->dir = removerAtual(atual);
            }else if(ant->esq == atual){
                ant->esq = removerAtual(atual);
            }
            return 1;
        }
        ant = atual;
        if(comparar < 0){
            atual = atual->esq;
        }else{
            atual = atual->dir;
        }
    }
    return 0;
}

int removerAluno(ABP* raiz, char nome[]){
```

```
    if(buscarAluno(raiz, nome) == 0){
        printf("-> Elemento inexistente!\n");
        return 0;
    }
    return removerIte(raiz, nome);
}

void em_ordem(NO* raiz, int nivel){
    if(raiz != NULL){
        em_ordem(raiz->esq, nivel+1);
        printf("-> [%s, %d, %.2lf, %d] \n", raiz->aluno.nome, raiz->aluno.matricula, raiz->aluno.nota, nivel);
        em_ordem(raiz->dir, nivel+1);
    }
}

void imprime(ABP* raiz){
    if(raiz == NULL) return;
    if(estaVazia(raiz)){
        printf("-> Arvore Vazia!\n");
        return;
    }
    em_ordem(*raiz, 0);
}

#endif
```

Console

```
root@DESKTOP-CJCQ0HI:/mnt/c/Users/MessiasFCM/Desktop/Lista 7/atv1.2# gcc main.c ABP.h -o tp2
root@DESKTOP-CJCQ0HI:/mnt/c/Users/MessiasFCM/Desktop/Lista 7/atv1.2# ./tp2
-=x Menu x=-
1 - Criar ABP
2 - Inserir um Aluno
3 - Buscar um Aluno pelo nome e imprimir suas informações
4 - Remover um Aluno pelo nome
5 - Imprimir a ABP em ordem
6 - Imprimir as informações do aluno com a maior nota
7 - Imprimir as informações do aluno com a menor nota
8 - Destruir a ABP
9 - Sair

Insira sua escolha: 1
-> ABP criada com sucesso!

Insira sua escolha: 2
Insira o nome do aluno: Messias
Insira a matrícula do aluno: 202201
Insira a nota do aluno: 8
-> Aluno adicionado com sucesso!

Insira sua escolha: 2
Insira o nome do aluno: Helen
Insira a matrícula do aluno: 202202
Insira a nota do aluno: 10
-> Aluno adicionado com sucesso!

Insira sua escolha: 2
Insira o nome do aluno: Heitor
Insira a matrícula do aluno: 202203
Insira a nota do aluno: 4
-> Aluno adicionado com sucesso!

Insira sua escolha: 5
- Árvore em Ordem:
-> [Heitor, 202203, 4.00, 2]
-> [Helen, 202202, 10.00, 1]
-> [Messias, 202201, 8.00, 0]


Insira sua escolha: 4
Insira o nome do aluno a ser removido: Heitor
-> Aluno removido com sucesso!

Insira sua escolha: 6
- Aluno com a maior nota:
-> Nome: Helen
-> Matrícula: 202202
-> Nota: 10.00
```

```
Insira sua escolha: 7
- Aluno com a menor nota:
-> Nome: Messias
-> Matrícula: 202201
-> Nota: 8.00


Insira sua escolha: 8
-> ABP destruida com sucesso!

Insira sua escolha: 9
root@DESKTOP-CJCQ0HI:/mnt/c/Users/MessiasFCM/Desktop/Lista 7/atv1.2#
```