**ABP.h**

```c
#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;
```

```
58      }else{
59          if((*raiz)->info == elem){
60              printf("Elemento Existente!\n");
61              return 0;
62          }
63          if(elem < (*raiz)->info)
64              return insereRec(&(*raiz)->esq, elem);
65          else if(elem > (*raiz)->info)
66              return insereRec(&(*raiz)->dir, elem);
67      }
68      return 1;
69  }
70
71  int insereElem(ABP* raiz, int elem){
72      if(raiz == NULL) return 0;
73      return insereRec(raiz, elem);
74  }
75
76  int pesquisaRec(NO** raiz, int elem){
77      if(*raiz == NULL) return 0;
78      if((*raiz)->info == elem) return 1;
79      if(elem < (*raiz)->info)
80          return pesquisaRec(&(*raiz)->esq, elem);
81      else
82          return pesquisaRec(&(*raiz)->dir, elem);
83  }
84
85  int pesquisa(ABP* raiz, int elem){
86      if(raiz == NULL) return 0;
87      if(estaVazia(raiz)) return 0;
88      return pesquisaRec(raiz, elem);
89  }
90
91  int removeRec(NO** raiz, int elem){
92      if(*raiz == NULL) return 0;
93      if((*raiz)->info == elem){
94          NO* aux;
95          if((*raiz)->esq == NULL && (*raiz)->dir == NULL){   //Caso 1 - NO sem filhos
96              printf("Caso 1: Liberando %d..\n", (*raiz)->info);
97              liberarNO(*raiz);
98              *raiz = NULL;
99          }else if((*raiz)->esq == NULL){     //Caso 2.1 - Possui apenas uma subarvore
    direita
100             printf("Caso 2.1: Liberando %d..\n", (*raiz)->info);
101             aux = *raiz;
102             *raiz = (*raiz)->dir;
103             liberarNO(aux);
104         }else if((*raiz)->dir == NULL){
105             //Caso 2.2 - Possui apenas uma subarvore esquerda
106             printf("Caso 2.2: Liberando %d..\n", (*raiz)->info);
107             aux = *raiz;
108             *raiz = (*raiz)->esq;
109             liberarNO(aux);
110         }else{  //Caso 3 - Possui as duas subarvores (esq e dir)
111             printf("Caso 3: Liberando %d..\n", (*raiz)->info);
112             NO* Filho = (*raiz)->esq;
113             while(Filho->dir != NULL)//Localiza o MAIOR valor da subarvore esquerda
114                 Filho = Filho->dir;
115             (*raiz)->info = Filho->info;
116             Filho->info = elem;
```

```
117             return removeRec(&(*raiz)->esq, elem);
118         }
119         return 1;
120     }else if(elem < (*raiz)->info)
121         return removeRec(&(*raiz)->esq, elem);
122     else
123         return removeRec(&(*raiz)->dir, elem);
124 }
125 
126 NO* removeAtual(NO* atual){
127     NO* no1, *no2;
128     //Ambos casos no if(atual->esq == NULL)
129     //Caso 1 - NO sem filhos
130     //Caso 2.1 - Possui apenas uma subarvore direita
131     if(atual->esq == NULL){
132         no2 = atual->dir;
133         liberarNO(atual);
134         return no2;
135     }
136     //Caso 3 - Possui as duas subarvores (esq e dir)
137     no1 = atual;
138     no2 = atual->esq;
139     while(no2->dir != NULL){
140         no1 = no2;
141         no2 = no2->dir;
142     }
143     if(no1 != atual){
144         no1->dir = no2->esq;
145         no2->esq = atual->esq;
146     }
147     no2->dir = atual->dir;
148     liberarNO(atual);
149     return no2;
150 }
151 
152 int removeIte(NO** raiz, int elem){
153     if(*raiz == NULL) return 0;
154     NO* atual = *raiz, *ant = NULL;
155     while(atual != NULL){
156         if(elem == atual->info){
157             if(atual == *raiz)
158                 *raiz = removeAtual(atual);
159             else{
160                 if(ant->dir == atual)
161                     ant->dir = removeAtual(atual);
162                 else
163                     ant->esq = removeAtual(atual);
164             }
165             return 1;
166         }
167         ant = atual;
168         if(elem < atual->info)
169             atual = atual->esq;
170         else
171             atual = atual->dir;
172     }
173     return 0;
174 }
175 
176 int removeElem(ABP* raiz, int elem){
```

```
177     if(pesquisa(raiz, elem) == 0){
178         printf("Elemento inexistente!\n");
179         return 0;
180     }
181     //return removeRec(raiz, elem);
182     return removeIte(raiz, elem);
183 }
184 
185 void em_ordem(NO* raiz, int nivel){
186     if(raiz != NULL){
187         em_ordem(raiz->esq, nivel+1);
188         printf("[%d, %d] ", raiz->info,nivel);
189         em_ordem(raiz->dir, nivel+1);
190     }
191 }
192 
193 void pre_ordem(NO* raiz, int nivel){
194     if(raiz != NULL){
195         printf("[%d, %d] ", raiz->info,nivel);
196         pre_ordem(raiz->esq, nivel+1);
197         pre_ordem(raiz->dir, nivel+1);
198     }
199 }
200 
201 void pos_ordem(NO* raiz, int nivel){
202     if(raiz != NULL){
203         pos_ordem(raiz->esq, nivel+1);
204         pos_ordem(raiz->dir, nivel+1);
205         printf("[%d, %d] ", raiz->info,nivel);
206     }
207 }
208 
209 void imprime(ABP* raiz){
210     if(raiz == NULL) return;
211     if(estaVazia(raiz)){
212         printf("Arvore Vazia!\n");
213         return;
214     }
215     printf("\nEm Ordem: "); em_ordem(*raiz, 0);
216     printf("\nPre Ordem: "); pre_ordem(*raiz, 0);
217     printf("\nPos Ordem: "); pos_ordem(*raiz, 0);
218     printf("\n");
219 }
220 
221 int tamanho(NO* raiz, int inicio){
222     if (raiz == NULL)
223         return 0;
224     int t = 1;
225     t += tamanho(raiz->esq, 0);
226     t += tamanho(raiz->dir, 0);
227     return t;
228 }
229 
230 #endif
```

