1.1/busca.h

```c
#ifndef BUSCA_H
#define BUSCA_H

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

//Medidas de Complexidade
int comp; //Num. de comparacoes

int* copiaVetor(int* v, int n){
    int i;
    int *v2;
    v2 = (int*) malloc (n*sizeof(int));
    for(i=0; i<n; i++) v2[i] = v[i];
    return v2;
}
void imprimeVetor(int* v, int n){
    int i, prim = 1;
    printf("[");
    for(i=0; i<n; i++)
        if(prim){ printf("%d", v[i]); prim = 0; }
        else printf(", %d", v[i]);
    printf("]\n");
}

void preencheAleatorio(int* v, int n, int ini, int fim){
    int i;
    for(i=0; i<n; i++)
        v[i] = ini + rand() % (fim-ini + 1);
}

void troca(int* a, int *b){
    int aux = *a;
    *a = *b;
    *b = aux;
}

int buscaSequencial(int *v, int n, int elem){
    int i;
    for(i=0; i<n; i++){
        comp++;
        if(v[i] == elem)
            return i; //Elemento encontrado
    }
    return -1; //Elemento encontrado
}

int particao(int *v, int ini, int fim){
    int i = ini, j = fim;
    int pivo = v[(ini+fim)/2];
    while (1) {
        while(v[i] < pivo){ i++; } //procura algum >= pivo do lado esquerdo
        while(v[j] > pivo){ j--; } //procura algum <= pivo do lado direito

        if(i<j){
            troca(&v[i], &v[j]); //troca os elementos encontrados
```

```
58              i++;
59              j--;
60          }else
61              return j; //retorna o local onde foi feita a particao
62      }
63  }
64
65  void QuickSort(int *v, int ini, int fim){
66      if(ini < fim ){
67          int q = particao(v, ini, fim);
68          QuickSort(v, ini, q);
69          QuickSort(v, q+1, fim);
70      }
71  }
72
73  int rec_buscaBinaria(int *v, int ini, int fim, int elem){
74      if(ini > fim) return -1;
75      int meio = (ini + fim)/2;
76      comp++;
77      if(v[meio] == elem)
78          return meio;
79      else
80          if(elem < v[meio])
81              return rec_buscaBinaria(v, ini, meio-1, elem);
82          else
83              return rec_buscaBinaria(v, meio+1, fim, elem);
84  }
85
86  int it_buscaBinaria(int *v, int ini, int fim, int elem){
87      int meio;
88      while(ini <= fim){
89          meio = (ini + fim)/2;
90          comp++;
91          if(elem == v[meio]) return meio;
92          else
93              if(elem < v[meio])
94                  fim = meio-1;
95              else
96                  ini = meio+1;
97      }
98      return -1;
99  }
100
101 #endif
```

1.1/main.c

```c
#include "busca.h"


int main(){

    //Atribuicoes iniciais
    srand(time(NULL));
    comp = 0;
    clock_t t;

    /*
    //Template de Calculo do Tempo de Execucao
    t = clock();
    //Chamada do Algoritmo aqui...
    t = clock() - t;
    printf ("It took me %d clicks (%f seconds).\n",t,((float)t)/CLOCKS_PER_SEC);
    */

    int *v;
    int n, x;
    printf("Digite o tamanho do vetor:\n");
    scanf("%d", &n);
    v = (int*) malloc (n*sizeof(int));


    preencheAleatorio(v, n, 1, n);
    //imprimeVetor(v, n);
    QuickSort(v, 0, n-1);

    printf("Digite um elemento para busca:\n");
    scanf("%d", &x);

    int ind;

    t = clock();
    ind = buscaSequencial(v, n, x);
    t = clock() - t;
    printf("-----Informacoes Busca Sequencial:\n");
    printf("Tempo Execucao:  %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
    printf("Comparacoes: %d\n", comp);

    //imprimeVetor(v, n);

    if(ind != -1)
        printf("O elemento %d foi encontrado na pos %d.\n", x, ind);
    else
        printf("O elemento %d NAO foi encontrado!\n", x);



    //imprimeVetor(v, n);

    comp = 0;
    t = clock();
    ind = rec_buscaBinaria(v, 0, n-1, x);
    t = clock() - t;
    printf("-----Informacoes Busca Binaria Recursiva:\n");
```

```c
    printf("Tempo Execucao:  %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
    printf("Comparacoes: %d\n", comp);

    if(ind != -1)
        printf("O elemento %d foi encontrado na pos %d.\n", x, ind);
    else
        printf("O elemento %d NAO foi encontrado!\n", x);


    comp = 0;
    t = clock();
    ind = it_buscaBinaria(v, 0, n-1, x);
    t = clock() - t;
    printf("-----Informacoes Busca Binaria Iterativa:\n");
    printf("Tempo Execucao:  %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
    printf("Comparacoes: %d\n", comp);

    if(ind != -1)
        printf("O elemento %d foi encontrado na pos %d.\n", x, ind);
    else
        printf("O elemento %d NAO foi encontrado!\n", x);


    free(v);
    return 0;
}
```

```
[lucascosta@fedora 1.1]$ ./main
Digite o tamanho do vetor:
7
Digite um elemento para busca:
25
-----Informacoes Busca Sequencial:
Tempo Execucao:  0.000004 seconds.
Comparacoes: 7
O elemento 25 NAO foi encontrado!
-----Informacoes Busca Binaria Recursiva:
Tempo Execucao:  0.000003 seconds.
Comparacoes: 3
O elemento 25 NAO foi encontrado!
-----Informacoes Busca Binaria Iterativa:
Tempo Execucao:  0.000002 seconds.
Comparacoes: 3
O elemento 25 NAO foi encontrado!
[lucascosta@fedora 1.1]$
```

