1.1/main.c

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

int main(){

    int c, elem;
    AVL* avl;

    printf("-----------------------------------------------------------------\n");
    printf("1 - Criar AVL;\n");
    printf("2 - Inserir um elemento;\n");
    printf("3 - Buscar um elemento;\n");
    printf("4 - Remover um elemento;\n");
    printf("5 - Imprimir a AVL em ordem;\n");
    printf("6 - Mostrar a quantidade de nós da AVL;\n");
    printf("7 - Destruir a AVL;\n");
    printf("8 - Sair.\n");
    printf("-----------------------------------------------------------------\n");

    do{
        printf("Operação: ");
        scanf("%d", &c);
        switch(c){
            case 1:
                avl = criaAVL();
                if(avl != NULL) printf("AVL criada com sucesso!\n");
                break;
            case 2:
                printf("Elemento a ser inserido: ");
                scanf("%d", &elem);
                if(insereElem(avl, elem)) printf("Elemento inserido com sucesso.\n"
);
                break;
            case 3:
                printf("Elemento a ser busacdo: ");
                scanf("%d", &elem);
                if(pesquisa(avl, elem)) printf("Elemento encontrado!\n");
                break;
            case 4:
                printf("Elemento a ser removido: ");
                scanf("%d", &elem);
                if(removeElem(avl, elem)) printf("Elemento removido com sucesso.\n"
);
                break;
            case 5:
                imprime(avl);
                break;
            case 6:
                elem = 0;
                numero_de_nos(avl, &elem);
                printf("Quantidade de nós: %d\n", elem);
                break;
            case 7:
                if(destroiAVL(avl)) printf("Árvore destruída com sucesso.\n");
                break;
            case 8:
```

```c
                break;
            default:
                printf("Comando inválido.\n");
                break;
        }
    printf("--------------------------------------------------------------------\n");
    }while(c != 8);

    return 0;

}
```

1.1/AVL.h

```c
/*---------------- File: AVL.h  ------------------------+
|Arvore AVL                                             |
|                                                       |
|                                                       |
| Implementado por Guilherme C. Pena em 23/10/2023      |
+-------------------------------------------------------+ */

#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;
}

int destroiAVL(AVL* raiz){
    if(raiz != NULL){
        destroiRec(*raiz);
        free(raiz);
        return 1;
    }
}

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

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

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

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

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

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

```
355         pre_ordem(raiz->esq, nivel+1);
356         pre_ordem(raiz->dir, nivel+1);
357     }
358 }
359 
360 void pos_ordem(NO* raiz, int nivel){
361     if(raiz != NULL){
362         pos_ordem(raiz->esq, nivel+1);
363         pos_ordem(raiz->dir, nivel+1);
364         printf("[%d, %d, %d] ", raiz->info, raiz->fb, nivel);
365     }
366 }
367 
368 void imprime(AVL* raiz){
369     if(raiz == NULL) return;
370     if(estaVazia(raiz)){
371         printf("Arvore Vazia!\n");
372         return;
373     }
374     //printf("\nEm Ordem: [INFO, FB, NIVEL]\n");
375     printf("Em Ordem: [INFO, FB, NIVEL, altura]\n");
376     em_ordem(*raiz, 0);
377     //printf("\nPre Ordem: "); pre_ordem(*raiz, 0);
378     //printf("\nPos Ordem: "); pos_ordem(*raiz, 0);
379     printf("\n");
380 }
381 
382 void numero_de_nos(NO** raiz, int* contador){
383     if(*raiz == NULL){
384         return;
385     }
386     numero_de_nos(&(*raiz)->esq, contador);
387     numero_de_nos(&(*raiz)->dir, contador);
388     (*contador)++;
389 }
390 
391 #endif
```

```
[lucascosta@fedora 1.1]$ gcc main.c -o main
[lucascosta@fedora 1.1]$ ./main
-----------------------------------------------------------------
1 - Criar AVL;
2 - Inserir um elemento;
3 - Buscar um elemento;
4 - Remover um elemento;
5 - Imprimir a AVL em ordem;
6 - Mostrar a quantidade de nós da AVL;
7 - Destruir a AVL;
8 - Sair.
-----------------------------------------------------------------
Operação: 1
AVL criada com sucesso!
-----------------------------------------------------------------
Operação: 2
Elemento a ser inserido: 5
Elemento inserido com sucesso.
-----------------------------------------------------------------
Operação: 2
Elemento a ser inserido: 6
Elemento inserido com sucesso.
-----------------------------------------------------------------
Operação: 2
Elemento a ser inserido: 7
Rotacao Simples a ESQUERDA!
Calculando FB do (6)..
Calculando FB do (5)..
-----------------------------------------------------------------
Operação: 3
Elemento a ser busacdo: 7
Elemento encontrado!
-----------------------------------------------------------------
Operação: 3
Elemento a ser busacdo: 9
-----------------------------------------------------------------
Operação: 5
Em Ordem: [INFO, FB, NIVEL, altura]
[5, 0, 1, 1] [6, 0, 0, 2] [7, 0, 1, 1]
-----------------------------------------------------------------
Operação: 6
Quantidade de nós: 3
-----------------------------------------------------------------
Operação: 4
Elemento a ser removido: 6
Caso 3: Liberando 6..
Caso 1: Liberando 6..
Elemento removido com sucesso.
-----------------------------------------------------------------
Operação: 7
Árvore destruída com sucesso.
-----------------------------------------------------------------
Operação:
```

