Aim:-

Insert the keys into a hash table of length m using open addressing using double hashing with h(k)=1+(k mod(m-1)).

Objective:-

To Insert the keys into a hash table of length m using open addressing using double hashing with h(k)=1+(k mod(m-1)).

Theory:-

Double hashing is a collision resolving technique in **Open Addressed** Hash tables. Double hashing uses the idea of applying a second hash function to key when a collision occurs.

Double hashing can be done using :
(hash1(key) + i * hash2(key)) % TABLE_SIZE
Here hash1() and hash2() are hash functions and TABLE_SIZE is size of hash table.
(We repeat by increasing i when collision occurs)

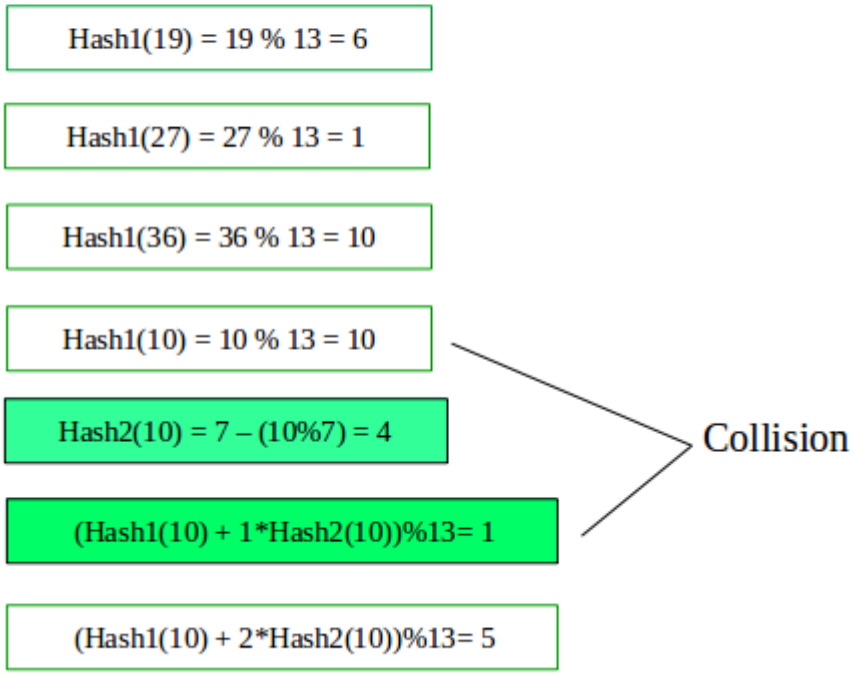
First hash function is typically hash1(key) = key % TABLE_SIZE

A popular second hash function is : **hash2(key) = PRIME – (key % PRIME)** where PRIME is a prime smaller than the TABLE_SIZE.
A good second Hash function is:

- It must never evaluate to zero
- Must make sure that all cells can be probed

Lets say, Hash1 (key) = key % 13

Hash2 (key) = 7 – (key % 7)



Algorithm:-

- Linear probing collision resolution leads to clusters in the table, because if two keys collide, the next position probed will be the same for both of them.
- The idea of double hashing: Make the offset to the next position probed depend on the key value, so it can be different for different keys
    - Need to introduce a second hash function $H_2(K)$, which is used as the offset in the probe sequence (think of linear probing as double hashing with $H_2(K) == 1$)
    - For a hash table of size M, $H_2(K)$ should have values in the range 1 through M-1; if M is prime, one common choice is H2(K) = 1 + ((K/M) mod (M-1))

- The insert algorithm for double hashing is then:

  1. Set indx = H(K); offset = H_2 (K)
  2. If table location indx already contains the key, no need to insert it. Done!
  3. Else if table location indx is empty, insert key there. Done!
  4. Else collision. Set indx = (indx + offset) mod M.
  5. If indx == H(K), table is full! (Throw an exception, or enlarge table.) Else go to 2.