1.2/busca.h

```c
#ifndef BUSCA_H
#define BUSCA_H

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

//Medidas de Complexidade
int comp; //Num. de comparacoes

int* copiaVetor(int* v, int n){
    int i;
    int *v2;
    v2 = (int*) malloc (n*sizeof(int));
    for(i=0; i<n; i++) v2[i] = v[i];
    return v2;
}
void imprimeVetor(int* v, int n){
    int i, prim = 1;
    printf("[");
    for(i=0; i<n; i++)
        if(prim){ printf("%d", v[i]); prim = 0; }
        else printf(", %d", v[i]);
    printf("]\n");
}

void preencheAleatorio(int* v, int n, int ini, int fim){
    int i;
    for(i=0; i<n; i++)
        v[i] = ini + rand() % (fim-ini + 1);
}

void troca(int* a, int *b){
    int aux = *a;
    *a = *b;
    *b = aux;
}

int buscaSequencial(int *v, int n, int elem){
    int i;
    for(i=0; i<n; i++){
        comp++;
        if(v[i] == elem)
            return i; //Elemento encontrado
    }
    return -1; //Elemento encontrado
}

int particao(int *v, int ini, int fim){
    int i = ini, j = fim;
    int pivo = v[(ini+fim)/2];
    while (1) {
        while(v[i] > pivo){ i++; } //procura algum >= pivo do lado esquerdo
        while(v[j] < pivo){ j--; } //procura algum <= pivo do lado direito

        if(i<j){
            troca(&v[i], &v[j]); //troca os elementos encontrados
```

```
            i++;
            j--;
        }else
            return j; //retorna o local onde foi feita a particao
    }
}

void QuickSort(int *v, int ini, int fim){
    if(ini < fim ){
        int q = particao(v, ini, fim);
        QuickSort(v, ini, q);
        QuickSort(v, q+1, fim);
    }
}

int rec_buscaBinaria(int *v, int ini, int fim, int elem){
    if(ini > fim) return -1;
    int meio = (ini + fim)/2;
    comp++;
    if(v[meio] == elem)
        return meio;
    else
        if(elem > v[meio])
            return rec_buscaBinaria(v, ini, meio-1, elem);
        else
            return rec_buscaBinaria(v, meio+1, fim, elem);
}

int it_buscaBinaria(int *v, int ini, int fim, int elem){
    int meio;
    while(ini <= fim){
        meio = (ini + fim)/2;
        comp++;
        if(elem == v[meio]) return meio;
        else
            if(elem > v[meio])
                fim = meio-1;
            else
                ini = meio+1;
    }
    return -1;
}

#endif
```

1.2/main.c

```c
#include "busca.h"


int main(){

    //Atribuicoes iniciais
    srand(time(NULL));
    comp = 0;
    clock_t t;

    /*
    //Template de Calculo do Tempo de Execucao
    t = clock();
    //Chamada do Algoritmo aqui...
    t = clock() - t;
    printf ("It took me %d clicks (%f seconds).\n",t,((float)t)/CLOCKS_PER_SEC);
    */

    int *v;
    int n, x;
    printf("Digite o tamanho do vetor:\n");
    scanf("%d", &n);
    v = (int*) malloc (n*sizeof(int));


    preencheAleatorio(v, n, 1, n);
    //imprimeVetor(v, n);
    QuickSort(v, 0, n-1);

    printf("Digite um elemento para busca:\n");
    scanf("%d", &x);

    int ind;

    t = clock();
    ind = buscaSequencial(v, n, x);
    t = clock() - t;
    printf("-----Informacoes Busca Sequencial:\n");
    printf("Tempo Execucao:  %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
    printf("Comparacoes: %d\n", comp);

    //imprimeVetor(v, n);

    if(ind != -1)
        printf("O elemento %d foi encontrado na pos %d.\n", x, ind);
    else
        printf("O elemento %d NAO foi encontrado!\n", x);



    //imprimeVetor(v, n);

    comp = 0;
    t = clock();
    ind = rec_buscaBinaria(v, 0, n-1, x);
    t = clock() - t;
    printf("-----Informacoes Busca Binaria Recursiva:\n");
```

```c
    printf("Tempo Execucao:  %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
    printf("Comparacoes: %d\n", comp);

    if(ind != -1)
        printf("O elemento %d foi encontrado na pos %d.\n", x, ind);
    else
        printf("O elemento %d NAO foi encontrado!\n", x);


    comp = 0;
    t = clock();
    ind = it_buscaBinaria(v, 0, n-1, x);
    t = clock() - t;
    printf("-----Informacoes Busca Binaria Iterativa:\n");
    printf("Tempo Execucao:  %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
    printf("Comparacoes: %d\n", comp);

    if(ind != -1)
        printf("O elemento %d foi encontrado na pos %d.\n", x, ind);
    else
        printf("O elemento %d NAO foi encontrado!\n", x);


    free(v);
    return 0;
}
```

```
[lucascosta@fedora 1.2]$ gcc main.c -o main
[lucascosta@fedora 1.2]$ ./main
Digite o tamanho do vetor:
75
Digite um elemento para busca:
16
-----Informacoes Busca Sequencial:
Tempo Execucao:  0.000004 seconds.
Comparacoes: 75
O elemento 16 NAO foi encontrado!
-----Informacoes Busca Binaria Recursiva:
Tempo Execucao:  0.000002 seconds.
Comparacoes: 6
O elemento 16 NAO foi encontrado!
-----Informacoes Busca Binaria Iterativa:
Tempo Execucao:  0.000003 seconds.
Comparacoes: 6
O elemento 16 NAO foi encontrado!
[lucascosta@fedora 1.2]$
```

1.3/busca.h

```c
#ifndef BUSCA_H
#define BUSCA_H

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#define MAX 200

typedef struct{

    char nome[MAX];
    int matricula;
    double nota1, nota2, nota3;

} Aluno;

//Medidas de Complexidade
int comp; //Num. de comparacoes

int* copiaVetor(int* v, int n){
    int i;
    int *v2;
    v2 = (int*) malloc (n*sizeof(int));
    for(i=0; i<n; i++) v2[i] = v[i];
    return v2;
}

Aluno* copiaVetor_aluno(Aluno* v, int n){
    int i;
    Aluno *v2;
    v2 = (Aluno*) malloc (n*sizeof(Aluno));
    for(i=0; i<n; i++) v2[i] = v[i];
    return v2;
}

void imprimeVetor(int* v, int n){
    int i, prim = 1;
    printf("[");
    for(i=0; i<n; i++)
        if(prim){ printf("%d", v[i]); prim = 0; }
        else printf(", %d", v[i]);
    printf("]\n");
}

void preencheAleatorio(int* v, int n, int ini, int fim){
    int i;
    for(i=0; i<n; i++)
        v[i] = ini + rand() % (fim-ini + 1);
}

void troca_nome(Aluno* a, Aluno *b){
    Aluno aux = *a;
    *a = *b;
    *b = aux;
}

