

```
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         *raiz = NULL;
43     }
44 }
45
46 int estaVazia(ABP* raiz) {
47     if (raiz == NULL) return 0;
48     return (*raiz == NULL);
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;
57         novo->dir = NULL;
58         *raiz = novo;
59     } else {
```

```

60     if ((*raiz)->info == elem) {
61         printf("Elemento Existente!\n");
62         return 0;
63     }
64     if (elem < (*raiz)->info)
65         return insereRec(&(*raiz)->esq, elem);
66     else if (elem > (*raiz)->info)
67         return insereRec(&(*raiz)->dir, elem);
68 }
69 return 1;
70 }
71
72 int insereElem(ABP* raiz, int elem) {
73     if (raiz == NULL) return 0;
74     return insereRec(raiz, elem);
75 }
76
77 int pesquisaRec(NO** raiz, int elem) {
78     if (*raiz == NULL) return 0;
79     if ((*raiz)->info == elem) return 1;
80     if (elem < (*raiz)->info)
81         return pesquisaRec(&(*raiz)->esq, elem);
82     else
83         return pesquisaRec(&(*raiz)->dir, elem);
84 }
85
86 int pesquisa(ABP* raiz, int elem) {
87     if (raiz == NULL) return 0;
88     if (estaVazia(raiz)) return 0;
89     return pesquisaRec(raiz, elem);
90 }
91
92 int removeRec(NO** raiz, int elem) {
93     if (*raiz == NULL) return 0;
94     if ((*raiz)->info == elem) {
95         NO* aux;
96         if ((*raiz)->esq == NULL && (*raiz)->dir == NULL) { // Caso 1 - NO sem
filhos
97             printf("Caso 1: Liberando %d..\n", (*raiz)->info);
98             liberarNO(*raiz);
99             *raiz = NULL;
100     } else if ((*raiz)->esq == NULL) { // Caso 2.1 - Possui apenas uma
subarvore 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) { // 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 subarvoret (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 int removeElem(ABP* raiz, int elem) {
127     if (pesquisa(raiz, elem) == 0) {
128         printf("Elemento inexistente!\n");
129         return 0;
130     }
131     return removeRec(raiz, elem);
132 }
133
134 void em_ordem(NO* raiz, int nivel) {
135     if (raiz != NULL) {
136         em_ordem(raiz->esq, nivel + 1);
137         printf("[%d, %d] ", raiz->info, nivel);
138         em_ordem(raiz->dir, nivel + 1);
139     }
140 }
141
142 void pre_ordem(NO* raiz, int nivel) {
143     if (raiz != NULL) {
144         printf("[%d, %d] ", raiz->info, nivel);
145         pre_ordem(raiz->esq, nivel + 1);
146         pre_ordem(raiz->dir, nivel + 1);
147     }
148 }
149
150 void pos_ordem(NO* raiz, int nivel) {
151     if (raiz != NULL) {
152         pos_ordem(raiz->esq, nivel + 1);
153         pos_ordem(raiz->dir, nivel + 1);
154         printf("[%d, %d] ", raiz->info, nivel);
155     }
156 }
157
158 int tamanho(NO* raiz, int inicial) {
159     if (raiz == NULL) return 0;
160
161     int t = 1;
162     t += tamanho(raiz->esq, 0);
163     t += tamanho(raiz->dir, 0);
164     return t;
165 }
166
167 #endif

```

```

1  #include "ex01-01.h"
2
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  enum {
7      EXIT = 0,
8      CREATE,
9      INSERT,
10     SEARCH,
11     REMOVE,
12     PRINT_INORDER,
13     PRINT_PREORDER,
14     PRINT_POSTORDER,
15     SIZE,
16     DESTROY
17 } Options;
18
19 int getOption() {
20     int option;
21
22     printf("\n=====\\n");
23     printf("(%) Criar\\n", CREATE);
24     printf("(%) Inserir elemento\\n", INSERT);
25     printf("(%) Buscar elemento\\n", SEARCH);
26     printf("(%) Remover elemento\\n", REMOVE);
27     printf("(%) Imprimir em ordem\\n", PRINT_INORDER);
28     printf("(%) Imprimir em pré-ordem\\n", PRINT_PREORDER);
29     printf("(%) Imprimir em pós-ordem\\n", PRINT_POSTORDER);
30     printf("(%) Tamanho\\n", SIZE);
31     printf("(%) Destruir\\n", DESTROY);
32     printf("(%) Sair\\n", EXIT);
33     printf("=====\\n");
34     printf("Operacao: ");
35
36     scanf("%d", &option);
37     printf("\\n");
38
39     return option;
40 }
41
42 int runMenu() {
43     ABP* abp = NULL;
44     int exit = 0, item;
45
46     do {
47         switch (getOption()) {
48             case CREATE:
49                 if (abp != NULL) {
50                     destroiABP(&abp);
51                 }
52                 abp = criaABP();
53                 break;
54
55             case INSERT:
56                 printf("Elemento a ser inserido: ");
57                 scanf("%d", &item);
58
59                 if (insereElem(abp, item)) {
60                     printf("Elemento inserido (%d)", item);
61                 } else {

