Jadavpur University

Department of Electronics and Telecommunication Engineering,

Faculty of Engineering & Technology

DSA LAB REPORT

2nd Year First Semester 2020



Name : RAHUL SAHA

Roll: 001910701009

Group 1

IMPLEMENTATION OF CIRCULAR QUEUE AND PRIORITY QUEUE

Date Of Submission: 22/04/2021

Assignments - 05

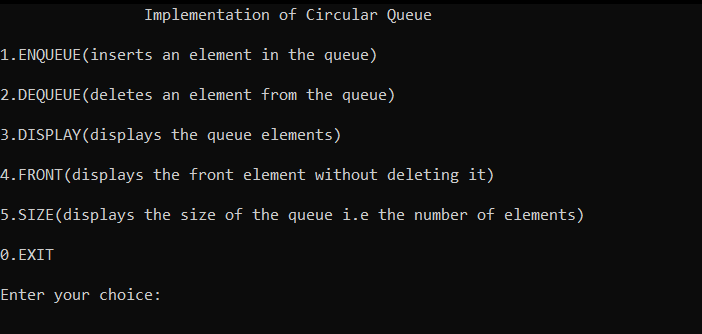
1. Implement a circular queue. Demonstrate that problems appearing in a linear queue can be circumvented in a circular queue.
2. Implement a min-priority queue. Your program should contain all the necessary functions like extract\_min() and others.
3. **Circular Queue**

Source Code:

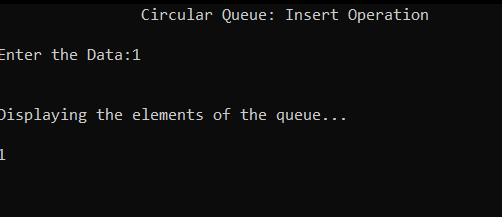
|  |
| --- |
| **#include<stdio.h> #include<conio.h> #include<stdlib.h> #define MAX 5  int cqueue[MAX];*//INITIALIZING THE QUEUE* int rear=-1; int front=-1;  *//FUNCTION TO DISPLAY THE QUEUE* void queue\_display(){  int i;  printf("\n\nDisplaying the elements of the queue...\n\n");  if(front==-1){  printf("\n\nThe queue is Empty");  getch();  return;  }  else{  for(i=front;i!=rear;i=(i+1)%MAX)**  **{**  ***//ITERATION UNTIL i IS EQUAL TO REAR*  printf("%d ",cqueue[i]);**  ***//PRINTS THE ELEMENT AT POSITION i OF THE ARRAY*  }  printf("%d ",cqueue[i]);*//PRINTS THE REAR ELEMENT*  }  getch(); }  *//FUNCTION TO INSERT AN ELEMENT INTO THE QUEUE* void enqueue(){  system("cls");  printf("\t\tCircular Queue: Insert Operation");  if((front==0 && rear==MAX-1)||(front==rear+1)){  printf("\n\nThe Queue is overflow");  getch();  return;  }  else{  if(front==-1){  front=0;  }  rear=(rear+1)%MAX;*//IF END OF ARRAY IS REACHED THEN ELEMENT WILL BE INSERTED IN THE ARRAY BEGINNING PROVIDED THERE IS SPACE HENCE CIRCULAR*  printf("\n\nEnter the Data:");  scanf("%d",&cqueue[rear]);  }  queue\_display();  return; }  *//DELETING AN ELEMENT FROM QUEUE* void dequeue(){  system("cls");  printf("\t\tCircular Queue: Deletion Operation");  if(front==-1){  printf("\n\nThe queue is Underflow");  getch();  return;  }  printf("\n\nThe deleted element from the Queue is: %d",cqueue[front]); *//DELETING THE FRONT ELEMENT*  if(front==rear)**  **{ *//IF ONLY ONE ELEMENT IS PRESENT RESETTING THE QUEUE*  front=-1;  rear=-1;  }  else{  front=(front+1)%MAX;   }  queue\_display();  return; }  *//DISPLAYS THE FRONT ELEMENT OF THE QUEUE* void queue\_front()**  **{  system("cls");  printf("\t\tCircular Queue: Displaying the front element of the Queue ");  if(front==-1){  printf("\n\nThe queue is underflow");  }  else{  printf("\n\nThe front element is: %d",cqueue[front]);**  ***//PRINTS THE FRONT ELEMENT*  }  getch();  return; }  *//DISPLAYS THE SIZE OF THE QUEUE* void queue\_size(){  system("cls");  int i,count=0;  printf("\t\tCircular Queue: Displaying the size of the Queue");  if(front==-1){  printf("\n\nThe size of the Queue is: 0");  }  else**  **{  for(i=front;i!=rear;i=(i+1)%MAX)**  **{  Count++; *//INCREMENT OF THE COUNT BY 1*   }  printf("\n\nThe size of the Queue is: %d",count+1);**  ***//PRINTS THE NUMBER OF ELEMENTS IN THE QUEUE*  }  getch();  return; } *//DRIVER FUNCTION WHICH HAS THE SWITCH CASE STRUCTURE* int main(){  int ch;  do{  system("cls");  printf("\t\tImplementation of Circular Queue");  printf("\n\n1.ENQUEUE(inserts an element in the queue)");  printf("\n\n2.DEQUEUE(deletes an element from the queue)");  printf("\n\n3.DISPLAY(displays the queue elements)");  printf("\n\n4.FRONT(displays the front element without deleting it)");  printf("\n\n5.SIZE(displays the size of the queue i.e the number of elements)");  printf("\n\n0.EXIT");  printf("\n\nEnter your choice: ");  scanf("%d",&ch);  switch(ch){  case 1: enqueue();  break;  case 2: dequeue();  break;  case 3: queue\_display();  break;  case 4: queue\_front();  break;  case 5: queue\_size();  break;  case 0: exit(0);    default: printf("Invalid Choice");  break;  }  }while(1); }** |
|  |
|  |