1.2/main.c

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

int main(){

    int c;
    AVL* avl;
    char nome[90];
    double salario;
    int contratacao;

    printf("-----------------------------------------------------------------\n");
    printf("1 - Criar AVL;\n");
    printf("2 - Inserir um funcionário pelo salário;\n");
    printf("3 - Buscar um funcionário pelo salário e imprimir suas informações;\n");
    printf("4 - Remover um funcionário pelo nome;\n");
    printf("5 - Imprimir a AVL em ordem;\n");
    printf("6 - Imprimir as informações do funcionário com maior salário;\n");
    printf("7 - Imprimir as informações do funcionário com menor salário;\n");
    printf("8 - Destruir a AVL;\n");
    printf("9 - Sair.\n");
    printf("-----------------------------------------------------------------\n");

    do{
        printf("Operação: ");
        scanf("%d", &c);
        switch(c){
            case 1:
                avl = criaAVL();
                if(avl != NULL) printf("AVL criada com sucesso!\n");
                break;
            case 2:
                printf("Informações do funcionário:\n");
                printf("Nome: ");
                getchar();
                fgets(nome, 90, stdin);
                nome[strlen(nome)-1] = '\0';
                printf("Salário: ");
                scanf("%lf", &salario);
                printf("Ano de contratação: ");
                scanf("%d", &contratacao);
                if(insereElem(avl, nome, salario, contratacao)) printf("Funcionário cadastrado com sucesso.\n");
                break;
            case 3:
                printf("Salário a ser buscado: ");
                scanf("%lf", &salario);
                NO* no = pesquisaFuncionario(avl, salario);
                imprimeFuncionario(no);
                break;
            case 4:
                printf("Funcionário a ser removido: ");
                getchar();
                fgets(nome, 90, stdin);
                nome[strlen(nome)-1] = '\0';
```

```c
                if(removeElem(avl, nome)) printf("Funcionário removido com sucesso.\n");
                break;
            case 5:
                imprime(avl);
                break;
            case 6:
                imprimeMaiorSalario(avl);
                break;
            case 7:
                imprimeMenorSalario(avl);
                break;
            case 8:
                if(destroiAVL(avl)) printf("Árvore destruída com sucesso.\n");
                break;
            case 9:
                break;
            default:
                printf("Comando inválido.\n");
                break;
        }
    printf("--------------------------------------------------------------------\n");
    }while(c != 9);

    return 0;

}
```

1.2/AVL.h

```c
#ifndef AVL_H
#define AVL_H

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

typedef struct NO{
    int fb, alt;
    struct NO* esq;
    struct NO* dir;
    char nome[90];
    double salario;
    int contratacao;
}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;
}

int destroiAVL(AVL* raiz){
    if(raiz != NULL){
        destroiRec(*raiz);
        free(raiz);
        return 1;
    }
}

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

//Calcula FB
int altura(NO* raiz){
```

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

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

