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 }
```

```
178
179 int buscaHash_EnderAberto_lin(Hash* h, int elem, int *p){
180     if(h == NULL) return 0;
181     int i, pos, newPos;
182     pos = chaveDivisao(elem, h->tam);
183     for(i=0; i<h->tam; i++){
184         newPos = sondagemLinear(pos, i, h->tam);
185         //newPos = sondagemQuadratica(pos, i, h->tam);
186         //newPos = sondagemDuploHash(pos, elem, i, h->tam);
187         if(h->tabela[newPos] == NULL) return 0;
188         if(*(h->tabela[newPos]) == elem){
189             *p = *(h->tabela[newPos]);
190             return 1;
191         }
192     }
193     return 0;
194 }
195
196 int buscaHash_EnderAberto_quad(Hash* h, int elem, int *p){
197     if(h == NULL) return 0;
198     int i, pos, newPos;
199     pos = chaveDivisao(elem, h->tam);
200     for(i=0; i<h->tam; i++){
201         // newPos = sondagemLinear(pos, i, h->tam);
202         newPos = sondagemQuadratica(pos, i, h->tam);
203         //newPos = sondagemDuploHash(pos, elem, i, h->tam);
204         if(h->tabela[newPos] == NULL) return 0;
205         if(*(h->tabela[newPos]) == elem){
206             *p = *(h->tabela[newPos]);
207             return 1;
208         }
209     }
210     return 0;
211 }
212
213 int buscaHash_EnderAberto_duplo(Hash* h, int elem, int *p){
214     if(h == NULL) return 0;
215     int i, pos, newPos;
216     pos = chaveDivisao(elem, h->tam);
217     for(i=0; i<h->tam; i++){
218         //newPos = sondagemLinear(pos, i, h->tam);
219         //newPos = sondagemQuadratica(pos, i, h->tam);
220         newPos = sondagemDuploHash(pos, elem, i, h->tam);
221         if(h->tabela[newPos] == NULL) return 0;
222         if(*(h->tabela[newPos]) == elem){
223             *p = *(h->tabela[newPos]);
224             return 1;
225         }
226     }
227     return 0;
228 }
229
230
231
232 void imprimeHash(Hash *h){
233     if(h == NULL) return;
234     int i;
235     for(i=0; i<h->tam; i++){
236         printf("%d: ", i);
237         if(h->tabela[i] == NULL) printf("NULL\n");
```

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