arvore1-1/ABP.h

```h
#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);
    }
}

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{
        if(elem < ant->info) ant->esq = novo;
        else ant->dir = novo;
    }
    return 1;
}

int insereElem(ABP* raiz, int elem){
    if(raiz == NULL) return 0;
    return insereRec(raiz, elem);
}

int pesquisaRec(NO** raiz, int elem){
    if(*raiz == NULL) return 0;
    if((*raiz)->info == elem) return 1;
    if(elem < (*raiz)->info)
        return pesquisaRec(&(*raiz)->esq, elem);
    else
        return pesquisaRec(&(*raiz)->dir, elem);
}

int pesquisaIte(NO** raiz, int elem){
    NO* aux = *raiz;
    while(aux != NULL){
        if(aux->info == elem) return 1;
        if(elem < aux->info)
            aux = aux->esq;
        else
            aux = aux->dir;
    }
```

```
118     return 0;
119 }
120 
121 int pesquisa(ABP* raiz, int elem){
122     if(raiz == NULL) return 0;
123     if(estaVazia(raiz)) return 0;
124     return pesquisaRec(raiz, elem);
125 }
126 
127 int removeRec(NO** raiz, int elem){
128     if(*raiz == NULL) return 0;
129     if((*raiz)->info == elem){
130         NO* aux;
131         if((*raiz)->esq == NULL && (*raiz)->dir == NULL){
132             //Caso 1 - NO sem filhos
133             printf("Caso 1: Liberando %d..\n", (*raiz)->info);
134             liberarNO(*raiz);
135             *raiz = NULL;
136         }else if((*raiz)->esq == NULL){
137             printf("Caso 2.1: Liberando %d..\n", (*raiz)->info);
138             aux = *raiz;
139             *raiz = (*raiz)->dir;
140             liberarNO(aux);
141         }else if((*raiz)->dir == NULL){
142             printf("Caso 2.2: Liberando %d..\n", (*raiz)->info);
143             aux = *raiz;
144             *raiz = (*raiz)->esq;
145             liberarNO(aux);
146         }else{
147             printf("Caso 3: Liberando %d..\n", (*raiz)->info);
148             //Estrategia 3.1:
149             NO* Filho = (*raiz)->esq;
150             while(Filho->dir != NULL)
151                 Filho = Filho->dir;
152             (*raiz)->info = Filho->info;
153             Filho->info = elem;
154             return removeRec(&(*raiz)->esq, elem);
155         }
156         return 1;
157     }else if(elem < (*raiz)->info)
158         return removeRec(&(*raiz)->esq, elem);
159     else
160         return removeRec(&(*raiz)->dir, elem);
161 }
162 
163 NO* removeAtual(NO* atual){
164     NO* no1, *no2;
165     if(atual->esq == NULL){
166         no2 = atual->dir;
167         liberarNO(atual);
168         return no2;
169     }
170     no1 = atual;
171     no2 = atual->esq;
172     while(no2->dir != NULL){
173         no1 = no2;
174         no2 = no2->dir;
175     }
176     if(no1 != atual){
177         no1->dir = no2->esq;
```

```
178         no2->esq = atual->esq;
179     }
180     no2->dir = atual->dir;
181     liberarNO(atual);
182     return no2;
183 }
184 
185 int removeIte(NO** raiz, int elem){
186     if(*raiz == NULL) return 0;
187     NO* atual = *raiz, *ant = NULL;
188     while(atual != NULL){
189         if(elem == atual->info){
190             if(atual == *raiz)
191                 *raiz = removeAtual(atual);
192             else{
193                 if(ant->dir == atual)
194                     ant->dir = removeAtual(atual);
195                 else
196                     ant->esq = removeAtual(atual);
197             }
198             return 1;
199         }
200         ant = atual;
201         if(elem < atual->info)
202             atual = atual->esq;
203         else
204             atual = atual->dir;
205     }
206     return 0;
207 }
208 
209 int removeElem(ABP* raiz, int elem){
210     if(pesquisa(raiz, elem) == 0){
211         printf("Elemento inexistente!\n");
212         return 0;
213     }
214     return removeIte(raiz, elem);
215 }
216 
217 void em_ordem(NO* raiz, int nivel){
218     if(raiz != NULL){
219         em_ordem(raiz->esq, nivel+1);
220         printf("[%d, %d] \n", raiz->info,nivel);
221         em_ordem(raiz->dir, nivel+1);
222     }
223 }
224 
225 void pre_ordem(NO* raiz, int nivel){
226     if(raiz != NULL){
227         printf("[%d, %d] \n", raiz->info,nivel);
228         pre_ordem(raiz->esq, nivel+1);
229         pre_ordem(raiz->dir, nivel+1);
230     }
231 }
232 
233 void pos_ordem(NO* raiz, int nivel){
234     if(raiz != NULL){
235         pos_ordem(raiz->esq, nivel+1);
236         pos_ordem(raiz->dir, nivel+1);
237         printf("[%d, %d] \n", raiz->info,nivel);
```

```
238     }
239 }
240 
241 void imprime(ABP* raiz){
242     if(raiz == NULL) return;
243     if(estaVazia(raiz)){
244         printf("Arvore Vazia!\n");
245         return;
246     }
247     printf("\nEm Ordem: "); em_ordem(*raiz, 0);
248     printf("\nPre Ordem: "); pre_ordem(*raiz, 0);
249     printf("\nPos Ordem: "); pos_ordem(*raiz, 0);
250     printf("\n");
251 }
252 
253 void numero_de_nos(NO** raiz, int* contador){
254     if(*raiz == NULL){
255         return;
256     }
257     numero_de_nos(&(*raiz)->esq,  contador);
258     numero_de_nos(&(*raiz)->dir, contador);
259     (*contador)++;
260 }
261 
262 
263 #endif
```