**exercicio1.c**

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

int main(){
    int opcao, elemento;
    ABP* abp = NULL;

    do {
        printf("1. Criar\n");
        printf("2. Inserir\n");
        printf("3. Buscar\n");
        printf("4. Remover\n");
        printf("5. Imprimir em Ordem\n");
        printf("6. Imprimir em Pre-Ordem\n");
        printf("7. Imprimir em Pos-Ordem\n");
        printf("8. Tamanho\n");
        printf("9. Destruir\n");
        printf("10. Sair\n");
        printf("Escolha uma opcao: ");
        scanf("%d", &opcao);

        switch(opcao){
            case 1:
                if (abp != NULL)
                    destroiABP(abp);
                abp = criaABP();
                break;
            case 2:
                printf ("Informe o elemento: ");
                scanf ("%d", &elemento);
                if (insereElem(abp, elemento))
                    printf ("Inseriu (%d).", elemento);
                else
                    printf ("Falha ao inserir.");
                break;
            case 3:
                printf ("Informe o elemento a ser buscado: ");
                scanf ("%d", &elemento);
                if (pesquisa(abp, elemento))
                    printf ("Elemento encontrado.");
                else
                    printf ("Elemento nao encontrado.");
                break;
            case 4:
                printf ("Informe o elemento a ser removido: ");
                scanf ("%d", &elemento);
                if (removeElem(abp, elemento))
                    printf ("Removeu com sucesso.");
                else
                    printf ("Falha ao remover elemento.");
                break;
            case 5:
                em_ordem(*abp, 0);
                break;
            case 6:
                pre_ordem(*abp, 0);
```

```
                break;
            case 7:
                pos_ordem(*abp, 0);
                break;
            case 8:
                printf ("Tamanho = %d", tamanho(*abp, 0));
                break;
            case 9:
                destroiABP(abp);
                break;
            case 10:
                printf ("Saindo.");
                break;
            default:
                printf ("Opcao invalida.");
                break;
        }
        printf("\n");
    } while (opcao != 10);
}
```

```
PS C:\Users\USER\OneDrive\Área de Trabalho\LabProg2\Lista 8\output> & .\'exercicio1.exe'
1. Criar
2. Inserir
3. Buscar
4. Remover
5. Imprimir em Ordem
6. Imprimir em Pre-Ordem
7. Imprimir em Pos-Ordem
8. Tamanho
9. Destruir
10. Sair
Escolha uma opcao: 1

