

ABP.h

```
1  #ifndef ABP_H
2  #define ABP_H
3
4  #include <stdio.h>
5  #include <stdlib.h>
6
7  typedef struct NO{
8      int info;
9      struct NO* esq;
10     struct NO* dir;
11 }NO;
12
13 typedef struct NO* ABP;
14
15 NO* alocarNO(){
16     return (NO*) malloc (sizeof(NO));
17 }
18
19 void liberarNO(NO* q){
20     free(q);
21 }
22
23 ABP* criaABP(){
24     ABP* raiz = (ABP*) malloc (sizeof(ABP));
25     if(raiz != NULL)
26         *raiz = NULL;
27     return raiz;
28 }
29
30 void destroiRec(NO* no){
31     if(no == NULL) return;
32     destroiRec(no->esq);
33     destroiRec(no->dir);
34     liberarNO(no);
35     no = NULL;
36 }
37
38 void destroiABP(ABP* raiz){
39     if(raiz != NULL){
40         destroiRec(*raiz);
41         free(raiz);
42     }
43 }
44
45 int estaVazia(ABP* raiz){
46     if(raiz == NULL) return 0;
47     return (*raiz == NULL);
48 }
49
50
51 int insereRec(NO** raiz, int elem){
52     if(*raiz == NULL){
53         NO* novo = alocarNO();
54         if(novo == NULL) return 0;
55         novo->info = elem;
56         novo->esq = NULL; novo->dir = NULL;
57         *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
100             direita
101                 printf("Caso 2.1: Liberando %d..\n", (*raiz)->info);
102                 aux = *raiz;
103                 *raiz = (*raiz)->dir;
104                 liberarNO(aux);
105             }else if((*raiz)->dir == NULL){
106                 //Caso 2.2 - Possui apenas uma subarvore esquerda
107                 printf("Caso 2.2: Liberando %d..\n", (*raiz)->info);
108                 aux = *raiz;
109                 *raiz = (*raiz)->esq;
110                 liberarNO(aux);
111             }else{ //Caso 3 - Possui as duas subarvorea (esq e dir)
112                 printf("Caso 3: Liberando %d..\n", (*raiz)->info);
113                 NO* Filho = (*raiz)->esq;
114                 while(Filho->dir != NULL)//Localiza o MAIOR valor da subarvore esquerda
115                     Filho = Filho->dir;
116                 (*raiz)->info = Filho->info;
117                 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 subarvoretas (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

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "ABP.h"
4
5  int main(){
6      int opcao, elemento;
7      ABP* abp = NULL;
8
9      do {
10         printf("1. Criar\n");
11         printf("2. Inserir\n");
12         printf("3. Buscar\n");
13         printf("4. Remover\n");
14         printf("5. Imprimir em Ordem\n");
15         printf("6. Imprimir em Pre-Ordem\n");
16         printf("7. Imprimir em Pos-Ordem\n");
17         printf("8. Tamanho\n");
18         printf("9. Destruir\n");
19         printf("10. Sair\n");
20         printf("Escolha uma opcao: ");
21         scanf("%d", &opcao);
22
23         switch(opcao){
24             case 1:
25                 if (abp != NULL)
26                     destroiABP(abp);
27                 abp = criaABP();
28                 break;
29             case 2:
30                 printf ("Informe o elemento: ");
31                 scanf ("%d", &elemento);
32                 if (insereElem(abp, elemento))
33                     printf ("Inseriu (%d).", elemento);
34                 else
35                     printf ("Falha ao inserir.");
36                 break;
37             case 3:
38                 printf ("Informe o elemento a ser buscado: ");
39                 scanf ("%d", &elemento);
40                 if (pesquisa(abp, elemento))
41                     printf ("Elemento encontrado.");
42                 else
43                     printf ("Elemento nao encontrado.");
44                 break;
45             case 4:
46                 printf ("Informe o elemento a ser removido: ");
47                 scanf ("%d", &elemento);
48                 if (removeElem(abp, elemento))
49                     printf ("Removeu com sucesso.");
50                 else
51                     printf ("Falha ao remover elemento.");
52                 break;
53             case 5:
54                 em_ordem(*abp, 0);
55                 break;
56             case 6:
57                 pre_ordem(*abp, 0);
```

```
58         break;
59     case 7:
60         pos_ordem(*abp, 0);
61         break;
62     case 8:
63         printf ("Tamanho = %d", tamanho(*abp, 0));
64         break;
65     case 9:
66         destroiABP(abp);
67         break;
68     case 10:
69         printf ("Saindo.");
70         break;
71     default:
72         printf ("Opcao invalida.");
73         break;
74     }
75     printf("\n");
76 } while (opcao != 10);
77 }
```

