AVL.h

```h
#ifndef AVL_H
#define AVL_H
#include <stdio.h>
#include <stdlib.h>
#define MAIOR(a, b) ((a > b) ? (a) : (b))

typedef struct NO{
    int info, fb, alt;
    struct NO* esq;
    struct NO* dir;
}NO;
typedef struct NO* AVL;

NO* alocarNO(){
    return (NO*) malloc (sizeof(NO));
}

void liberarNO(NO* q){
    free(q);
}

AVL* criaAVL(){
    AVL* raiz = (AVL*) malloc (sizeof(AVL));
    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 destroiAVL(AVL* raiz){
    if(raiz != NULL){
        destroiRec(*raiz);
        free(raiz);
    }
}

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

int altura(NO* raiz){
    if(raiz == NULL) return 0;
    if(raiz->alt > 0)
        return raiz->alt;
    else{
        //printf("Calculando altura do (%d)..\n", raiz->info);
        return MAIOR(altura(raiz->esq), altura(raiz->dir)) + 1;
    }
}
```

```
58
59  int FB(NO* raiz){
60      if(raiz == NULL) return 0;
61      printf("Calculando FB do (%d)..\n", raiz->info);
62      return altura(raiz->esq) - altura(raiz->dir);
63  }
64
65  void avl_RotDir(NO** raiz){
66     printf("Rotacao Simples a DIREITA!\n");
67     NO *aux;
68     aux = (*raiz)->esq;
69     (*raiz)->esq = aux->dir;
70     aux->dir = *raiz;
71
72     //Acertando alturas e FB dos NOs afetados
73      (*raiz)->alt = aux->alt = -1;
74      aux->alt = altura(aux);
75      (*raiz)->alt = altura(*raiz);
76      aux->fb = FB(aux);
77      (*raiz)->fb = FB(*raiz);
78
79     *raiz = aux;
80  }
81
82  void avl_RotEsq(NO** raiz){
83     printf("Rotacao Simples a ESQUERDA!\n");
84     NO *aux;
85     aux = (*raiz)->dir;
86     (*raiz)->dir = aux->esq;
87     aux->esq = *raiz;
88
89     //Acertando alturas e Fatores de Balanceamento dos NOs afetados
90      (*raiz)->alt = aux->alt = -1;
91      aux->alt = altura(aux);
92      (*raiz)->alt = altura(*raiz);
93      aux->fb = FB(aux);
94      (*raiz)->fb = FB(*raiz);
95
96     *raiz = aux;
97  }
98
99
100 //Funcoes de Rotacao Dupla
101 void avl_RotEsqDir(NO** raiz){
102     printf("Rotacao Dupla ESQUERDA-DIREITA!\n");
103     NO *fe; //filho esquerdo
104     NO *ffd; //filho filho direito
105
106     fe = (*raiz)->esq;
107     ffd = fe->dir;
108
109     fe->dir = ffd->esq;
110     ffd->esq = fe;
111
112     (*raiz)->esq = ffd->dir;
113     ffd->dir = *raiz;
114
115     //Acertando alturas e Fatores de Balanceamento dos NOs afetados
116     (*raiz)->alt = fe->alt = ffd->alt = -1;
117     fe->alt = altura(fe);
```

```
118     ffd->alt = altura(ffd);
119     (*raiz)->alt = altura(*raiz);
120     fe->fb = FB(fe);
121     ffd->fb = FB(ffd);
122     (*raiz)->fb = FB(*raiz);
123 
124     *raiz = ffd;
125 }
126 
127 
128 void avl_RotDirEsq(NO** raiz){
129     printf("Rotacao Dupla DIREITA-ESQUERDA!\n");
130     NO* fd; //filho direito
131     NO* ffe; //filho filho esquerdo
132 
133     fd = (*raiz)->dir;
134     ffe = fd->esq;
135 
136     fd->esq = ffe->dir;
137     ffe->dir = fd;
138 
139     (*raiz)->dir = ffe->esq;
140     ffe->esq = *raiz;
141 
142     //Acertando alturas e Fatores de Balanceamento dos NOs afetados
143     (*raiz)->alt = fd->alt = ffe->alt = -1;
144     fd->alt = altura(fd);
145     ffe->alt = altura(ffe);
146     (*raiz)->alt = altura(*raiz);
147     fd->fb = FB(fd);
148     ffe->fb = FB(ffe);
149     (*raiz)->fb = FB(*raiz);
150 
151     *raiz = ffe;
152 }
153 
154 void avl_RotEsqDir2(NO** raiz){
155     printf("Rotacao Dupla 2 ESQUERDA-DIREITA!\n");
156     avl_RotEsq(&(*raiz)->esq);
157     avl_RotDir(raiz);
158 }
159 
160 void avl_RotDirEsq2(NO** raiz){
161     printf("Rotacao Dupla 2 DIREITA-ESQUERDA!\n");
