

roteiro-12/ex01-01.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  int comp;
6
7  void preencheAleatorio(int *v, int n, int ini, int fim) {
8      int i;
9      for (i = 0; i < n; i++)
10         v[i] = ini + rand() % (fim - ini + 1);
11 }
12
13 void troca(int *a, int *b) {
14     int aux = *a;
15     *a = *b;
16     *b = aux;
17 }
18
19 int buscaSequencial(int *v, int n, int elem) {
20     int i;
21     for (i = 0; i < n; i++) {
22         comp++;
23         if (v[i] == elem)
24             return i; // Elemento encontrado
25     }
26     return -1; // Elemento nao encontrado
27 }
28
29 int particao(int *v, int ini, int fim) {
30     int i = ini, j = fim;
31     int pivo = v[(ini + fim) / 2];
32     while (1) {
33         while (v[i] < pivo) {
34             i++;
35         } // procura algum >= pivo do lado esquerdo
36         while (v[j] > pivo) {
37             j--;
38         } // procura algum <= pivo do lado direito
39
40         if (i < j) {
41             troca(&v[i], &v[j]); // troca os elementos encontrados
42             i++;
43             j--;
44         } else
45             return j; // retorna o local onde foi feita a particao
46     }
47 }
48
49 void QuickSort(int *v, int ini, int fim) {
50     if (ini < fim) {
51         int q = particao(v, ini, fim);
52         QuickSort(v, ini, q);
53         QuickSort(v, q + 1, fim);
54     }
55 }
56
```

```

57 int rec_buscaBinaria(int *v, int ini, int fim, int elem) {
58     if (ini > fim) return -1;
59     int meio = (ini + fim) / 2;
60     comp++;
61     if (v[meio] == elem)
62         return meio;
63     else if (elem < v[meio])
64         return rec_buscaBinaria(v, ini, meio - 1, elem);
65     else
66         return rec_buscaBinaria(v, meio + 1, fim, elem);
67 }
68
69 int it_buscaBinaria(int *v, int ini, int fim, int elem) {
70     int meio;
71     while (ini <= fim) {
72         meio = (ini + fim) / 2;
73         comp++;
74         if (elem == v[meio])
75             return meio;
76         else if (elem < v[meio])
77             fim = meio - 1;
78         else
79             ini = meio + 1;
80     }
81     return -1;
82 }
83
84 int main() {
85     srand(time(NULL));
86     comp = 0;
87     clock_t t;
88
89     int *v;
90     int n, x;
91     printf("Digite o tamanho do vetor:\n");
92     scanf("%d", &n);
93     v = (int *)malloc(n * sizeof(int));
94
95     preencheAleatorio(v, n, 1, n);
96     QuickSort(v, 0, n - 1);
97
98     printf("Digite um elemento para busca:\n");
99     scanf("%d", &x);
100
101     int ind;
102
103     t = clock();
104     ind = buscaSequencial(v, n, x);
105     t = clock() - t;
106     printf("-----Informacoes Busca Sequencial:\n");
107     printf("Tempo Execucao:  %f seconds.\n", ((float)t) / CLOCKS_PER_SEC);
108     printf("Comparacoes: %d\n", comp);
109
110     if (ind != -1)
111         printf("0 elemento %d foi encontrado na pos %d.\n", x, ind);
112     else
113         printf("0 elemento %d NAO foi encontrado!\n", x);
114
115     comp = 0;

```

