

Lista 8 de Laboratório de Programação II

Nome : Messias Feres Curi Melo

Matrícula : 2022003764

Problema 1

1.1

main.c

```
C main.c > main()
1  #include <stdio.h>
2  #include "ABP.h"
3
4  int main()
5  {
6      ABP* A;
7      int escolha, elemento, nivel, resultado;
8
9      printf("--x Menu x--\n");
10     printf("1 - Criar ABP\n");
11     printf("2 - Inserir um elemento\n");
12     printf("3 - Buscar um elemento\n");
13     printf("4 - Remover um elemento\n");
14     printf("5 - Imprimir a ABP em ordem\n");
15     printf("6 - Imprimir a ABP em pré-ordem\n");
16     printf("7 - Imprimir a ABP em pós-ordem\n");
17     printf("8 - Mostrar a quantidade de nós na ABP\n");
18     printf("9 - Destruir a ABP\n");
19     printf("10 - Sair\n");
20
21     do{
22         printf("\nInsira sua escolha: ");
23         scanf("%d", &escolha);
24         if(escolha == 1){
25             A = criaABP();
26             printf("-> ABP criada com sucesso!");
27         }else if(escolha == 2){
28             printf("- Insira o elemento a ser adicionado: ");
29             scanf("%d", &elemento);
30             resultado = insereElem(A, elemento);
31             if(resultado){
32                 printf("-> Elemento adicionado com sucesso!");
33             }else{
34                 printf("-> Elemento não foi adicionado!");
35             }
36         }else if(escolha == 3){
37             printf("- Insira o elemento a ser buscado: ");
38             scanf("%d", &elemento);
39             resultado = pesquisa(A, elemento);
40             if(resultado){
41                 printf("-> Elemento encontrado!");
42             }else{
43                 printf("-> Elemento não encontrado!");
44             }
45         }else if(escolha == 4){
46             printf("- Insira o elemento a ser removido: ");
47             scanf("%d", &elemento);
48             resultado = removeElem(A, elemento);
49             if(resultado){
```

```

49         printf("-> Elemento removido com sucesso!!");
50     }else{
51         printf("-> Elemento não foi removido!");
52     }
53 }else if(escolha == 5){
54     printf("- Insira o nível: ");
55     scanf("%d", &nivel);
56     em_ordem(*A, nivel);
57 }else if(escolha == 6){
58     printf("- Insira o nível: ");
59     scanf("%d", &nivel);
60     pre_ordem(*A, nivel);
61 }else if(escolha == 7){
62     printf("- Insira o nível: ");
63     scanf("%d", &nivel);
64     pos_ordem(*A, nivel);
65 }else if(escolha == 8){
66     resultado = contaNos(A);
67     printf("-> Quantidade de nós: %d", resultado);
68 }else if(escolha == 9){
69     destroiABP(A);
70     printf("-> ABP destruída com sucesso!");
71 }
72 }while (escolha != 10);
73
74 return 0;
75 }

```

ABP.h

```
C ABP.h > ...
1  /*----- File: ABP.h -----+
2  |Arvore Binaria de Pesquisa (ABP)|
3  |                               |
4  |                               |
5  | Implementado por Guilherme C. Pena em 12/10/2023 |
6  +-----+ */
7
8  #ifndef ABP_H
9  #define ABP_H
10
11  #include <stdio.h>
12  #include <stdlib.h>
13
14  typedef struct NO{
15      int info;
16      struct NO* esq;
17      struct NO* dir;
18  }NO;
19
20  typedef struct NO* ABP;
21
22  NO* alocarNO(){
23      return (NO*) malloc (sizeof(NO));
24  }
25
26  void liberarNO(NO* q){
27      free(q);
28  }
29
30  ABP* criaABP(){
31      ABP* raiz = (ABP*) malloc (sizeof(ABP));
32      if(raiz != NULL)
33          *raiz = NULL;
34      return raiz;
35  }
36
37  void destroiRec(NO* no){
38      if(no == NULL) return;
39      destroiRec(no->esq);
40      destroiRec(no->dir);
41      liberarNO(no);
42      no = NULL;
43  }
44
45  void destroiABP(ABP* raiz){
46      if(raiz != NULL){
47          destroiRec(*raiz);
48          free(raiz);
49      }
50  }
```

```

49     }
50 }
51
52 int estaVazia(ABP* raiz){
53     if(raiz == NULL) return 0;
54     return (*raiz == NULL);
55 }
56
57
58 int insereRec(NO** raiz, int elem){
59     if(*raiz == NULL){
60         NO* novo = alocarNO();
61         if(novo == NULL) return 0;
62         novo->info = elem;
63         novo->esq = NULL; novo->dir = NULL;
64         *raiz = novo;
65     }else{
66         if((*raiz)->info == elem){
67             printf("-> Elemento Existente!\n");
68             return 0;
69         }
70         if(elem < (*raiz)->info)
71             return insereRec(&(*raiz)->esq, elem);
72         else if(elem > (*raiz)->info)
73             return insereRec(&(*raiz)->dir, elem);
74     }
75     return 1;
76 }
77
78 int insereIte(NO** raiz, int elem){
79     NO *aux = *raiz, *ant = NULL;
80     while (aux != NULL){
81         ant = aux;
82         if(aux->info == elem){
83             printf("-> Elemento Existente!\n");
84             return 0;
85         }
86         if(elem < aux->info) aux = aux->esq;
87         else aux = aux->dir;
88     }
89     NO* novo = alocarNO();
90     if(novo == NULL) return 0;
91     novo->info = elem;
92     novo->esq = NULL; novo->dir = NULL;
93     if(ant == NULL){
94         *raiz = novo;
95     }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 }