162     avl_RotDir(&(*raiz)->dir);
163     avl_RotEsq(raiz);
164 }
165 
166 
167 //Funcoes Auxiliares referentes a cada filho
168 void avl_AuxFE(NO **raiz){
169    NO* fe;
170    fe = (*raiz)->esq;
171    if(fe->fb == +1) /* Sinais iguais e positivo*/
172      avl_RotDir(raiz);
173    else /* Sinais diferentes*/
174      avl_RotEsqDir(raiz);
175 }
176 
177 void avl_AuxFD(NO **raiz){
```

```
178     NO* fd;
179     fd = (*raiz)->dir;
180     if(fd->fb == -1) /* Sinais iguais e negativos*/
181       avl_RotEsq(raiz);
182     else /* Sinais diferentes*/
183       avl_RotDirEsq(raiz);
184 }
185 
186 int insereRec(NO** raiz, int elem){
187     int ok; //Controle para as chamadas recursivas
188     if(*raiz == NULL){
189         NO* novo = alocarNO();
190         if(novo == NULL) return 0;
191         novo->info = elem; novo->fb = 0, novo->alt = 1;
192         novo->esq = NULL; novo->dir = NULL;
193         *raiz = novo; return 1;
194     }else{
195         if((*raiz)->info == elem){
196             printf("Elemento Existente!\n"); ok = 0;
197         }
198         if(elem < (*raiz)->info){
199             ok = insereRec(&(*raiz)->esq, elem);
200             if(ok){
201                 switch((*raiz)->fb){
202                     case -1:
203                         (*raiz)->fb = 0; ok = 0; break;
204                     case 0:
205                         (*raiz)->fb = +1;
206                         (*raiz)->alt++;
207                         break;
208                     case +1:
209                         avl_AuxFE(raiz); ok = 0; break;
210                 }
211             }
212         }
213         else if(elem > (*raiz)->info){
214             ok = insereRec(&(*raiz)->dir, elem);
215             if(ok){
216                 switch((*raiz)->fb){
217                     case +1:
218                         (*raiz)->fb = 0; ok = 0; break;
219                     case 0:
220                         (*raiz)->fb = -1; (*raiz)->alt++; break;
221                     case -1:
222                         avl_AuxFD(raiz); ok = 0; break;
223                 }
224             }
225         }
226     }
227     return ok;
228 }
229 
230 int insereElem(AVL* raiz, int elem){
231     if(raiz == NULL) return 0;
232     return insereRec(raiz, elem);
233 }
234 
235 int pesquisaRec(NO** raiz, int elem){
236     if(*raiz == NULL) return 0;
237     if((*raiz)->info == elem) return 1;
```

```
238     if(elem < (*raiz)->info)
239         return pesquisaRec(&(*raiz)->esq, elem);
240     else
241         return pesquisaRec(&(*raiz)->dir, elem);
242 }
243 
244 int pesquisa(AVL* raiz, int elem){
245     if(raiz == NULL) return 0;
246     if(estaVazia(raiz)) return 0;
247     return pesquisaRec(raiz, elem);
248 }
249 
250 int removeRec(NO** raiz, int elem){
251     if(*raiz == NULL) return 0;
252     int ok;
253     if((*raiz)->info == elem){
254         NO* aux;
255         if((*raiz)->esq == NULL && (*raiz)->dir == NULL){   //Caso 1 - NO sem filhos
256             printf("Caso 1: Liberando %d..\n", (*raiz)->info);
257             liberarNO(*raiz);
258             *raiz = NULL;
259         }else if((*raiz)->esq == NULL){ //Caso 2.1 - Possui apenas uma subarvore direita
260             printf("Caso 2.1: Liberando %d..\n", (*raiz)->info);
261             aux = *raiz;
262             *raiz = (*raiz)->dir;
263             liberarNO(aux);
264         }else if((*raiz)->dir == NULL){ //Caso 2.2 - Possui apenas uma subarvore esquerda
265             printf("Caso 2.2: Liberando %d..\n", (*raiz)->info);
266             aux = *raiz;
267             *raiz = (*raiz)->esq;
