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; }

    n_bits++;
    pos = chave;
    while(pos > tam){
        int metade_bits = n_bits/2;
        int parte1 = pos >> metade_bits;
        parte1 = parte1 << metade_bits;
        int parte2 = pos ^ parte1;
        parte1 = pos >> metade_bits;
        pos = parte1 ^ parte2;
        n_bits = n_bits/2;
    }
    return pos;
}

int valorString(char *str){
    int i, valor = 1;
    int tam = strlen(str);
    for(i=0; i<tam; i++)
        valor = 31*valor + (i+1)*((int) str[i]);
    return valor;
}

int insereHash_semTratar_div(Hash* h, int elem){
    if(h == NULL) return 0;
    int pos = chaveDivisao(elem, h->tam);

    if(h->tabela[pos] == NULL){
        int* novo = (int*) malloc (sizeof(int));
        if(novo == NULL) return 0;
        *novo = elem;
        h->tabela[pos] = novo;
        h->qtd++;
    }else *(h->tabela[pos]) = elem;
    return 1;
}

int insereHash_semTratar_mul(Hash* h, int elem){
    if(h == NULL) return 0;
    int pos = chaveMultiplicacao(elem, h->tam);

    if(h->tabela[pos] == NULL){
        int* novo = (int*) malloc (sizeof(int));
        if(novo == NULL) return 0;
        *novo = elem;
        h->tabela[pos] = novo;
        h->qtd++;
    }else *(h->tabela[pos]) = elem;
    return 1;
}

int insereHash_semTratar_dobra(Hash* h, int elem){
    if(h == NULL) return 0;
    int pos = chaveDobra(elem, h->tam);

    if(h->tabela[pos] == NULL){
        int* novo = (int*) malloc (sizeof(int));
        if(novo == NULL) return 0;
        *novo = elem;
        h->tabela[pos] = novo;
        h->qtd++;
    }else *(h->tabela[pos]) = elem;
    return 1;
}

int buscaHash_semTratar_div(Hash* h, int elem, int *p){
    if(h == NULL) return 0;
    int pos = chaveDivisao(elem, h->tam);
    if(h->tabela[pos] == NULL) return 0;
    if(*(h->tabela[pos]) == elem){
        *p = *(h->tabela[pos]);
        return 1;
    }
    return 0;
}

int buscaHash_semTratar_mul(Hash* h, int elem, int *p){
    if(h == NULL) return 0;
    int pos = chaveMultiplicacao(elem, h->tam);
    if(h->tabela[pos] == NULL) return 0;
    if(*(h->tabela[pos]) == elem){
        *p = *(h->tabela[pos]);
        return 1;
    }
    return 0;
}

int buscaHash_semTratar_dobra(Hash* h, int elem, int *p){
    if(h == NULL) return 0;
    int pos = chaveDobra(elem, h->tam);
    if(h->tabela[pos] == NULL) return 0;
    if(*(h->tabela[pos]) == elem){
        *p = *(h->tabela[pos]);
        return 1;
    }
    return 0;
}

int sondagemLinear(int pos, int i, int tam){
    return ( (pos + i) & 0x7FFFFFFF) % tam;
}

int sondagemQuadratica(int pos, int i, int tam){
    pos = pos + 2*i + 5*i*i;
    return ( pos & 0x7FFFFFFF) % tam;
}

int sondagemDuploHash(int H1, int chave, int i, int tam){
    int H2 = chaveDivisao(chave, tam-1) + 1;
    return ( (H1 + i*H2) & 0x7FFFFFFF) % tam;
}

int insereHash_EnderAberto(Hash* h, int elem){
  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){
        int* novo = (int*) malloc (sizeof(int));
        if(novo == NULL) return 0;
        *novo = elem;
        h->tabela[newPos] = novo;
        h->qtd++;
        return 1;
    }
  }
  return 0;
}

int buscaHash_EnderAberto(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");
        else printf("%d\n", *(h->tabela[i]));
    }
}

#endif
```