```c
void avl_AuxFE(NO **raiz){
   NO* fe;
   fe = (*raiz)->esq;
   if(fe->fb == +1) /* Sinais iguais e positivo*/
     avl_RotDir(raiz);
   else /* Sinais diferentes*/
     avl_RotEsqDir(raiz);
}

void avl_AuxFD(NO **raiz){
   NO* fd;
   fd = (*raiz)->dir;
   if(fd->fb == -1) /* Sinais iguais e negativos*/
     avl_RotEsq(raiz);
   else /* Sinais diferentes*/
     avl_RotDirEsq(raiz);
}

int insereRec(NO** raiz, char nome[], double salario, int contratacao){
    int ok; //Controle para as chamadas recursivas
    if(*raiz == NULL){
        NO* novo = alocarNO();
        if(novo == NULL) return 0;
        strcpy(novo->nome, nome);
        novo->salario = salario;
        novo->contratacao = contratacao;
        novo->fb = 0, novo->alt = 1;
        novo->esq = NULL; novo->dir = NULL;
        *raiz = novo; return 1;
    }else{
        if(!strcmp((*raiz)->nome, nome)){
            printf("Elemento Existente!\n"); ok = 0;
        }
        if(salario < (*raiz)->salario){
            ok = insereRec(&(*raiz)->esq, nome, salario, contratacao);
            if(ok){
                switch((*raiz)->fb){
                    case -1:
                        (*raiz)->fb = 0; ok = 0; break;
                    case 0:
                        (*raiz)->fb = +1;
                        (*raiz)->alt++;
                        break;
                    case +1:
                        avl_AuxFE(raiz); ok = 0; break;
                }
            }
        }
        else if(salario > (*raiz)->salario){
            ok = insereRec(&(*raiz)->dir, nome, salario, contratacao);
            if(ok){
                switch((*raiz)->fb){
                    case +1:
                        (*raiz)->fb = 0; ok = 0; break;
                    case 0:
                        (*raiz)->fb = -1; (*raiz)->alt++; break;
                    case -1:
                        avl_AuxFD(raiz); ok = 0; break;
                }
            }
```

```
238         }
239     }
240     return ok;
241 }
242 
243 int insereElem(AVL* raiz, char nome[], double salario, int contratacao){
244     if(raiz == NULL) return 0;
245     return insereRec(raiz, nome, salario, contratacao);
246 }
247 
248 int pesquisaRec(NO** raiz, double salario){
249     if(*raiz == NULL) return 0;
250     if((*raiz)->salario == salario) return 1;
251     if(salario < (*raiz)->salario)
252         return pesquisaRec(&(*raiz)->esq, salario);
253     else
254         return pesquisaRec(&(*raiz)->dir, salario);
255 }
256 
257 int pesquisa(AVL* raiz, double salario){
258     if(raiz == NULL) return 0;
259     if(estaVazia(raiz)) return 0;
260     return pesquisaRec(raiz, salario);
261 }
262 
263 int pesquisaRecN(NO** raiz, char nome[]){
264     int i = 0;
265     if(*raiz == NULL) return 0;
266     if(!strcmp((*raiz)->nome, nome)) return 1;
267     i = pesquisaRecN(&(*raiz)->esq, nome);
268     i = pesquisaRecN(&(*raiz)->dir, nome);
269     return i;
270 }
271 
272 int pesquisaN(AVL* raiz, char nome[]){
273     if(raiz == NULL) return 0;
274     if(estaVazia(raiz)) return 0;
275     return pesquisaRecN(raiz, nome);
276 }
277 
278 double encontraSalario(AVL* raiz, char nome[]){
279     double salario = 0;
280     if((*raiz) == NULL) return salario;
281     if(!strcmp((*raiz)->nome, nome)){
282         salario = (*raiz)->salario;
283     }
284     salario = encontraSalario(&(*raiz)->esq, nome);
285     salario = encontraSalario(&(*raiz)->dir, nome);
286     return salario;
287 }
288 
289 int removeRec(NO** raiz, char nome[]){
290     if(*raiz == NULL) return 0;
291     int ok;
292     double salario = encontraSalario(raiz, nome);
293     if(!strcmp((*raiz)->nome, nome)){
294         NO* aux;
295         if((*raiz)->esq == NULL && (*raiz)->dir == NULL){
296             //Caso 1 - NO sem filhos
297             printf("Caso 1: Liberando %s..\n", (*raiz)->nome);
```