```

```

190
191 int removeIte(NO** raiz, int elem){
192     if(*raiz == NULL) return 0;
193     NO* atual = *raiz, *ant = NULL;
194     while(atual != NULL){
195         if(elem == atual->info){
196             if(atual == *raiz)
197                 *raiz = removeAtual(atual);
198             else{
199                 if(ant->dir == atual)
200                     ant->dir = removeAtual(atual);
201                 else
202                     ant->esq = removeAtual(atual);
203             }
204             return 1;
205         }
206         ant = atual;
207         if(elem < atual->info)
208             atual = atual->esq;
209         else
210             atual = atual->dir;
211     }
212     return 0;
213 }
214
215 int removeElem(ABP* raiz, int elem){
216     if(pesquisa(raiz, elem) == 0){
217         printf("-> Elemento inexistente!\n");
218         return 0;
219     }
220     return removeIte(raiz, elem);
221 }
222
223 void em_ordem(NO* raiz, int nivel){
224     if(raiz != NULL){
225         em_ordem(raiz->esq, nivel+1);
226         printf("-> [%d, %d] ", raiz->info, nivel);
227         em_ordem(raiz->dir, nivel+1);
228     }
229 }
230
231 void pre_ordem(NO* raiz, int nivel){
232     if(raiz != NULL){
233         printf("-> [%d, %d] ", raiz->info, nivel);
234         pre_ordem(raiz->esq, nivel+1);
235         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 destruída com sucesso!
Insira sua escolha: 10
root@DESKTOP-CJCQ0HI:/mnt/c/Users/MessiasFCM/Desktop/Lista 7/atv1.1#
```