```
116     t = clock();
117     ind = rec_buscaBinaria(v, 0, n - 1, x);
118     t = clock() - t;
119     printf("-----Informacoes Busca Binaria Recursiva:\n");
120     printf("Tempo Execucao: %f seconds.\n", ((float)t) / CLOCKS_PER_SEC);
121     printf("Comparacoes: %d\n", comp);
122
123     if (ind != -1)
124         printf("O elemento %d foi encontrado na pos %d.\n", x, ind);
125     else
126         printf("O elemento %d NAO foi encontrado!\n", x);
127
128     comp = 0;
129     t = clock();
130     ind = it_buscaBinaria(v, 0, n - 1, x);
131     t = clock() - t;
132     printf("-----Informacoes Busca Binaria Iterativa:\n");
133     printf("Tempo Execucao: %f seconds.\n", ((float)t) / CLOCKS_PER_SEC);
134     printf("Comparacoes: %d\n", comp);
135
136     if (ind != -1)
137         printf("O elemento %d foi encontrado na pos %d.\n", x, ind);
138     else
139         printf("O elemento %d NAO foi encontrado!\n", x);
140
141     free(v);
142     return 0;
143 }
```



gabriel-dp@gabriel-dp: ~/Desktop/dev/c/lab2/roteiro-12



gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-12\$./ex01-01

Digite o tamanho do vetor:

1000000

Digite um elemento para busca:

23

-----Informacoes Busca Sequencial:

Tempo Execucao: 0.002649 seconds.

Comparacoes: 1000000

O elemento 23 NAO foi encontrado!

-----Informacoes Busca Binaria Recursiva:

Tempo Execucao: 0.000002 seconds.

Comparacoes: 20

O elemento 23 NAO foi encontrado!

-----Informacoes Busca Binaria Iterativa:

Tempo Execucao: 0.000001 seconds.

Comparacoes: 20

O elemento 23 NAO foi encontrado!

gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-12\$

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  int comp;
6
7  void preencheAleatorio(int *v, int n, int ini, int fim) {
8      int i;
9      for (i = 0; i < n; i++)
10         v[i] = ini + rand() % (fim - ini + 1);
11 }
12
13 void troca(int *a, int *b) {
14     int aux = *a;
15     *a = *b;
16     *b = aux;
17 }
18
19 int particao(int *v, int ini, int fim) {
20     int i = ini, j = fim;
21     int pivo = v[(ini + fim) / 2];
22     while (1) {
23         while (v[i] > pivo) {
24             i++;
25         }
26         while (v[j] < pivo) {
27             j--;
28         }
29
30         if (i < j) {
31             troca(&v[i], &v[j]);
32             i++;
33             j--;
34         } else
35             return j;
36     }
37 }
38
39 void QuickSort(int *v, int ini, int fim) {
40     if (ini < fim) {
41         int q = particao(v, ini, fim);
42         QuickSort(v, ini, q);
43         QuickSort(v, q + 1, fim);
44     }
45 }
46
47 int rec_buscaBinaria(int *v, int ini, int fim, int elem) {
48     if (ini > fim) return -1;
49     int meio = (ini + fim) / 2;
50     comp++;
51     if (v[meio] == elem)
52         return meio;
53     else if (elem > v[meio])
54         return rec_buscaBinaria(v, ini, meio - 1, elem);
55     else
56         return rec_buscaBinaria(v, meio + 1, fim, elem);
57 }
58
59 int it_buscaBinaria(int *v, int ini, int fim, int elem) {
```

```

60     int meio;
61     while (ini <= fim) {
62         meio = (ini + fim) / 2;
63         comp++;
64         if (elem == v[meio])
65             return meio;
66         else if (elem > v[meio])
67             fim = meio - 1;
68         else
69             ini = meio + 1;
70     }
71     return -1;
72 }
73
74 int main() {
75     srand(time(NULL));
76     comp = 0;
77     clock_t t;
78
79     int *v;
80     int n, x;
81     printf("Digite o tamanho do vetor:\n");
82     scanf("%d", &n);
83     v = (int *)malloc(n * sizeof(int));
84
85     preencheAleatorio(v, n, 1, n);
86     QuickSort(v, 0, n - 1);
87
88     printf("Digite um elemento para busca:\n");
89     scanf("%d", &x);
90
91     int ind;
92
93     comp = 0;
94     t = clock();
95     ind = rec_buscaBinaria(v, 0, n - 1, x);
96     t = clock() - t;
97     printf("-----Informacoes Busca Binaria Recursiva Inversa:\n");
98     printf("Tempo Execucao: %f seconds.\n", ((float)t) / CLOCKS_PER_SEC);
99     printf("Comparacoes: %d\n", comp);
100
101     if (ind != -1)
102         printf("0 elemento %d foi encontrado na pos %d.\n", x, ind);
103     else
104         printf("0 elemento %d NAO foi encontrado!\n", x);
105
106     comp = 0;
107     t = clock();
108     ind = it_buscaBinaria(v, 0, n - 1, x);
109     t = clock() - t;
110     printf("-----Informacoes Busca Binaria Iterativa Inversa:\n");
111     printf("Tempo Execucao: %f seconds.\n", ((float)t) / CLOCKS_PER_SEC);
112     printf("Comparacoes: %d\n", comp);
113
114     if (ind != -1)
115         printf("0 elemento %d foi encontrado na pos %d.\n", x, ind);
116     else
117         printf("0 elemento %d NAO foi encontrado!\n", x);
118
119     free(v);
120     return 0;
121 }