1. Criar
2. Inserir
3. Buscar
4. Remover
5. Imprimir em Ordem
6. Imprimir em Pre-Ordem
7. Imprimir em Pos-Ordem
8. Tamanho
9. Destruir
10. Sair
Escolha uma opcao: 2
Informe o elemento: 10
Inseriu (10).
1. Criar
2. Inserir
3. Buscar
4. Remover
5. Imprimir em Ordem
6. Imprimir em Pre-Ordem
7. Imprimir em Pos-Ordem
8. Tamanho
9. Destruir
10. Sair
Escolha uma opcao: 2
Informe o elemento: 5
Inseriu (5).
1. Criar
2. Inserir
3. Buscar
4. Remover
5. Imprimir em Ordem
6. Imprimir em Pre-Ordem
7. Imprimir em Pos-Ordem
8. Tamanho
9. Destruir
10. Sair
Escolha uma opcao: 2
Informe o elemento: 15
Inseriu (15).
1. Criar
2. Inserir
3. Buscar
4. Remover
5. Imprimir em Ordem
6. Imprimir em Pre-Ordem
7. Imprimir em Pos-Ordem
8. Tamanho
9. Destruir
10. Sair
Escolha uma opcao: 3
Informe o elemento a ser buscado: 15
Elemento encontrado.
1. Criar
2. Inserir
3. Buscar
4. Remover
5. Imprimir em Ordem
6. Imprimir em Pre-Ordem
7. Imprimir em Pos-Ordem
8. Tamanho
9. Destruir
10. Sair
Escolha uma opcao: 5
[5, 1] [10, 0] [15, 1]
1. Criar
2. Inserir
3. Buscar
4. Remover
5. Imprimir em Ordem
6. Imprimir em Pre-Ordem
7. Imprimir em Pos-Ordem
8. Tamanho
9. Destruir
10. Sair
Escolha uma opcao: 6
[10, 0] [5, 1] [15, 1]
1. Criar
2. Inserir
3. Buscar
4. Remover
5. Imprimir em Ordem
6. Imprimir em Pre-Ordem
7. Imprimir em Pos-Ordem
8. Tamanho
9. Destruir
10. Sair
Escolha uma opcao: 7
[5, 1] [15, 1] [10, 0]
1. Criar
2. Inserir
3. Buscar
4. Remover
5. Imprimir em Ordem
6. Imprimir em Pre-Ordem
7. Imprimir em Pos-Ordem
8. Tamanho
9. Destruir
10. Sair
Escolha uma opcao: 4
Informe o elemento a ser removido: 15
Removeu com sucesso.
1. Criar
2. Inserir
3. Buscar
4. Remover
5. Imprimir em Ordem
6. Imprimir em Pre-Ordem
7. Imprimir em Pos-Ordem
8. Tamanho
9. Destruir
10. Sair
Escolha uma opcao: 8
Tamanho = 2
1. Criar
2. Inserir
3. Buscar
4. Remover
5. Imprimir em Ordem
6. Imprimir em Pre-Ordem
7. Imprimir em Pos-Ordem
8. Tamanho
9. Destruir
10. Sair
Escolha uma opcao: 9
1. Criar
2. Inserir
3. Buscar
4. Remover
5. Imprimir em Ordem
6. Imprimir em Pre-Ordem
7. Imprimir em Pos-Ordem
8. Tamanho
9. Destruir
10. Sair
Escolha uma opcao: 10
Saindo.
```

**exercicio2.h**

```c
#ifndef EXERCICIO2_H
#define EXERCICIO2_H
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct{
    char nome[50];
    int matricula;
    double nota;
}Aluno;