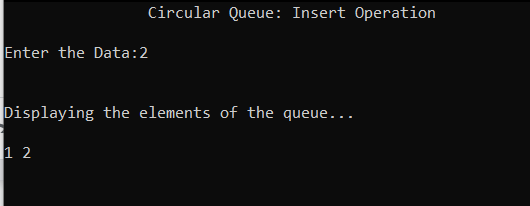
**OUTPUT(s) :**

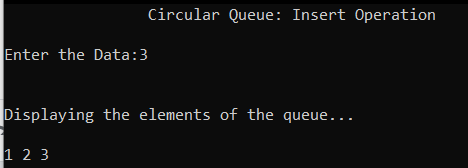
***Menu template of the user console:***

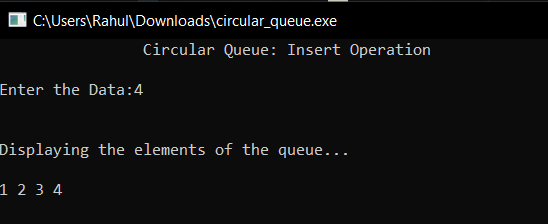
**

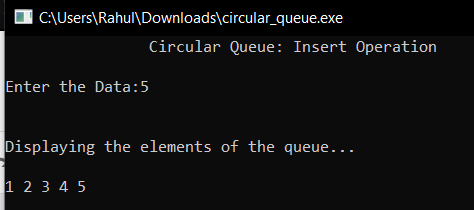
***User creates a circular queue containing elements 1,2,3,4,5 imputed in order.***