1.2

main.c

```
C main.c > main()
1  #include <stdio.h>
2  #include "ABP.h"
3
4  int main(){
5      ABP* A;
6      char nome[100];
7      int escolha, matricula, resultado;
8      double nota;
9
10     printf("--x Menu x--\n");
11     printf("1 - Criar ABP\n");
12     printf("2 - Inserir um Aluno\n");
13     printf("3 - Buscar um Aluno pelo nome e imprimir suas informações\n");
14     printf("4 - Remover um Aluno pelo nome\n");
15     printf("5 - Imprimir a ABP em ordem\n");
16     printf("6 - Imprimir as informações do aluno com a maior nota\n");
17     printf("7 - Imprimir as informações do aluno com a menor nota\n");
18     printf("8 - Destruir a ABP\n");
19     printf("9 - Sair\n");
20
21     do{
22         printf("\n\nInsira sua escolha: ");
23         scanf("%d", &escolha);
24         if(escolha == 1){
25             A = criarABP();
26             printf("-> ABP criada com sucesso!");
27         }else if(escolha == 2){
28             printf("Insira o nome do aluno: ");
29             scanf("%s", nome);
30             printf("Insira a matrícula do aluno: ");
31             scanf("%d", &matricula);
32             printf("Insira a nota do aluno: ");
33             scanf("%lf", &nota);
34             resultado = inserirAluno(A, nome, matricula, nota);
35             if(resultado){
36                 printf("-> Aluno adicionado com sucesso!");
37             }else{
38                 printf("-> Aluno não foi adicionado!");
39             }
40         }else if(escolha == 3){
41             printf("Insira o nome do aluno a ser buscado: ");
42             scanf("%s", nome);
43             resultado = buscarAluno(A, nome);
44             if(resultado){
45                 printf("-> Aluno encontrado!");
46             }else{
47                 printf("-> Aluno não encontrado!");
48             }
49         }else if(escolha == 4){
50             printf("Insira o nome do aluno a ser removido: ");
51             scanf("%s", nome);
52             resultado = removerAluno(A, nome);
53             if(resultado){
54                 printf("-> Aluno removido com sucesso!");
55             }else{
56                 printf("-> Aluno não foi removido!");
57             }
58         }else if(escolha == 5){
59             printf("- Árvore em Ordem:\n");
60             imprime(A);
61         }else if(escolha == 6){
62             Aluno maiorNota = encontrarMaiorNota(A);
63             printf("- Aluno com a maior nota:\n");
64             printf("-> Nome: %s\n", maiorNota.nome);
65             printf("-> Matrícula: %d\n", maiorNota.matricula);
66             printf("-> Nota: %.2lf\n", maiorNota.nota);
67         }else if(escolha == 7){
68             Aluno menorNota = encontrarMenorNota(A);
69             printf("- Aluno com a menor nota:\n");
70             printf("-> Nome: %s\n", menorNota.nome);
71             printf("-> Matrícula: %d\n", menorNota.matricula);
72             printf("-> Nota: %.2lf\n", menorNota.nota);
73         }else if(escolha == 8){
74             destruirABP(A);
75             printf("-> ABP destruída com sucesso!");
76         }
77     }while (escolha != 9);
78
79     return 0;
80 }
```

ABP.h

```
C ABP.h > ...
1  /*----- File: ABP.h -----+
2  |Arvore Binaria de Pesquisa (ABP) |
3  | | |
4  | | |
5  | Implementado por Guilherme C. Pena em 12/10/2023 |
6  +-----+ */
7
8  #ifndef ABP_H
9  #define ABP_H
10
11 #include <stdio.h>
12 #include <stdlib.h>
13 #include <string.h>
14
15 typedef struct Aluno {
16     char nome[100];
17     int matricula;
18     double nota;
19 } Aluno;
20
21 typedef struct NO{
22     Aluno aluno;
23     struct NO* esq;
24     struct NO* dir;
25 }NO;
26
27 typedef struct NO* ABP;
28
29 NO* alocarNO(char nome[], int matricula, double nota){
30     NO* novoAluno = (NO*) malloc (sizeof(NO));
31     if(novoAluno != NULL){
32         strcpy(novoAluno->aluno.nome, nome);
33         novoAluno->aluno.matricula = matricula;
34         novoAluno->aluno.nota = nota;
35         novoAluno->esq = NULL;
36         novoAluno->dir = NULL;
37     }
38     return novoAluno;
39 }
40
41 void liberarNO(NO* q){
42     free(q);
43 }
44
45 ABP* criarABP(){
46     ABP* raiz = (ABP*) malloc (sizeof(ABP));
47     if(raiz != NULL)
48         *raiz = NULL;
```

```

49     return raiz;
50 }
51
52 void destruirRec(NO* no){
53     if(no == NULL) return;
54     destruirRec(no->esq);
55     destruirRec(no->dir);
56     liberarNO(no);
57     no = NULL;
58 }
59
60 void destruirABP(ABP* raiz){
61     if(raiz != NULL){
62         destruirRec(*raiz);
63         free(raiz);
64     }
65 }
66
67 int estaVazia(ABP* raiz){
68     if(raiz == NULL) return 0;
69     return (*raiz == NULL);
70 }
71
72 int inserirRec(NO** raiz, char nome[], int matricula, double nota){
73     if(*raiz == NULL){
74         NO* novo = alocarNO(nome, matricula, nota);
75         if(novo == NULL) return 0;
76         *raiz = novo;
77     }else{
78         int comparacao = strcmp(nome, (*raiz)->aluno.nome);
79         if(comparacao == 0){
80             printf("-> Elemento Existente!\n");
81             return 0;
82         }
83         if(comparacao < 0){
84             return inserirRec(&(*raiz)->esq, nome, matricula, nota);
85         }else if(comparacao > 0){
86             return inserirRec(&(*raiz)->dir, nome, matricula, nota);
87         }
88     }
89     return 1;
90 }
91
92 int inserirAluno(ABP* raiz, char nome[], int matricula, double nota){
93     if(raiz == NULL) return 0;
94     return inserirRec(raiz, nome, matricula, nota);
95 }