typedef struct NO{
    Aluno 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, Aluno elem){
    if(*raiz == NULL){
```

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

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

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

int pesquisa(ABP* raiz, Aluno elem){
    if(raiz == NULL) return 0;
    if(estaVazia(raiz)) return 0;
    return pesquisaRec(raiz, elem);
}

int removeRec(NO** raiz, Aluno elem){
    if(*raiz == NULL) return 0;
    int compara = strcmp((*raiz)->info.nome, elem.nome);
    if(compara == 0){
        NO* aux;
        if((*raiz)->esq == NULL && (*raiz)->dir == NULL){   //Caso 1 - NO sem filhos
            printf("Caso 1: Liberando %s..\n", (*raiz)->info.nome);
            liberarNO(*raiz);
            *raiz = NULL;
        }else if((*raiz)->esq == NULL){     //Caso 2.1 - Possui apenas uma subarvore direita
            printf("Caso 2.1: Liberando %s..\n", (*raiz)->info.nome);
            aux = *raiz;
            *raiz = (*raiz)->dir;
            liberarNO(aux);
        }else if((*raiz)->dir == NULL){
            //Caso 2.2 - Possui apenas uma subarvore esquerda
```

```
117             printf("Caso 2.2: Liberando %s..\n", (*raiz)->info.nome);
118             aux = *raiz;
119             *raiz = (*raiz)->esq;
120             liberarNO(aux);
121         }else{  //Caso 3 - Possui as duas subarvores (esq e dir)
122             printf("Caso 3: Liberando %s..\n", (*raiz)->info.nome);
123             NO* Filho = (*raiz)->esq;
124             while(Filho->dir != NULL)//Localiza o MAIOR valor da subarvore esquerda
125                 Filho = Filho->dir;
126             (*raiz)->info = Filho->info;
127             Filho->info = elem;
128             return removeRec(&(*raiz)->esq, elem);
129         }
130         return 1;
131     }else if(compara < 0)
132         return removeRec(&(*raiz)->esq, elem);
133     else
134         return removeRec(&(*raiz)->dir, elem);
135 }
136 
137 int removeElem(ABP* raiz, Aluno elem){
138     if (pesquisa(raiz, elem) == 0){
139         printf ("Elemento nao existe.\n");
140         return 0;
141     }
142     return removeRec(raiz, elem);
143 }
144 
145 void em_ordem(NO* raiz, int nivel){
146     if(raiz != NULL){
147         em_ordem(raiz->esq, nivel+1);
148         printf("[%s, %d] ", raiz->info.nome, nivel);
149         em_ordem(raiz->dir, nivel+1);
150     }
151 }
152 
153 int tamanho(NO* raiz, int inicio){
154     if (raiz == NULL)
155         return 0;
156     int t = 1;
157     t += tamanho(raiz->esq, 0);
158     t += tamanho(raiz->dir, 0);
159     return t;
160 }
161 
162 Aluno* maiorNota(NO* raiz){
163     if (raiz != NULL){
164         Aluno *dir = maiorNota(raiz->dir);
165         Aluno *esq = maiorNota(raiz->esq);
166         Aluno *maior;
167         if (dir == NULL || (esq != NULL && esq->nota > dir->nota))
168             maior = esq;
169         else
170             maior = dir;
171 
172         if (maior == NULL || raiz->info.nota >= maior->nota)
173             return &(raiz->info);
174         else
175             return maior;
176     }
```

```
177     return NULL;
178 }
179 
180 Aluno* menorNota(NO* raiz){
181     if (raiz != NULL){
182         Aluno *dir = maiorNota(raiz->dir);
183         Aluno *esq = maiorNota(raiz->esq);
184         Aluno *maior;
185         if (dir == NULL || (esq != NULL && esq->nota < dir->nota))
186             maior = esq;
187         else
188             maior = dir;
189 
190         if (maior == NULL || raiz->info.nota <= maior->nota)
191             return &(raiz->info);
192         else
193             return maior;
194     }
195     return NULL;
196 }
197 
198 #endif  //EXERCICIO2_H
```