```



gabriel-dp@gabriel-dp: ~/Desktop/dev/c/lab2/roteiro-12



gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-12\$./ex01-02

Digite o tamanho do vetor:

1000000

Digite um elemento para busca:

23
-----Informacoes Busca Binaria Recursiva Inversa:

Tempo Execucao: 0.000002 seconds.

Comparacoes: 20

0 elemento 23 NAO foi encontrado!

-----Informacoes Busca Binaria Iterativa Inversa:

Tempo Execucao: 0.000001 seconds.

Comparacoes: 20

0 elemento 23 NAO foi encontrado!

gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-12\$

roteiro-12/ex01-03.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <time.h>
5
6  #define NOME_TAMANHO 50
7
8  typedef struct {
9      char nome[50 + 1];
10     int matricula;
11 } Aluno;
12
13 int comp;
14
15 void troca(Aluno *a, Aluno *b) {
16     Aluno aux = *a;
17     *a = *b;
18     *b = aux;
19 }
20
21 int particao(Aluno *v, int ini, int fim, int (*compara)(Aluno *, Aluno *)) {
22     int i = ini, j = fim;
23     Aluno pivo = v[(ini + fim) / 2];
24     while (1) {
25         while (compara(&(v[i]), &pivo) < 0) {
26             i++;
27         }
28         while (compara(&(v[j]), &pivo) > 0) {
29             j--;
30         }
31
32         if (i < j) {
33             troca(&v[i], &v[j]);
34             i++;
35             j--;
36         } else
37             return j;
38     }
39 }
40
41 void QuickSort(Aluno *v, int ini, int fim, int (*compara)(Aluno *, Aluno *)) {
42     if (ini < fim) {
43         int q = particao(v, ini, fim, compara);
44         QuickSort(v, ini, q, compara);
45         QuickSort(v, q + 1, fim, compara);
46     }
47 }
48
49 int rec_buscaBinaria(Aluno *v, int ini, int fim, Aluno elem, int (*compara)
(Aluno *, Aluno *)) {
50     if (ini > fim) return -1;
51     int meio = (ini + fim) / 2;
52     comp++;
53     if (compara(&(v[meio]), &elem) == 0)
54         return meio;
55     else if (compara(&(v[meio]), &elem) > 0)
```