arvore1-1/main.c

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

int main(){

    ABP* A;
    int n, elem;

    printf("------------------------------------------------------------------------
-------------------------\n");
    printf("ABP Sequencial Dinâmica:\n");
    printf("1 - Criar ABP;\n");
    printf("2 - Inserir elemento;\n");
    printf("3 - Buscar elemento;\n");
    printf("4 - Remover elemento;\n");
    printf("5 - Imprimir ABP em ordem;\n");
    printf("6 - Imprimir ABP em pré-ordem;\n");
    printf("7 - Imprimir 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");
    printf("------------------------------------------------------------------------
-------------------------\n");

    do{

        scanf("%d", &n);

        switch(n){

            case 1:
                A = criaABP();
                if(A != NULL) printf("ABP criada com sucesso.\n");
                break;

            case 2:
                printf("Elemento a ser inserido: ");
                scanf("%d", &elem);
                if(insereElem(A, elem)) printf("Elemento inserido com sucesso.\n");
                break;

            case 3:
                printf("Elemento a ser buscado: ");
                scanf("%d", &elem);
                if(pesquisa(A, elem)) printf("Elemento encontrado.\n");
                else
                    printf("Elemento não foi encontrado.\n");
                break;

            case 4:
                printf("Elemento a ser removido: ");
                scanf("%d", &elem);
                if(removeElem(A, elem)) printf("Elemento removido com sucesso.\n");
                break;

            case 5:
```

```c
                em_ordem(*A, 0);
                break;

            case 6:
                pre_ordem(*A, 0);
                break;

            case 7:
                pos_ordem(*A, 0);
                break;

            case 8:
                int qtd_nos = 0;
                numero_de_nos(A, (&qtd_nos));
                printf("Numero de nós: %d\n", qtd_nos);
                break;

            case 9:
                destroiABP(A);
                printf("ABP destruida com sucesso\n");
                break;

        }

        printf("------------------------------------------------------------------------
----------------------\n");

    } while(n != 10);

    printf("------------------------------------------------------------------------
-------------------------\n");

    return 0;

}
```

```
[lucascosta@fedora arvore1-1]$ gcc  main.c -o main
[lucascosta@fedora arvore1-1]$ ./main
-----------------------------------------------------------------------------------------------
ABP Sequencial Dinâmica:
1 - Criar ABP;
2 - Inserir elemento;
3 - Buscar elemento;
4 - Remover elemento;
5 - Imprimir ABP em ordem;
6 - Imprimir ABP em pré-ordem;
7 - Imprimir ABP em pós-ordem;
8 - Mostrar a quantidade de nós na ABP;
9 - Destruir a ABP;
10 - Sair.
-----------------------------------------------------------------------------------------------
1
ABP criada com sucesso.
-----------------------------------------------------------------------------------------------
2
Elemento a ser inserido: 15
Elemento inserido com sucesso.
-----------------------------------------------------------------------------------------------
2
Elemento a ser inserido: 3
Elemento inserido com sucesso.
-----------------------------------------------------------------------------------------------
2
Elemento a ser inserido: 8
Elemento inserido com sucesso.
-----------------------------------------------------------------------------------------------
2
Elemento a ser inserido: 6
Elemento inserido com sucesso.
-----------------------------------------------------------------------------------------------
5
[3, 1]
[6, 3]
[8, 2]
[15, 0]
-----------------------------------------------------------------------------------------------
6
[15, 0]
[3, 1]
[8, 2]
[6, 3]
-----------------------------------------------------------------------------------------------
7
[6, 3]
[8, 2]
[3, 1]
[15, 0]
-----------------------------------------------------------------------------------------------
8
Numero de nós: 4
-----------------------------------------------------------------------------------------------
9
ABP destruida com sucesso
-----------------------------------------------------------------------------------------------
10
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
[lucascosta@fedora arvore1-1]$
```

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
```

