 **Bharatiya Vidya Bhavan’s**

**SARDAR PATEL INSTITUTE OF TECHNOLOGY**

(Autonomous Institute Affiliated to University of Mumbai)

Munshi Nagar, Andheri (W), Mumbai – 400 058.

COMPS Department

|  |  |
| --- | --- |
| **Experiment** | 1 |
| **Aim** | To Implement Circular Queue |
| **Name** | David Daniels |
| **UID** | 202330038 |
| **Class** | Div -A |
| **Batch** | C |
| **Date of Submission** | 24-8-24 |

|  |  |
| --- | --- |
| **Theory** | A Queue Data Structure is used for storing and managing data in a specific order similar to a line at a ticket counter. It follows the principle of First in, First out (FIFO), where the first element added to the queue is the first one to be removed.  A Circular Queue is an extended version of a normal queue where the last element of the queue is connected to the first element of the queue forming a circle.  In a normal Queue, we can insert elements until queue becomes full. But once queue becomes full, we can not insert the next element even if there is a space in front of queue.  Basic Operations of Queue Data Structure   * Enqueue (Insert): Adds an element to the rear of the queue. * Dequeue (Delete): Removes and returns the element from the front of the queue. * Empty: Checks if the queue is empty. * Full: Checks if the queue is full.   Queues are versatile data structures with many real world applications  1. Scheduling Tasks in CPU: All instructions in CPU first go to buffered queue in CPU to ensure pipelining.  2. Buffering (I/O and Priniting): Queues are used to line up for ex print jobs and user keyboards presses  3. Data Streaming: Queues are used in video streaming to load small chunk of video beforehand and to ensure correct processing of data packets  4. Breadth-First Search (BFS):  In graph algorithms, BFS uses a queue to explore nodes level by level. This is useful for finding the shortest path in an unweighted graph.  5. Call Center Systems: Queues are used to manage call systems , making sure that the first person to call is being served  6. Round-Robin Scheduling: Queue is used in multi-threading to make sure that each thread gets fair share of CPU time  To implement a queue in C, we must make a structure containing 4 elemenents  1) int front  2) int rear  3) int size  3) (datatype)\* arr  **Enqueue Operation**: In enqueue operation, the  element is inserted at the rear position, and the rear is updated (circular increment for circular queue and r++ for normal). If the queue is full, a message indicating this is displayed.  **Dequeue Operation**: In dequeue operation , the element at the front is removed, and the front is updated (circular increment for circular queue and r++ for normal) If the queue is empty, a message indicating this is displayed. |
| **Algorithm** | 1. Initialisation of queue 2. Make the struct object using malloc 3. Ask size of queue from user and allocate the array in heap using malloc(size \* sizeof(datatype)) 4. Set rear and front to 0 5. Is\_full(queue q) 6. If (rear+1)%size == front, then queue must be full 7. Is\_empty(queue q) 8. If rear==front , then queue must be empty 9. Enqueue(queue q , int data) 10. Check is\_full 11. If element is available then circular increment rear and put data in arr[rear] of array 12. Dequeue(queue q) 13. Check is\_empty 14. If element is available then circular increment front and return data of arr[front] and make arr[front]=0 afterwards 15. Display (queue q) 16. Run a for loop from queue front+1 till queue rear; By circular incrementing the number i 17. Print arr[i] |
| **Problem Solving** |  |
| **Program (Code)** | #include <stdio.h>  #include <stdbool.h>  #include <stdlib.h>  struct circular\_queue  {  int \* arr;  int f,r,size;  };  bool is\_full(struct circular\_queue \* q)  {  if ((q->r+1)%q->size==q->f)  {  return true;  }  return false;  }  bool is\_empty(struct circular\_queue \* q)  {  if (q->f==q->r)  {  return true;  }  return false;  }  void enqueue(struct circular\_queue \* q, int data)  {  if (is\_full(q))  {  printf("FAIL\n");  return;  }  q->r=(q->r+1)%q->size;  q->arr[q->r]=data;  return;  }  int dequeue(struct circular\_queue \* q)  {  int num=0;  if (is\_empty(q))  {  printf("EMPTY\n");  return -1;  }  q->f=(q->f+1)%q->size;  num = q->arr[q->f];  q->arr[q->f]=0;  return num;  }  void display(struct circular\_queue \* q)  {  printf("Queue is: \n");  for (int i = q->f+1; i != q->r; i=(i+1)%q->size)  {  printf("%d\n",q->arr[i]);  }  printf("%d",q->arr[q->r]);  printf("\n\n");  }  void take\_input(struct