**SVKM’s NMIMS**

**Mukesh Patel School of Technology Management & Engineering**

**Computer Engineering Department**

Program: B.Tech Integrated. Sem V

**Course: Basic Data Structures**

**Experiment No.06**

PART A

(PART A : TO BE REFFERED BY STUDENTS)

**A.1 Aim:**

To study and implement concept of circular queue data structure.

TASK 1: Write a Program to implement a circular Queue

TASK 2: Call Center Management System

Scenario: A call center receives calls from customers, and each call is placed in a waiting queue if all agents are busy. The system has a limited number of slots in the queue to hold these calls. When an agent becomes available, the next call in line is connected. If the queue is full and a new call arrives, the system can either reject the call or replace the oldest one.

Problem:

* The call queue has a fixed size due to system limitations.
* Calls must be handled in the order they arrive (FIFO).
* The system must efficiently manage the fixed number of slots in the queue.

**A.2 Prerequisite:**

1. Knowledge of different operations performed on Queue data structure

2. Fundamental concepts of C\C++.

**A.3 Outcome:**

**After successful completion of this experiment students will be able to**

1. Identify the steps of queue data structure selection.
2. Implement Queue data structure to solve the given problem
3. Enlist the applications of stack data structure.
4. Differentiate between different types of Queue.

**A.4 Theory:**

**A.4.1. Introduction of Queue**

**A.5 Procedure/Algorithm:**

**A.5.1:**

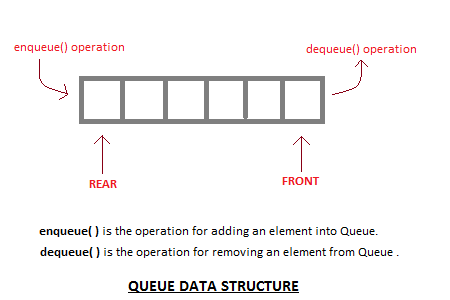
Queue is a linear data structure in which items are inserted at one end called ‘Rear’ and deleted from the other end called ‘Front’. Queues are based on the First-In-First-Out (FIFO) principle that means the data item that is inserted first in the queue is also the first one to be removed from the queue.

A queue or FIFO (first in, first out) is an abstract data type that serves as a collection of elements, with two principal operations: enqueue, the process of adding an element to the collection.(The element is added from the rear side) and dequeue, the process of removing the first element that was added. (The element is removed from the front side). It can be implemented by using both array and linked list.

**Representation of Queue:**

1. Static representation using arrays.
2. Dynamic representation using linked list.

Array representation of a queue needs two indices: FRONT and REAR.



Array declaration in C++ as follows: int queue[maxsize]

**Conditions to be assumed:**

1. FRONT =REAR or FRONT=REAR=-1, if the queue empty
2. Whenever an element is deleted from the queue, FRONT=FRONT+1
3. Whenever an element is inserted into the queue, REAR=REAR+1

**Operations on Queue:**

|  |  |  |
| --- | --- | --- |
| Operation | Description | Restriction |
| Create Queue | It creates a new empty queue. This operation must be carried out in order to make the queue logically accessible | - |
| Qinsert | It inserts a new data item at the rear of the queue | Queue must not be full |
| Qdelete | It deletes and then return the data item at the front of the queue. | Queue must not be empty |
| Queue full | It returns true if the queue is full. Otherwise it returns false. | - |
| Queue Empty | It returns true if the queue is empty. Otherwise it returns false. |  |

PART B

(PART B : TO BE COMPLETED BY STUDENTS)

***(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case the there is no Black board access available)***

|  |  |
| --- | --- |
| Roll No. C146 | Name: Manan Gandhi |
| Class : D | Batch : D1 |
| Date of Experiment: | Date of Submission |
| Grade : | Time of Submission: |
| Date of Grading: |  |