268             liberarNO(aux);
269         }else{  //Caso 3 - Possui as duas subarvores (esq e dir)
270             //3.1 - Substituir pelo NO com o MAIOR valor da subarvore esquerda
271             //3.2 - Substituir pelo NO com o MENOR valor da subarvore direita
272             printf("Caso 3: Liberando %d..\n", (*raiz)->info);
273             //Estrategia 3.1:
274             NO* Filho = (*raiz)->esq;
275             while(Filho->dir != NULL)   //Localiza o MAIOR valor da subarvore esquerda
276                 Filho = Filho->dir;
277             (*raiz)->info = Filho->info;
278             Filho->info = elem;
279             return removeRec(&(*raiz)->esq, elem);
280         }
281         return 1;
282     }else if(elem < (*raiz)->info){
283         ok = removeRec(&(*raiz)->esq, elem);
284         if(ok){
285             switch((*raiz)->fb){
286                 case +1:
287                 case 0:
288                     //Acertando alturas e Fatores de Balanceamento dos NOs afetados
289                     (*raiz)->alt = -1;
290                     (*raiz)->alt = altura(*raiz);
291                     (*raiz)->fb = FB(*raiz);
292                     break;
293                 case -1:
294                     avl_AuxFD(raiz); break;
295             }
296         }
297     }
```

```
298     else{
299         ok = removeRec(&(*raiz)->dir, elem);
300         if(ok){
301             switch((*raiz)->fb){
302                 case -1:
303                 case 0:
304                     //Acertando alturas e Fatores de Balanceamento dos NOs afetados
305                     (*raiz)->alt = -1;
306                     (*raiz)->alt = altura(*raiz);
307                     (*raiz)->fb = FB(*raiz);
308                     break;
309                 case +1:
310                     avl_AuxFE(raiz); break;
311             }
312         }
313     }
314     return ok;
315 }
316 
317 int removeElem(AVL* raiz, int elem){
318     if(pesquisa(raiz, elem) == 0){
319         printf("Elemento inexistente!\n");
320         return 0;
321     }
322     return removeRec(raiz, elem);
323 }
324 
325 void em_ordem(NO* raiz, int nivel){
326     if(raiz != NULL){
327         em_ordem(raiz->esq, nivel+1);
328         //printf("[%d, %d, %d] ", raiz->info, raiz->fb, nivel);
329         printf("[%d, %d, %d, %d] ", raiz->info, raiz->fb, nivel, raiz->alt);
330         em_ordem(raiz->dir, nivel+1);
331     }
332 }
333 
334 void pre_ordem(NO* raiz, int nivel){
335     if(raiz != NULL){
336         printf("[%d, %d, %d] ", raiz->info, raiz->fb, nivel);
337         pre_ordem(raiz->esq, nivel+1);
338         pre_ordem(raiz->dir, nivel+1);
339     }
340 }
341 
342 void pos_ordem(NO* raiz, int nivel){
343     if(raiz != NULL){
344         pos_ordem(raiz->esq, nivel+1);
345         pos_ordem(raiz->dir, nivel+1);
346         printf("[%d, %d, %d] ", raiz->info, raiz->fb, nivel);
347     }
348 }
349 
350 void imprime(AVL* raiz){
351     if(raiz == NULL) return;
352     if(estaVazia(raiz)){
353         printf("Arvore Vazia!\n");
354         return;
355     }
356     //printf("\nEm Ordem: [INFO, FB, NIVEL]\n");
357     printf("\nEm Ordem: [INFO, FB, NIVEL, altura]\n");
```

```
358     em_ordem(*raiz, 0);
359     //printf("\nPre Ordem: "); pre_ordem(*raiz, 0);
360     //printf("\nPos Ordem: "); pos_ordem(*raiz, 0);
361     printf("\n");
362 }
363 
364 int tamanho(NO* raiz, int inicio){
365     if (raiz == NULL)
366         return 0;
367     int tam = 1;
368     tam += tamanho(raiz->esq, 0);
369     tam += tamanho(raiz->dir, 0);
370     return tam;
371 }
372 
373 #endif  //AVL_H
```