```

56         return rec_buscaBinaria(v, ini, meio - 1, elem, compara);
57     else
58         return rec_buscaBinaria(v, meio + 1, fim, elem, compara);
59 }
60
61 int comparaNome(Aluno *a, Aluno *b) {
62     return strcmp(a->nome, b->nome);
63 }
64
65 int comparaMatricula(Aluno *a, Aluno *b) {
66     return a->matricula - b->matricula;
67 }
68
69 int main() {
70     // Atribuicoes iniciais
71     srand(time(NULL));
72     comp = 0;
73     clock_t t;
74
75     Aluno *v, x;
76     int n;
77     printf("Digite o tamanho do vetor:\n");
78     scanf("%d", &n);
79     v = (Aluno *)malloc(n * sizeof(Aluno));
80
81     for (int i = 0; i < n; i++) {
82         printf("Aluno %d\n", i + 1);
83         printf("Nome: ");
84         fgets(v[i].nome, NOME_TAMANHO, stdin);
85         fgets(v[i].nome, NOME_TAMANHO, stdin);
86         printf("Matricula: ");
87         scanf("%d", &(v[i].matricula));
88     }
89
90     printf("Digite um elemento para busca:\n");
91     printf("Nome: ");
92     fgets(x.nome, NOME_TAMANHO, stdin);
93     fgets(x.nome, NOME_TAMANHO, stdin);
94     printf("Matricula: ");
95     scanf("%d", &(x.matricula));
96
97     int ind;
98
99     QuickSort(v, 0, n - 1, &comparaNome);
100    comp = 0;
101    t = clock();
102    ind = rec_buscaBinaria(v, 0, n - 1, x, &comparaNome);
103    t = clock() - t;
104    printf("\n-----Informacoes Busca Binaria Recursiva (Nome):\n");
105    printf("Tempo Execucao: %f seconds.\n", ((float)t) / CLOCKS_PER_SEC);
106    printf("Comparacoes: %d\n", comp);
107
108    if (ind != -1)
109        printf("0 elemento foi encontrado na pos %d.\n", ind);
110    else
111        printf("0 elemento NAO foi encontrado!\n");
112
113    QuickSort(v, 0, n - 1, comparaMatricula);
114    comp = 0;

```

```
115 | t = clock();
116 | ind = rec_buscaBinaria(v, 0, n - 1, x, comparaMatricula);
117 | t = clock() - t;
118 | printf("\n-----Informacoes Busca Binaria Recursiva (Matricula):\n");
119 | printf("Tempo Execucao: %f seconds.\n", ((float)t) / CLOCKS_PER_SEC);
120 | printf("Comparacoes: %d\n", comp);
121 |
122 | if (ind != -1)
123 |     printf("0 elemento foi encontrado na pos %d.\n", ind);
124 | else
125 |     printf("0 elemento NAO foi encontrado!\n");
126 |
127 | free(v);
128 | return 0;
129 | }
```



gabriel-dp@gabriel-dp: ~/Desktop/dev/c/lab2/roteiro-12



gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-12\$./ex01-03

Digite o tamanho do vetor:

3

Aluno 1

Nome: Gabriel

Matricula: 203

Aluno 2

Nome: Davi

Matricula: 205

Aluno 3

Nome: Was

Matricula: 208

Digite um elemento para busca:

Nome: Was

Matricula: 203

-----Informacoes Busca Binaria Recursiva (Nome):

Tempo Execucao: 0.000002 seconds.

Comparacoes: 2

O elemento foi encontrado na pos 2.

-----Informacoes Busca Binaria Recursiva (Matricula):

Tempo Execucao: 0.000001 seconds.

Comparacoes: 2

O elemento foi encontrado na pos 0.

gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-12\$

roteiro-12/hash.h

```
1  #ifndef HASH_H
2  #define HASH_H
3
4  #include <stdio.h>
5  #include <stdlib.h>
6
7  typedef struct {
8      int** tabela;
9      int tam, qtd;
10 } Hash;
11
12 Hash* criaHash(int t) {
13     Hash* h;
14     h = (Hash*)malloc(sizeof(Hash));
15     if (h != NULL) {
16         h->tam = t;
17         h->qtd = 0;
18         h->tabela = (int**)malloc(t * sizeof(int*));
19         if (h->tabela == NULL) return NULL;
20         int i;
21         for (i = 0; i < t; i++)
22             h->tabela[i] = NULL;
23     }
24     return h;
25 }
26
27 void destroiHash(Hash* h) {
28     if (h != NULL) {
29         int i;
30         for (i = 0; i < h->tam; i++)
31             if (h->tabela[i] != NULL)
32                 free(h->tabela[i]);
33         free(h->tabela);
34         free(h);
35     }
36 }
37
38 int chaveDivisao(int chave, int tam) {
39     return (chave & 0x7FFFFFFF) % tam;
40 }
41
42 int chaveMultiplicacao(int chave, int tam) {
43     float A = 0.6180339887; // constante: 0 < A < 1
44     float val = chave * A;
45     val = val - (int)val;
46     return (int)(tam * val);
47 }
48
49 int chaveDobra(int chave, int tam) {
50     int pos, n_bits = 30;
51
52     int p = 1;
53     int r = p << n_bits;
54     while ((chave & r) != r) {
55         n_bits--;
56         r = p << n_bits;
```

