

**CSE-225  
 Assignment-3**

Name: Md. Najmus Sakib.

ID: 1813345642.

Section: 14.

# Ans. To The Question: Queue implementation using linked list

This program is the implementation of queue using linked list. The queue operations “**IsEmpty”**, “**enQueue”**, and “**DeQueue”** is implemented. This program allow user to add item in the queue and remove item from the queue and print the removed item. This program have a function “**PrintQueue**” to print all the items in the queue.

**Here is the code of “main.cpp” driver file**

#include <iostream>

#include "quetype.cpp" // .cpp file added to the main driver file

using namespace std;

int main()

{

QueType<int> link; // object creation

int number;

int item,item\_delete;

int i=0;

cout<<"Enter the number of items you want to add: ";

cin>>number; // the number of items we want to add will store in this variable

cout<<"\nEnter the items: ";

while(i< number) // loop to take user inputs and add them into the queue list until the number

{

cin>>item;

link.enQueue(item); // every item will be enqueued into the queue list

i++;

}

cout<<"\nRemoved item is: "<<endl;

link.DeQueue(item\_delete); // it will remove the first item of the queue list

if(link.IsEmpty()==true) // finding if the list is empty or not

{

cout<<"\nThe list is empty"<<endl; // if the list is empty then show this as message.

}

else

cout<<"\nThe list is not empty"<<endl; // if the list is not empty then show this as message.

cout<<"\nThe list of values are: "<<endl;

link.PrintQueeue(); // print the items in the queue list

return 0;

}

**Here is the “quetype.h” header file-**

#ifndef QUETYPE\_H\_INCLUDED

#define QUETYPE\_H\_INCLUDED

class FullQueue

{};

class EmptyQueue

{};

template <class ItemType> // standard template library

class QueType

{

struct NodeType //structure

{

ItemType info; // holds the item or data

NodeType\* next; //next pointer

};

public: // public variables and methods

QueType(); // constructor

void MakeEmpty(); // delete all the items and make the queue empty

void enQueue(ItemType); // method of add item

void DeQueue(ItemType&); // method of removing an item

bool IsEmpty(); // method of finding if the list of queue is empty

bool IsFull(); // method of finding if the list of queue is full

void PrintQueeue(); // print method to print the list of queue

private: // private variables and methods

NodeType \*front, \*rear; /\*private pointer where front pointer holds the address of the starting item and rear pointer holds the address of the last item\*/

};

#endif // QUETYPE\_H\_INCLUDED

**Here is the code of “quetype.cpp” file where all the functions are implemented-**

#include "quetype.h"

#include <iostream>

using namespace std;

template <class ItemType>

QueType<ItemType>::QueType()

{

front = NULL; // front set to NULL by default when object is created

rear = NULL; // rear set to NULL by default when object is created

}

template <class ItemType>

bool QueType<ItemType>::IsEmpty()

{

return (front == NULL); // to make the list of queue empty, the front is set to NULL since front pointer holds the first item's address of the list

}

template<class ItemType>

bool QueType<ItemType>::IsFull()

{

NodeType\* location;

try // try and catch block used to handle exception

{

location = new NodeType;

delete location;

return false;

}

catch(bad\_alloc& exception)

{

return true;

}

}

template <class ItemType>

void QueType<ItemType>::enQueue(ItemType newItem)

{

if (IsFull())

throw FullQueue(); // throw exception if the queue list is full

else

{

NodeType\* newNode;

newNode = new NodeType; // new node pointer created

newNode->info = newItem; //new node's info will have the new item to be stored in the queue list

newNode->next = NULL; // next node is set to null

if (rear == NULL) // check if the rear is NULL or not

front = newNode; //if it is NULL then front node pointer indicated the new node pointer

else

rear->next = newNode; //otherwise rear's next node indicated the new node pointer

rear = newNode; // at last rear indicates ne node pointer

}

}

template <class ItemType>

void QueType<ItemType>::DeQueue(ItemType& item)