exercicio1.c

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

int main(){
    int opcao;
    AVL* avl = NULL;
    int elemento;

    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. Quantidade de Nos\n");
        printf("7. Destruir\n");
        printf("8. Sair\n");
        printf("Escolha uma opcao: ");
        scanf("%d", &opcao);

        switch(opcao){
            case 1:
                if (avl != NULL)
                    destroiAVL(avl);
                avl = criaAVL();
                break;
            case 2:
                printf ("Informe o elemento: ");
                scanf ("%d", &elemento);
                if (insereElem(avl, 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(avl, elemento))
                    printf ("Elemento encontrado.");
                else
                    printf ("Elemento nao encontrado.");
                break;
            case 4:
                printf ("Informe o elemento a ser removido: ");
                scanf ("%d", &elemento);
                if (removeElem(avl, elemento))
                    printf ("Removeu com sucesso.");
                else
                    printf ("Falha ao remover elemento.");
                break;
            case 5:
                em_ordem(*avl, 0);
                break;
            case 6:
                printf ("Tamanho = %d", tamanho(*avl, 0));
                break;
            case 7:
```

```c
                destroiAVL(avl);
                break;
            case 8:
                printf ("Saindo.");
                break;
            default:
                printf ("Opcao invalida.");
                break;
        }
        printf("\n");
    } while (opcao != 8);
    return 0;
}
```

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

1. Criar
2. Inserir
3. Buscar
4. Remover
5. Imprimir em Ordem
6. Quantidade de Nos
7. Destruir
8. Sair
Escolha uma opcao: 2
Informe o elemento: 10
Inseriu (10).
1. Criar
2. Inserir
3. Buscar
4. Remover
5. Imprimir em Ordem
6. Quantidade de Nos
7. Destruir
8. Sair
Escolha uma opcao: 2
Informe o elemento: 15
Inseriu (15).
1. Criar
2. Inserir
3. Buscar
4. Remover
5. Imprimir em Ordem
6. Quantidade de Nos
7. Destruir
8. 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. Quantidade de Nos
7. Destruir
8. Sair
Escolha uma opcao: 5
[10, -1, 0, 2] [15, 0, 1, 1]
1. Criar
2. Inserir
3. Buscar
4. Remover
5. Imprimir em Ordem
6. Quantidade de Nos
7. Destruir
8. Sair
Escolha uma opcao: 6
Tamanho = 2
1. Criar
2. Inserir
3. Buscar
4. Remover
5. Imprimir em Ordem
6. Quantidade de Nos
7. Destruir
8. Sair
Escolha uma opcao: 7

1. Criar
2. Inserir
3. Buscar
4. Remover
5. Imprimir em Ordem
6. Quantidade de Nos
7. Destruir
8. Sair
Escolha uma opcao: 8
Saindo.
```

exercicio2.h

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

#define MAIOR(a, b) ((a > b) ? (a) : (b))

typedef struct{
    char nome[50];
    int contratacao;
    double salario;
}Funcionario;

typedef struct NO{
    int fb, alt;
    Funcionario funcionario;
    struct NO* esq;
    struct NO* dir;
}NO;
typedef struct NO* AVL;

NO* alocarNO() {
    return (NO*)malloc(sizeof(NO));
}

void liberarNO(NO* q) {
    free(q);
}

AVL* criaAVL() {
    AVL* raiz = (AVL*)malloc(sizeof(AVL));
    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 destroiAVL(AVL** raiz) {
    if (*raiz != NULL) {
        destroiRec(**raiz);
        free(*raiz);
        *raiz = NULL;
    }
}

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

```

```
58 int altura(NO* raiz){
59     if(raiz == NULL) return 0;
60     if(raiz->alt > 0)
61         return raiz->alt;
62     else{
63         //printf("Calculando altura do (%d)..\n", raiz->info);
64         return MAIOR(altura(raiz->esq), altura(raiz->dir)) + 1;
65     }
66 }
67
68 int FB(NO* raiz){
69     if(raiz == NULL) return 0;
70     printf("Calculando FB do (%d)..\n", raiz->funcionario);
71     return altura(raiz->esq) - altura(raiz->dir);
72 }
73
74 void avl_RotDir(NO** raiz){
75    printf("Rotacao Simples a DIREITA!\n");
76    NO *aux;
77    aux = (*raiz)->esq;
78    (*raiz)->esq = aux->dir;
79    aux->dir = *raiz;
80
81    //Acertando alturas e FB dos NOs afetados
82     (*raiz)->alt = aux->alt = -1;
83     aux->alt = altura(aux);
84     (*raiz)->alt = altura(*raiz);
85     aux->fb = FB(aux);
86     (*raiz)->fb = FB(*raiz);
87
88    *raiz = aux;
89 }
90
91 void avl_RotEsq(NO** raiz){
92    printf("Rotacao Simples a ESQUERDA!\n");
93    NO *aux;
94    aux = (*raiz)->dir;
95    (*raiz)->dir = aux->esq;
96    aux->esq = *raiz;
97
98    //Acertando alturas e Fatores de Balanceamento dos NOs afetados
99     (*raiz)->alt = aux->alt = -1;
100    aux->alt = altura(aux);
101    (*raiz)->alt = altura(*raiz);
102    aux->fb = FB(aux);
103    (*raiz)->fb = FB(*raiz);
104
105   *raiz = aux;
106 }
107
108
109 //Funcoes de Rotacao Dupla
110 void avl_RotEsqDir(NO** raiz){
111     printf("Rotacao Dupla ESQUERDA-DIREITA!\n");
112     NO *fe; //filho esquerdo
113     NO *ffd; //filho filho direito
114
115     fe = (*raiz)->esq;
116     ffd = fe->dir;
117
```