```

```c
int buscaSequencial_nome(Aluno *v, int n, char *elem){
    int i;
    for(i=0; i<n; i++){
        comp++;
        if(strcmp(v[i].nome, elem) == 0)
            return i; //Elemento encontrado
    }
    return -1; //Elemento nao encontrado
}

int particao_nome(Aluno *v, int ini, int fim){
    int i = ini, j = fim;
    Aluno pivo = v[(ini+fim)/2];
    while (1) {
        while(strcmp(v[i].nome, pivo.nome) < 0){ i++; } //procura algum >= pivo do lado esquerdo
        while(strcmp(v[j].nome, pivo.nome) > 0){ j--; } //procura algum <= pivo do lado direito

        if(i<j){
            troca_nome(&v[i], &v[j]); //troca os elementos encontrados
            i++;
            j--;
        }else
            return j; //retorna o local onde foi feita a particao
    }
}

void QuickSort_nome(Aluno *v, int ini, int fim){
    if(ini < fim ){
        int q = particao_nome(v, ini, fim);
        QuickSort_nome(v, ini, q);
        QuickSort_nome(v, q+1, fim);
    }
}

int rec_buscaBinaria_nome(Aluno *v, int ini, int fim, char *elem){
    if(ini > fim) return -1;
    int meio = (ini + fim)/2;
    comp++;
    if(strcmp(v[meio].nome, elem) == 0)
        return meio;
    else
        if(strcmp(v[meio].nome, elem) > 0)
            return rec_buscaBinaria_nome(v, ini, meio-1, elem);
        else
            return rec_buscaBinaria_nome(v, meio+1, fim, elem);
}

int it_buscaBinaria_nome(Aluno *v, int ini, int fim, char *elem){
    int meio;
    while(ini <= fim){
        meio = (ini + fim)/2;
        comp++;
        if(strcmp(v[meio].nome, elem) == 0) return meio;
        else
            if(strcmp(v[meio].nome, elem) > 0)
                fim = meio-1;
            else
                ini = meio+1;
```

```
116     }
117     return -1;
118 }
119 
120 void troca_matricula(Aluno* a, Aluno *b){
121     Aluno aux = *a;
122     *a = *b;
123     *b = aux;
124 }
125 
126 int buscaSequencial_matricula(Aluno *v, int n, int elem){
127     int i;
128     for(i=0; i<n; i++){
129         comp++;
130         if(v[i].matricula == elem)
131             return i; //Elemento encontrado
132     }
133     return -1; //Elemento encontrado
134 }
135 
136 int particao_matricula(Aluno *v, int ini, int fim){
137     int i = ini, j = fim;
138     Aluno pivo = v[(ini+fim)/2];
139     while (1) {
140         while(v[i].matricula < pivo.matricula){ i++; } //procura algum >= pivo do
    lado esquerdo
141         while(v[j].matricula > pivo.matricula){ j--; } //procura algum <= pivo do
    lado direito
142 
143         if(i<j){
144             troca_matricula(&v[i], &v[j]); //troca os elementos encontrados
145             i++;
146             j--;
147         }else
148             return j; //retorna o local onde foi feita a particao
149     }
150 }
151 
152 void QuickSort_matricula(Aluno *v, int ini, int fim){
153     if(ini < fim ){
154         int q = particao_matricula(v, ini, fim);
155         QuickSort_matricula(v, ini, q);
156         QuickSort_matricula(v, q+1, fim);
157     }
158 }
159 
160 int rec_buscaBinaria_matricula(Aluno *v, int ini, int fim, int elem){
161     if(ini > fim) return -1;
162     int meio = (ini + fim)/2;
163     comp++;
164     if(v[meio].matricula == elem)
165         return meio;
166     else
167         if(elem < v[meio].matricula)
168             return rec_buscaBinaria_matricula(v, ini, meio-1, elem);
169         else
170             return rec_buscaBinaria_matricula(v, meio+1, fim, elem);
171 }
172 
173 int it_buscaBinaria_matricula(Aluno *v, int ini, int fim, int elem){
```

```
174     int meio;
175     while(ini <= fim){
176         meio = (ini + fim)/2;
177         comp++;
178         if(elem == v[meio].matricula) return meio;
179         else
180             if(elem < v[meio].matricula)
181                 fim = meio-1;
182             else
183                 ini = meio+1;
184     }
185     return -1;
186 }
187 
188 // void troca(int* a, int *b){
189 //  int aux = *a;
190 //  *a = *b;
191 //  *b = aux;
192 // }
193 
194 // int buscaSequencial(int *v, int n, int elem){
195 //     int i;
196 //     for(i=0; i<n; i++){
197 //         comp++;
198 //         if(v[i] == elem)
199 //             return i; //Elemento encontrado
200 //     }
201 //     return -1; //Elemento encontrado
202 // }
203 
204 // int particao(int *v, int ini, int fim){
205 //     int i = ini, j = fim;
206 //     int pivo = v[(ini+fim)/2];
207 //     while (1) {
208 //         while(v[i] < pivo){ i++; } //procura algum >= pivo do lado esquerdo
209 //         while(v[j] > pivo){ j--; } //procura algum <= pivo do lado direito
210 
211 //         if(i<j){
212 //             troca(&v[i], &v[j]); //troca os elementos encontrados
213 //             i++;
214 //             j--;
215 //         }else
216 //             return j; //retorna o local onde foi feita a particao
217 //     }
218 // }
219 
220 // void QuickSort(int *v, int ini, int fim){
221 //     if(ini < fim ){
222 //         int q = particao(v, ini, fim);
223 //         QuickSort(v, ini, q);
224 //         QuickSort(v, q+1, fim);
225 //     }
226 // }
227 
228 // int rec_buscaBinaria(int *v, int ini, int fim, int elem){
229 //     if(ini > fim) return -1;
230 //     int meio = (ini + fim)/2;
231 //     comp++;
232 //     if(v[meio] == elem)
233 //         return meio;
```

```
234 //     else
235 //         if(elem < v[meio])
236 //             return rec_buscaBinaria(v, ini, meio-1, elem);
237 //         else
238 //             return rec_buscaBinaria(v, meio+1, fim, elem);
239 // }
240 
241 // int it_buscaBinaria(int *v, int ini, int fim, int elem){
242 //     int meio;
243 //     while(ini <= fim){
244 //         meio = (ini + fim)/2;
245 //         comp++;
246 //         if(elem == v[meio]) return meio;
247 //         else
248 //             if(elem < v[meio])
249 //                 fim = meio-1;
250 //             else
251 //                 ini = meio+1;
252 //     }
253 //     return -1;
254 // }
255 
256 #endif
```

1.3/main.c

```c
#include "busca.h"


int main(){

    //Atribuicoes iniciais
    srand(time(NULL));
    comp = 0;
    clock_t t;

    /*
    //Template de Calculo do Tempo de Execucao
    t = clock();
    //Chamada do Algoritmo aqui...
    t = clock() - t;
    printf ("It took me %d clicks (%f seconds).\n",t,((float)t)/CLOCKS_PER_SEC);
    */

    Aluno *v;
    int n, matricula;
    char nome[MAX];
    char buffer[MAX];
    printf("Digite o tamanho do vetor:\n");
    scanf("%d", &n);
    v = (Aluno*) malloc (n*sizeof(Aluno));

    for(int i = 0; i < n; i++){
        printf("Digite a matricula do Aluno:\n");
        scanf("%d", &v[i].matricula);
        printf("Digite o nome do Aluno:\n");
        fgets(buffer, sizeof(buffer), stdin);
        scanf("%200[^\n]", v[i].nome);
        v[i].nota1 = 10;
        v[i].nota2 = 10;
        v[i].nota3 = 10;
    }
    //preencheAleatorio(v, n, 1, n);
    //imprimeVetor(v, n);

    //por nome
    QuickSort_nome(v, 0, n-1);

    printf("Digite um nome para busca:\n");
    fgets(buffer, sizeof(buffer), stdin);
    scanf("%200[^\n]", nome);

    int ind;

    t = clock();
    ind = buscaSequencial_nome(v, n, nome);
    t = clock() - t;
    printf("-----Informacoes Busca Sequencial:\n");
    printf("Tempo Execucao:  %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
    printf("Comparacoes: %d\n", comp);

    //imprimeVetor(v, n);

