Batch: B2 Roll No.: 16010121110
Experiment / assignment / tutorial No.
Grade: AA / AB / BB / BC / CC / CD /DD
Signature of the Staff In-charge with date

Title: Implementation of Basic operations on queue for the assigned application using Array and Linked List- Create, Insert, Delete, Destroy

Objective: To implement Basic Operations on Queue i.e. Create, Push, Pop, Destroy for the given application

Expected Outcome of Experiment:

CO	Outcome
1	Explain the different data structures used in problem solving

Books/ Journals/ Websites referred:

1. *Fundamentals Of Data Structures In C* – Ellis Horowitz, Satraj Sahni, Susan Anderson-Fred
2. *An Introduction to data structures with applications* – Jean Paul Tremblay, Paul G. Sorenson
3. *Data Structures A Pseudo Approach with C* – Richard F. Gilberg & Behrouz A. Forouzan
4.

Abstract:+ queue ADT

Definition:

An ordered collection of homogenous data items

Where elements are added at rear and removed from
the front end

Operations:

Create an empty queue

check if it is empty and/or full

Enqueue:
add an element at the rear

Dequeue:
remove the element in front

Destroy : remove all the elements one by one and
destroy the data structure

(Define Queue, enlist queue operations).
The Queue ADT: Value definition

Abstract typedef QueueType(ElementType ele)

Condition: none

Queue ADT: Operator definition

1. Abstract QueueType CreateQueue()

Precondition: none

Postcondition: Empty Queue is created

2. Abstract QueueType Enqueue(QueueType Queue,

ElementType Element)

Precondition: Queue not full or NotFull(Queue)= True

Postcondition: Queue = Queue' + Element at the rear

Or Queue = original queue with new Element at the rear

Queue ADT: Operator definition

3. Abstract ElementType dequeue(QueueType Queue)

Precondition: Queue not empty or NotEmpty(Queue)= True

Postcondition: Dequeue= element at the front

Queue= Queue - Element at the front

Or Queue = original queue with front element deleted

4. Abstract DestroyQueue(QueueType Queue)

Precondition: Queue not empty or NotEmpty(Queue)= True

Postcondition: Element from the Queue are removed one by one

starting from front to rear.

NotEmpty(Queue)= False

Queue ADT: Operator definition

5. Abstract Boolean NotFull(QueueType Queue)

Precondition: none

Postcondition: NotFull(Queue)= true if Queue is not full

NotFull(Queue)= False if Queue is full.

6. Abstract Boolean NotEmpty(QueueType Queue)

Precondition: none

Postcondition: NotEmpty(Queue)= true if queue is not empty

NotEmpty(Queue)= False if Queue is empty.

List 5 Real Life applications of Queue:

1) **Queue of people waiting at busstop**
2) **Queue of work to be done. (priority queue)**
3) **Queue of objects in conveyer belt**
4) **Queue of action sequences in microprocessor.**
5) **Interrupt (priority queue)**

Define and explain various types of queue with suitable diagram and their application(s):

1) **Prioriy queue - used to keep track of priorities of the objects.**
2) **Doubly ended queue- insertions and deletions can be done at any end.**
3) **Circular queue- queue used to maximize space and circular algorithm simplicity.**

Algorithm for Queue operations using array/Linked list : (Write only the algorithm for the assigned type)

PUSH(HEAD, DATA, PRIORITY):

Step 1: Create new node with DATA and PRIORITY
Step 2: Check if HEAD has lower priority. If true follow Steps 3-4 and end. Else goto Step 5.
Step 3: NEW -> NEXT = HEAD
Step 4: HEAD = NEW
Step 5: Set TEMP to head of the list
Step 6: While TEMP -> NEXT != NULL and TEMP -> NEXT -> PRIORITY > PRIORITY
Step 7: TEMP = TEMP -> NEXT
[END OF LOOP]
Step 8: NEW -> NEXT = TEMP -> NEXT
Step 9: TEMP -> NEXT = NEW
Step 10: End
POP(HEAD):

Step 1: Set the head of the list to the next node in the list. HEAD = HEAD -> NEXT.
Step 2: Free the node at the head of the list
Step 3: End
PEEK(HEAD):

Step 1: Return HEAD -> DATA
Step 2: End

Ref geeksforgeeks.org/priority-queue-using-linked-list/

Implementation Details:

1) **Mention the application assigned to you and explain how you implemented the solution using the assigned type of Queue.**

Used priority queue. Used OOP concepts for making classes for node and nodelist. Made a separate metho for sort for sprting the pripority queue. (insertion sort).

Program source code:

```
/*********************************************************************
*********
priority queue.

*********************************************************************
*********/
#include <iostream>

using namespace std;
class queueElement
{
public:
 int queueElementPriority;
 string queueElementStringValue;
   queueElement ()
 {
   //default constructor
 }
 queueElement (string queueElementStringValue, int queueElementPriority)
 {
   this->queueElementStringValue = queueElementStringValue;

   this->queueElementPriority = queueElementPriority;
```

```
  }
};

class queue
{
public:

  queueElement * queueArray;
  int top;
  int bottom;
  int size;
public:
    queue ()
  {
    //default constructor
  }
  queue (int size)
  {
    queueArray = new queueElement[size];
    this->size = size;
    top = 0;
    bottom = 0;
  }
  void add (string queueElementStringValue, int queueElementPriority)
  {
```

```
  if ((top) >= size)
   {
       cout << "Overflow" << "\n";

   }
  else
   {

       queueElement *elementToBeAdded = new queueElement (queueElementStringValue, queueElementPriority);

       queueArray[top] = *elementToBeAdded;
       top++;
       sort ();
   }
 }

 void sort ()
 {
  if (top > 1)
   {
       for (int i = top - 1; i > 0; i--)
        {
                if (queueArray[i].queueElementPriority < queueArray[i - 1].queueElementPriority)
            {
```

```
                //swap;
                queueElement temp = queueArray[i];
                queueArray[i] = queueArray[i - 1];
                queueArray[i - 1] = temp;
            }
          }
    }
}
 queueElement pop ()
{
  if (bottom == top)
    {
        cout << "underflow" << "\n";
        top = 0;
        bottom = 0;

    }
  else
    {

        bottom++;
        return queueArray[bottom - 1];     //alternative to temp method of removal
    }
}
 void display ()
```

```
 {
   for (int i = bottom; i < top; i++)
     {
          cout << queueArray[i].queueElementStringValue << " , ";
     }
   cout << "\n";
 }
};

int
main ()
{
  queue *myBookQueue = new queue (4);
  myBookQueue->add ("enid blyton", 4);
  myBookQueue->add ("Harry potter", 2);
  myBookQueue->add ("sherlock homes", 1);
  myBookQueue->add ("pokemon", 3);



  myBookQueue->display ();


  return 0;
}
```

Output Screenshots:

sherlock homes , Harry potter , pokemon , enid blyton ,

Applications of Queue in computer science:

Application of Queue in Data Structure

Managing requests on a single shared resource such as CPU scheduling and disk scheduling.

Handling hardware or real-time systems interrupts.

Handling website traffic.

Routers and switches in networking.

Maintaining the playlist in media players.

Ref-
https://www.naukri.com/learning/articles/queue-data-structure-types-implementation-applications/

Conclusion:-
Thus we have successfully understood the working behind stacks and understood how to implement stack using various methods like static and dynamic implementation. We applied stacked into the priority queue and designed algorithms and code for priority queue for sorting books.