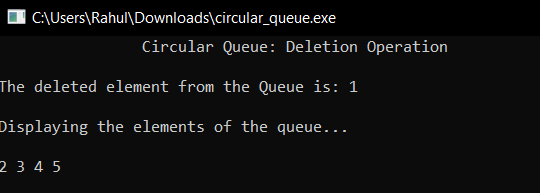


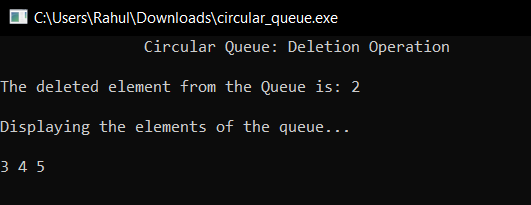




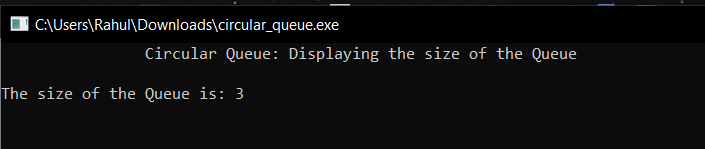


***Deletion ( In FIFO Manner ):***

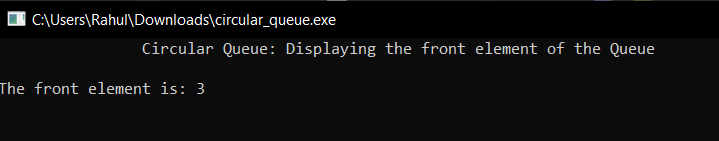




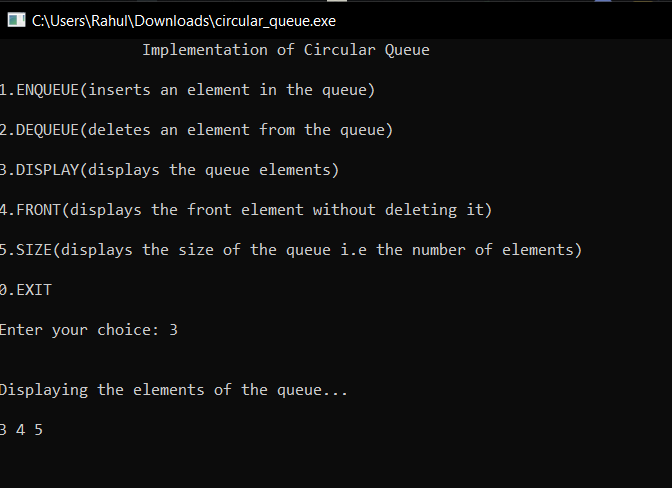
***At any point of the code execution, we can check the size of the queue***



***Let’s say user wants to display the front element of the queue without dequeue the item at the same time***



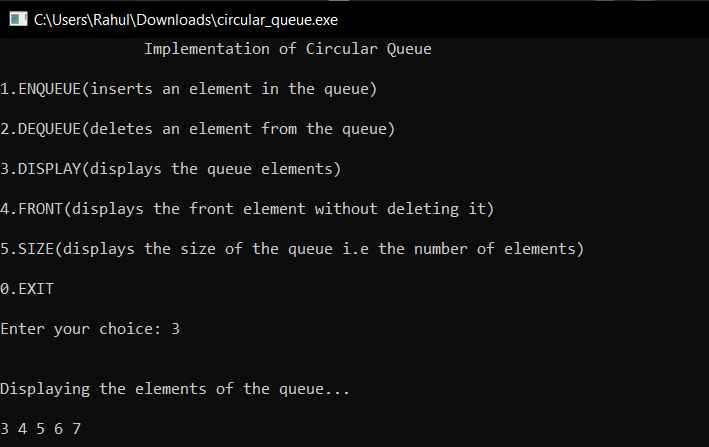
***Displaying the updated queue***



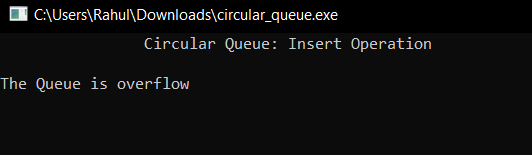
**Overflow Criteria :**

Since the size of the queue was 5 let’s fill the other 2 blank places with two other items

and display the list

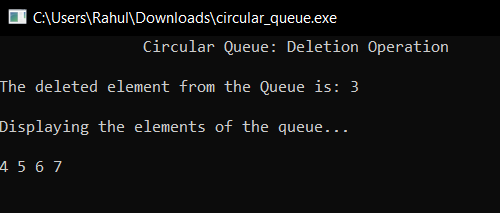


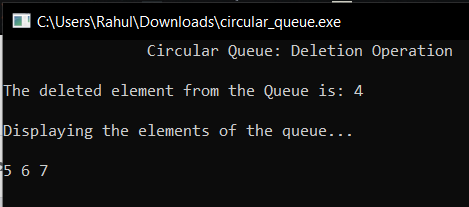
Now if I want to add a new element to the queue, it sends a message as Queue Overflow