```
298             liberarNO(*raiz);
299             *raiz = NULL;
300         }else if((*raiz)->esq == NULL){
301             //Caso 2.1 - Possui apenas uma subarvore direita
302             printf("Caso 2.1: Liberando %s..\n", (*raiz)->nome);
303             aux = *raiz;
304             *raiz = (*raiz)->dir;
305             liberarNO(aux);
306         }else if((*raiz)->dir == NULL){
307             //Caso 2.2 - Possui apenas uma subarvore esquerda
308             printf("Caso 2.2: Liberando %s..\n", (*raiz)->nome);
309             aux = *raiz;
310             *raiz = (*raiz)->esq;
311             liberarNO(aux);
312         }else{
313             //Caso 3 - Possui as duas subarvores (esq e dir)
314             //Duas estrategias:
315             //3.1 - Substituir pelo NO com o MAIOR valor da subarvore esquerda
316             //3.2 - Substituir pelo NO com o MENOR valor da subarvore direita
317             printf("Caso 3: Liberando %s..\n", (*raiz)->nome);
318             //Estrategia 3.1:
319             NO* Filho = (*raiz)->esq;
320             NO* aux = (*raiz);
321             while(Filho->dir != NULL)//Localiza o MAIOR valor da subarvore
    esquerda
322                 Filho = Filho->dir;
323             strcpy((*raiz)->nome, Filho->nome);
324             (*raiz)->salario = Filho->salario;
325             (*raiz)->contratacao = Filho->contratacao;
326             strcpy(Filho->nome, aux->nome);
327             Filho->salario = aux->salario;
328             Filho->contratacao = aux->contratacao;
329             return removeRec(&(*raiz)->esq, nome);
330         }
331         return 1;
332     }else if(salario < (*raiz)->salario){
333         ok = removeRec(&(*raiz)->esq, nome);
334         if(ok){
335             switch((*raiz)->fb){
336                 case +1:
337                 case 0:
338                     //Acertando alturas e Fatores de Balanceamento dos NOs
    afetados
339                     (*raiz)->alt = -1;
340                     (*raiz)->alt = altura(*raiz);
341                     (*raiz)->fb = FB(*raiz);
342                     break;
343                 case -1:
344                     avl_AuxFD(raiz); break;
345             }
346         }
347     }
348     else{
349         ok = removeRec(&(*raiz)->dir, nome);
350         if(ok){
351             switch((*raiz)->fb){
352                 case -1:
353                 case 0:
354                     //Acertando alturas e Fatores de Balanceamento dos NOs
    afetados
355                     (*raiz)->alt = -1;
```

```
356                     (*raiz)->alt = altura(*raiz);
357                     (*raiz)->fb = FB(*raiz);
358                     break;
359                 case +1:
360                     avl_AuxFE(raiz); break;
361             }
362         }
363     }
364     return ok;
365 }
366 
367 int removeElem(AVL* raiz, char nome[]){
368     if(!pesquisaN(raiz, nome)){
369         printf("Funcionário inexistente!\n");
370         return 0;
371     }
372     return removeRec(raiz, nome);
373 }
374 
375 void em_ordem(NO* raiz, int nivel){
376     if(raiz != NULL){
377         em_ordem(raiz->esq, nivel+1);
378         //printf("[%d, %d, %d] ", raiz->info, raiz->fb, nivel);
379         printf("[%s, R$%.2lf, %d, %d, %d, %d] ", raiz->nome, raiz->salario, raiz->
    contratacao, raiz->fb, nivel, raiz->alt);
380         em_ordem(raiz->dir, nivel+1);
381     }
382 }
383 
384 /*
385 void pre_ordem(NO* raiz, int nivel){
386     if(raiz != NULL){
387         printf("[%d, %d, %d] ", raiz->info, raiz->fb, nivel);
388         pre_ordem(raiz->esq, nivel+1);
389         pre_ordem(raiz->dir, nivel+1);
390     }
391 }
392 */
393 
394 /*
395 void pos_ordem(NO* raiz, int nivel){
396     if(raiz != NULL){
397         pos_ordem(raiz->esq, nivel+1);
398         pos_ordem(raiz->dir, nivel+1);
399         printf("[%d, %d, %d] ", raiz->info, raiz->fb, nivel);
400     }
401 }
402 */
403 
404 void imprime(AVL* raiz){
405     if(raiz == NULL) return;
406     if(estaVazia(raiz)){
407         printf("Arvore Vazia!\n");
408         return;
409     }
410     //printf("\nEm Ordem: [INFO, FB, NIVEL]\n");
411     printf("Em Ordem: [NOME, SALÁRIO, ANO DE CONTRATAÇÃO, FB, NIVEL, ALTURA]\n");
412     em_ordem(*raiz, 0);
413     //printf("\nPre Ordem: "); pre_ordem(*raiz, 0);
414     //printf("\nPos Ordem: "); pos_ordem(*raiz, 0);
```