```

```

62         printf("Falha ao inserir (%d)", item);
63     }
64     break;
65
66     case SEARCH:
67         printf("Elemento a ser pesquisado: ");
68         scanf("%d", &item);
69
70         if (pesquisa(abp, item)) {
71             printf("Elemento presente (%d)", item);
72         } else {
73             printf("Elemento nao encontrado (%d)", item);
74         }
75         break;
76
77     case REMOVE:
78         printf("Elemento a ser removido: ");
79         scanf("%d", &item);
80
81         if (removeElem(abp, item)) {
82             printf("Elemento removido (%d)", item);
83         } else {
84             printf("Falha ao remover (%d)", item);
85         }
86         break;
87
88     case PRINT_INORDER:
89         em_ordem(*abp, 0);
90         break;
91
92     case PRINT_PREORDER:
93         pre_ordem(*abp, 0);
94         break;
95     case PRINT_POSTORDER:
96         pos_ordem(*abp, 0);
97         break;
98
99     case SIZE:
100         printf("Tamanho = %d", tamanho(*abp, 0));
101         break;
102
103     case DESTROY:
104         destroiABP(&abp);
105         break;
106
107     case EXIT:
108         if (abp != NULL) {
109             destroiABP(&abp);
110         }
111         printf("Programa encerrado");
112         exit = 1;
113         break;
114
115     default:
116         printf("Opcao desconhecida, tente novamente");
117     }
118     printf("\n");
119 } while (!exit);
120 }
121
122 int main() {
123     runMenu();
124     return 0;
125 }

```

```
gabriel-dp@gabriel-dp: ~/Desktop/dev/c/lab2/roteiro-08
gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-08$ ./ex01-01

=====
(1) Criar
(2) Inserir elemento
(3) Buscar elemento
(4) Remover elemento
(5) Imprimir em ordem
(6) Imprimir em pré-ordem
(7) Imprimir em pós-ordem
(8) Tamanho
(9) Destruir
(0) Sair
=====
Operacao: 1

=====
(1) Criar
(2) Inserir elemento
(3) Buscar elemento
(4) Remover elemento
(5) Imprimir em ordem
(6) Imprimir em pré-ordem
(7) Imprimir em pós-ordem
(8) Tamanho
(9) Destruir
(0) Sair
=====
Operacao: 2

Elemento a ser inserido: 23
Elemento inserido (23)

=====
(1) Criar
(2) Inserir elemento
(3) Buscar elemento
(4) Remover elemento
(5) Imprimir em ordem
(6) Imprimir em pré-ordem
(7) Imprimir em pós-ordem
(8) Tamanho
(9) Destruir
(0) Sair
=====
Operacao: 2

Elemento a ser inserido: 19
Elemento inserido (19)

=====
(1) Criar
(2) Inserir elemento
(3) Buscar elemento
(4) Remover elemento
(5) Imprimir em ordem
(6) Imprimir em pré-ordem
(7) Imprimir em pós-ordem
(8) Tamanho
(9) Destruir
(0) Sair
=====
Operacao: 2

Elemento a ser inserido: 24
Elemento inserido (24)

=====
(1) Criar
(2) Inserir elemento
(3) Buscar elemento
(4) Remover elemento
(5) Imprimir em ordem
(6) Imprimir em pré-ordem
(7) Imprimir em pós-ordem
(8) Tamanho
(9) Destruir
(0) Sair
=====
Operacao: 5

[19, 1] [23, 0] [24, 1]

=====
(1) Criar
(2) Inserir elemento
(3) Buscar elemento
(4) Remover elemento
(5) Imprimir em ordem
(6) Imprimir em pré-ordem
(7) Imprimir em pós-ordem
(8) Tamanho
(9) Destruir
(0) Sair
=====
Operacao: 6

[23, 0] [19, 1] [24, 1]

=====
(1) Criar
(2) Inserir elemento
(3) Buscar elemento
(4) Remover elemento
(5) Imprimir em ordem
(6) Imprimir em pré-ordem
(7) Imprimir em pós-ordem
(8) Tamanho
(9) Destruir
(0) Sair
=====
Operacao: 7

[19, 1] [24, 1] [23, 0]

=====
(1) Criar
(2) Inserir elemento
(3) Buscar elemento
(4) Remover elemento
(5) Imprimir em ordem
(6) Imprimir em pré-ordem
(7) Imprimir em pós-ordem
(8) Tamanho
(9) Destruir
(0) Sair
=====
Operacao: 8

Tamanho = 3

=====
(1) Criar
(2) Inserir elemento
(3) Buscar elemento
(4) Remover elemento
(5) Imprimir em ordem
(6) Imprimir em pré-ordem
(7) Imprimir em pós-ordem
(8) Tamanho
(9) Destruir
(0) Sair
=====
Operacao: 0

Programa encerrado
gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-08$
```