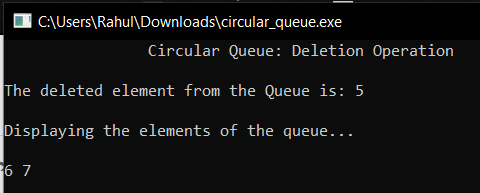


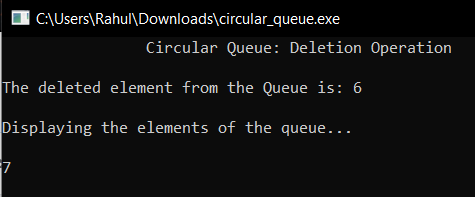
**Underflow Criterion:**

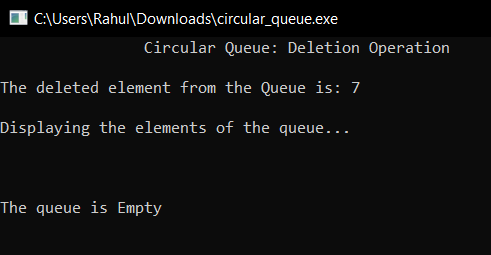
Suppose from the existing list we keep on removing the elements one by one. A point will come when there will be no elements on the list and if we still try to delete the same the queue will undergo through an Underflow Condition

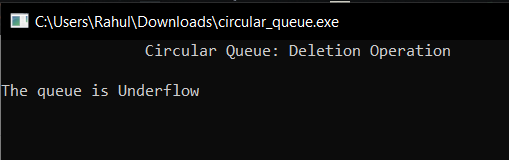












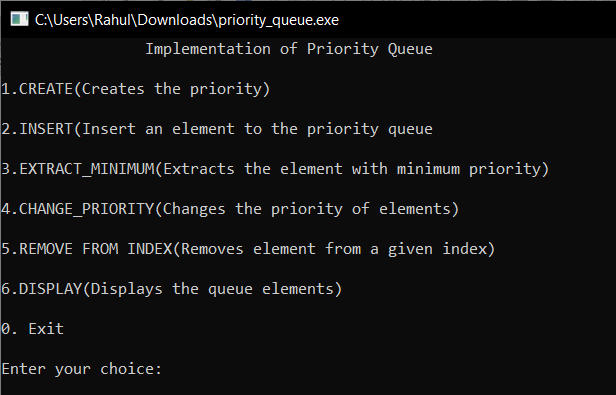
1. **Priority Queue**

Source Code:

|  |
| --- |
| **#include<stdio.h> #include<conio.h> #include<stdlib.h> #define MAX 100  int array[MAX]; int ptr=-1;  *//FUNCTION TO SWAP ELEMENTS* void element\_swap(int \*elem1,int \*elem2){  int temp;  temp=\*elem1;  \*elem1=\*elem2;  \*elem2=temp; }  *//FUNCTION TO FIND THE SIZE OF THE QUEUE* int size(){  int i,count=0;  if(ptr==-1){  return 0;  }  for(i=0;i<=ptr;i++){  count++;  }  return count; }  *//FUNCTION TO DISPLAY THE PRIORITY QUEUE* void display(){  int i;  if(ptr==-1){  printf("The queue is empty");  getch();  return;  }  for(i=0;i<=ptr;i++){  printf("%d ",array[i]);  }  getch();  return; }  *//FUNCTION WHICH RETURNS THE INDEX OF THE PARENT NODE OF A GIVEN NODE* int parent(int index){  return (index-1)/2; }  *//FUNCTION WHICH WHICH RETURNS THE INDEX OF THE RIGHT CHILD OF A GIVEN NODE* int right\_child(int index){  return(2\*index+2);  }  *//FUNCTION WHICH WHICH RETURNS THE INDEX OF THE LEFT CHILD OF A GIVEN NODE* int left\_child(int index){  return(2\*index+1); }  *//FUNCTION TO RETURN THE MINIMUM PRIORITY ELEMENT* int get\_min(){  return array[0]; }  *//FUNCTION TO SHIFT UP THE DOWN THE NODE MAINTAIN THE HEAP PROPERTY* void heap\_minimum(int index){  int index\_left\_child, index\_right\_child,minimum;  index\_left\_child=left\_child(index);  index\_right\_child=right\_child(index);   minimum=index;  if(index\_left\_child<=ptr and array[index\_left\_child]<array[minimum]){  minimum=index\_left\_child;  }   if(index\_right\_child<=ptr and array[index\_right\_child]<array[minimum]){   minimum=index\_right\_child;  }   if(minimum!=index){   element\_swap(&array[index],&array[minimum]);  heap\_minimum(minimum);   } }  *//FUNCTION TO SHIFT UP THE NODE TO MAINTAIN HEAP PROPERTY* void decrease\_val(int index){  while((index>0) && (array[parent(index)]>array[index])){  element\_swap(&array[index],&array[parent(index)]);  index=parent(index);  } }  *//INSERTS A NEW ELEMENT TO THE BINARY HEAP* void insert(int element){  ptr=ptr+1;  array[ptr]=element;  decrease\_val(ptr); }  *//FUNCTION TO CREATE A PRIORITY QUEUE* void create(){  system("cls");  printf("\t\tCreating a Priority Queue");  int number\_of\_elements,i;  printf("\n\nEnter the number of elements to be inserted: ");  scanf("%d",&number\_of\_elements);  for(i=0;i<number\_of\_elements;i++){  int element;  printf("\n\nEnter the data:");  scanf("%d",&element);  insert(element);  printf("\nThe priority Queue is...\n");  display();  } }  *//FUNCTION TO INSERT AN ELEMENT IN THE PRIORITY QUEUE* void op\_insert(){  system("cls");  printf("\t\tInsertion in Priority Queue");  int element;  printf("\n\nEnter the Element:");  scanf("%d",&element);  insert(element);  printf("\nThe priority Queue is...\n");  display(); }  *//FUNCTION TO EXTRACT THE MINIMUM PRIORITY ELEMENT* void extract\_min(){  system("cls");  printf("\t\tExtract The minimum element in the priority Queue");  if(ptr==-1){  printf("\n\nThe priority Queue is empty");  getch();  return;  }  printf("\n\nPriority Queue before Extracting Minimum...\n");  display();  printf("\n\nThe minimum element in the priority Queue: %d",array[0]);  array[0]=array[ptr];  ptr--;  heap\_minimum(0);  printf("\n\nPriority Queue after Extracting Minimum...\n");  display(); }  *//FUNCTION TO CHANGE PRIORITY OF AN ELEMENT* void change\_priority(){  system("cls");  printf("\t\t Changing Priority of Element in the Priority Queue");  int new\_priority,position,old\_priority;  printf("\n\nEnter the new priority: ");  scanf("%d",&new\_priority);  printf("\n\nEnter the index: ");  scanf("%d",&position);  if(position>size()-1){  printf("\n\nInvalid index");  getch();  return;  }  printf("\n\nThe Queue before change in priority..\n");  display();  old\_priority=array[position];  array[position]=new\_priority;  if(new\_priority>old\_priority){  heap\_minimum(position);  }  else{  decrease\_val(position);  }  printf("\n\nThe Queue after change in priority\n");  display(); }  *//FUNCTION TO REMOVE AN ELEMENT FROM A GIVEN INDEX IN THE PRIORITY QUEUE* void remove\_index(){  system("cls");  printf("\t\t Remove Element from a given index in the Priority Queue");  int position;  printf("\n\nEnter the Index: ");  scanf("%d",&position);  if(position>size()-1){  printf("\n\nInvalid index");  getch();  return;  }  printf("The Element at index %d is: %d",position,array[position]);  printf("\n\nPriority Queue before Removing the element from Index...\n");  display();  array[position]=get\_min()-2;  decrease\_val(position);  array[0]=array[ptr];  ptr--;  heap\_minimum(0);  printf("\n\nPriority Queue after Removing the element from Index...\n");  display(); }  *//THE MAIN DRIVER FUNCTION* int main(){  int ch;  do{  system("cls");  printf("\t\tImplementation of Priority Queue");  printf("\n\n1.CREATE(Creates the priority)");  printf("\n\n2.INSERT(Insert an element to the priority queue");  printf("\n\n3.EXTRACT\_MINIMUM(Extracts the element with minimum priority)");  printf("\n\n4.CHANGE\_PRIORITY(Changes the priority of elements)");  printf("\n\n5.REMOVE FROM INDEX(Removes element from a given index)");  printf("\n\n6.DISPLAY(Displays the queue elements)");  printf("\n\n0. Exit");  printf("\n\nEnter your choice: ");  scanf("%d",&ch);  switch(ch){  case 1: create();  break;  case 2: op\_insert();  break;  case 3: extract\_min();  break;  case 4: change\_priority();  break;  case 5: remove\_index();  break;  case 6: display();  break;  case 0: exit(0);  default: printf("Invalid Choice");  break;  }  }while(1); }** |

