***// HASH TABLE***

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <stdbool.h>

typedef struct

{

int \*table;

int size;

} HashTable;

*// Create a hash table of the required size*

HashTable \*create\_table(int size);

*// Search for an element in the hash table*

*// Return 1 if the element is present in the table*

*// Return 0 otherwise*

int search(HashTable \*htable, int element);

*// Insert the element into the hash table*

void insert(HashTable \*htable, int element);

*// Delete the element from the hash table if present*

void delete (HashTable \*htable, int element);

*// Display all the hash table elements*

void display\_table(HashTable \*htable);

*// Destroy the hash table*

void destroy\_table(HashTable \*htable);

int main()

{

int size, choice, loop = 1;

int element, find;

scanf("%d", &size);

HashTable \*htable = create\_table(size);

while (loop)

{

scanf("%d", &choice);

switch (choice)

{

case 1:

*// Insert element*

scanf("%d", &element);

insert(htable, element);

break;

case 2:

*// Delete element*

scanf("%d", &element);

delete (htable, element);

break;

case 3:

*// Search element*

scanf("%d", &element);

find = search(htable, element);

if (find)

printf("YES\n");

else

printf("NO\n");

break;

case 4:

*// Print all elements in the hash table*

display\_table(htable);

break;

default:

*// Destroy tree and exit the loop*

destroy\_table(htable);

loop = 0;

break;

}

}

}

HashTable \*create\_table(int size)

{

HashTable \*a = (HashTable\*)malloc(sizeof(HashTable));

a->table = (int\*)malloc(sizeof(int)\*size);

a->size = size;

for(int i=0;i<size;i++)

{

a->table[i] = -1;

}

return a;

}

void insert(HashTable \*htable, int element)

{

int index = element%(htable->size);

if(htable->table[index]==-1)

{

htable->table[index] = element;

}

else

{

int flag = 0;

for(int i = index+1;i<htable->size;i++)

{

if(htable->table[i]==-1)

{

htable->table[i] = element;

flag = 1;

break;

}

}

if(flag==0)

{

for(int j = 0;j<index;j++)

{

if(htable->table[j]==-1)

{

htable->table[j] = element;

break;

}

}

}

}

}

int search(HashTable \*htable, int element)

{

int index = element%(htable->size);

if(htable->table[index]==element)

{

return 1;

}

else

{

int i = index+1;

while(htable->table[i]!=-1 && i<htable->size)

{

if(htable->table[i]==element)

{

return 1;

}

i++;

}

if(htable->table[i]==-1)

{

return 0;

}

if(i==htable->size)

{

int j = 0;

while(htable->table[j]!=-1 && j<index)

{

if(htable->table[j]==element)

{

return 1;

}

j++;

}

return 0;

}

}

}

void delete (HashTable \*htable, int element)

{

int pres = search(htable,element);

if(pres)

{

int index = element%(htable->size);

if(htable->table[index]==element)

{

htable->table[index] = -1;

return;

}

else

{

int i = index+1;

while(htable->table[i]!=-1 && i<htable->size)

{

if(htable->table[i]==element)

{

htable->table[i] = -1;

return;

}

i++;

}

if(i==htable->size)

{

int j = 0;

while(htable->table[j]!=-1 && j<index)

{

if(htable->table[j]==element)

{

htable->table[j] = -1;

return;

}

j++;

}

}

}

}

}

void display\_table(HashTable \*htable)

{

for(int i = 0;i<htable->size;i++)

{

printf("%d ",htable->table[i]);

}

printf("\n");

}

void destroy\_table(HashTable \*htable)

{

free(htable->table);

}