

2.1/hash.h

```
1  #ifndef HASH_H
2  #define HASH_H
3
4  #include <stdio.h>
5  #include <stdlib.h>
6  #include <string.h>
7
8  typedef struct{
9      int **tabela;
10     int tam, qtd;
11 }Hash;
12
13
14 Hash* criaHash(int t){
15     Hash* h;
16     h = (Hash*) malloc (sizeof(Hash));
17     if(h != NULL){
18         h->tam = t; h->qtd = 0;
19         h->tabela = (int**) malloc (t*sizeof(int*));
20         if(h->tabela == NULL) return NULL;
21         int i;
22         for(i = 0; i<t; i++)
23             h->tabela[i] = NULL;
24     }
25     return h;
26 }
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                 free(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 chaveMultiplicacao(int chave, int tam){
45     float A = 0.6180339887; //constante: 0 < A < 1
46     float val = chave * A;
47     val = val - (int) val;
48     return (int) (tam * val);
49 }
50
51 int chaveDobra(int chave, int tam){
52     int pos, n_bits = 30;
53
54     int p = 1;
55     int r = p << n_bits;
56     while((chave & r) != r){ n_bits--; r = p << n_bits; }
57 }
```

```
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     }
78     return valor;
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
```