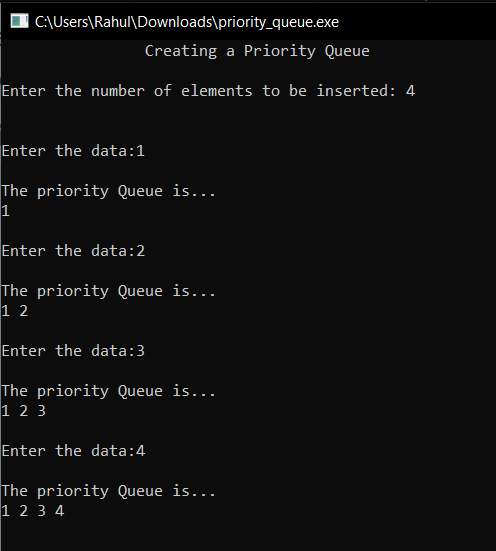
**OUTPUT(s):**

***The console menu***

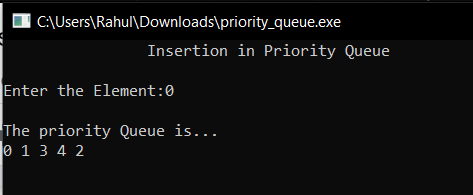


***First we create a minpriority queue containing 4 elements***

Let 1,2,3,4 are the values that are to be added into the queue

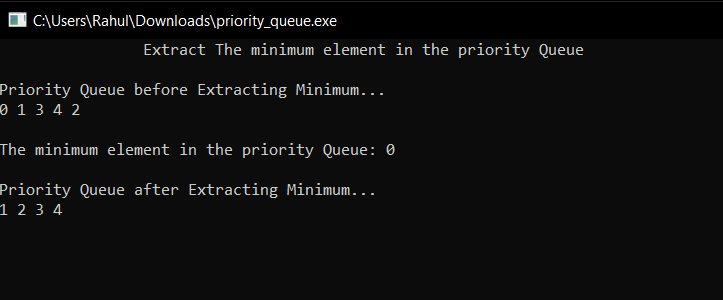


***Inserting another element in the priority queue***

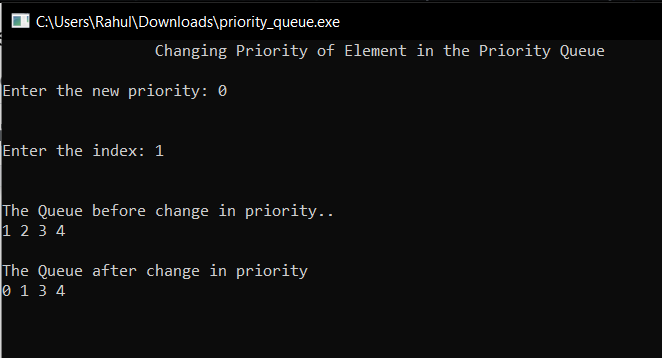


Since, 0 is the smallest of all existing values so according to the min priority rule, the element is placed at the front of the queue.