- With prime table size, double hashing works very well in practice

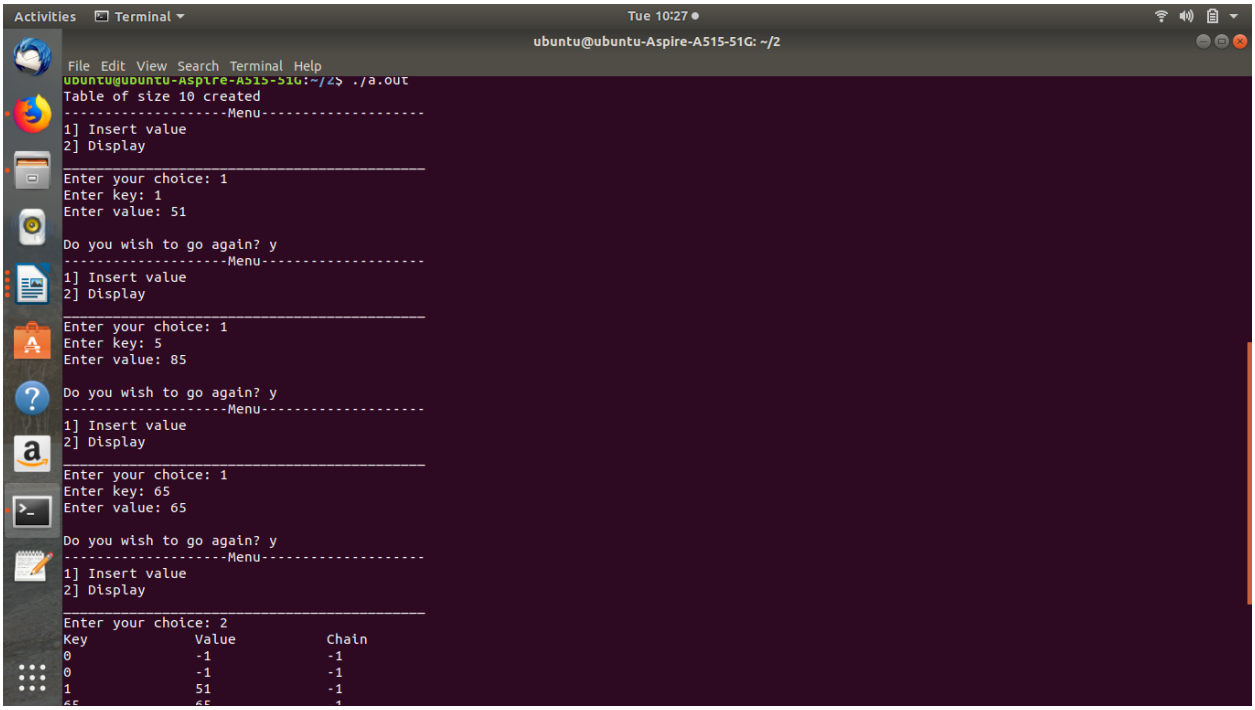
Code:-

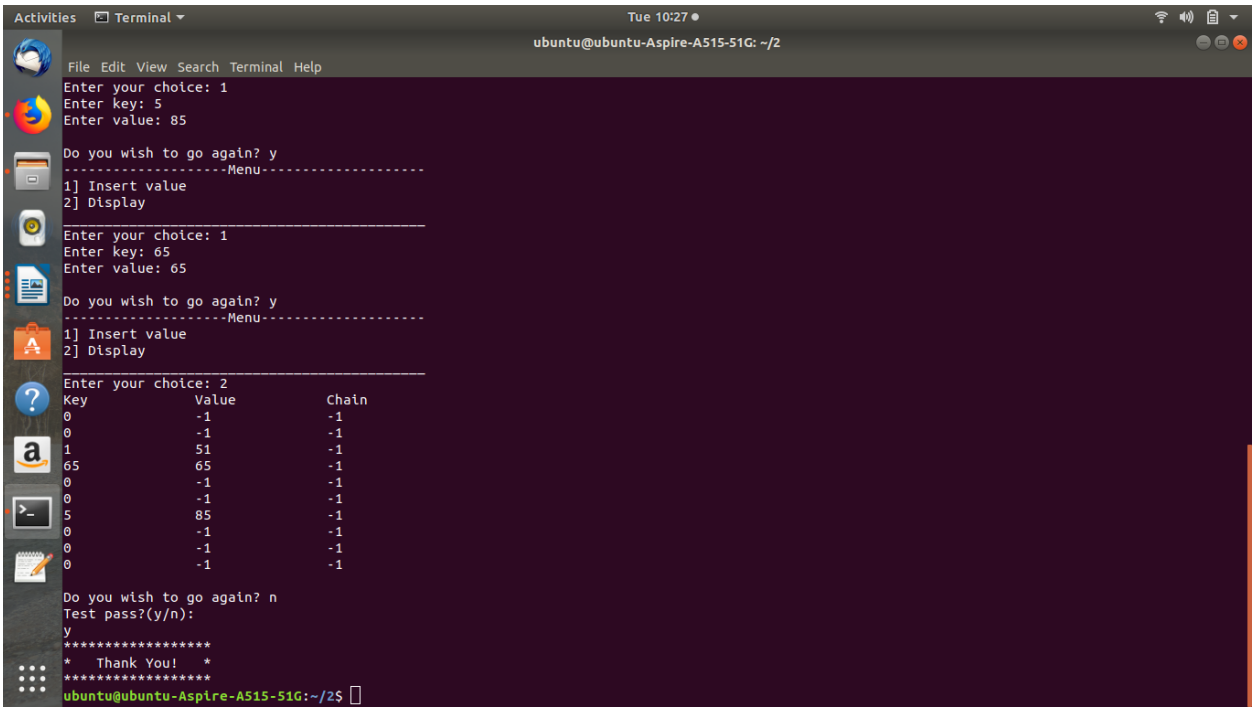
```
       #include <iostream>
using namespace std;
class dr
{
    int n=10;
    int arr[100][3];
    int c;
public:
    dr()
    {
        cout<<"Table of size "<<n<<" created\n";
        for(int i=0;i<n;i++)
        {
            arr[i][0]=0;
            arr[i][1]=-1;
            arr[i][2]=-1;
        }
        c=0;
    }
    void add(int,int);
    int find_key(int);
    void display();
    void update_val(int,int);
};
void dr::add(int key,int value)
{
    int new_hash_addr1,new_hash_addr2,main_hash_addr=-1,j=0;
    if(this->find_key(key)!=-1)
    {
        cout<<"Key already exists\n";
        return;
    }
    if(c==(n-1))
    {
        cout<<"Table full, request denied\n";
    }
    new_hash_addr1=(key)%n;
    new_hash_addr1=1+(key%(n-1));
    if(arr[new_hash_addr1][1]==-1)
    {
        arr[new_hash_addr1][0]=key;
        arr[new_hash_addr1][1]=value;
    }
    else if(arr[new_hash_addr2][1]==-1)
    {
        arr[new_hash_addr2][0]=key;
        arr[new_hash_addr2][1]=value;
    }
    else
    {
        while(arr[new_hash_addr2][2]!=-1)
```

```
        {
            main_hash_addr=new_hash_addr2;
            new_hash_addr2=arr[main_hash_addr][2];
        }
        main_hash_addr=new_hash_addr2;
        for(int i=0;i<n;i++)
        {
            new_hash_addr2=(main_hash_addr+i)%n;
            if(arr[new_hash_addr2][1]==-1)
            {
                arr[new_hash_addr2][0]=key;
                arr[new_hash_addr2][1]=value;
                arr[main_hash_addr][2]=new_hash_addr2;
                c++;
                break;
            }
        }
    }
}
void dr::display()
{