arvore1-2/main.c

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

int main(){

    ABP* A;
    int n, matricula;
    char nome[200];
    double nota;

    printf("--------------------------------------------------------------------------
-------------------------\n");
    printf("ABP Sequencial Dinâmica:\n");
    printf("1 - Criar ABP;\n");
    printf("2 - Inserir Aluno;\n");
    printf("3 - Buscar Aluno;\n");
    printf("4 - Remover Aluno pelo nome;\n");
    printf("5 - Imprimir ABP em ordem;\n");
    printf("6 - Imprimir informações do aluno com maior nota;\n");
    printf("7 - Imprimir informações do aluno com menor nota;\n");
    printf("8 - Destruir a ABP;\n");
    printf("9 - Sair.\n");
    printf("--------------------------------------------------------------------------
-------------------------\n");

    do{

        scanf("%d", &n);

        switch(n){

            case 1:
                A = criaABP();
                if(A != NULL) printf("ABP criada com sucesso.\n");
                break;

            case 2:
                printf("Matricula do aluno: ");
                scanf("%d", &matricula);
                printf("Nome do aluno: ");
                scanf("%s", nome);
                printf("Nota do aluno: ");
                scanf("%lf", &nota);
                if(inserematricula(A, matricula, nome, nota)) printf("Matricula
inserido com sucesso.\n");
                break;

            case 3:
                printf("Nome a ser buscado: ");
                scanf("%s", nome);
                printf("Matricula do aluno: ");
                scanf("%d", &matricula);
                if(pesquisa(A, matricula, nome)) printf("Matricula encontrado.\n");
                else
                    printf("Matricula não foi encontrado.\n");
                break;

```

```c
            case 4:
                printf("Nome a ser removido: ");
                scanf("%s", nome);
                printf("Matricula do aluno: ");
                scanf("%d", &matricula);
                if(removematricula(A, matricula, nome)) printf("Aluno removido com sucesso.\n");
                break;

            case 5:
                em_ordem(*A, 0);
                break;

            case 6:
                matricula = achaMaior(*A, (*A)->nota, (*A)->matricula);
                imprimeMaior(*A, matricula);
                break;

            case 7:
                matricula = achaMenor(*A, (*A)->nota, (*A)->matricula);
                imprimeMenor(*A, matricula);
                break;

            case 8:
                destroiABP(A);
                printf("ABP destruida com sucesso\n");
                break;

        }

        printf("--------------------------------------------------------------------------------\n");

    } while(n != 9);

    printf("--------------------------------------------------------------------------------\n");

    return 0;

}
```

```
[lucascosta@fedora arvore1-2]$ ./main
-----------------------------------------------------------------------------------------------
ABP Sequencial Dinâmica:
1 - Criar ABP;
2 - Inserir Aluno;
3 - Buscar Aluno;
4 - Remover Aluno pelo nome;
5 - Imprimir ABP em ordem;
6 - Imprimir informações do aluno com maior nota;
7 - Imprimir informações do aluno com menor nota;
8 - Destruir a ABP;
9 - Sair.
-----------------------------------------------------------------------------------------------
1
ABP criada com sucesso.
-----------------------------------------------------------------------------------------------
2
Matricula do aluno: 2020
Nome do aluno: Lucas
Nota do aluno: 10
Matricula inserido com sucesso.
-----------------------------------------------------------------------------------------------
2
Matricula do aluno: 2021
Nome do aluno: Lask
Nota do aluno: 7
Matricula inserido com sucesso.
-----------------------------------------------------------------------------------------------

2
Matricula do aluno: 2022
Nome do aluno: Luana
Nota do aluno: 5
Matricula inserido com sucesso.
-----------------------------------------------------------------------------------------------
5
[Lucas, 2020, 10.00, 0]
[Lask, 2021, 7.00, 1]
[Luana, 2022, 5.00, 2]
-----------------------------------------------------------------------------------------------
6
[Lucas, 2020, 10.00]
-----------------------------------------------------------------------------------------------
7
[Luana, 2022, 5.00]
-----------------------------------------------------------------------------------------------
4
Nome a ser removido: Lucas
Matricula do aluno: 2020
Caso 2.1: Liberando Lucas..
Aluno removido com sucesso.
-----------------------------------------------------------------------------------------------
8
ABP destruida com sucesso
-----------------------------------------------------------------------------------------------
9
-----------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------
[lucascosta@fedora arvore1-2]$
```