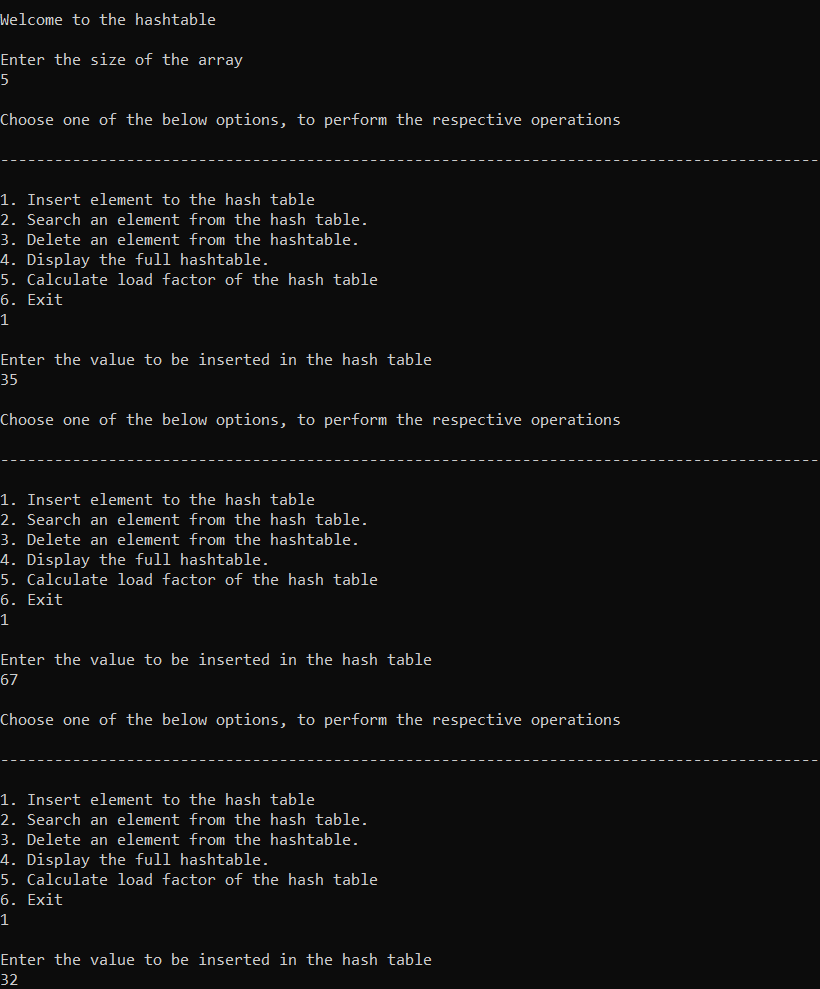
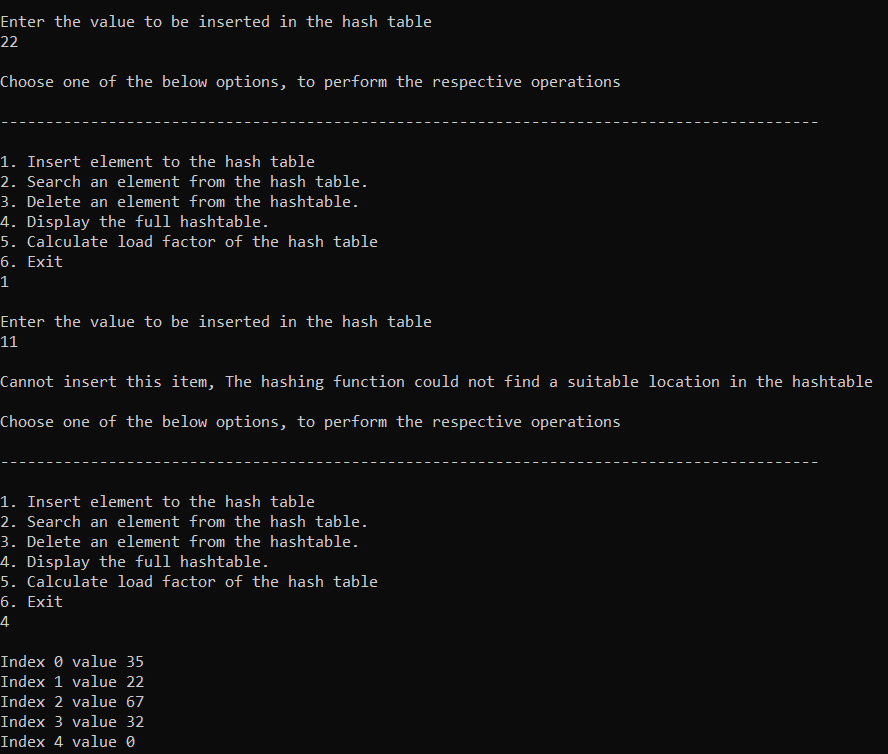
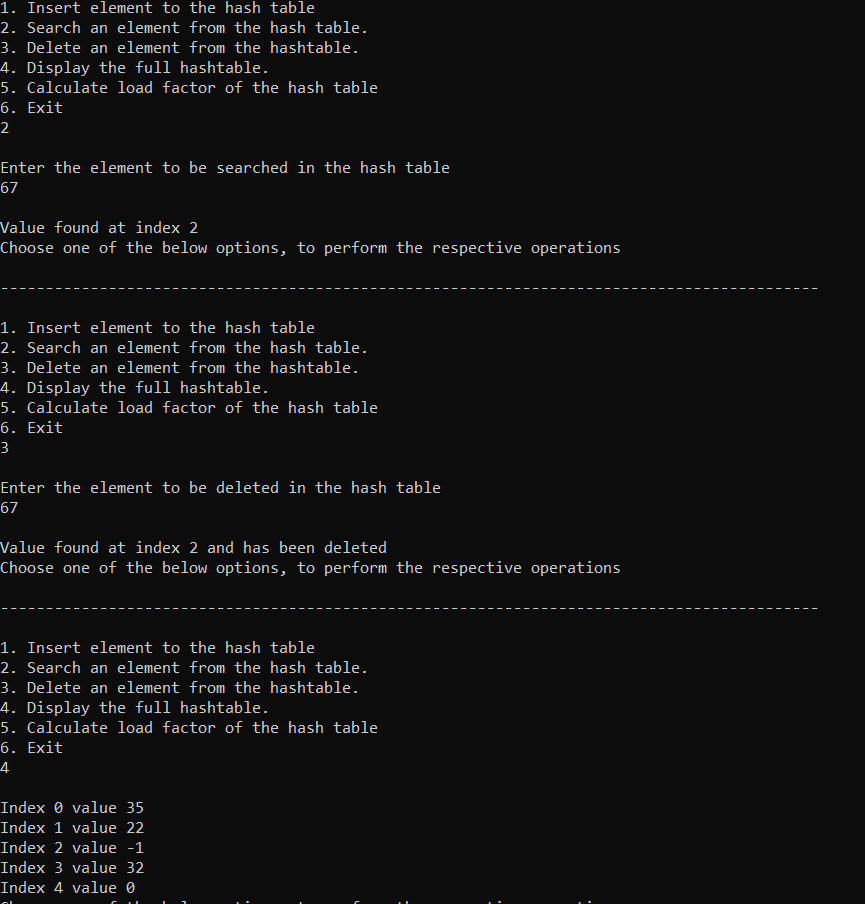
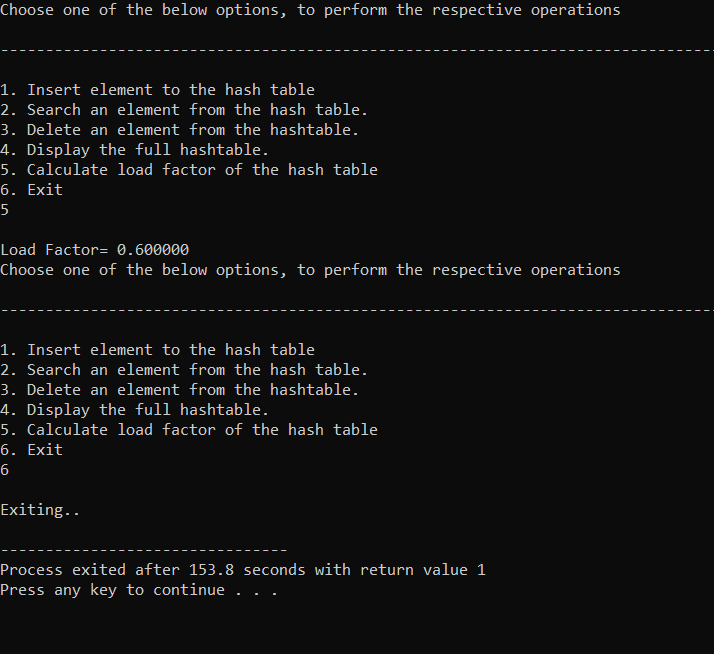
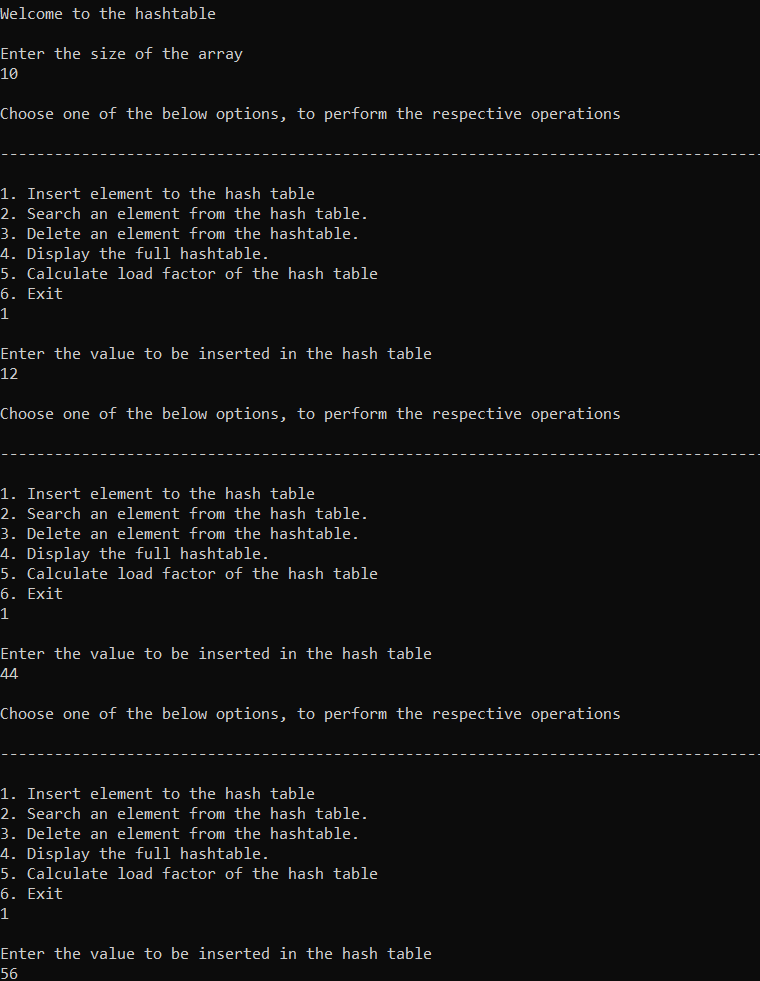
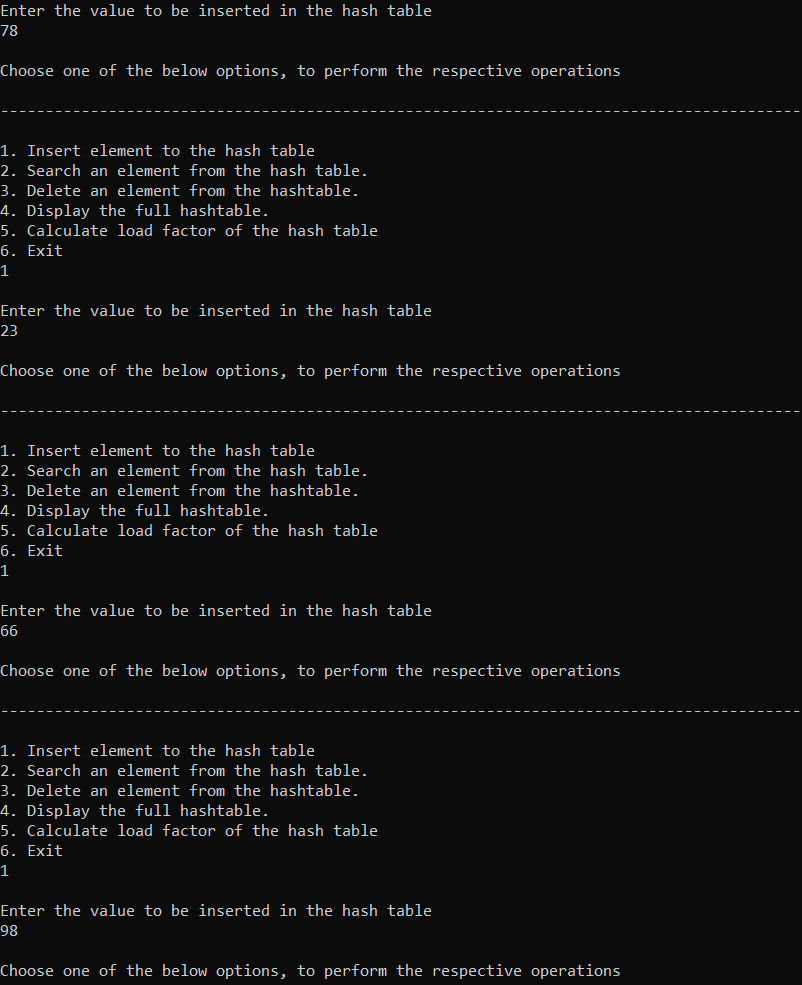