```

```c
    if(ind != -1)
        printf("O elemento %s foi encontrado na pos %d.\n", nome, ind);
    else
        printf("O elemento %s NAO foi encontrado!\n", nome);



    //imprimeVetor(v, n);

    comp = 0;
    t = clock();
    ind = rec_buscaBinaria_nome(v, 0, n-1, nome);
    t = clock() - t;
    printf("-----Informacoes Busca Binaria Recursiva:\n");
    printf("Tempo Execucao:  %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
    printf("Comparacoes: %d\n", comp);

    if(ind != -1)
        printf("O elemento %s foi encontrado na pos %d.\n", nome, ind);
    else
        printf("O elemento %s NAO foi encontrado!\n", nome);


    comp = 0;
    t = clock();
    ind = it_buscaBinaria_nome(v, 0, n-1, nome);
    t = clock() - t;
    printf("-----Informacoes Busca Binaria Iterativa:\n");
    printf("Tempo Execucao:  %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
    printf("Comparacoes: %d\n", comp);

    if(ind != -1)
        printf("O elemento %s foi encontrado na pos %d.\n", nome, ind);
    else
        printf("O elemento %s NAO foi encontrado!\n", nome);

    //por matricula
    QuickSort_matricula(v, 0, n-1);

    printf("Digite uma matricula para busca:\n");
    scanf("%d", &matricula);


    t = clock();
    ind = buscaSequencial_matricula(v, n, matricula);
    t = clock() - t;
    printf("-----Informacoes Busca Sequencial:\n");
    printf("Tempo Execucao:  %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
    printf("Comparacoes: %d\n", comp);

    //imprimeVetor(v, n);

    if(ind != -1)
        printf("O elemento %d foi encontrado na pos %d.\n", matricula, ind);
    else
        printf("O elemento %d NAO foi encontrado!\n", matricula);



    //imprimeVetor(v, n);
```

```c

    comp = 0;
    t = clock();
    ind = rec_buscaBinaria_matricula(v, 0, n-1, matricula);
    t = clock() - t;
    printf("-----Informacoes Busca Binaria Recursiva:\n");
    printf("Tempo Execucao:  %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
    printf("Comparacoes: %d\n", comp);

    if(ind != -1)
        printf("O elemento %d foi encontrado na pos %d.\n", matricula, ind);
    else
        printf("O elemento %d NAO foi encontrado!\n", matricula);


    comp = 0;
    t = clock();
    ind = it_buscaBinaria_matricula(v, 0, n-1, matricula);
    t = clock() - t;
    printf("-----Informacoes Busca Binaria Iterativa:\n");
    printf("Tempo Execucao:  %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
    printf("Comparacoes: %d\n", comp);

    if(ind != -1)
        printf("O elemento %d foi encontrado na pos %d.\n", matricula, ind);
    else
        printf("O elemento %d NAO foi encontrado!\n", matricula);

    free(v);
    return 0;
}
```

```
[lucascosta@fedora 1.3]$ ./main
Digite o tamanho do vetor:
3
Digite a matricula do Aluno:
1245
Digite o nome do Aluno:
Lucas
Digite a matricula do Aluno:
1456
Digite o nome do Aluno:
Larissa
Digite a matricula do Aluno:
1485
Digite o nome do Aluno:
Luana
Digite um nome para busca:
Lucas
-----Informacoes Busca Sequencial:
Tempo Execucao:  0.000005 seconds.
Comparacoes: 3
O elemento Lucas foi encontrado na pos 2.
-----Informacoes Busca Binaria Recursiva:
Tempo Execucao:  0.000003 seconds.
Comparacoes: 2
O elemento Lucas foi encontrado na pos 2.
-----Informacoes Busca Binaria Iterativa:
Tempo Execucao:  0.000002 seconds.
Comparacoes: 2
O elemento Lucas foi encontrado na pos 2.
Digite uma matricula para busca:
1485
-----Informacoes Busca Sequencial:
Tempo Execucao:  0.000005 seconds.
Comparacoes: 5
O elemento 1485 foi encontrado na pos 2.
-----Informacoes Busca Binaria Recursiva:
Tempo Execucao:  0.000003 seconds.
Comparacoes: 2
O elemento 1485 foi encontrado na pos 2.
-----Informacoes Busca Binaria Iterativa:
Tempo Execucao:  0.000002 seconds.
Comparacoes: 2
O elemento 1485 foi encontrado na pos 2.
[lucascosta@fedora 1.3]$
```

2.1/hash.h

```c
#ifndef HASH_H
#define HASH_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct{
    int **tabela;
    int tam, qtd;
}Hash;


Hash* criaHash(int t){
    Hash* h;
    h = (Hash*) malloc (sizeof(Hash));
    if(h != NULL){
        h->tam = t; h->qtd = 0;
        h->tabela = (int**) malloc (t*sizeof(int*));
        if(h->tabela == NULL) return NULL;
        int i;
        for(i = 0; i<t; i++)
            h->tabela[i] = NULL;
    }
    return h;
}


void destroiHash(Hash *h){
    if(h != NULL){
        int i;
        for(i = 0; i<h->tam; i++)
            if(h->tabela[i] != NULL)
                free(h->tabela[i]);
        free(h->tabela);
        free(h);
    }
}

int chaveDivisao(int chave, int tam){
    return (chave & 0x7FFFFFFF) % tam;
}

int chaveMultiplicacao(int chave, int tam){
    float A = 0.6180339887; //constante: 0 < A < 1
    float val = chave * A;
    val = val - (int) val;
    return (int) (tam * val);
}

int chaveDobra(int chave, int tam){
    int pos, n_bits = 30;

    int p = 1;
    int r = p << n_bits;
    while((chave & r) != r){ n_bits--; r = p << n_bits; }