## roteiro-08/ex01-02.h

```
1  #ifndef ABP_ALUNO_H
2  #define ABP_ALUNO_H
3
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <string.h>
7
8  typedef struct {
9      char nome[50];
10     int matricula;
11     double nota;
12 } Aluno;
13
14 typedef struct NO {
15     Aluno 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         *raiz = NULL;
50     }
51 }
52
53 int estaVazia(ABP* raiz) {
54     if (raiz == NULL) return 0;
55     return (*raiz == NULL);
56 }
```

```

57
58 int insereRec(N0** raiz, Aluno elem) {
59     if (*raiz == NULL) {
60         N0* novo = alocarN0();
61         if (novo == NULL) return 0;
62         novo->info = elem;
63         novo->esq = NULL;
64         novo->dir = NULL;
65         *raiz = novo;
66     } else {
67         int comp = strcmp((*raiz)->info.nome, elem.nome);
68         if (comp == 0) {
69             printf("Elemento Existente!\n");
70             return 0;
71         }
72         if (comp < 0)
73             return insereRec(&(*raiz)->esq, elem);
74         else if (comp > 0)
75             return insereRec(&(*raiz)->dir, elem);
76     }
77     return 1;
78 }
79
80 int insereElem(ABP* raiz, Aluno elem) {
81     if (raiz == NULL) return 0;
82     return insereRec(raiz, elem);
83 }
84
85 int pesquisaRec(N0** raiz, Aluno elem) {
86     if (*raiz == NULL) return 0;
87     int comp = strcmp((*raiz)->info.nome, elem.nome);
88     if (comp == 0)
89         return 1;
90     else if (comp < 0)
91         return pesquisaRec(&(*raiz)->esq, elem);
92     else
93         return pesquisaRec(&(*raiz)->dir, elem);
94 }
95
96 int pesquisa(ABP* raiz, Aluno elem) {
97     if (raiz == NULL) return 0;
98     if (estaVazia(raiz)) return 0;
99     return pesquisaRec(raiz, elem);
100 }
101
102 int removeRec(N0** raiz, Aluno elem) {
103     if (*raiz == NULL) return 0;
104     int comp = strcmp((*raiz)->info.nome, elem.nome);
105     if (comp == 0) {
106         N0* aux;
107         if ((*raiz)->esq == NULL && (*raiz)->dir == NULL) { // Caso 1 - N0 sem
filhos
108             printf("Caso 1: Liberando %s..\n", (*raiz)->info.nome);
109             liberarN0(*raiz);
110             *raiz = NULL;
111         } else if ((*raiz)->esq == NULL) { // Caso 2.1 - Possui apenas uma
subarvore direita
112             printf("Caso 2.1: Liberando %s..\n", (*raiz)->info.nome);
113             aux = *raiz;

```



```

114         *raiz = (*raiz)->dir;
115         liberarN0(aux);
116     } 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         liberarN0(aux);
121     } else { // Caso 3 - Possui as duas subarvoret (esq e dir)
122         printf("Caso 3: Liberando %s..\n", (*raiz)->info.nome);
123         N0* 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 (comp < 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 inexistente!\n");
140         return 0;
141     }
142     return removeRec(raiz, elem);
143 }
144
145 void em_ordem(N0* 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(N0* raiz, int inicial) {
154     if (raiz == NULL) return 0;
155
156     int t = 1;
157     t += tamanho(raiz->esq, 0);
158     t += tamanho(raiz->dir, 0);
159     return t;
160 }
161
162 Aluno* melhor(N0* raiz) {
163     if (raiz != NULL) {
164         Aluno *dir = melhor(raiz->dir), *esq = melhor(raiz->esq), *maior;
165         if (dir == NULL || (esq != NULL && esq->nota > dir->nota)) {
166             maior = esq;
167         } else {
168             maior = dir;
169         }
170         if (maior == NULL || raiz->info.nota >= maior->nota) {