```
415     printf("\n");
416 }
417 
418 void numero_de_nos(NO** raiz, int* contador){
419     if(*raiz == NULL){
420         return;
421     }
422     numero_de_nos(&(*raiz)->esq, contador);
423     numero_de_nos(&(*raiz)->dir, contador);
424     (*contador)++;
425 }
426 
427 void imprimeMaiorSalario(AVL* raiz){
428     if((*raiz)->dir == NULL){
429         printf("FUNCIONÀRIO COM MAIOR SALÁRIO:\n");
430         printf("NOME: %s\n", (*raiz)->nome);
431         printf("SALÁRIO: $%.2lf\n", (*raiz)->salario);
432         printf("ANO DE CONTRATAÇÃO: %d\n", (*raiz)->contratacao);
433         return;
434     }
435     imprimeMaiorSalario(&(*raiz)->dir);
436 }
437 
438 void imprimeMenorSalario(AVL* raiz){
439     if((*raiz)->esq == NULL){
440         printf("FUNCIONÀRIO COM MENOR SALÁRIO:\n");
441         printf("NOME: %s\n", (*raiz)->nome);
442         printf("SALÁRIO: $%.2lf\n", (*raiz)->salario);
443         printf("ANO DE CONTRATAÇÃO: %d\n", (*raiz)->contratacao);
444         return;
445     }
446     imprimeMenorSalario(&(*raiz)->esq);
447 }
448 
449 NO* pesquisaFuncionarioRec(AVL *raiz, double salario) {
450     if (*raiz == NULL) return NULL;
451     if ((*raiz)->salario == salario) return *raiz;
452     if (salario < (*raiz)->salario)
453         return pesquisaFuncionarioRec(&(*raiz)->esq, salario);
454     else
455         return pesquisaFuncionarioRec(&(*raiz)->dir, salario);
456 }
457 
458 NO* pesquisaFuncionario(AVL *raiz, double salario) {
459     if (raiz == NULL) return NULL;
460     if (estaVazia(raiz)) return NULL;
461     return pesquisaFuncionarioRec(raiz, salario);
462 }
463 
464 
465 
466 void imprimeFuncionario(NO* no) {
467     printf("Nome: %s, Salario: %.2lf, Ano de Contratacao: %d\n",
468            no->nome, no->salario, no->contratacao);
469 }
470 
471 
472 #endif
```

```
5 - Imprimir a AVL em ordem;
6 - Imprimir as informações do funcionário com maior salário;
7 - Imprimir as informações do funcionário com menor salário;
8 - Destruir a AVL;
9 - Sair.
---------------------------------------------------------------
Operação: 1
AVL criada com sucesso!
---------------------------------------------------------------
Operação: 2
Informações do funcionário:
Nome: Lucas
Salário: 1422
Ano de contratação: 2015
Funcionário cadastrado com sucesso.
---------------------------------------------------------------
Operação: 3
Salário a ser buscado: 1422
Nome: Lucas, Salario: 1422.00, Ano de Contratacao: 2015
---------------------------------------------------------------
Operação: 2
Informações do funcionário:
Nome: Larissa
Salário: 4500
Ano de contratação: 2013
Funcionário cadastrado com sucesso.
---------------------------------------------------------------
Operação: 2
Informações do funcionário:
Nome: Luana
Salário: 7555
Ano de contratação: 2011
Rotacao Simples a ESQUERDA!
Calculando FB de (Larissa)..
Calculando FB de (Lucas)..
---------------------------------------------------------------
Operação: 5
Em Ordem: [NOME, SALÁRIO, ANO DE CONTRATAÇÃO, FB, NIVEL, ALTURA]
[Lucas, R$1422.00, 2015, 0, 1, 1] [Larissa, R$4500.00, 2013, 0, 0, 2] [Luana, R$7555.00, 2011, 0, 1, 1]
---------------------------------------------------------------
Operação: 6
FUNCIONÁRIO COM MAIOR SALÁRIO:
NOME: Luana
SALÁRIO: $7555.00
ANO DE CONTRATAÇÃO: 2011
---------------------------------------------------------------
Operação: 7
FUNCIONÁRIO COM MENOR SALÁRIO:
NOME: Lucas
SALÁRIO: $1422.00
ANO DE CONTRATAÇÃO: 2015
---------------------------------------------------------------
Operação: 4
Funcionário a ser removido: Luana
---------------------------------------------------------------
Operação: 8
Árvore destruída com sucesso.
---------------------------------------------------------------
Operação: 9
---------------------------------------------------------------
[lucascosta@fedora 1.2]$
```