```

```
58      n_bits++;
59      pos = chave;
60      while(pos > tam){
61          int metade_bits = n_bits/2;
62          int parte1 = pos >> metade_bits;
63          parte1 = parte1 << metade_bits;
64          int parte2 = pos ^ parte1;
65          parte1 = pos >> metade_bits;
66          pos = parte1 ^ parte2;
67          n_bits = n_bits/2;
68      }
69      return pos;
70  }
71
72  int valorString(char *str){
73      int i, valor = 1;
74      int tam = strlen(str);
75      for(i=0; i<tam; i++)
76          valor = 31*valor + (i+1)*((int) str[i]);
77      return valor;
78  }
79
80  int insereHash_semTratar_div(Hash* h, int elem){
81      if(h == NULL) return 0;
82      int pos = chaveDivisao(elem, h->tam);
83
84      if(h->tabela[pos] == NULL){
85          int* novo = (int*) malloc (sizeof(int));
86          if(novo == NULL) return 0;
87          *novo = elem;
88          h->tabela[pos] = novo;
89          h->qtd++;
90      }else *(h->tabela[pos]) = elem;
91      return 1;
92  }
93
94  int insereHash_semTratar_mul(Hash* h, int elem){
95      if(h == NULL) return 0;
96      int pos = chaveMultiplicacao(elem, h->tam);
97
98      if(h->tabela[pos] == NULL){
99          int* novo = (int*) malloc (sizeof(int));
100         if(novo == NULL) return 0;
101         *novo = elem;
102         h->tabela[pos] = novo;
103         h->qtd++;
104     }else *(h->tabela[pos]) = elem;
105     return 1;
106 }
107
108 int insereHash_semTratar_dobra(Hash* h, int elem){
109     if(h == NULL) return 0;
110     int pos = chaveDobra(elem, h->tam);
111
112     if(h->tabela[pos] == NULL){
113         int* novo = (int*) malloc (sizeof(int));
114         if(novo == NULL) return 0;
115         *novo = elem;
116         h->tabela[pos] = novo;
117         h->qtd++;
```

```
118     }else *(h->tabela[pos]) = elem;
119     return 1;
120 }
121 
122 int buscaHash_semTratar_div(Hash* h, int elem, int *p){
123     if(h == NULL) return 0;
124     int pos = chaveDivisao(elem, h->tam);
125     if(h->tabela[pos] == NULL) return 0;
126     if(*(h->tabela[pos]) == elem){
127         *p = *(h->tabela[pos]);
128         return 1;
129     }
130     return 0;
131 }
132 
133 int buscaHash_semTratar_mul(Hash* h, int elem, int *p){
134     if(h == NULL) return 0;
135     int pos = chaveMultiplicacao(elem, h->tam);
136     if(h->tabela[pos] == NULL) return 0;
137     if(*(h->tabela[pos]) == elem){
138         *p = *(h->tabela[pos]);
139         return 1;
140     }
141     return 0;
142 }
143 
144 int buscaHash_semTratar_dobra(Hash* h, int elem, int *p){
145     if(h == NULL) return 0;
146     int pos = chaveDobra(elem, h->tam);
147     if(h->tabela[pos] == NULL) return 0;
148     if(*(h->tabela[pos]) == elem){
149         *p = *(h->tabela[pos]);
150         return 1;
151     }
152     return 0;
153 }
154 
155 int sondagemLinear(int pos, int i, int tam){
156     return ( (pos + i) & 0x7FFFFFFF) % tam;
157 }
158 
159 int sondagemQuadratica(int pos, int i, int tam){
160     pos = pos + 2*i + 5*i*i;
161     return ( pos & 0x7FFFFFFF) % tam;
162 }
163 
164 int sondagemDuploHash(int H1, int chave, int i, int tam){
165     int H2 = chaveDivisao(chave, tam-1) + 1;
166     return ( (H1 + i*H2) & 0x7FFFFFFF) % tam;
167 }
168 
169 int insereHash_EnderAberto(Hash* h, int elem){
170   if(h == NULL) return 0;
171   int i, pos, newPos;
172   pos = chaveDivisao(elem, h->tam);
173   for(i=0; i<h->tam; i++){
174     newPos = sondagemLinear(pos, i, h->tam);
175     //newPos = sondagemQuadratica(pos, i, h->tam);
176     //newPos = sondagemDuploHash(pos, elem, i, h->tam);
177     if(h->tabela[newPos] == NULL){
```

```
178         int* novo = (int*) malloc (sizeof(int));
179         if(novo == NULL) return 0;
180         *novo = elem;
181         h->tabela[newPos] = novo;
182         h->qtd++;
183         return 1;
184     }
185   }
186   return 0;
187 }
188 
189 int buscaHash_EnderAberto(Hash* h, int elem, int *p){
190     if(h == NULL) return 0;
191     int i, pos, newPos;
192     pos = chaveDivisao(elem, h->tam);
193     for(i=0; i<h->tam; i++){
194         newPos = sondagemLinear(pos, i, h->tam);
195         //newPos = sondagemQuadratica(pos, i, h->tam);
196         //newPos = sondagemDuploHash(pos, elem, i, h->tam);
197         if(h->tabela[newPos] == NULL) return 0;
198         if(*(h->tabela[newPos]) == elem){
199             *p = *(h->tabela[newPos]);
200             return 1;
201         }
202     }
203     return 0;
204 }
205 
206 void imprimeHash(Hash *h){
207     if(h == NULL) return;
208     int i;
209     for(i=0; i<h->tam; i++){
210         printf("%d: ", i);
211         if(h->tabela[i] == NULL) printf("NULL\n");
212         else printf("%d\n", *(h->tabela[i]));
213     }
214 }
215 
216 #endif
```

2.1/main.c

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

// void binary(int n){
//  if(n<2)
//           printf("%d", n%2);
//  else{
//           binary(n/2);
//    printf("%d", n%2);
//  }
// }

int main(){

    // int x = 1;
    // int d;
    // printf("%d: ", x); binary(x); printf("\n");

    // d = x << 4;

    // printf("%d: ", d); binary(d); printf("\n");

    // d = d >> 2;

    // printf("%d: ", d); binary(d); printf("\n");

    Hash *H;;
    H = criaHash(5);

    int busca;


    insereHash_semTratar_div(H, 31);
    insereHash_semTratar_div(H, 32);
    insereHash_semTratar_div(H, 64);

    printf("Divisão:.\n");

    if(!buscaHash_semTratar_div(H, 64, &busca))
        printf("Elemento NAO encontrado.\n");
    else
        printf("Elemento %d encontrado.\n", busca);
    if(!buscaHash_semTratar_div(H, 31, &busca))
        printf("Elemento NAO encontrado.\n");
    else
        printf("Elemento %d encontrado.\n", busca);
    if(!buscaHash_semTratar_div(H, 33, &busca))
        printf("Elemento NAO encontrado.\n");
    else
        printf("Elemento %d encontrado.\n", busca);

    imprimeHash(H);
    destroiHash(H);
    H = criaHash(5);

    insereHash_semTratar_mul(H, 31);
    insereHash_semTratar_mul(H, 32);
```

```c
    insereHash_semTratar_mul(H, 64);

    printf("Multiplicação:.\n");

    if(!buscaHash_semTratar_mul(H, 64, &busca))
        printf("Elemento NAO encontrado.\n");
    else
        printf("Elemento %d encontrado.\n", busca);
    if(!buscaHash_semTratar_mul(H, 31, &busca))
        printf("Elemento NAO encontrado.\n");
    else
        printf("Elemento %d encontrado.\n", busca);
        if(!buscaHash_semTratar_mul(H, 33, &busca))
        printf("Elemento NAO encontrado.\n");
    else
        printf("Elemento %d encontrado.\n", busca);

    imprimeHash(H);
    destroiHash(H);
    H = criaHash(10);

    insereHash_semTratar_dobra(H, 31);
    insereHash_semTratar_dobra(H, 32);
    insereHash_semTratar_dobra(H, 64);

    printf("Dobra:.\n");

    if(!buscaHash_semTratar_dobra(H, 64, &busca))
        printf("Elemento NAO encontrado.\n");
    else
        printf("Elemento %d encontrado.\n", busca);
    if(!buscaHash_semTratar_dobra(H, 31, &busca))
        printf("Elemento NAO encontrado.\n");
    else
        printf("Elemento %d encontrado.\n", busca);
        if(!buscaHash_semTratar_dobra(H, 33, &busca))
        printf("Elemento NAO encontrado.\n");
    else
        printf("Elemento %d encontrado.\n", busca);

    imprimeHash(H);
    destroiHash(H);