```

57     }
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 insereHash_semTratar(Hash* h, int elem, int (*funChave)(int, int)) {
74     if (h == NULL) return 0;
75     int pos = funChave(elem, h->tam);
76
77     if (h->tabela[pos] == NULL) {
78         int* novo = (int*)malloc(sizeof(int));
79         if (novo == NULL) return 0;
80         *novo = elem;
81         h->tabela[pos] = novo;
82         h->qtd++;
83     } else
84         *(h->tabela[pos]) = elem;
85     return 1;
86 }
87
88 int buscaHash_semTratar(Hash* h, int elem, int (*funChave)(int, int), int* p) {
89     if (h == NULL) return 0;
90     int pos = funChave(elem, h->tam);
91     if (h->tabela[pos] == NULL) return 0;
92     if (*(h->tabela[pos]) == elem) {
93         *p = *(h->tabela[pos]);
94         return 1;
95     }
96     return 0;
97 }
98
99 int sondagemLinear(int pos, int chave, int i, int tam, int (*funChave)(int,
100 int)) {
101     return ((pos + i) & 0x7FFFFFFF) % tam;
102 }
103
104 int sondagemQuadratica(int pos, int chave, int i, int tam, int (*funChave)(int,
105 int)) {
106     pos = pos + 2 * i + 5 * i * i;
107     return (pos & 0x7FFFFFFF) % tam;
108 }
109
110 int sondagemDuploHash(int H1, int chave, int i, int tam, int (*funChave)(int,
111 int)) {
112     int H2 = funChave(chave, tam - 1) + 1;
113     return ((H1 + i * H2) & 0x7FFFFFFF) % tam;
114 }

```

```

113 int insereHash_EnderAberto(Hash* h, int elem, int (*funChave)(int, int), int
    (*funSondagem)(int, int, int, int, int (*)(int, int))) {
114     if (h == NULL) return 0;
115     int i, pos, newPos;
116     pos = funChave(elem, h->tam);
117     for (i = 0; i < h->tam; i++) {
118         newPos = funSondagem(pos, elem, i, h->tam, funChave);
119         if (h->tabela[newPos] == NULL) {
120             int* novo = (int*)malloc(sizeof(int));
121             if (novo == NULL) return 0;
122             *novo = elem;
123             h->tabela[newPos] = novo;
124             h->qtd++;
125             return 1;
126         }
127     }
128     return 0;
129 }
130
131 int buscaHash_EnderAberto(Hash* h, int elem, int* p, int (*funChave)(int, int),
    int (*funSondagem)(int, int, int, int, int (*)(int, int))) {
132     if (h == NULL) return 0;
133     int i, pos, newPos;
134     pos = funChave(elem, h->tam);
135     for (i = 0; i < h->tam; i++) {
136         newPos = funSondagem(pos, elem, i, h->tam, funChave);
137         if (h->tabela[newPos] == NULL) return 0;
138         if (*(h->tabela[newPos]) == elem) {
139             *p = *(h->tabela[newPos]);
140             return 1;
141         }
142     }
143     return 0;
144 }
145
146 void imprimeHash(Hash* h) {
147     if (h == NULL) return;
148     int i;
149     for (i = 0; i < h->tam; i++) {
150         printf("%d: ", i);
151         if (h->tabela[i] == NULL)
152             printf("NULL\n");
153         else
154             printf("%d\n", *(h->tabela[i]));
155     }
156 }
157
158 #endif

```

roteiro-12/ex02-01.c