```

```

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{

```

```

144     noMenorNota = encontrarMenorNotaRec(raiz->dir, nota);
145     if(noMenorNota == NULL || noMenorNota->aluno.nota > raiz->aluno.nota){
146         noMenorNota = raiz;
147     }
148 }
149 return noMenorNota;
150 }
151
152 Aluno encontrarMenorNota(ABP* raiz){
153     NO* noMenorNota = encontrarMenorNotaRec(*raiz, 9999);
154     return noMenorNota->aluno;
155 }
156
157 int removerRec(NO** raiz, char nome[]){
158     if(*raiz == NULL) return 0;
159     int comparar = strcmp(nome, (*raiz)->aluno.nome);
160     if(comparar == 0){
161         NO* aux;
162         if((*raiz)->esq == NULL && (*raiz)->dir == NULL){
163             liberarNO(*raiz);
164             *raiz = NULL;
165         }else if((*raiz)->esq == NULL){
166             aux = *raiz;
167             *raiz = (*raiz)->dir;
168             liberarNO(aux);
169         }else if((*raiz)->dir == NULL){
170             aux = *raiz;
171             *raiz = (*raiz)->esq;
172             liberarNO(aux);
173         }else{
174             NO* Filho = (*raiz)->esq;
175             while (Filho->dir != NULL){
176                 Filho = Filho->dir;
177             }
178             (*raiz)->aluno = Filho->aluno;
179             return removerRec(&(*raiz)->esq, nome);
180         }
181         return 1;
182     }else if(comparar < 0){
183         return removerRec(&(*raiz)->esq, nome);
184     }
185     else{
186         return removerRec(&(*raiz)->dir, nome);
187     }
188 }
189
190 NO* removerAtual(NO* atual){

```

```

191     NO* no1, *no2;
192     if(atual->esq == NULL){
193         no2 = atual->dir;
194         liberarNO(atual);
195         return no2;
196     }
197
198     no1 = atual;
199     no2 = atual->esq;
200     while(no2->dir != NULL){
201         no1 = no2;
202         no2 = no2->dir;
203     }
204     if(no1 != atual){
205         no1->dir = no2->esq;
206         no2->esq = atual->esq;
207     }
208     no2->dir = atual->dir;
209     liberarNO(atual);
210     return no2;
211 }
212
213 int removerIte(NO* raiz, char nome[]){
214     if(*raiz == NULL) return 0;
215     NO* atual = *raiz, *ant = NULL;
216     while(atual != NULL){
217         int comparar = strcmp(nome, atual->aluno.nome);
218         if(comparar == 0){
219             if(atual == *raiz){
220                 *raiz = removerAtual(atual);
221             }else if(ant->dir == atual){
222                 ant->dir = removerAtual(atual);
223             }else if(ant->esq == atual){
224                 ant->esq = removerAtual(atual);
225             }
226             return 1;
227         }
228         ant = atual;
229         if(comparar < 0){
230             atual = atual->esq;
231         }else{
232             atual = atual->dir;
233         }
234     }
235     return 0;
236 }
237
238 int removerAluno(ABP* raiz, char nome[]){
239     if(buscarAluno(raiz, nome) == 0){
240         printf("-> Elemento inexistente!\n");
241         return 0;
242     }
243     return removerIte(raiz, nome);
244 }
245
246 void em_ordem(NO* raiz, int nivel){
247     if(raiz != NULL){
248         em_ordem(raiz->esq, nivel+1);
249         printf("-> [%s, %d, %.2lf, %d] \n", raiz->aluno.nome, raiz->aluno.matricula, raiz->aluno.nota, nivel);
250         em_ordem(raiz->dir, nivel+1);
251     }
252 }
253
254 void imprime(ABP* raiz){
255     if(raiz == NULL) return;
256     if(estaVazia(raiz)){
257         printf("-> Arvore Vazia!\n");
258         return;
259     }
260     em_ordem(*raiz, 0);
261 }
262
263 #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 destruída com sucesso!

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