```
118     fe->dir = ffd->esq;
119     ffd->esq = fe;
120 
121     (*raiz)->esq = ffd->dir;
122     ffd->dir = *raiz;
123 
124     //Acertando alturas e Fatores de Balanceamento dos NOs afetados
125     (*raiz)->alt = fe->alt = ffd->alt = -1;
126     fe->alt = altura(fe);
127     ffd->alt = altura(ffd);
128     (*raiz)->alt = altura(*raiz);
129     fe->fb = FB(fe);
130     ffd->fb = FB(ffd);
131     (*raiz)->fb = FB(*raiz);
132 
133     *raiz = ffd;
134 }
135 
136 
137 void avl_RotDirEsq(NO** raiz){
138     printf("Rotacao Dupla DIREITA-ESQUERDA!\n");
139     NO* fd; //filho direito
140     NO* ffe; //filho filho esquerdo
141 
142     fd = (*raiz)->dir;
143     ffe = fd->esq;
144 
145     fd->esq = ffe->dir;
146     ffe->dir = fd;
147 
148     (*raiz)->dir = ffe->esq;
149     ffe->esq = *raiz;
150 
151     //Acertando alturas e Fatores de Balanceamento dos NOs afetados
152     (*raiz)->alt = fd->alt = ffe->alt = -1;
153     fd->alt = altura(fd);
154     ffe->alt = altura(ffe);
155     (*raiz)->alt = altura(*raiz);
156     fd->fb = FB(fd);
157     ffe->fb = FB(ffe);
158     (*raiz)->fb = FB(*raiz);
159 
160     *raiz = ffe;
161 }
162 
163 void avl_RotEsqDir2(NO** raiz){
164     printf("Rotacao Dupla 2 ESQUERDA-DIREITA!\n");
165     avl_RotEsq(&(*raiz)->esq);
166     avl_RotDir(raiz);
167 }
168 
169 void avl_RotDirEsq2(NO** raiz){
170     printf("Rotacao Dupla 2 DIREITA-ESQUERDA!\n");
171     avl_RotDir(&(*raiz)->dir);
172     avl_RotEsq(raiz);
173 }
174 
175 
176 //Funcoes Auxiliares referentes a cada filho
177 void avl_AuxFE(NO **raiz){
```

```
178     NO* fe;
179     fe = (*raiz)->esq;
180     if(fe->fb == +1) /* Sinais iguais e positivo*/
181       avl_RotDir(raiz);
182     else /* Sinais diferentes*/
183       avl_RotEsqDir(raiz);
184 }
185 
186 void avl_AuxFD(NO **raiz){
187     NO* fd;
188     fd = (*raiz)->dir;
189     if(fd->fb == -1) /* Sinais iguais e negativos*/
190       avl_RotEsq(raiz);
191     else /* Sinais diferentes*/
192       avl_RotDirEsq(raiz);
193 }
194 
195 int insereRec(NO** raiz, Funcionario elemento){
196     int ok;
197     if (*raiz == NULL){
198         NO* novo = alocarNO();
199         if (novo == NULL)
200             return 0;
201         novo->funcionario = elemento;
202         novo->fb = 0;
203         novo->alt = 1;
204         novo->esq = NULL;
205         novo->dir = NULL;
206         *raiz = novo;
207         return 1;
208     } else{
209         if ((*raiz)->funcionario.salario == elemento.salario){
210             printf ("Elemento existente.\n");
211             ok = 0;
212         }
213         if (elemento.salario < (*raiz)->funcionario.salario){
214             ok = insereRec(&(*raiz)->esq, elemento);
215             if(ok){
216                 switch((*raiz)->fb){
217                     case -1:
218                         (*raiz)->fb = 0; ok = 0; break;
219                     case 0:
220                         (*raiz)->fb = +1;
221                         (*raiz)->alt++;
222                         break;
223                     case +1:
224                         avl_AuxFE(raiz); ok = 0; break;
225                 }
226             }
227         }else if(elemento.salario > (*raiz)->funcionario.salario){
228             ok = insereRec(&(*raiz)->dir, elemento);
229             if(ok){
230                 switch((*raiz)->fb){
231                     case +1:
232                         (*raiz)->fb = 0; ok = 0; break;
233                     case 0:
234                         (*raiz)->fb = -1; (*raiz)->alt++; break;
235                     case -1:
236                         avl_AuxFD(raiz); ok = 0; break;
237                 }
```