```
1  #include <stdio.h>
2
3  #include "hash.h"
4
5  int main() {
6      int elementos[] = {23, 19, 81, 2, 49, 7, 25, 99};
7      char* funsNome[] = {"Divisao", "Multiplicacao", "Dobra"};
8      int (*funsChave[])(int, int) = {&chaveDivisao, &chaveMultiplicacao, &
chaveDobra};
9
10     printf("{ ");
11     for (int j = 0; j < (sizeof(elementos) / sizeof(int)); j++) {
12         printf("%d ", elementos[j]);
13     }
14     printf("}\n\n");
15
16     for (int i = 0; i < (sizeof(funsChave) / sizeof(funsChave[0])); i++) {
17         Hash* tabela = criaHash(10);
18
19         for (int j = 0; j < (sizeof(elementos) / sizeof(int)); j++)
20             insereHash_semTratar(tabela, elementos[j], funsChave[i]);
21
22         printf("%s\n", funsNome[i]);
23         imprimeHash(tabela);
24         printf("\n");
25
26         destroiHash(tabela);
27     }
28
29     return 0;
30 }
```



gabriel-dp@gabriel-dp: ~/Desktop/dev/c/lab2/roteiro-12



```
gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-12$ ./ex02-01
```

```
{ 23 19 81 2 49 7 25 99 }
```

Divisao

0: NULL

1: 81

2: 2

3: 23

4: NULL

5: 25

6: NULL

7: 7

8: NULL

9: 99

Multiplicacao

0: 81

1: 99

2: 49

3: 7

4: 25

5: NULL

6: NULL

7: 19

8: NULL

9: NULL

Dobra

0: NULL

1: NULL

2: 2

3: NULL

4: 81

5: NULL

6: 99

7: 25

8: NULL

9: NULL

```
gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-12$
```


roteiro-12/ex02-02.c

```
1  #include <stdio.h>
2
3  #include "hash.h"
4
5  int main() {
6      int elementos[] = {23, 19, 81, 2, 49, 7, 25, 99};
7      char* funsNome[] = {"Linear", "Quadratica", "Duplo Hash"};
8      int (*funsSondagem[])(int, int, int, int, int (*)(int, int)) = {&
sondagemLinear, &sondagemQuadratica, &sondagemDuploHash};
9
10     printf("{ ");
11     for (int j = 0; j < (sizeof(elementos) / sizeof(int)); j++)
12         printf("%d ", elementos[j]);
13     printf("}\n\n");
14
15     for (int i = 0; i < (sizeof(funsSondagem) / sizeof(funsSondagem[0])); i++) {
16         Hash* tabela = criaHash(10);
17
18         for (int j = 0; j < (sizeof(elementos) / sizeof(int)); j++)
19             insereHash_EnderAberto(tabela, elementos[j], &chaveDivisao,
funsSondagem[i]);
20
21         printf("%s\n", funsNome[i]);
22         imprimeHash(tabela);
23         printf("\n");
24
25         destroiHash(tabela);
26     }
27
28     return 0;
29 }
```



gabriel-dp@gabriel-dp: ~/Desktop/dev/c/lab2/roteiro-12



```
gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-12$ ./ex02-02
```

```
{ 23 19 81 2 49 7 25 99 }
```

Linear

```
0: 49
1: 81
2: 2
3: 23
4: 99
5: 25
6: NULL
7: 7
8: NULL
9: 19
```

Quadratica

```
0: 99
1: 81
2: 2
3: 23
4: NULL
5: 25
6: 49
7: 7
8: NULL
9: 19
```

Duplo Hash

```
0: 99
1: 81
2: 2
3: 23
4: 49
5: 25
6: NULL
7: 7
8: NULL
9: 19
```

```
gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-12$
```

roteiro-12/hash-lse.h