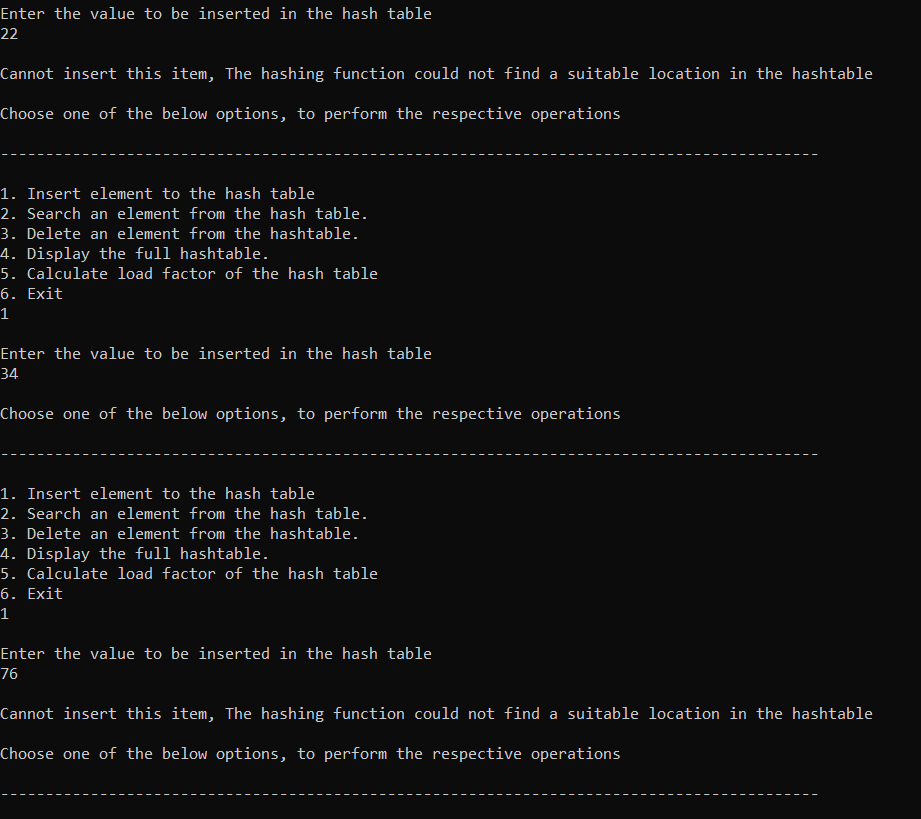
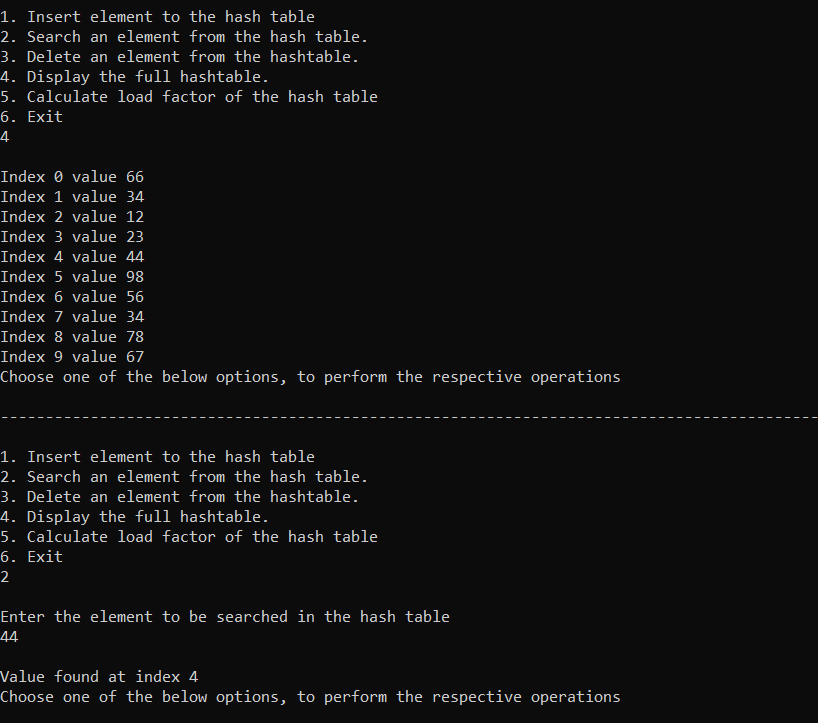
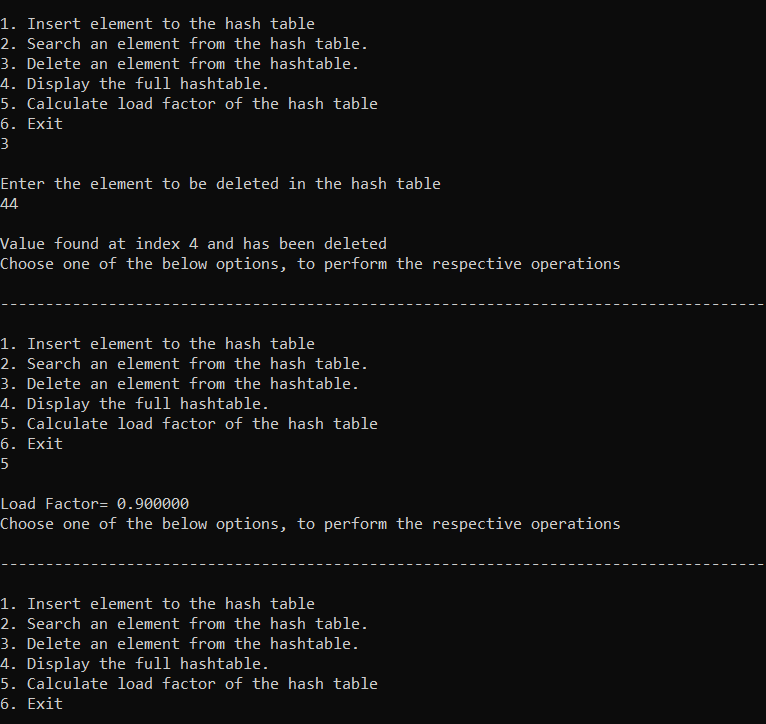
**Question 2: Design a HashTable (atleast size &gt;= 10) for Integers  
(Use Collision Resolution Technique : -Double Hashing)**

* The problem is solved by making use of singly circular linked list.
* Program is written on C with these basic functions.
* **(i)** **insertValue(int arr[]):** This funciton is used to insert the elements into the hash table one by one, until the hashtable is full.
* **(ii) loadfact(int arr[]):** This function calculates the load factor of the hashtable at any given point of time.
* **(iii) searchValue(int arr[]):** This function searches an element input by the user into the hashtable and provides the index of the valur if found, else prints not found.
* **(iv) display(int arr[]):**  This function is used to display my whole hashtable.