```
238             }
239         }
240     }
241     return ok;
242 }
243 
244 int insereElem(AVL* raiz, Funcionario elemento){
245     if(raiz == NULL)
246         return 0;
247     return insereRec(raiz, elemento);
248 }
249 
250 int pesquisaRec(NO** raiz, Funcionario elemento){
251     if(*raiz == NULL) return 0;
252     if((*raiz)->funcionario.salario == elemento.salario) return 1;
253     if(elemento.salario < (*raiz)->funcionario.salario)
254         return pesquisaRec(&(*raiz)->esq, elemento);
255     else
256         return pesquisaRec(&(*raiz)->dir, elemento);
257 }
258 
259 int pesquisa(AVL* raiz, Funcionario elemento){
260     if(raiz == NULL) return 0;
261     if(estaVazia(raiz)) return 0;
262     return pesquisaRec(raiz, elemento);
263 }
264 
265 int removeRec(NO** raiz, Funcionario elemento){
266     if(*raiz == NULL) return 0;
267     int ok;
268     if((*raiz)->funcionario.salario == elemento.salario){
269         NO* aux;
270         if((*raiz)->esq == NULL && (*raiz)->dir == NULL){   //Caso 1 - NO sem filhos
271             printf("Caso 1: Liberando %d..\n", (*raiz)->funcionario);
272             liberarNO(*raiz);
273             *raiz = NULL;
274         }else if((*raiz)->esq == NULL){ //Caso 2.1 - Possui apenas uma subarvore direita
275             printf("Caso 2.1: Liberando %d..\n", (*raiz)->funcionario);
276             aux = *raiz;
277             *raiz = (*raiz)->dir;
278             liberarNO(aux);
279         }else if((*raiz)->dir == NULL){ //Caso 2.2 - Possui apenas uma subarvore esquerda
280             printf("Caso 2.2: Liberando %d..\n", (*raiz)->funcionario);
281             aux = *raiz;
282             *raiz = (*raiz)->esq;
283             liberarNO(aux);
284         }else{  //Caso 3 - Possui as duas subarvores (esq e dir)
285             //3.1 - Substituir pelo NO com o MAIOR valor da subarvore esquerda
286             //3.2 - Substituir pelo NO com o MENOR valor da subarvore direita
287             printf("Caso 3: Liberando %d..\n", (*raiz)->funcionario);
288             //Estrategia 3.1:
289             NO* Filho = (*raiz)->esq;
290             while(Filho->dir != NULL)   //Localiza o MAIOR valor da subarvore esquerda
291                 Filho = Filho->dir;
292             (*raiz)->funcionario = Filho->funcionario;
293             Filho->funcionario = elemento;
294             return removeRec(&(*raiz)->esq, elemento);
295         }
296         return 1;
297     }else if(elemento.salario < (*raiz)->funcionario.salario){
```

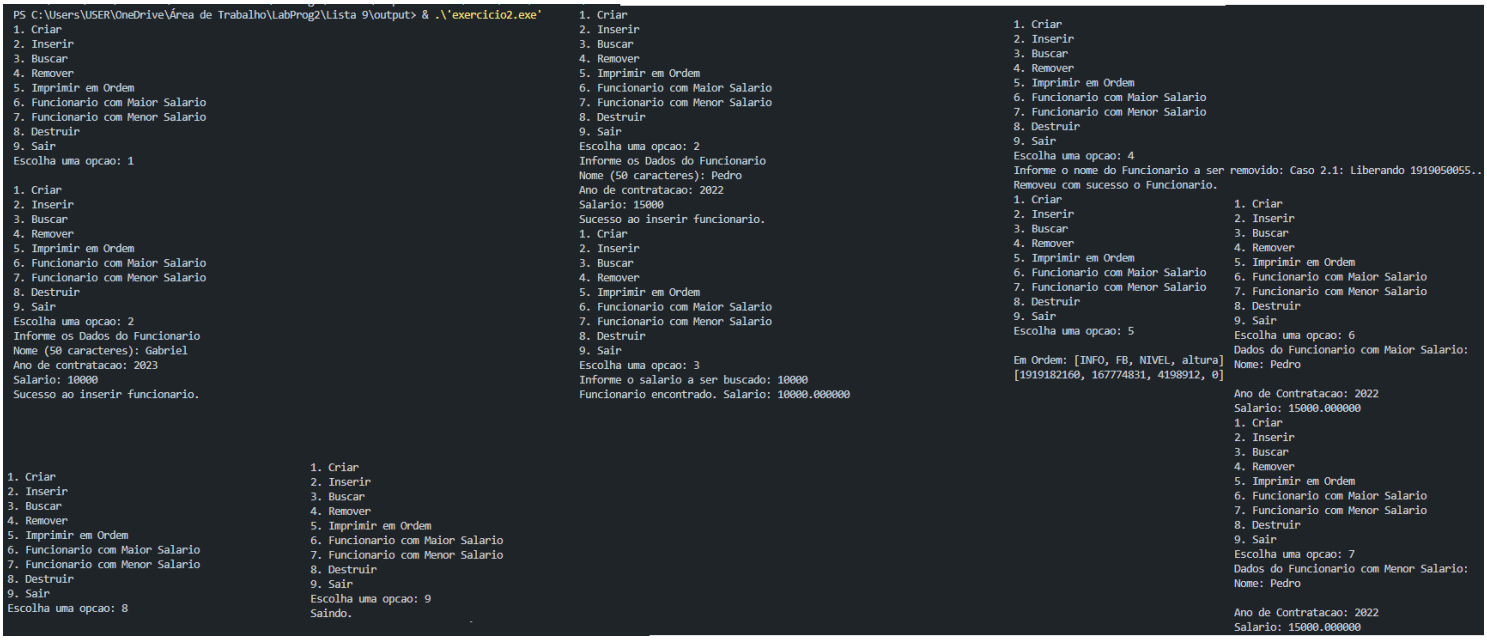
```
298         ok = removeRec(&(*raiz)->esq, elemento);
299         if(ok){
300             switch((*raiz)->fb){
301                 case +1:
302                 case 0:
303                     //Acertando alturas e Fatores de Balanceamento dos NOs afetados
304                     (*raiz)->alt = -1;
305                     (*raiz)->alt = altura(*raiz);
306                     (*raiz)->fb = FB(*raiz);
307                     break;
308                 case -1:
309                     avl_AuxFD(raiz); break;
310             }
311         }
312     }
313     else{
314         ok = removeRec(&(*raiz)->dir, elemento);
315         if(ok){
316             switch((*raiz)->fb){
317                 case -1:
318                 case 0:
319                     //Acertando alturas e Fatores de Balanceamento dos NOs afetados
320                     (*raiz)->alt = -1;
321                     (*raiz)->alt = altura(*raiz);
322                     (*raiz)->fb = FB(*raiz);
323                     break;
324                 case +1:
325                     avl_AuxFE(raiz); break;
326             }
327         }
328     }
329     return ok;
330 }
331
332 int removeElem(AVL* raiz, Funcionario elemento){
333     if(pesquisa(raiz, elemento) == 0){
334         printf("Funcionario inexistente!\n");
335         return 0;
336     }
337     return removeRec(raiz, elemento);
338 }
339
340 void em_ordem(NO* raiz, int nivel){
341     if(raiz != NULL){
342         em_ordem(raiz->esq, nivel+1);
343         //printf("[%d, %d, %d] ", raiz->info, raiz->fb, nivel);
344         printf("[%d, %d, %d, %d] ", raiz->funcionario, raiz->fb, nivel, raiz->alt);
345         em_ordem(raiz->dir, nivel+1);
346     }
347 }
348
349 void imprime(AVL* raiz){
350     if(raiz == NULL) return;
351     if(estaVazia(raiz)){
352         printf("Arvore Vazia!\n");
353         return;
354     }
355     printf("\nEm Ordem: [INFO, FB, NIVEL, altura]\n");
356     em_ordem(*raiz, 0);
357     printf("\n");
```