    // insereHash_EnderAberto(H, 31);
    // insereHash_EnderAberto(H, 32);
    // insereHash_EnderAberto(H, 62);

    // if(!buscaHash_EnderAberto(H, 31, &busca))
    //     printf("Elemento NAO encontrado.\n");
    // else
    //     printf("Elemento %d encontrado.\n", busca);

    // if(!buscaHash_EnderAberto(H, 62, &busca))
    //     printf("Elemento NAO encontrado.\n");
    // else
    //     printf("Elemento %d encontrado.\n", busca);

    return 0;
}
```

```
[lucascosta@fedora 2.1]$ gcc main.c -o main
[lucascosta@fedora 2.1]$ ./main
Divisão:.
Elemento 64 encontrado.
Elemento 31 encontrado.
Elemento NAO encontrado.
0: NULL
1: 31
2: 32
3: NULL
4: 64
Multiplicação:.
Elemento 64 encontrado.
Elemento 31 encontrado.
Elemento NAO encontrado.
0: 31
1: NULL
2: 64
3: 32
4: NULL
Dobra:.
Elemento 64 encontrado.
Elemento NAO encontrado.
Elemento NAO encontrado.
0: NULL
1: NULL
2: NULL
3: NULL
4: 32
5: NULL
6: NULL
7: NULL
8: 64
9: NULL
[lucascosta@fedora 2.1]$
```

2.2/hash.h

```c
#ifndef HASH_H
#define HASH_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct{
    int **tabela;
    int tam, qtd;
}Hash;


Hash* criaHash(int t){
    Hash* h;
    h = (Hash*) malloc (sizeof(Hash));
    if(h != NULL){
        h->tam = t; h->qtd = 0;
        h->tabela = (int**) malloc (t*sizeof(int*));
        if(h->tabela == NULL) return NULL;
        int i;
        for(i = 0; i<t; i++)
            h->tabela[i] = NULL;
    }
    return h;
}


void destroiHash(Hash *h){
    if(h != NULL){
        int i;
        for(i = 0; i<h->tam; i++)
            if(h->tabela[i] != NULL)
                free(h->tabela[i]);
        free(h->tabela);
        free(h);
    }
}

int chaveDivisao(int chave, int tam){
    return (chave & 0x7FFFFFFF) % tam;
}

int chaveMultiplicacao(int chave, int tam){
    float A = 0.6180339887; //constante: 0 < A < 1
    float val = chave * A;
    val = val - (int) val;
    return (int) (tam * val);
}

int chaveDobra(int chave, int tam){
    int pos, n_bits = 30;

    int p = 1;
    int r = p << n_bits;
    while((chave & r) != r){ n_bits--; r = p << n_bits; }

```

```
58      n_bits++;
59      pos = chave;
60      while(pos > tam){
61          int metade_bits = n_bits/2;
62          int parte1 = pos >> metade_bits;
63          parte1 = parte1 << metade_bits;
64          int parte2 = pos ^ parte1;
65          parte1 = pos >> metade_bits;
66          pos = parte1 ^ parte2;
67          n_bits = n_bits/2;
68      }
69      return pos;
70  }
71
72  int valorString(char *str){
73      int i, valor = 1;
74      int tam = strlen(str);
75      for(i=0; i<tam; i++)
76          valor = 31*valor + (i+1)*((int) str[i]);
77      return valor;
78  }
79
80  int insereHash_semTratar(Hash* h, int elem){
81      if(h == NULL) return 0;
82      int pos = chaveDivisao(elem, h->tam);
83
84      if(h->tabela[pos] == NULL){
85          int* novo = (int*) malloc (sizeof(int));
86          if(novo == NULL) return 0;
87          *novo = elem;
88          h->tabela[pos] = novo;
89          h->qtd++;
90      }else *(h->tabela[pos]) = elem;
91      return 1;
92  }
93
94  int buscaHash_semTratar(Hash* h, int elem, int *p){
95      if(h == NULL) return 0;
96      int pos = chaveDivisao(elem, h->tam);
97      if(h->tabela[pos] == NULL) return 0;
98      if(*(h->tabela[pos]) == elem){
99          *p = *(h->tabela[pos]);
100         return 1;
101     }
102     return 0;
103 }
104
105 int sondagemLinear(int pos, int i, int tam){
106     return ( (pos + i) & 0x7FFFFFFF) % tam;
107 }
108
109 int sondagemQuadratica(int pos, int i, int tam){
110     pos = pos + 2*i + 5*i*i;
111     return ( pos & 0x7FFFFFFF) % tam;
112 }
113
114 int sondagemDuploHash(int H1, int chave, int i, int tam){
115     int H2 = chaveDivisao(chave, tam-1) + 1;
116     return ( (H1 + i*H2) & 0x7FFFFFFF) % tam;
117 }
```

```
118
119 int insereHash_EnderAberto_lin(Hash* h, int elem){
120   if(h == NULL) return 0;
121   int i, pos, newPos;
122   pos = chaveDivisao(elem, h->tam);
123   for(i=0; i<h->tam; i++){
124     newPos = sondagemLinear(pos, i, h->tam);
125     //newPos = sondagemQuadratica(pos, i, h->tam);
126     //newPos = sondagemDuploHash(pos, elem, i, h->tam);
127     if(h->tabela[newPos] == NULL){
128         int* novo = (int*) malloc (sizeof(int));
129         if(novo == NULL) return 0;
130         *novo = elem;
131         h->tabela[newPos] = novo;
132         h->qtd++;
133         return 1;
134     }
135   }
136   return 0;
137 }
138
139 int insereHash_EnderAberto_quad(Hash* h, int elem){
140   if(h == NULL) return 0;
141   int i, pos, newPos;
142   pos = chaveDivisao(elem, h->tam);
143   for(i=0; i<h->tam; i++){
144     //newPos = sondagemLinear(pos, i, h->tam);
145     newPos = sondagemQuadratica(pos, i, h->tam);
146     //newPos = sondagemDuploHash(pos, elem, i, h->tam);
147     if(h->tabela[newPos] == NULL){
148         int* novo = (int*) malloc (sizeof(int));
149         if(novo == NULL) return 0;
150         *novo = elem;
151         h->tabela[newPos] = novo;
152         h->qtd++;
153         return 1;
154     }
155   }
156   return 0;
157 }
158
159 int insereHash_EnderAberto_duplo(Hash* h, int elem){
160   if(h == NULL) return 0;
161   int i, pos, newPos;
162   pos = chaveDivisao(elem, h->tam);
163   for(i=0; i<h->tam; i++){
164     //newPos = sondagemLinear(pos, i, h->tam);
165     //newPos = sondagemQuadratica(pos, i, h->tam);
166     newPos = sondagemDuploHash(pos, elem, i, h->tam);
167     if(h->tabela[newPos] == NULL){
168         int* novo = (int*) malloc (sizeof(int));
169         if(novo == NULL) return 0;
170         *novo = elem;
171         h->tabela[newPos] = novo;
172         h->qtd++;
173         return 1;
174     }
175   }
176   return 0;
177 }
```