**B.1 Software Code written by student: (Task 1)**

***(Paste your code completed during the 2 hours of practical in the lab here)***

**Task1:**

*// Circular Queue*

*#include* <iostream>

*using* *namespace* std;

*const* int *n* *=* *10*;

int *front* *=* *-1*, *rear* *=* *-1*;

struct *QueueElement*

{

int *value*;

};

*QueueElement* *queue*[*n*];

void *enqueue*(int);

void *dequeue*();

void *display*();

int *main*()

{

int *choice* *=* *0*;

bool *running* *=* *true*;

int *value*;

*while* (*running*)

{

*cout* *<<* *endl*

*<<* "1. Insert" *<<* *endl*

*<<* "2. Delete" *<<* *endl*

*<<* "3. Display" *<<* *endl*

*<<* "0. Exit" *<<* *endl*

*<<* "Enter choice: ";

*cin* *>>* *choice*;

*switch* (*choice*)

{

*case* *1*:

*cout* *<<* "Enter value: ";

*cin* *>>* *value*;

*enqueue*(*value*);

*break*;

*case* *2*:

*dequeue*();

*break*;

*case* *3*:

*display*();

*break*;

*case* *0*:

*running* *=* *false*;

*break*;

*default*:

*cout* *<<* "Invalid choice" *<<* *endl*;

*break*;

}

}

*return* *0*;

}

void *enqueue*(int *value*)

{

*if* ((*front* *==* *0* *&&* *rear* *==* *n* *-* *1*) *||* (*rear* *==* (*front* *-* *1*) *%* (*n* *-* *1*)))

{

*cout* *<<* "Overflow" *<<* *endl*;

*return*;

}

*if* (*front* *==* *-1*)

{

*front* *=* *0*;

*rear* *=* *0*;

}

*else* *if* (*rear* *==* *n* *-* *1* *&&* *front* *!=* *0*)

{

*rear* *=* *0*;

}

*else*

{

*rear++*;

}

*queue*[*rear*].*value* *=* *value*;

}

void *dequeue*()

{

*if* (*front* *==* *-1*)

{

*cout* *<<* "Underflow" *<<* *endl*;

*return*;

}

int *value* *=* *queue*[*front*].*value*;

*if* (*front* *==* *rear*)

{

*front* *=* *-1*;

*rear* *=* *-1*;

}

*else* *if* (*front* *==* *n* *-* *1*)

{

*front* *=* *0*;

}

*else*

{

*front++*;

}

*cout* *<<* "Deleted value: " *<<* *value* *<<* *endl*;

}

void *display*()

{

*if* (*front* *==* *-1* *&&* *rear* *==* *-1*)

{

*cout* *<<* "Empty" *<<* *endl*;

*return*;

}

*if* (*front* *<=* *rear*)

{

*for* (int *i* *=* *front*; *i* *<=* *rear*; *i++*)

{

*cout* *<<* *queue*[*i*].*value* *<<* "*\t*";

}

}

*else*

{

*for* (int *i* *=* *front*; *i* *<* *n*; *i++*)

{

*cout* *<<* *queue*[*i*].*value* *<<* "*\t*";

}

*for* (int *i* *=* *0*; *i* *<=* *rear*; *i++*)

{

*cout* *<<* *queue*[*i*].*value* *<<* "*\t*";

}

}

*cout* *<<* *endl*;

}

**Task2:**

*/\**

*Call Center Management System*

*Scenario: A call center receives calls from customers, and each call is placed in a waiting queue if all agents are busy. The system has a limited number of slots in the queue to hold these calls. When an agent becomes available, the next call in line is connected. If the queue is full and a new call arrives, the system can either reject the call or replace the oldest one.*

*Problem:*

*\* The call queue has a fixed size due to system limitations.*

*\* Calls must be handled in the order they arrive (FIFO).*

*\* The system must efficiently manage the fixed number of slots in the queue.*

*\*/*

*#include* <iostream>

*using* *namespace* std;