* **(v) deleteValue(int arr[]):** This function is used to delete an element from the hashtable if it exists in the hashtable and replaces it by -1.
* **(vi) main():** driver function, with menu driven program, supported by simple if and else statements, and use of **goto** statements whenever required.

**Execution Screenshots:  
** ****  

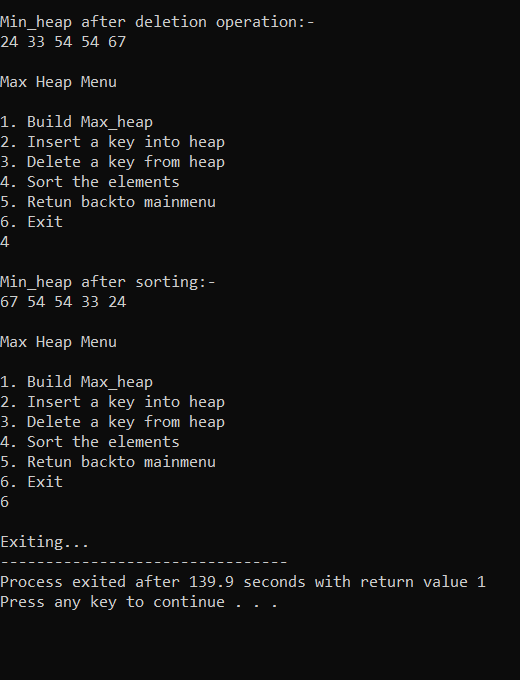
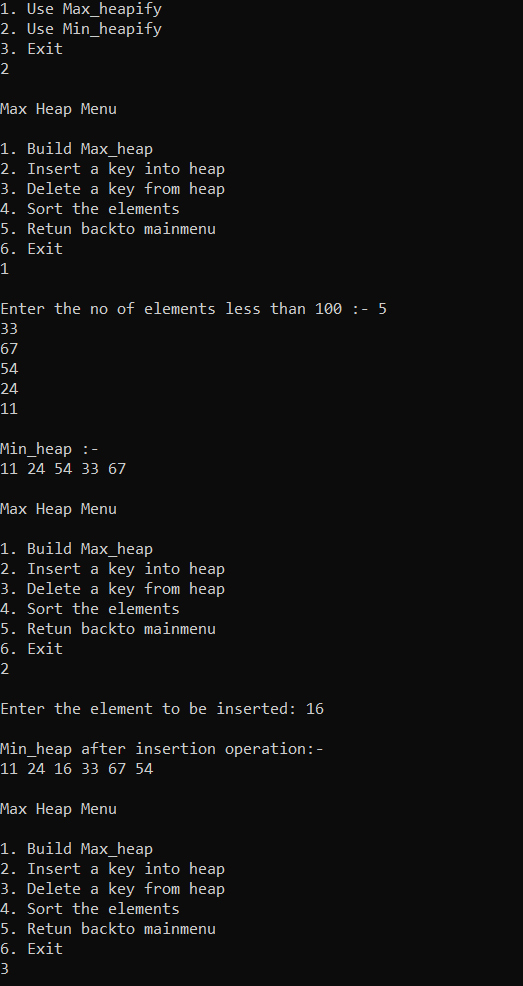
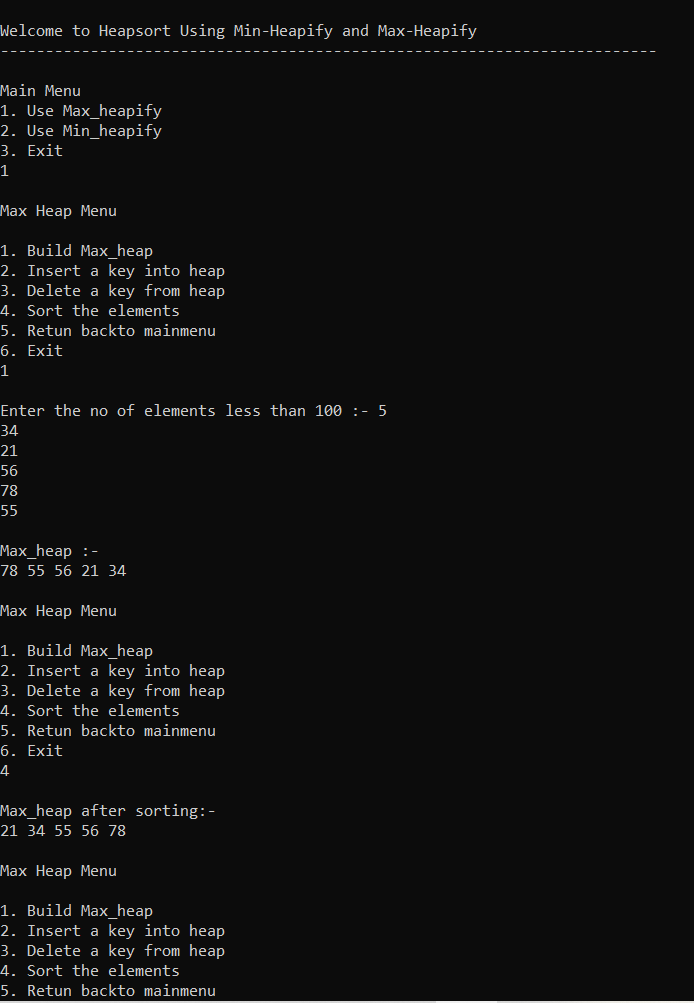
**Question 2: Design a HashTable (atleast size &gt;= 10) for Integers  
(Use Collision Resolution Technique : -Double Hashing)**

* The problem is solved by making use of singly circular linked list.
* Program is written on C with these basic functions.
* **(i) void getprime():** function to calculate the prime number pair that will be used for the hashing algorithm for implementing the double hashing algorithm
* **(ii)** **insertValue(int arr[]):** This funciton is used to insert the elements into the hash table one by one, until the hashtable is full.
* **(iii) loadfact(int arr[]):** This function calculates the load factor of the hashtable at any given point of time.
* **(iv) searchValue(int arr[]):** This function searches an element input by the user into the hashtable and provides the index of the valur if found, else prints not found.
* **(v) display(int arr[]):**  This function is used to display my whole hashtable.
* **(vi) deleteValue(int arr[]):** This function is used to delete an element from the hashtable if it exists in the hashtable and replaces it by -1.
* **(vii) main():** driver function, with menu driven program, supported by simple if and else statements, and use of **goto** statements whenever required.

**Execution Screenshots:** ****   

**Question 3: Implement Heap-sort Algorithm Using Min-Heapify and Max-Heapify Procedures.**

* The problem is solved by making use of global array for storing the heap
* Program is written on C with these basic functions.
