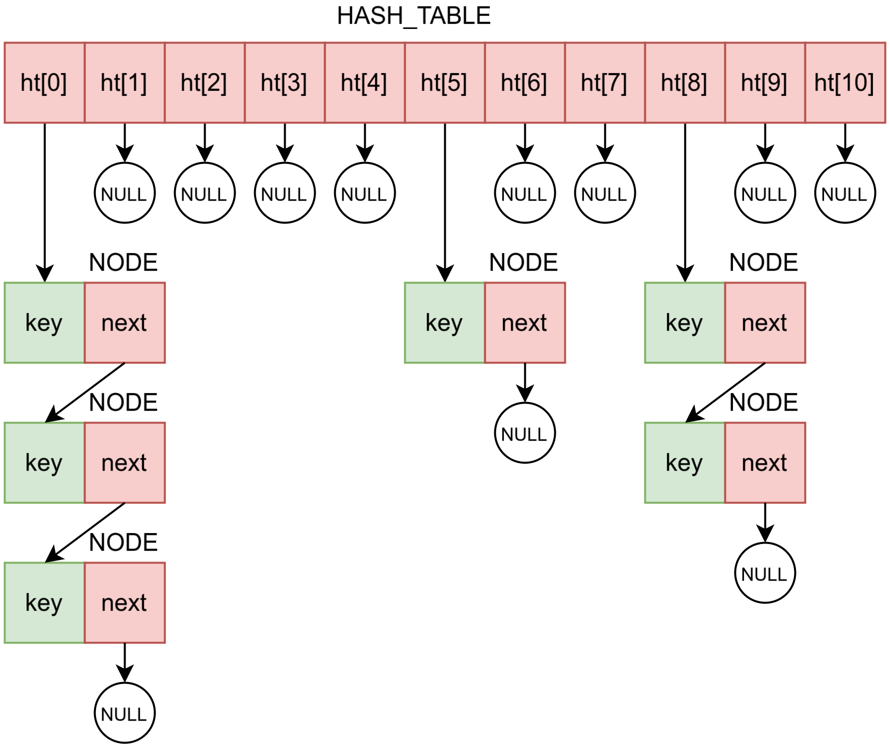
Lab 4 - Hash Tables

1. Implement a hash table that uses collision resolution by chaining using a data strucure similar to the following example:

```
typedef struct _NODE{
    int key;
    struct _NODE* next;
} NODE;

typedef struct{
    NODE* first;
} SLL;

typedef struct {
    // the size of the table should be a prime number
    SLL v[17];
} HASH_TABLE;
```



Create a function for each of the following operations:

(a) Initialise the hashtable ★

(b) Insert a value in the hashtable ★

(c) Search a value in the hashtable ★

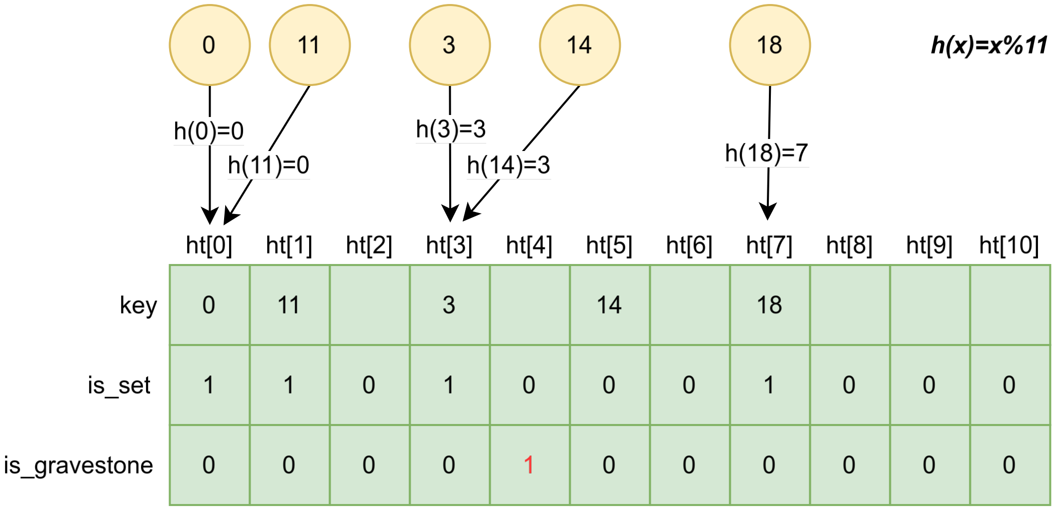
(d) Delete a value from the hashtable ★★

(e) Deallocate the hashtable (free the memory for the nodes and the table) ★

2. Implement a hash table that uses collision resolution by open addressing using a data strucure similar to the following example:

```
typedef struct {
    int key;
    char is_set;
    char is_gravestone;
} HT_ENTRY;

typedef struct {
    // a vector of HT_ENTRIES
    HT_ENTRY *v;
    // the size of the table should be a prime number
    int ht_size;
    // optional , used to compute the load factor
    int nr_occupied;
} HASH_TABLE;
```



	ht[0]	ht[1]	ht[2]	ht[3]	ht[4]	ht[5]	ht[6]	ht[7]	ht[8]	ht[9]	ht[10]
key	0	11		3		14		18			
is_set	1	1	0	1	0	0	0	1	0	0	0
is_gravestone	0	0	0	0	1	0	0	0	0	0	0

HASH_TABLE

(a) Implement the following operations by using linear probing (`pos = (h(x) + i) % ht_size`)

i. Initialise a table of a given size ★

ii. Insert a value in the hashtable ★★

iii. Search a value in the hashtable ★★

iv. Delete a value from the hashtable ★★

v. Get the hash table's load factor ★★

vi. Increase the table size and rehash the table if the load factor is above 0.7 ★★★

vii. Rehash the table if the number of gravestones is above 30% of the hashtable's size ★★★

viii. Deallocate the table ★

(b) Implement the following operations by using quadratic probing (`pos = (h(x) + i*i) % ht_size`)

i. Initialise a table of a given size ★

ii. Insert a value in the hashtable ★★

iii. Search a value in the hashtable ★★

iv. Delete a value from the hashtable ★★

v. Get the hash table's load factor ★★

vi. Increase the table size and rehash the table if the load factor is above 0.7 ★★★

vii. Rehash the table if the number of gravestones is above 30% of the hashtable's size ★★★

   viii. Deallocate the table ★

(c) Implement the following operations by using double hashing (`pos = (h(x)+ i*h2(x)) % ht_size`)

   i. Initialise a table of a given size ★

   ii. Insert a value in the hashtable ★★

   iii. Search a value in the hashtable ★★

   iv. Delete a value from the hashtable ★★

   v. Get the hash table's load factor ★★

   vi. Increase the table size and rehash the table if the load factor is above 0.7 ★★★

   vii. Rehash the table if the number of gravestones is above 30% of the hashtable's size ★★★

   viii. Deallocate the table ★

Note: Leave a comment with the text PB1, PB2.A.II, ... PB10 above every function that implements the respective lab task. (upper case text, no space between the text and the problem number)