{

if (IsEmpty())

throw EmptyQueue(); // throw exception if the queue list doesn't have item

else

{

NodeType\* tempPtr; // temporary pointer created.

tempPtr = front; // temporary pointer indicates the front pointer

item = front->info; // item holds the item which will delete

cout<<item<<endl; // prints the item which will be deleted

front = front->next; // front is updated by its next node

if (front == NULL) // check if front pointer is NULL or not

rear = NULL; // if null then rear pointer is set to NULL too

delete tempPtr; // at last delete temporary pointer since it wont be used afterwards

}

}

template <class ItemType>

void QueType<ItemType>::PrintQueeue()

{

if (IsEmpty())

throw EmptyQueue(); // throw exception if the the the queue list doesn't have item

else{

while(!IsEmpty()) // iterate till the list is empty.

{

NodeType\* tempPtr; // new pointer created

tempPtr = front; // temporary pointer indicates

cout<<tempPtr->info<<endl; // it prints the item whose are in the queue list

front = front->next; // front indicates its next node

delete tempPtr; // deleted temporary pointer since it wont be used afterwards

}

}

}

**Sample input output:-**

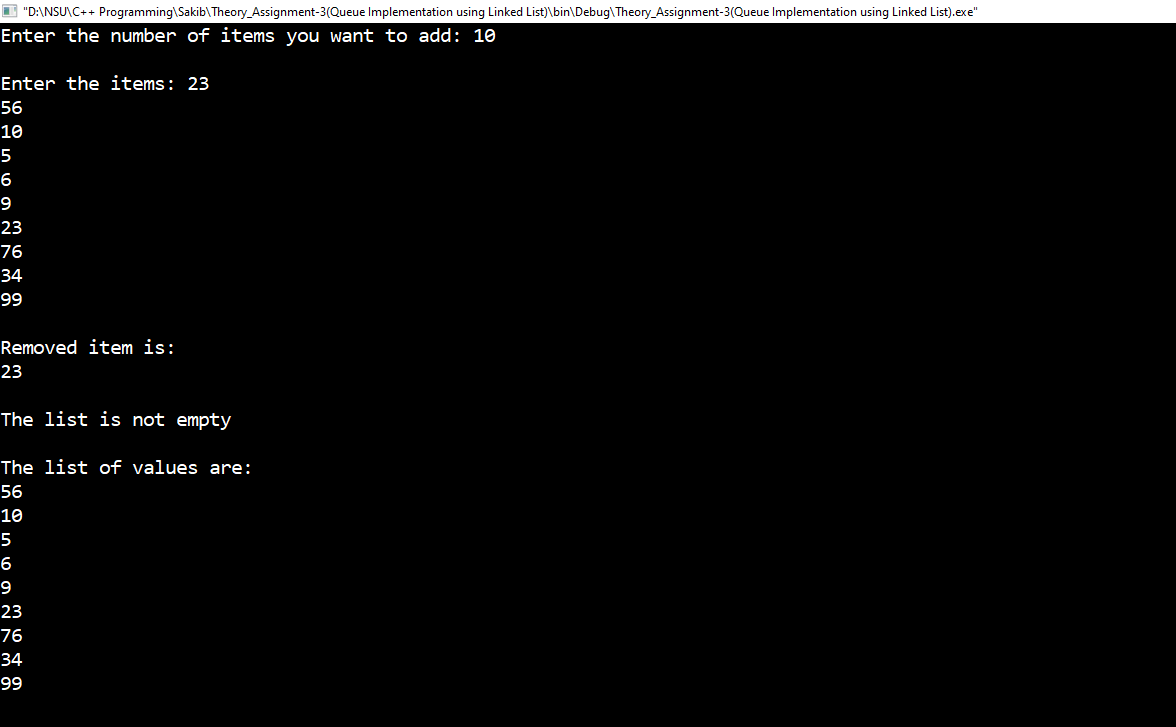


Figure 1: Sample input and output of the program.