|  |  |
| --- | --- |
| **Data Structures & Algorithms**  Diploma in CSF, IT  Year 2/3 (2020/21) Semester 4/6 | **Week 5** |
| **1-2 Hours** |
| **Tutorial 5 – Queues** | |

1. After the following statements execute, what are the contents of the queue?

num1 = 5;

num2 = 1;

num3 = 4;

q.enqueue(num2);

q.enqueue(num3);

q.dequeue()

q.enqueue(num1 – num2)

q.dequeue(num1)

q.dequeue(num2)

cout << num2 << “ “ << num1 << “ “ << num3 << endl;

2. The specification of the Queue ADT implemented using Circular Array is given below.

|  |
| --- |
| *// Queue.h - Specification of Queue ADT (implemented using Circular Array)*  #pragma once  #include<string>  #include<iostream>  using namespace std;  typedef int ItemType;  class Queue  {  private:  ItemType items[MAX\_SIZE];  int front;  int back;  bool isFull;  public:  // constructor  Queue();  *// enqueue (add) item at back of the queue*  bool enqueue(ItemType item);  *// dequeue (remove) item from front of the queue*  bool dequeue();  *// retrieve (get) and dequeue item from front of the queue*  bool dequeue(ItemType& item);  *// retrieve (get) item from front of queue*  void getFront(ItemType& item);  *// check if the queue is empty*  bool isEmpty();  }; |

Implement the following operations of the Queue ADT

1. Queue();

Queue::Queue(){

front = 0;

back = MAX\_SIZE -1;

isFull = false;

}

1. bool enqueue(ItemType item);

bool Queue::enqueue(ItemType item){

if (!isFull){

if (back+1%MAX\_SIZE == front)){

isFull = true;

}

back = (back+1) % MAX\_SIZE;

items[back] = item;

return true;

}

Else{

cout << “Queue full ,cannot enqueue” << endl;

return false;

}

}

1. bool dequeue(ItemType& item);

bool Queue::dequeue(ItemType& Item){

if (!isEmpty()){

item = items[front]

front = (front+1) % MAX\_SIZE;

isFull = false;

return true;

} else {

cout << “Queue Empty, Cannot dequeue“ << endl;

return false;

}

}

1. bool isEmpty();

bool Queue::isEmpty(){

return (!isFull() && (back+1)% MAX\_SIZE == front)  
}

3. The specification of the Queue ADT implemented using Pointers is given below.

|  |
| --- |
| *// Queue.h - Specification of Queue ADT (implemented using Pointers)*  #pragma once  #include<string>  #include<iostream>  using namespace std;  typedef string ItemType;  class Queue  {  private:  struct Node  {  ItemType item; // item  Node \*next; // pointer pointing to next item  };  Node \*frontNode; // point to the front node  Node \*backNode; // point to the back node  public:  // constructor  Queue();  ~Queue();  *// enqueue (add) item at back of the queue*  bool enqueue(ItemType item);  *// dequeue (remove) item from front of the queue*  bool dequeue();  *// retrieve (get) and dequeue item from front of the queue*  bool dequeue(ItemType& item);  *// retrieve (get) item from front of queue*  void getFront(ItemType& item);  *// check if the queue is empty*  bool isEmpty();  *// returns no of elements in queue*  int getNoOfElements();  }; |

1. Write a client function getLastElement()that returns the last element of the queue while leaving the queue unchanged. This function can call any of the functions in the Queue ADT. Function prototype is given as follows:

bool getLastElement(Queue& q, ItemType& item);

bool Queue::getLastElement(Queue& q, ItemType& item){

Queue temp;

ItemType tempItem;

While (!q.isEmpty(){

q.dequeue(tempItem);

temp.enqueue(tempItem);

}

item = tempItem;

While (!temp.isEmpty()){

temp.dequeue(tempItem);

q.enqueue(tempItem);

}

return true;

} else {

cout << “Queue is empty , cannot get element” << endl;

return false

}

1. Implement a function getNoOfElements() for ADT Queue that returns the no of elements in the queue. Function prototype is given as follows:

int getNoOfElements();

int Queue::getNoOfElements(){

int count = 0;

if (!isEmpty()){

Queue q;

ItemType tempItem;

while(!isEmpty()){

dequeue(tempItem);

q.enqueue(temp(Item);

count++;

}

while (!q.isEmpty()){

enqueue(tempItem);

q.dequeue(tempItem);

}

}

return count;

}

1. Imagine the case if the pointer- based implementation of ADT Queue does not come with a backNode. Discuss how this will affect the efficiency in terms of computation time for enqueue and dequeue functions.

Without a backNode, as the size of the element increases, the higher inefficiency for enqueueing. Whenever enqueue is required, it has to calculate where the end of the node is in order to enqueue the value in. Dequeue is not affected as it is removing the frontNode’s item