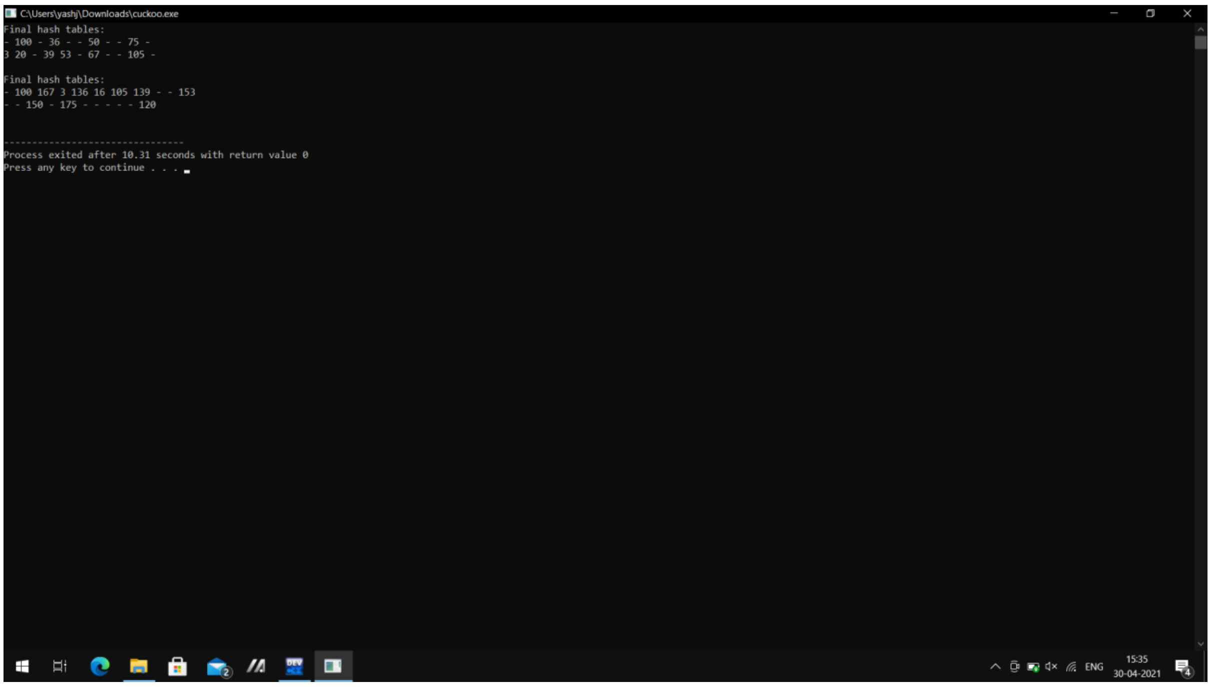
```
// C++ program to demonstrate working of Cuckoo
// hashing.

/*
Name:Yash C Jaware
Roll number:142
seat number:S204068

*/


#include<bits/stdc++.h>
// upper bound on number of elements in our set
#define MAXN 11
// choices for position
#define ver 2
// Auxiliary space bounded by a small multiple
// of MAXN, minimizing wastage
int hashtable[ver][MAXN];
// Array to store possible positions for a key
int pos[ver];
/* function to fill hash table with dummy value
* dummy value: INT_MIN
* number of hashtables: ver */
void initTable()
{
for (int j=0; j<MAXN; j++)
for (int i=0; i<ver; i++)
hashtable[i][j] = INT_MIN;
}
/* return hashed value for a key
```

```
* function: ID of hash function according to which
key has to hashed
* key: item to be hashed */
int hash(int function, int key)
{
switch (function)
{
case 1: return key%MAXN;
case 2: return (key/MAXN)%MAXN;
}
}
/* function to place a key in one of its possible positions
* tableID: table in which key has to be placed, also equal
to function according to which key must be hashed
* cnt: number of times function has already been called
in order to place the first input key
* n: maximum number of times function can be recursively
called before stopping and declaring presence of cycle */
void place(int key, int tableID, int cnt, int n)
{
/* if function has been recursively called max number
of times, stop and declare cycle. Rehash. */
if (cnt==n)
{
printf("%d unpositioned\n", key);
printf("Cycle present. REHASH.\n");
return;
}
/* calculate and store possible positions for the key.
* check if key already present at any of the positions.
If YES, return. */
for (int i=0; i<ver; i++)
{
pos[i] = hash(i+1, key);
if (hashtable[i][pos[i]] == key)
return;
}
/* check if another key is already present at the
position for the new key in the table
* If YES: place the new key in its position
* and place the older key in an alternate position
for it in the next table */
if (hashtable[tableID][pos[tableID]]!=INT_MIN)
{
int dis = hashtable[tableID][pos[tableID]];
hashtable[tableID][pos[tableID]] = key;
place(dis, (tableID+1)%ver, cnt+1, n);
}
else //else: place the new key in its position
hashtable[tableID][pos[tableID]] = key;
}
/* function to print hash table contents */
void printTable()
{
printf("Final hash tables:\n");
for (int i=0; i<ver; i++, printf("\n"))
for (int j=0; j<MAXN; j++)
(hashtable[i][j]==INT_MIN)? printf("- "):
printf("%d ", hashtable[i][j]);
printf("\n");
}
/* function for Cuckoo-hashing keys
```

```
* keys[]: input array of keys
* n: size of input array */
void cuckoo(int keys[], int n)
{
// initialize hash tables to a dummy value (INT-MIN)
// indicating empty position
initTable();
// start with placing every key at its position in
// the first hash table according to first hash
// function
for (int i=0, cnt=0; i<n; i++, cnt=0)
place(keys[i], 0, cnt, n);
//print the final hash tables
printTable();
}
/* driver function */
int main()
{
/* following array doesn't have any cycles and
hence all keys will be inserted without any
rehashing */
int keys_1[] = {20, 50, 53, 75, 100, 67, 105,
3, 36, 39};
int n = sizeof(keys_1)/sizeof(int);
cuckoo(keys_1, n);
/* following array has a cycle and hence we will
have to rehash to position every key */
int keys_2[] = {120,150,153,175, 100,167,105,
3,136,139,16};
int m = sizeof(keys_2)/sizeof(int);
cuckoo(keys_2, m);
return 0;

}
```



```
C:\Users\yashj\Downloads\cuckoo.exe
Final hash tables:
- 100 - 36 - - 50 - - 75 -
3 20 - 39 53 - 67 - - 105 -

Final hash tables:
- 100 167 3 136 16 105 139 - - 153
- - 150 - 175 - - - - - 120


--------------------------------
Process exited after 10.31 seconds with return value 0
Press any key to continue . . .
```