```

```
171         return &(raiz->info);
172     } else {
173         return maior;
174     }
175 }
176 return NULL;
177 }
178
179 Aluno* pior(NO* raiz) {
180     if (raiz != NULL) {
181         Aluno *dir = melhor(raiz->dir), *esq = melhor(raiz->esq), *maior;
182         if (dir == NULL || (esq != NULL && esq->nota < dir->nota)) {
183             maior = esq;
184         } else {
185             maior = dir;
186         }
187         if (maior == NULL || raiz->info.nota <= maior->nota) {
188             return &(raiz->info);
189         } else {
190             return maior;
191         }
192     }
193     return NULL;
194 }
195
196 void fixString(char str[50]) {
197     int length = strlen(str);
198     if (str[length - 1] == '\n') str[length - 1] = '\0';
199 }
200
201 #endif
```

## roteiro-08/ex01-02.c

```
1  #include "ex01-02.h"
2
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  enum {
7      EXIT = 0,
8      CREATE,
9      INSERT,
10     SEARCH,
11     REMOVE,
12     PRINT_INORDER,
13     PRINT_BEST,
14     PRINT_WORST,
15     DESTROY
16 } Options;
17
18 int getOption() {
19     int option;
20
21     printf("\n=====\\n");
22     printf("(%) Criar\\n", CREATE);
23     printf("(%) Inserir aluno\\n", INSERT);
24     printf("(%) Buscar aluno\\n", SEARCH);
25     printf("(%) Remover aluno\\n", REMOVE);
26     printf("(%) Imprimir em ordem\\n", PRINT_INORDER);
27     printf("(%) Imprimir melhor aluno\\n", PRINT_BEST);
28     printf("(%) Imprimir pior aluno\\n", PRINT_WORST);
29     printf("(%) Destruir\\n", DESTROY);
30     printf("(%) Sair\\n", EXIT);
31     printf("=====\\n");
32     printf("Operacao: ");
33
34     scanf("%d", &option);
35     printf("\\n");
36
37     return option;
38 }
39
40 int runMenu() {
41     ABP* abp = NULL;
42     int exit = 0;
43     Aluno item, *itemp;
44     char nome[50];
45     int matricula;
46     double nota;
47
48     do {
49         switch (getOption()) {
50             case CREATE:
51                 if (abp != NULL) {
52                     destroiABP(&abp);
53                 }
54                 abp = criaABP();
55
56                 break;
```

```

57
58     case INSERT:
59         printf("Aluno a ser inserido\n");
60         printf("Nome (Max 50 caracteres): ");
61         setbuf(stdin, NULL);
62         fgets(item.nome, 50, stdin);
63         fixString(item.nome);
64         setbuf(stdin, NULL);
65         printf("Matricula: ");
66         scanf("%d", &item.matricula);
67         printf("Nota: ");
68         scanf("%lf", &item.nota);
69
70         if (insereElem(abp, item)) {
71             printf("Aluno inserido (%s)", item.nome);
72         } else {
73             printf("Falha ao inserir (%s)", item.nome);
74         }
75         break;
76
77     case SEARCH:
78         printf("Nome para buscar (Max 50 caracteres): ");
79         setbuf(stdin, NULL);
80         fgets(item.nome, 50, stdin);
81         fixString(item.nome);
82         setbuf(stdin, NULL);
83
84         if (pesquisa(abp, item)) {
85             printf("Aluno presente (%s)", item.nome);
86         } else {
87             printf("Aluno nao encontrado (%s)", item.nome);
88         }
89         break;
90
91     case REMOVE:
92         printf("Nome para remover (Max 50 caracteres): ");
93         setbuf(stdin, NULL);
94         fgets(item.nome, 50, stdin);
95         fixString(item.nome);
96         setbuf(stdin, NULL);
97
98         if (removeElem(abp, item)) {
99             printf("Aluno removido (%s)", item.nome);
100         } else {
101             printf("Falha ao remover (%s)", item.nome);
102         }
103         break;
104
105     case PRINT_INORDER:
106         em_ordem(*abp, 0);
107         break;
108
109     case PRINT_BEST:
110         itemp = melhor(*abp);
111         printf("Melhor aluno\n");
112         printf("Nome = %s\nMatricula = %d\nNota = %.2lf", itemp->nome,
113 itemp->matricula, itemp->nota);
114         break;