    cout<<"Key\t\tValue\t\tChain\n";
    for(int i=0;i<n;i++)
    {
        cout<<arr[i][0]<<"\t\t"<<arr[i][1]<<"\t\t"<<arr[i][2]<<endl;
    }
}
int dr::find_key(int key)
{
    int search_addr=key%n,f=0;
    while(arr[search_addr][0]!=key && arr[search_addr][2]!=-1)
    {
        search_addr=arr[search_addr][2];
    }
    if(arr[search_addr][0]==key)
    {
        return arr[search_addr][1];
    }
    else if(arr[search_addr][2]==-1)
    {
        return -1;
    }
}
int main()
{
    char r;
    do
    {
        char op;
        dr table;
        int c;
        do
        {
            cout<<"--------------------Menu--------------------\n";
            cout<<"1] Insert value\n2] Display\n";
            cout<<"_____________________________________________\n";
            cout<<"Enter your choice: ";
            cin>>c;
            switch(c)
            {
                case 1: {
                            int key,val;
                            cout<<"Enter key: ";
                            cin>>key;
                            cout<<"Enter value: ";
                            cin>>val;
                            table.add(key,val);
                        }
                        break;
                case 2: table.display();
                        break;
                default:cout<<"Invalid\n";
            }
            cout<<"\nDo you wish to go again? ";
            cin>>op;
        }while(op=='y' || op=='Y');
        cout << "Test pass?(y/n): " << endl;
```

```
        cin>>r;
    }while(r=='n' || r=='N');
    cout<<"********************\n";
    cout<<"*   Thank You!    *\n";
    cout<<"********************\n";
    return 0;
}
```

Output Screenshot:-





Conclusion:-

We Successfully implemented the Heap datastructure.