*const* int *n* *=* *10*;

int *front* *=* *-1*, *rear* *=* *-1*;

struct *QueueElement*

{

int *value*;

};

*QueueElement* *queue*[*n*];

void *enqueue*(int);

void *dequeue*();

void *display*();

int *main*()

{

int *choice* *=* *0*;

bool *running* *=* *true*;

int *value*;

*while* (*running*)

{

*cout* *<<* *endl*

*<<* "1. Insert" *<<* *endl*

*<<* "2. Delete" *<<* *endl*

*<<* "3. Display" *<<* *endl*

*<<* "0. Exit" *<<* *endl*

*<<* "Enter choice: ";

*cin* *>>* *choice*;

*switch* (*choice*)

{

*case* *1*:

*cout* *<<* "Enter value: ";

*cin* *>>* *value*;

*enqueue*(*value*);

*break*;

*case* *2*:

*dequeue*();

*break*;

*case* *3*:

*display*();

*break*;

*case* *0*:

*running* *=* *false*;

*break*;

*default*:

*cout* *<<* "Invalid choice" *<<* *endl*;

*break*;

}

}

*return* *0*;

}

void *enqueue*(int *value*)

{

*if* ((*front* *==* *0* *&&* *rear* *==* *n* *-* *1*) *||* (*rear* *==* (*front* *-* *1*) *%* (*n* *-* *1*)))

{

*cout* *<<* "Queue is full. Replace the oldest call? (y/n): ";

char *choice*;

*cin* *>>* *choice*;

*if* (*choice* *==* 'n')

{

*return*;

}

*front* *=* (*front* *+* *1*) *%* *n*;

}

*else* *if* (*front* *==* *-1*)

{

*front* *=* *0*;

*rear* *=* *0*;

}

*else* *if* (*rear* *==* *n* *-* *1* *&&* *front* *!=* *0*)

{

*rear* *=* *0*;

}

*else*

{

*rear++*;

}

*queue*[*rear*].*value* *=* *value*;

}

void *dequeue*()

{

*if* (*front* *==* *-1*)

{

*cout* *<<* "Queue is empty" *<<* *endl*;

*return*;

}

*cout* *<<* "Removed call with value: " *<<* *queue*[*front*].*value* *<<* *endl*;

*if* (*front* *==* *rear*)

{

*front* *=* *-1*;

*rear* *=* *-1*;

}

*else*

{

*front* *=* (*front* *+* *1*) *%* *n*;

}

}

void *display*()

{

*if* (*front* *==* *-1*)

{

*cout* *<<* "Queue is empty" *<<* *endl*;

*return*;

}

*cout* *<<* "Calls in queue: ";

*if* (*rear* *>=* *front*)

{

*for* (int *i* *=* *front*; *i* *<=* *rear*; *i++*)

{

*cout* *<<* *queue*[*i*].*value* *<<* " ";

}

}

*else*

{

*for* (int *i* *=* *front*; *i* *<* *n*; *i++*)

{

*cout* *<<* *queue*[*i*].*value* *<<* " ";

}

*for* (int *i* *=* *0*; *i* *<=* *rear*; *i++*)

{

*cout* *<<* *queue*[*i*].*value* *<<* " ";

}

}

*cout* *<<* *endl*;

}

**B.2 Input and Output: (Task 1)**

***(Paste your program input and output in following format, If there is error then paste the specific error in the output part. In case of error with due permission of the faculty extension can be given to submit the error free code with output in due course of time. Students will be graded accordingly.)***

**Task1:**

**B.3 Observations and learning [w.r.t. all tasks]:**

***(Students are expected to comment on the output obtained with clear observations and learning for each task/ sub part assigned)***

**B.4 Conclusion:**

*(****Students must write the conclusion as per the attainment of individual outcome listed above and learning/observation noted in section B.3)***

**B.5 Question of Curiosity**

***(To be answered by student based on the practical performed and learning/observations)***

**Discuss the limitation of linear queue and elaborate any three applications of queue.**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*