* (i) **void max\_heapify(int arr[], int i, int heapsize):** function to construct the max heap, with the help of recursion building the tree from the bottom upto the root node.
* (ii) **int \*buildMaxHeap(int arr[], int items):** we create a heap and run maxheapify to make it maxheap.
* (iii) **void insert\_max(int element, int items):** function to insert the values in the max heap and then max heapify on every insertion.
* (iv) **void delete\_max(int arr[],int items):** function to delete the values(root node) in the max heap and then max heapify to satisfy max heap property.
* (v) **void heapsort\_max(int arr[],int items):** sorts the elements of the max heap.
* (vi) **void min\_heapify(int arr[], int i, int heapsize):** function to construct the min heap, with the help of recursion building the tree from the bottom upto the root node.
* (vii) **int \*buildMinHeap(int arr[], int items):** we create a heap and run minheapify to make it maxheap.
* (viii) **void insert\_min(int element, int items):** function to insert the values in the min heap and then min heapify on every insertion.
* (ix) **void delete\_min(int arr[],int items):** function to delete the values(root node) in the min heap and then min heapify to satisfy min heap property.
* (x) **void heapsort\_min(int arr[],int items):** sorts the elements of the min heap.
* (xi) **void printarr(int arr[], int n):** Function to print my array of heap.
* (xii) **main()**: main driver function that helps the user make use of all the functionalities provided by the program. It uses the global array for the heap. And user gets the chance to choose between making use of the min heap or the max heap as per requirement, to be chosen from the main menu in the start of the execution of the program.
* The input is checked against any exception, by using if and else to check for expected input.  
  If unexpected input is received, The program returns Invalid Input and exits.

**Execution Screenshots:  
**