```c

int buscaHash_EnderAberto_lin(Hash* h, int elem, int *p){
    if(h == NULL) return 0;
    int i, pos, newPos;
    pos = chaveDivisao(elem, h->tam);
    for(i=0; i<h->tam; i++){
        newPos = sondagemLinear(pos, i, h->tam);
        //newPos = sondagemQuadratica(pos, i, h->tam);
        //newPos = sondagemDuploHash(pos, elem, i, h->tam);
        if(h->tabela[newPos] == NULL) return 0;
        if(*(h->tabela[newPos]) == elem){
            *p = *(h->tabela[newPos]);
            return 1;
        }
    }
    return 0;
}

int buscaHash_EnderAberto_quad(Hash* h, int elem, int *p){
    if(h == NULL) return 0;
    int i, pos, newPos;
    pos = chaveDivisao(elem, h->tam);
    for(i=0; i<h->tam; i++){
        // newPos = sondagemLinear(pos, i, h->tam);
        newPos = sondagemQuadratica(pos, i, h->tam);
        //newPos = sondagemDuploHash(pos, elem, i, h->tam);
        if(h->tabela[newPos] == NULL) return 0;
        if(*(h->tabela[newPos]) == elem){
            *p = *(h->tabela[newPos]);
            return 1;
        }
    }
    return 0;
}

int buscaHash_EnderAberto_duplo(Hash* h, int elem, int *p){
    if(h == NULL) return 0;
    int i, pos, newPos;
    pos = chaveDivisao(elem, h->tam);
    for(i=0; i<h->tam; i++){
        //newPos = sondagemLinear(pos, i, h->tam);
        //newPos = sondagemQuadratica(pos, i, h->tam);
        newPos = sondagemDuploHash(pos, elem, i, h->tam);
        if(h->tabela[newPos] == NULL) return 0;
        if(*(h->tabela[newPos]) == elem){
            *p = *(h->tabela[newPos]);
            return 1;
        }
    }
    return 0;
}



void imprimeHash(Hash *h){
    if(h == NULL) return;
    int i;
    for(i=0; i<h->tam; i++){
        printf("%d: ", i);
        if(h->tabela[i] == NULL) printf("NULL\n");
```

```
238             else printf("%d\n", *(h->tabela[i]));
239         }
240     }
241
242     #endif
```

2.2/main.c

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

// void binary(int n){
//  if(n<2)
//           printf("%d", n%2);
//  else{
//           binary(n/2);
//    printf("%d", n%2);
//  }
// }

int main(){

    // int x = 1;
    // int d;
    // printf("%d: ", x); binary(x); printf("\n");

    // d = x << 4;

    // printf("%d: ", d); binary(d); printf("\n");

    // d = d >> 2;

    // printf("%d: ", d); binary(d); printf("\n");

    Hash *H;;
    H = criaHash(5);

    int busca;

    insereHash_EnderAberto_lin(H, 31);
    insereHash_EnderAberto_lin(H, 32);
    insereHash_EnderAberto_lin(H, 64);

    printf("Linear:.\n");

    if(!buscaHash_EnderAberto_lin(H, 64, &busca))
        printf("Elemento NAO encontrado.\n");
    else
        printf("Elemento %d encontrado.\n", busca);
    if(!buscaHash_EnderAberto_lin(H, 31, &busca))
        printf("Elemento NAO encontrado.\n");
    else
        printf("Elemento %d encontrado.\n", busca);
    if(!buscaHash_EnderAberto_lin(H, 33, &busca))
        printf("Elemento NAO encontrado.\n");
    else
        printf("Elemento %d encontrado.\n", busca);

    imprimeHash(H);
    destroiHash(H);
    H = criaHash(5);

    insereHash_EnderAberto_quad(H, 31);
    insereHash_EnderAberto_quad(H, 32);
    insereHash_EnderAberto_quad(H, 64);
```

```c

    printf("Quadratica:.\n");

    if(!buscaHash_EnderAberto_quad(H, 64, &busca))
        printf("Elemento NAO encontrado.\n");
    else
        printf("Elemento %d encontrado.\n", busca);
    if(!buscaHash_EnderAberto_quad(H, 31, &busca))
        printf("Elemento NAO encontrado.\n");
    else
        printf("Elemento %d encontrado.\n", busca);
    if(!buscaHash_EnderAberto_quad(H, 33, &busca))
        printf("Elemento NAO encontrado.\n");
    else
        printf("Elemento %d encontrado.\n", busca);

    imprimeHash(H);
    destroiHash(H);
    H = criaHash(5);

    insereHash_EnderAberto_duplo(H, 31);
    insereHash_EnderAberto_duplo(H, 32);
    insereHash_EnderAberto_duplo(H, 64);

    printf("Duplo:.\n");

    if(!buscaHash_EnderAberto_duplo(H, 64, &busca))
        printf("Elemento NAO encontrado.\n");
    else
        printf("Elemento %d encontrado.\n", busca);
    if(!buscaHash_EnderAberto_duplo(H, 31, &busca))
        printf("Elemento NAO encontrado.\n");
    else
        printf("Elemento %d encontrado.\n", busca);
    if(!buscaHash_EnderAberto_duplo(H, 33, &busca))
        printf("Elemento NAO encontrado.\n");
    else
        printf("Elemento %d encontrado.\n", busca);

    imprimeHash(H);
    destroiHash(H);

    return 0;

}
```

```
[lucascosta@fedora 2.2]$ gcc main.c -o main
[lucascosta@fedora 2.2]$ ./main
Linear:.
Elemento 64 encontrado.
Elemento 31 encontrado.
Elemento NAO encontrado.
0: NULL
1: 31
2: 32
3: NULL
4: 64
Quadratica:.
Elemento 64 encontrado.
Elemento 31 encontrado.
Elemento NAO encontrado.
0: NULL
1: 31
2: 32
3: NULL
4: 64
Duplo:.
Elemento 64 encontrado.
Elemento 31 encontrado.
Elemento NAO encontrado.
0: NULL
1: 31
2: 32
3: NULL
4: 64
[lucascosta@fedora 2.2]$
```