```

```
115
116         case PRINT_WORST:
117             itemp = pior(*abp);
118             printf("Pior aluno\n");
119             printf("Nome = %s\nMatricula = %d\nNota = %.2lf", itemp->nome,
itemp->matricula, itemp->nota);
120             break;
121
122         case DESTROY:
123             destroiABP(&abp);
124             break;
125
126         case EXIT:
127             if (abp != NULL) {
128                 destroiABP(&abp);
129             }
130             printf("Programa encerrado");
131             exit = 1;
132             break;
133
134         default:
135             printf("Opcao desconhecida, tente novamente");
136     }
137     printf("\n");
138 } while (!exit);
139 }
140
141 int main() {
142     runMenu();
143     return 0;
144 }
```

**gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-08\$ ./ex01-02**

=====

(1) Criar  
(2) Inserir aluno  
(3) Buscar aluno  
(4) Remover aluno  
(5) Imprimir em ordem  
(6) Imprimir melhor aluno  
(7) Imprimir pior aluno  
(8) Destruir  
(0) Sair  
=====

Operacao: 1

=====

(1) Criar  
(2) Inserir aluno  
(3) Buscar aluno  
(4) Remover aluno  
(5) Imprimir em ordem  
(6) Imprimir melhor aluno  
(7) Imprimir pior aluno  
(8) Destruir  
(0) Sair  
=====

Operacao: 2

Aluno a ser inserido

Nome (Max 50 caracteres): Gabriel de Paula

Matricula: 222050003

Nota: 7.6

Aluno inserido (Gabriel de Paula)

=====

(1) Criar  
(2) Inserir aluno  
(3) Buscar aluno  
(4) Remover aluno  
(5) Imprimir em ordem  
(6) Imprimir melhor aluno  
(7) Imprimir pior aluno  
(8) Destruir  
(0) Sair  
=====

Operacao: 2

Aluno a ser inserido

Nome (Max 50 caracteres): Davi dos Reis

Matricula: 222050008

Nota: 8.5

Aluno inserido (Davi dos Reis)

=====

(1) Criar  
(2) Inserir aluno  
(3) Buscar aluno  
(4) Remover aluno  
(5) Imprimir em ordem  
(6) Imprimir melhor aluno  
(7) Imprimir pior aluno  
(8) Destruir  
(0) Sair  
=====

Operacao: 2

Aluno a ser inserido

Nome (Max 50 caracteres): Giancalo Prados

Matricula: 222050001

Nota: 9.4

Aluno inserido (Giancalo Prados)

=====

(1) Criar  
(2) Inserir aluno  
(3) Buscar aluno  
(4) Remover aluno  
(5) Imprimir em ordem  
(6) Imprimir melhor aluno  
(7) Imprimir pior aluno  
(8) Destruir  
(0) Sair  
=====

Operacao: 5

[Giancalo Prados, 1] [Gabriel de Paula, 0] [Davi dos Reis, 1]

=====

(1) Criar  
(2) Inserir aluno  
(3) Buscar aluno  
(4) Remover aluno  
(5) Imprimir em ordem  
(6) Imprimir melhor aluno  
(7) Imprimir pior aluno  
(8) Destruir  
(0) Sair  
=====

Operacao: 6

Melhor aluno

Nome = Giancalo Prados

Matricula = 222050001

Nota = 9.40

=====

(1) Criar  
(2) Inserir aluno  
(3) Buscar aluno  
(4) Remover aluno  
(5) Imprimir em ordem  
(6) Imprimir melhor aluno  
(7) Imprimir pior aluno  
(8) Destruir  
(0) Sair  
=====

Operacao: 7

Pior aluno

Nome = Gabriel de Paula

Matricula = 222050003

Nota = 7.60

=====

(1) Criar  
(2) Inserir aluno  
(3) Buscar aluno  
(4) Remover aluno  
(5) Imprimir em ordem  
(6) Imprimir melhor aluno  
(7) Imprimir pior aluno  
(8) Destruir  
(0) Sair  
=====

Operacao: 0

Programa encerrado

**gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-08\$** □