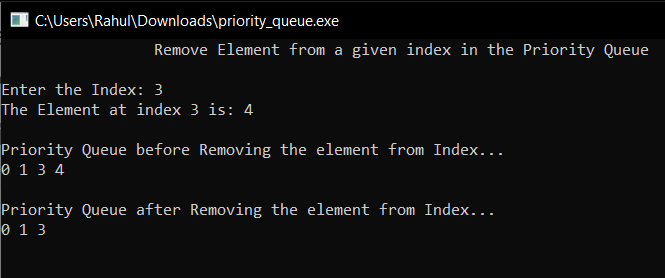
***Extracting the element with minimum priority***



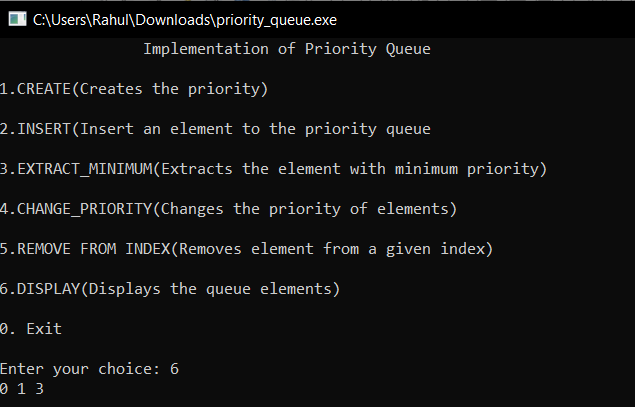
***The user can also change the priority at a given index, so the new input becomes the minimum priority***



***The user can also delete the element from the specified index which is according to the priority***



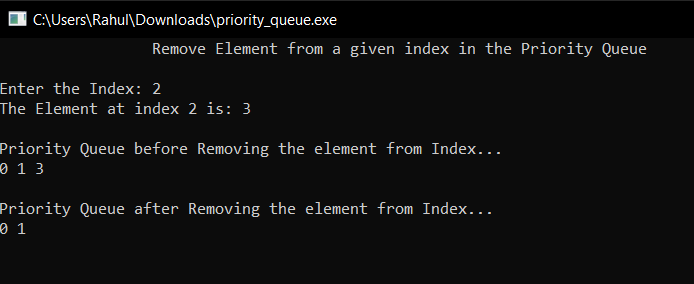
***Atlast, display the updated queue***

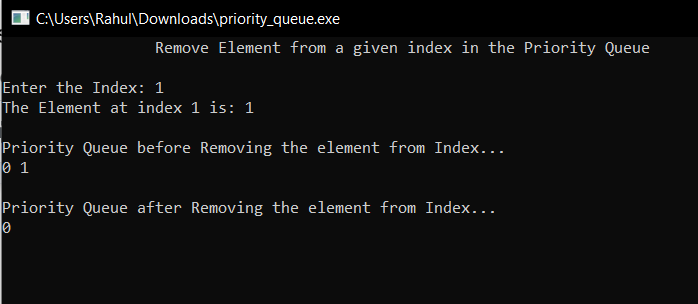


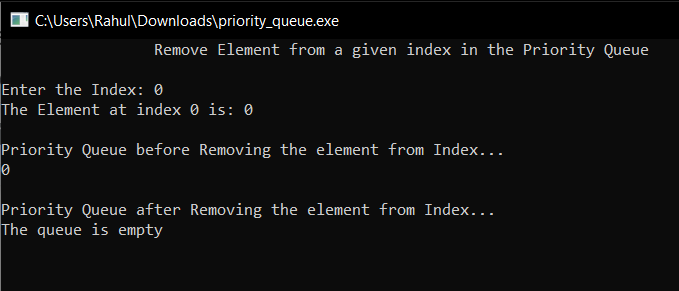
**Extreme cases**

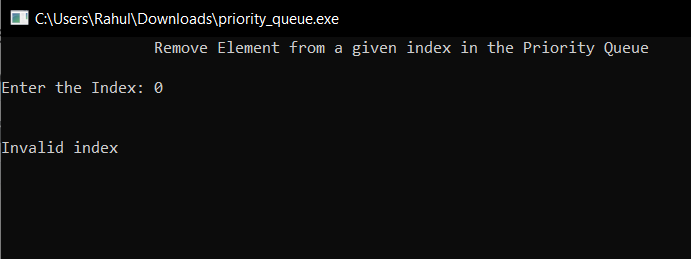
**Underflow Detection:**

If we keep on removing elements from the priority queue a point will come when there is no elements to delete, the condition is the under flow condition which is properly demonstrated in my program as follows:









Also, if user tries to access the index outside the range of priority queue, same response is obtained

