arvore1-2/ABP.h

```c
#ifndef ABP_H
#define ABP_H

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

typedef struct NO{
    int matricula;
    char nome[200];
    double nota;
    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);
    }
}

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


int insereRec(NO** raiz, int matricula, char* nome, double nota){
    if(*raiz == NULL){
        NO* novo = alocarNO();
        if(novo == NULL) return 0;
```

```
58          novo->matricula = matricula;
59          strcpy(novo->nome, nome);
60          novo->nota = nota;
61          novo->esq = NULL; novo->dir = NULL;
62          *raiz = novo;
63      }else{
64          if((*raiz)->matricula == matricula){
65              printf("Matricula Existente!\n");
66              return 0;
67          }
68          if(matricula < (*raiz)->matricula)
69              return insereRec(&(*raiz)->esq, matricula, nome, nota);
70          else if(matricula > (*raiz)->matricula)
71              return insereRec(&(*raiz)->dir, matricula, nome, nota);
72      }
73      return 1;
74  }
75
76  int inserematricula(ABP* raiz, int matricula, char* nome, double nota){
77      if(raiz == NULL) return 0;
78      return insereRec(raiz, matricula, nome, nota);
79  }
80
81  int pesquisaRec(NO** raiz, int matricula, char* nome){
82      if(*raiz == NULL) return 0;
83      if((*raiz)->matricula == matricula && !strcmp((*raiz)->nome, nome)) return 1;
84      if(matricula < (*raiz)->matricula)
85          return pesquisaRec(&(*raiz)->esq, matricula, nome);
86      else
87          return pesquisaRec(&(*raiz)->dir, matricula, nome);
88  }
89
90  int pesquisa(ABP* raiz, int matricula, char* nome){
91      if(raiz == NULL) return 0;
92      if(estaVazia(raiz)) return 0;
93      return pesquisaRec(raiz, matricula, nome);
94  }
95
96  int removeRec(NO** raiz, int matricula, char* nome){
97      if(*raiz == NULL) return 0;
98      if((*raiz)->matricula == matricula && !strcmp((*raiz)->nome, nome)){
99          NO* aux;
100         if((*raiz)->esq == NULL && (*raiz)->dir == NULL){
101             //Caso 1 - NO sem filhos
102             printf("Caso 1: Liberando %s..\n", (*raiz)->nome);
103             liberarNO(*raiz);
104             *raiz = NULL;
105         }else if((*raiz)->esq == NULL){
106             //Caso 2.1 - Possui apenas uma subarvore direita
107             printf("Caso 2.1: Liberando %s..\n", (*raiz)->nome);
108             aux = *raiz;
109             *raiz = (*raiz)->dir;
110             liberarNO(aux);
111         }else if((*raiz)->dir == NULL){
112             //Caso 2.2 - Possui apenas uma subarvore esquerda
113             printf("Caso 2.2: Liberando %s..\n", (*raiz)->nome);
114             aux = *raiz;
115             *raiz = (*raiz)->esq;
116             liberarNO(aux);
117         }else{
```

```
118             //Caso 3 - Possui as duas subarvores (esq e dir)
119             //Duas estrategias:
120             //3.1 - Substituir pelo NO com o MAIOR valor da subarvore esquerda
121             //3.2 - Substituir pelo NO com o MENOR valor da subarvore direita
122             printf("Caso 3: Liberando %s..\n", (*raiz)->nome);
123             //Estrategia 3.1:
124             NO* Filho = (*raiz)->esq;
125             while(Filho->dir != NULL)//Localiza o MAIOR valor da subarvore esquerda
126                 Filho = Filho->dir;
127             (*raiz)->matricula = Filho->matricula;
128             Filho->matricula = matricula;
129             return removeRec(&(*raiz)->esq, matricula, nome);
130         }
131         return 1;
132     }else if(matricula < (*raiz)->matricula)
133         return removeRec(&(*raiz)->esq, matricula, nome);
134     else
135         return removeRec(&(*raiz)->dir, matricula, nome);
136 }
137
138 int removematricula(ABP* raiz, int matricula, char* nome){
139     if(pesquisa(raiz, matricula, nome) == 0){
140         printf("Matricula inexistente!\n");
141         return 0;
142     }
143     return removeRec(raiz, matricula, nome);
144 }
145
146 void em_ordem(NO* raiz, int nivel){
147     if(raiz != NULL){
148         em_ordem(raiz->esq, nivel+1);
149         printf("[%s, %d, %.2lf, %d] \n", raiz->nome, raiz->matricula, raiz->nota, nivel);
150         em_ordem(raiz->dir, nivel+1);
151     }
152 }
153
154 void imprime(ABP* raiz){
155     if(raiz == NULL) return;
156     if(estaVazia(raiz)){
157         printf("Arvore Vazia!\n");
158         return;
159     }
160     em_ordem(*raiz, 0);
161     printf("\n");
162 }
163
164 int achaMaior(NO* raiz, int maiorNota, int matricula){
165     if(raiz == NULL) return matricula;
166     if(raiz->nota > maiorNota){
167         maiorNota = raiz->nota;
168         matricula = raiz->matricula;
169     }
170     matricula = achaMaior(raiz->esq, maiorNota, matricula);
171     matricula = achaMaior(raiz->dir, maiorNota, matricula);
172     return matricula;
173 }
174
175 void imprimeMaior(NO* raiz, int matricula){
```

```
176     if(raiz == NULL) return;
177     if(raiz->matricula == matricula){
178         printf("[%s, %d, %.2lf] \n", raiz->nome, raiz->matricula, raiz->nota);
179     }
180     if(matricula < raiz->matricula)
181         return imprimeMaior(raiz->esq, matricula);
182     else
183         return imprimeMaior(raiz->dir, matricula);
184 }
185 
186 int achaMenor(NO* raiz, int menorNota, int matricula){
187     if(raiz == NULL) return matricula;
188     if(raiz->nota < menorNota){
189         menorNota = raiz->nota;
190         matricula = raiz->matricula;
191     }
192     matricula = achaMenor(raiz->esq, menorNota, matricula);
193     matricula = achaMenor(raiz->dir, menorNota, matricula);
194     return matricula;
195 }
196 
197 void imprimeMenor(NO* raiz, int matricula){
198     if(raiz == NULL) return;
199     if(raiz->matricula == matricula){
200         printf("[%s, %d, %.2lf] \n", raiz->nome, raiz->matricula, raiz->nota);
201     }
202     if(matricula < raiz->matricula)
203         return imprimeMenor(raiz->esq, matricula);
204     else
205         return imprimeMenor(raiz->dir, matricula);
206 }
207 
208 #endif
```