```
}
//Complexidade: O(n)
Funcionario* maiorSalario(NO* raiz){
    if (raiz != NULL){
        Funcionario *dir = maiorSalario(raiz->dir);
        Funcionario *esq = maiorSalario(raiz->esq);
        Funcionario* maior;

        if (dir == NULL || (esq != NULL && esq->salario > dir->salario))
            maior = dir;
        else
            maior = esq;

        if (maior == NULL || raiz->funcionario.salario >= maior->salario)
            return &(raiz->funcionario);
        else
            return maior;
    }
    return NULL;
}
// Complexidade: O(n).
Funcionario* menorSalario(NO* raiz){
    if (raiz != NULL){
        Funcionario *dir = maiorSalario(raiz->dir);
        Funcionario *esq = maiorSalario(raiz->esq);
        Funcionario* maior;

        if (dir == NULL || (esq != NULL && esq->salario < dir->salario))
            maior = dir;
        else
            maior = esq;

        if (maior == NULL || raiz->funcionario.salario <= maior->salario)
            return &(raiz->funcionario);
        else
            return maior;
    }
    return NULL;
}
#endif
```

exercicio2.c

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

int main(){
    int opcao;
    AVL* avl = NULL;
    Funcionario elemento, *auxiliar;

    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. Funcionario com Maior Salario\n");
        printf("7. Funcionario com Menor Salario\n");
        printf("8. Destruir\n");
        printf("9. Sair\n");
        printf("Escolha uma opcao: ");
        scanf("%d", &opcao);

        switch(opcao){
            case 1:
                if (avl != NULL)
                    destroiAVL(avl);
                avl = criaAVL();
                break;
            case 2:
                printf("Informe os Dados do Funcionario\n");
                printf("Nome (50 caracteres): ");
                setbuf(stdin, NULL);
                fgets(elemento.nome, 50, stdin);
                setbuf(stdin, NULL);
                printf("Ano de contratacao: ");
                scanf("%d", &elemento.contratacao);
                printf("Salario: ");
                scanf("%lf", &elemento.salario);

                if (insereElem(avl, elemento)){
                    printf ("Sucesso ao inserir funcionario.");
                } else {
                    printf ("Falha ao inserir funcionario");
                }
                break;
            case 3:
                printf ("Informe o salario a ser buscado: ");
                scanf ("%lf", &elemento.salario);
                if (pesquisa(avl, elemento))
                    printf ("Funcionario encontrado. Salario: %lf", elemento.salario);
                else
                    printf ("Funcionario nao encontrado.");
                break;
            case 4:
                printf ("Informe o nome do Funcionario a ser removido: ");
                fgets(elemento.nome, 50, stdin);
                setbuf (stdin, NULL);
                if (removeElem(avl, elemento))
```

```c
                    printf ("Removeu com sucesso o Funcionario.");
                else
                    printf ("Falha ao remover funcionario.");
                break;
            case 5:
                imprime(avl);
                break;
            case 6:
                auxiliar = maiorSalario(*avl);
                printf ("Dados do Funcionario com Maior Salario:\n");
                printf ("Nome: %s\nAno de Contratacao: %d\nSalario: %lf", auxiliar->nome, auxiliar->contratacao, auxiliar->salario);
                break;
            case 7:
                auxiliar = menorSalario(*avl);
                printf ("Dados do Funcionario com Menor Salario:\n");
                printf ("Nome: %s\nAno de Contratacao: %d\nSalario: %lf", auxiliar->nome, auxiliar->contratacao, auxiliar->salario);
                break;
            case 8:
                destroiAVL(&avl);
                break;
            case 9:
                if (avl != NULL){
                    destroiAVL(&avl);
                }
                printf ("Saindo.");
                break;
            default:
                printf ("Opcao invalida.");
                break;
        }
        printf("\n");
    } while (opcao != 9);
    return 0;
}
```