2.3/hash.h

```c
#ifndef HASH_H
#define HASH_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "LSE.h"

typedef struct{
    Lista **tabela;
    int tam, qtd;
}Hash;


Hash* criaHash(int t){
    Hash* h;
    h = (Hash*) malloc (sizeof(Hash));
    if(h != NULL){
        h->tam = t; h->qtd = 0;
        h->tabela = (Lista**) malloc (t*sizeof(Lista*));
        if(h->tabela == NULL) return NULL;
        int i;
        for(i = 0; i<t; i++)
            h->tabela[i] = NULL;
    }
    return h;
}


void destroiHash(Hash *h){
    if(h != NULL){
        int i;
        for(i = 0; i<h->tam; i++)
            if(h->tabela[i] != NULL)
                destroiLista(h->tabela[i]);
        free(h->tabela);
        free(h);
    }
}

int chaveDivisao(int chave, int tam){
    return (chave & 0x7FFFFFFF) % tam;
}

int chaveMultiplicacao(int chave, int tam){
    float A = 0.6180339887; //constante: 0 < A < 1
    float val = chave * A;
    val = val - (int) val;
    return (int) (tam * val);
}

int chaveDobra(int chave, int tam){
    int pos, n_bits = 30;

    int p = 1;
    int r = p << n_bits;
    while((chave & r) != r){ n_bits--; r = p << n_bits; }
```

```
58
59      n_bits++;
60      pos = chave;
61      while(pos > tam){
62          int metade_bits = n_bits/2;
63          int parte1 = pos >> metade_bits;
64          parte1 = parte1 << metade_bits;
65          int parte2 = pos ^ parte1;
66          parte1 = pos >> metade_bits;
67          pos = parte1 ^ parte2;
68          n_bits = n_bits/2;
69      }
70      return pos;
71  }
72
73  int valorString(char *str){
74      int i, valor = 1;
75      int tam = strlen(str);
76      for(i=0; i<tam; i++)
77          valor = 31*valor + (i+1)*((int) str[i]);
78      return valor;
79  }
80
81  int insereHashLSE(Hash* h, int elem){
82      if(h == NULL) return 0;
83      int pos = chaveDivisao(elem, h->tam);
84      if(h->tabela[pos] == NULL)
85          h->tabela[pos] = criaLista();
86      insereIni(h->tabela[pos], elem);
87      h->qtd++;
88      return 1;
89  }
90
91  int buscaHashLSE(Hash* h, int elem, int *p){
92      if(h == NULL) return 0;
93      int pos = chaveDivisao(elem, h->tam);
94      if(h->tabela[pos] == NULL) return 0;
95      return listaBuscaElem(h->tabela[pos], elem, p);
96  }
97
98  void imprimeHash(Hash *h){
99      if(h == NULL) return;
100     int i;
101     for(i=0; i<h->tam; i++){
102         printf("%d: ", i);
103         if(h->tabela[i] == NULL) printf("NULL\n");
104         else imprimeLista(h->tabela[i]);
105     }
106 }
107
108 #endif
```

2.3/LSE.h

```c
#ifndef LISTASE_H
#define LISTASE_H

#include <stdio.h>
#include <stdlib.h>

typedef struct NO{
    int info;
    struct NO* prox;
}NO;

typedef struct NO* Lista;

Lista* criaLista(){
    Lista *li;
    li = (Lista*) malloc (sizeof(Lista));
    if(li != NULL){
        *li = NULL;
    }
    return li;
}

int listaVazia(Lista *li){
    if(li == NULL) return 1;
    if(*li == NULL) return 1;//True - Vazia!
    return 0;//False - tem elemento!
}

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

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

int listaBuscaElem(Lista* li, int elem, int *p){
    if(li == NULL) return 0;
    NO* aux = *li;
    while(aux != NULL){
        if(aux->info == elem){
            *p = aux->info;
            return 1;
        }
        aux = aux->prox;
    }
    return 0;
}

int insereIni(Lista* li, int elem){
    if(li == NULL) return 0;
    NO* novo = alocarNO();
    if(novo == NULL) return 0;
    novo->info = elem;
    novo->prox = *li;
    *li = novo;
    return 1;
```

```
58  }
59
60  int insereFim(Lista* li, int elem){
61      if(li == NULL) return 0;
62      NO* novo = alocarNO();
63      if(novo == NULL) return 0;
64      novo->info = elem;
65      novo->prox = NULL;
66      if(listaVazia(li)){
67          *li = novo;
68      }else{
69          NO* aux = *li;
70          while(aux->prox != NULL)
71              aux = aux->prox;
72          aux->prox = novo;
73      }
74      return 1;
75  }
76
77  int removeIni(Lista* li){
78      if(li == NULL) return 0;
79      if(listaVazia(li)) return 0;
80      NO* aux = *li;
81      *li = aux->prox;
82      liberarNO(aux);
83      return 1;
84  }
85
86  int removeFim(Lista* li){
87      if(li == NULL) return 0;
88      if(listaVazia(li)) return 0;
89      NO* ant, *aux = *li;
90      while(aux->prox != NULL){
91          ant = aux;
92          aux = aux->prox;
93      }
94      if(aux == *li)
95          *li = aux->prox;
96      else
97          ant->prox = aux->prox;
98      liberarNO(aux);
99      return 1;
100 }
101
102 void imprimeLista(Lista* li){
103     if(li == NULL) return;
104     if(listaVazia(li)){
105         printf("Lista Vazia!\n");
106         return;
107     }
108     //printf("Elementos:\n");
109     NO* aux = *li;
110     while(aux != NULL){
111         printf("%d ", aux->info);
112         aux = aux->prox;
113     }
114     printf("\n");
115 }
116
117 void destroiLista(Lista* li){
```

```
118     if(li != NULL){
119         NO* aux;
120         while((*li) != NULL){
121             aux = *li;
122             *li = (*li)->prox;
123             liberarNO(aux);
124         }
125         free(li);
126     }
127 }
128 
129 #endif
```

2.3/main.c

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



int main(){

    Hash *H;
    H = criaHash(15);

    int busca;

    int i;
    for(i=0; i<100; i++)
        insereHashLSE(H, i);

    imprimeHash(H);

    if(buscaHashLSE(H, 12, &busca)){
        printf("Elemento %d encontrado\n", busca);
    }
    else{
        printf("Elemento NÃO encontrado\n");
    }

    if(buscaHashLSE(H, 102, &busca)){
        printf("Elemento %d encontrado\n", busca);
    }
    else{
        printf("Elemento NÃO encontrado\n");
    }

    destroiHash(H);

    return 0;
}
```

```
[lucascosta@fedora 2.3]$ gcc main.c -o main
[lucascosta@fedora 2.3]$ ./main
0: 90 75 60 45 30 15 0
1: 91 76 61 46 31 16 1
2: 92 77 62 47 32 17 2
3: 93 78 63 48 33 18 3
4: 94 79 64 49 34 19 4
5: 95 80 65 50 35 20 5
6: 96 81 66 51 36 21 6
7: 97 82 67 52 37 22 7
8: 98 83 68 53 38 23 8
9: 99 84 69 54 39 24 9
10: 85 70 55 40 25 10
11: 86 71 56 41 26 11
12: 87 72 57 42 27 12
13: 88 73 58 43 28 13
14: 89 74 59 44 29 14
Elemento 12 encontrado
Elemento NÃO encontrado
[lucascosta@fedora 2.3]$
```