**exercicio2.c**

```c
#include <stdio.h>
#include "exercicio2.h"

int main(){
    int opcao;
    Aluno elemento, *auxiliar;
    ABP* abp = NULL;

    do{
        printf("1. Criar\n");
        printf("2. Inserir\n");
        printf("3. Buscar\n");
        printf("4. Remover\n");
        printf("5. Imprimir em Ordem\n");
        printf("6. Imprimir infos do Aluno com maior nota\n");
        printf("7. Imprimir infos do Aluno com menor nota\n");
        printf("8. Destruir\n");
        printf("9. Sair\n");
        printf("Escolha uma opcao: ");
        scanf("%d", &opcao);

        switch (opcao){
            case 1:
                if (abp != NULL)
                    destroiABP(abp);
                abp = criaABP();
                printf ("Criada com sucesso.");
                break;
            case 2:
                printf ("Aluno a ser inserido: \n");
                printf ("Nome (50 caracteres): ");
                setbuf(stdin, NULL);
                fgets(elemento.nome, 50, stdin);
                setbuf(stdin, NULL);
                printf ("Matricula: ");
                scanf ("%d", &elemento.matricula);
                printf ("Nota: ");
                scanf ("%lf", &elemento.nota);

                if (insereElem(abp, elemento))
                    printf ("Inseriu com sucesso");
                else
                    printf ("Falha ao inserir.");
                break;
            case 3:
                printf ("Nome para buscar (50 caracteres): ");
                setbuf(stdin, NULL);
                fgets(elemento.nome, 50, stdin);
                setbuf(stdin, NULL);

                if (pesquisa(abp, elemento))
                    printf ("Aluno encontrado.");
                else
                    printf ("Aluno nao encontrado.");
                break;
            case 4:
                printf ("Nome para remover (50 caracteres): ");
```

```c
                setbuf (stdin, NULL);
                fgets(elemento.nome, 50, stdin);
                setbuf(stdin, NULL);

                if (removeElem(abp, elemento))
                    printf ("Removeu o aluno.");
                else
                    printf ("Falha ao remover o aluno.");
                break;
            case 5:
                em_ordem(*abp, 0);
                break;
            case 6:
                auxiliar = maiorNota(*abp);
                printf ("Melhor Aluno:\n");
                printf ("Nome = %s\nMatricula = %d\nNota = %.2lf", auxiliar->nome, auxiliar->matricula, auxiliar->nota);
                break;
            case 7:
                auxiliar = menorNota(*abp);
                printf ("Pior Aluno:\n");
                printf ("Nome = %s\nMatricula = %d\nNota = %.2lf", auxiliar->nome, auxiliar->matricula, auxiliar->nota);
                break;
            case 8:
                destroiABP(abp);
                break;
            case 9:
                printf ("Saindo.");
                break;
            default:
                printf ("Opcao invalida");
                break;
        }
        printf ("\n");
    } while(opcao != 9);
}
```

```
PS C:\Users\USER\OneDrive\Área de Trabalho\LabProg2\Lista 8\output> & .\'exercicio2.exe'
1. Criar
2. Inserir
3. Buscar
4. Remover
5. Imprimir em Ordem
6. Imprimir infos do Aluno com maior nota
7. Imprimir infos do Aluno com menor nota
8. Destruir
9. Sair
Escolha uma opcao: 1
Criada com sucesso.
1. Criar
2. Inserir
3. Buscar
4. Remover
5. Imprimir em Ordem
6. Imprimir infos do Aluno com maior nota
7. Imprimir infos do Aluno com menor nota
8. Destruir
9. Sair
Escolha uma opcao: 2
Aluno a ser inserido:
Nome (50 caracteres): Gabriel
Matricula: 1
Nota: 10
Inseriu com sucesso
1. Criar
2. Inserir
3. Buscar
4. Remover
5. Imprimir em Ordem
6. Imprimir infos do Aluno com maior nota
7. Imprimir infos do Aluno com menor nota
8. Destruir
9. Sair
Escolha uma opcao: 2
Aluno a ser inserido:
Nome (50 caracteres): Pedro
Matricula: 2
Nota: 5
Inseriu com sucesso
1. Criar
2. Inserir
3. Buscar
4. Remover
5. Imprimir em Ordem
6. Imprimir infos do Aluno com maior nota
7. Imprimir infos do Aluno com menor nota
8. Destruir
9. Sair
Escolha uma opcao: 5
[Pedro
, 1] [Gabriel
, 0]
1. Criar
2. Inserir
3. Buscar
4. Remover
5. Imprimir em Ordem
6. Imprimir infos do Aluno com maior nota
7. Imprimir infos do Aluno com menor nota
8. Destruir
9. Sair
Escolha uma opcao: 3
Nome para buscar (50 caracteres): Gabriel
Aluno encontrado.
1. Criar
2. Inserir
3. Buscar
4. Remover
5. Imprimir em Ordem
6. Imprimir infos do Aluno com maior nota
7. Imprimir infos do Aluno com menor nota
8. Destruir
9. Sair
Escolha uma opcao: 6
Melhor Aluno:
Nome = Gabriel

Matricula = 1
Nota = 10.00
1. Criar
2. Inserir
3. Buscar
4. Remover
5. Imprimir em Ordem
6. Imprimir infos do Aluno com maior nota
7. Imprimir infos do Aluno com menor nota
8. Destruir
9. Sair
Escolha uma opcao: 7
Pior Aluno:
Nome = Pedro

Matricula = 2
Nota = 5.00
1. Criar
2. Inserir
3. Buscar
4. Remover
5. Imprimir em Ordem
6. Imprimir infos do Aluno com maior nota
7. Imprimir infos do Aluno com menor nota
8. Destruir
9. Sair
Escolha uma opcao: 4
Nome para remover (50 caracteres): Pedro
Caso 1: Liberando Pedro
..
Removeu o aluno.
1. Criar
2. Inserir
3. Buscar
4. Remover
5. Imprimir em Ordem
6. Imprimir infos do Aluno com maior nota
7. Imprimir infos do Aluno com menor nota
8. Destruir
9. Sair
Escolha uma opcao: 8
```