circular\_queue \* q)  {  int choice=0;  int d=0;  printf("Enter 1 to enqueue\n 2 to dequeue\n 3 to print\n 0 to quit\n\n");  while (true)  {  printf("Enter Choice: ");  scanf("%d",&choice);  printf("\n\n");  if (choice==0)  {  printf("Goodbye\n");  return;  }    switch (choice)  {  case 1:  {  printf("Enter number to enqueue: ");  scanf("%d",&d);  printf("\n");  enqueue(q,d);  break;  }  case 2:  {  d=dequeue(q);  printf("Dequeue is: %d",d);  printf("\n");  break;  }  case 3:  {  display(q);  break;  }    default:  {  printf("USER IS AN IDIOT");  break;  }    }  }  }  int main(int argc, char const \*argv[])  {  int s;  printf("Enter size of queue: ");  scanf("%d",&s);  printf("\n\n ");  struct circular\_queue \* q = (struct circular\_queue \*) malloc(sizeof(struct circular\_queue));  q->arr=(int \*)malloc(sizeof(int) \* s);  q->f=0;  q->r=0;  q->size=s;  int deq =0;  take\_input(q);  return 0;  }  **PETROL PROBLEM**  #include <stdio.h>  #include <stdlib.h>  #include <stdbool.h>  struct node  {      int petrol\_at\_station;      int distance\_to\_next\_node;      struct node\* next;  };  struct node\* insert\_at\_end(struct node \* head, int pet , int dist)  {      struct node \* new\_end = malloc(sizeof(struct node));      struct node \* ptr =head;      new\_end->petrol\_at\_station=pet;      new\_end->distance\_to\_next\_node=dist;      while(ptr->next!= head)      {          ptr = ptr->next;      }      ptr->next=new\_end;      new\_end->next=head;      return head;  }  // all get insered at end anyways  void insert\_all\_pumps(struct node \* head)  {      int no=0;      printf("Enter number of pumps: ");      scanf("%d",&no);      printf("\n\n");      int temp\_pet=0;      int temp\_dist=0;      for (int i = 0; i < no; i++)      {          printf("Enter fuel you found at pump number %d: ",i+1);          scanf("%d",&temp\_pet);          printf("\n");          printf("Enter distance to next pump: ");          scanf("%d",&temp\_dist);          printf("\n");          head=insert\_at\_end(head,temp\_pet,temp\_dist);      }  }  int length\_of\_cl(struct node \*head)  {      int length=0;      struct node \* ptr = head;      do      {          length++;          ptr = ptr->next;      }      while (ptr!=head);      printf("\n\n");      free(ptr);      return length;  }  void check\_for\_fuel(struct node \* head,struct node \* ptr\_current,int pump\_no)  {      int current\_fuel=0;      int distance\_to\_travel=0;      struct node \* ptr = ptr\_current;      do      {          current\_fuel+=ptr->petrol\_at\_station;          distance\_to\_travel+=ptr->distance\_to\_next\_node;          if (current\_fuel-distance\_to\_travel<=0)          {              printf("Trip not possible for pump no %d",pump\_no);              free(ptr);              return;          }          ptr=ptr->next;        } while (ptr->next!=ptr\_current);      if (ptr->next==ptr\_current && ptr->petrol\_at\_station-ptr->distance\_to\_next\_node<=0)      {          printf("Trip possible: for pump no %d",pump\_no);          free(ptr);      }      else      {         printf("Trip not possible for pump no %d",pump\_no);          free(ptr);          return;      }      }  void simulate (struct node \* head)  {      int pump\_no=0;      struct node \* ptr=head;      for (int i = 0; i < length\_of\_cl(head)-1; i++)      {          ptr=ptr->next;          check\_for\_fuel(head,ptr,pump\_no);          pump\_no++;      }      }  void free\_list(struct node\* head)  {      struct node \* ptr = head;      do      {          ptr=head;          head = head->next;          free(ptr);      }      while (ptr!=head);      free(head);  }  int main(int argc, char const \*argv[])  {      struct node \* head = (struct node \*)malloc(sizeof(struct node));      struct node \* intial\_location = (struct node \*)malloc(sizeof(struct node));      head->next=intial\_location;      //to make circular      printf("Enter fuel you found at 1st pump: ");      scanf("%d",&intial\_location->petrol\_at\_station);      printf("\n");      printf("Enter distance to next pump: ");      scanf("%d",&intial\_location->distance\_to\_next\_node);      printf("\n");      intial\_location->next=head;      insert\_all\_pumps(head);      simulate(head);      free\_list(head);      return 0;  } |
| **Output** | **PETROL PROBLEM** |
| **Conclusion** | This I have learned to implement a circular queue and its methods using c language  I have also implemented the petrol problem using circular linked lists in c |