PS C:\Users\USER\OneDrive\Área de Trabalho\LabProg2\Lista 8\output> & .\'exercicio1.exe'

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

exercicio2.h

```
1  #ifndef EXERCICIO2_H
2  #define EXERCICIO2_H
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <string.h>
6
7  typedef struct{
8      char nome[50];
9      int matricula;
10     double nota;
11 }Aluno;
12
13 typedef struct NO{
14     Aluno info;
15     struct NO* esq;
16     struct NO* dir;
17 }NO;
18
19 typedef struct NO* ABP;
20
21 NO* alocarNO() {
22     return (NO*)malloc(sizeof(NO));
23 }
24
25 void liberarNO(NO* q) {
26     free(q);
27 }
28
29 ABP* criaABP(){
30     ABP* raiz = (ABP*) malloc (sizeof(ABP));
31     if(raiz != NULL)
32         *raiz = NULL;
33     return raiz;
34 }
35
36 void destroiRec(NO* no){
37     if(no == NULL) return;
38     destroiRec(no->esq);
39     destroiRec(no->dir);
40     liberarNO(no);
41     no = NULL;
42 }
43
44 void destroiABP(ABP* raiz){
45     if(raiz != NULL){
46         destroiRec(*raiz);
47         free(raiz);
48     }
49 }
50
51 int estaVazia(ABP* raiz){
52     if(raiz == NULL) return 0;
53     return (*raiz == NULL);
54 }
55
56 int insereRec(NO** raiz, Aluno elem){
57     if(*raiz == NULL){
```



```

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

```
1  #include <stdio.h>
2  #include "exercicio2.h"
3
4  int main(){
5      int opcao;
6      Aluno elemento, *auxiliar;
7      ABP* abp = NULL;
8
9      do{
10         printf("1. Criar\n");
11         printf("2. Inserir\n");
12         printf("3. Buscar\n");
13         printf("4. Remover\n");
14         printf("5. Imprimir em Ordem\n");
15         printf("6. Imprimir infos do Aluno com maior nota\n");
16         printf("7. Imprimir infos do Aluno com menor nota\n");
17         printf("8. Destruir\n");
18         printf("9. Sair\n");
19         printf("Escolha uma opcao: ");
20         scanf("%d", &opcao);
21
22         switch (opcao){
23             case 1:
24                 if (abp != NULL)
25                     destroiABP(abp);
26                 abp = criaABP();
27                 printf ("Criada com sucesso.");
28                 break;
29             case 2:
30                 printf ("Aluno a ser inserido: \n");
31                 printf ("Nome (50 caracteres): ");
32                 setbuf(stdin, NULL);
33                 fgets(elemento.nome, 50, stdin);
34                 setbuf(stdin, NULL);
35                 printf ("Matricula: ");
36                 scanf ("%d", &elemento.matricula);
37                 printf ("Nota: ");
38                 scanf ("%lf", &elemento.nota);
39
40                 if (insereElem(abp, elemento))
41                     printf ("Inseriu com sucesso");
42                 else
43                     printf ("Falha ao inserir.");
44                 break;
45             case 3:
46                 printf ("Nome para buscar (50 caracteres): ");
47                 setbuf(stdin, NULL);
48                 fgets(elemento.nome, 50, stdin);
49                 setbuf(stdin, NULL);
50
51                 if (pesquisa(abp, elemento))
52                     printf ("Aluno encontrado.");
53                 else
54                     printf ("Aluno nao encontrado.");
55                 break;
56             case 4:
57                 printf ("Nome para remover (50 caracteres): ");
```

```
58     setbuf (stdin, NULL);
59     fgets(elemento.nome, 50, stdin);
60     setbuf(stdin, NULL);
61
62     if (removeElem(abp, elemento))
63         printf ("Removeu o aluno.");
64     else
65         printf ("Falha ao remover o aluno.");
66     break;
67 case 5:
68     em_ordem(*abp, 0);
69     break;
70 case 6:
71     auxiliar = maiorNota(*abp);
72     printf ("Melhor Aluno:\n");
73     printf ("Nome = %s\nMatricula = %d\nNota = %.2lf", auxiliar->nome,
auxiliar->matricula, auxiliar->nota);
74     break;
75 case 7:
76     auxiliar = menorNota(*abp);
77     printf ("Pior Aluno:\n");
78     printf ("Nome = %s\nMatricula = %d\nNota = %.2lf", auxiliar->nome,
auxiliar->matricula, auxiliar->nota);
79     break;
80 case 8:
81     destroiABP(abp);
82     break;
83 case 9:
84     printf ("Saindo.");
85     break;
86 default:
87     printf ("Opcao invalida");
88     break;
89 }
90 printf ("\n");
91 } while(opcao != 9);
92 }
```

```
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
```