PS C:\Users\USER\OneDrive\Área de Trabalho\LabProg2\Lista 9\output> & .\'exercicio2.exe'
1. Criar
2. Inserir
3. Buscar
4. Remover
5. Imprimir em Ordem
6. Funcionario com Maior Salario
7. Funcionario com Menor Salario
8. Destruir
9. Sair
Escolha uma opcao: 1
1. Criar
2. Inserir
3. Buscar
4. Remover
5. Imprimir em Ordem
6. Funcionario com Maior Salario
7. Funcionario com Menor Salario
8. Destruir
9. Sair
Escolha uma opcao: 2
Informe os Dados do Funcionario
Nome (50 caracteres): Gabriel
Ano de contratacao: 2023
Salario: 10000
Sucesso ao inserir funcionario.
1. Criar
2. Inserir
3. Buscar
4. Remover
5. Imprimir em Ordem
6. Funcionario com Maior Salario
7. Funcionario com Menor Salario
8. Destruir
9. Sair
Escolha uma opcao: 8
1. Criar
2. Inserir
3. Buscar
4. Remover
5. Imprimir em Ordem
6. Funcionario com Maior Salario
7. Funcionario com Menor Salario
8. Destruir
9. Sair
Escolha uma opcao: 9
Saindo.
1. Criar
2. Inserir
3. Buscar
4. Remover
5. Imprimir em Ordem
6. Funcionario com Maior Salario
7. Funcionario com Menor Salario
8. Destruir
9. Sair
Escolha uma opcao: 2
Informe os Dados do Funcionario
Nome (50 caracteres): Pedro
Ano de contratacao: 2022
Salario: 15000
Sucesso ao inserir funcionario.
1. Criar
2. Inserir
3. Buscar
4. Remover
5. Imprimir em Ordem
6. Funcionario com Maior Salario
7. Funcionario com Menor Salario
8. Destruir
9. Sair
Escolha uma opcao: 3
Informe o salario a ser buscado: 10000
Funcionario encontrado. Salario: 10000.000000
1. Criar
2. Inserir
3. Buscar
4. Remover
5. Imprimir em Ordem
6. Funcionario com Maior Salario
7. Funcionario com Menor Salario
8. Destruir
9. Sair
Escolha uma opcao: 4
Informe o nome do Funcionario a ser removido: Caso 2.1: Liberando 1919050055..
Removeu com sucesso o Funcionario.
1. Criar
2. Inserir
3. Buscar
4. Remover
5. Imprimir em Ordem
6. Funcionario com Maior Salario
7. Funcionario com Menor Salario
8. Destruir
9. Sair
Escolha uma opcao: 5
Em Ordem: [INFO, FB, NIVEL, altura]
[1919182160, 167774831, 4198912, 0]
1. Criar
2. Inserir
3. Buscar
4. Remover
5. Imprimir em Ordem
6. Funcionario com Maior Salario
7. Funcionario com Menor Salario
8. Destruir
9. Sair
Escolha uma opcao: 6
Dados do Funcionario com Maior Salario:
Nome: Pedro
Ano de Contratacao: 2022
Salario: 15000.000000
1. Criar
2. Inserir
3. Buscar
4. Remover
5. Imprimir em Ordem
6. Funcionario com Maior Salario
7. Funcionario com Menor Salario
8. Destruir
9. Sair
Escolha uma opcao: 7
Dados do Funcionario com Menor Salario:
Nome: Pedro
Ano de Contratacao: 2022
Salario: 15000.000000