```
1  #ifndef HASH_LSE_H
2  #define HASH_LSE_H
3
4  #include <stdio.h>
5  #include <stdlib.h>
6
7  #include "lse.h"
8
9  typedef struct {
10     Lista **tabela;
11     int tam, qtd;
12 } Hash;
13
14 Hash *criaHash(int t) {
15     Hash *h;
16     h = (Hash *)malloc(sizeof(Hash));
17     if (h != NULL) {
18         h->tam = t;
19         h->qtd = 0;
20         h->tabela = (Lista **)malloc(t * sizeof(Lista *));
21         if (h->tabela == NULL) return NULL;
22         int i;
23         for (i = 0; i < t; i++)
24             h->tabela[i] = NULL;
25     }
26     return h;
27 }
28
29 void destroiHash(Hash *h) {
30     if (h != NULL) {
31         int i;
32         for (i = 0; i < h->tam; i++)
33             if (h->tabela[i] != NULL)
34                 destroiLista(h->tabela[i]);
35         free(h->tabela);
36         free(h);
37     }
38 }
39
40 int chaveDivisao(int chave, int tam) {
41     return (chave & 0x7FFFFFFF) % tam;
42 }
43
44 int insereHashLSE(Hash *h, int elem) {
45     if (h == NULL) return 0;
46     int pos = chaveDivisao(elem, h->tam);
47     if (h->tabela[pos] == NULL)
48         h->tabela[pos] = criaLista();
49     insereIni(h->tabela[pos], elem);
50     h->qtd++;
51     return 1;
52 }
53
54 int buscaHashLSE(Hash *h, int elem, int *p) {
55     if (h == NULL) return 0;
56     int pos = chaveDivisao(elem, h->tam);
```

```
57     if (h->tabela[pos] == NULL) return 0;
58     return listaBuscaElem(h->tabela[pos], elem, p);
59 }
60
61 void imprimeHash(Hash *h) {
62     if (h == NULL) return;
63     int i;
64     for (i = 0; i < h->tam; i++) {
65         printf("%d: ", i);
66         if (h->tabela[i] == NULL)
67             printf("NULL\n");
68         else
69             imprimeLista(h->tabela[i]);
70     }
71 }
72
73 #endif
```

roteiro-12/lse.h

```
1  #ifndef LISTASE_H
2  #define LISTASE_H
3
4  #include <stdio.h>
5  #include <stdlib.h>
6
7  typedef struct NO {
8      int info;
9      struct NO* prox;
10 } NO;
11
12 typedef struct NO* Lista;
13
14 Lista* criaLista() {
15     Lista* li;
16     li = (Lista*)malloc(sizeof(Lista));
17     if (li != NULL) {
18         *li = NULL;
19     }
20     return li;
21 }
22
23 int listaVazia(Lista* li) {
24     if (li == NULL) return 1;
25     if (*li == NULL) return 1; // True - Vazia!
26     return 0;                 // False - tem elemento!
27 }
28
29 NO* alocarNO() {
30     return (NO*)malloc(sizeof(NO));
31 }
32
33 void liberarNO(NO* q) {
34     free(q);
35 }
36
37 int listaBuscaElem(Lista* li, int elem, int* p) {
38     if (li == NULL) return 0;
39     NO* aux = *li;
40     while (aux != NULL) {
41         if (aux->info == elem) {
42             *p = aux->info;
43             return 1;
44         }
45         aux = aux->prox;
46     }
47     return 0;
48 }
49
50 int insereIni(Lista* li, int elem) {
51     if (li == NULL) return 0;
52     NO* novo = alocarNO();
53     if (novo == NULL) return 0;
54     novo->info = elem;
55     novo->prox = *li;
56     *li = novo;
```

```

57     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
130
```

roteiro-12/ex02-03.c

```
1  #include <stdio.h>
2
3  #include "hash-lse.h"
4
5  int main() {
6      int elementos[] = {23, 19, 81, 2, 49, 7, 25, 99};
7
8      Hash* tabela = criaHash(10);
9
10     printf("{ ");
11     for (int j = 0; j < (sizeof(elementos) / sizeof(int)); j++) {
12         printf("%d ", elementos[j]);
13         insereHashLSE(tabela, elementos[j]);
14     }
15     printf("}\n\n");
16
17     imprimeHash(tabela);
18
19     destroiHash(tabela);
20
21     return 0;
22 }
```




gabriel-dp@gabriel-dp: ~/Desktop/dev/c/lab2/roteiro-12



```
gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-12$ ./ex02-03
```

```
{ 23 19 81 2 49 7 25 99 }
```

```
0: NULL
```

```
1: 81
```

```
2: 2
```

```
3: 23
```

```
4: NULL
```

```
5: 25
```

```
6: NULL
```

```
7: 7
```

```
8: NULL
```

```
9: 99 49 19
```

```
gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-12$
```