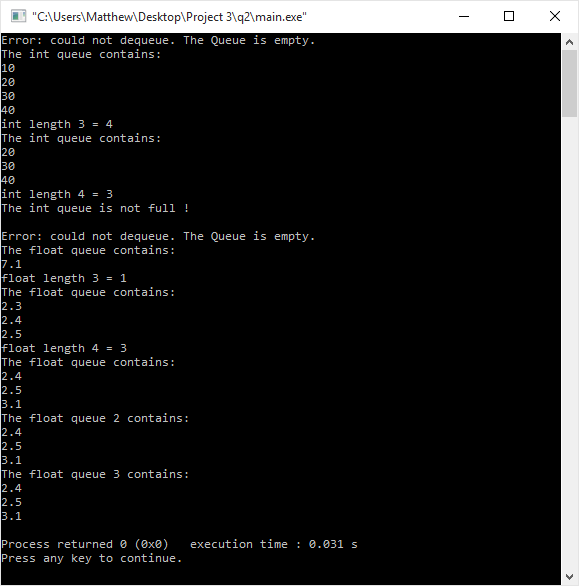
Matthew Wills

Machine Type: AMD Phenom II X4 970 Processor 3.50 GHz

Compiler used: mingw

**Question 2**



**Source code:**

**main.cpp**

#include "queue.h"

int main()

{

Queue<int>IntQueue;

int x;

IntQueue.MakeEmpty();

IntQueue.Dequeue(x);

IntQueue.Enqueue(10);

IntQueue.Enqueue(20);

IntQueue.Enqueue(30);

IntQueue.Enqueue(40);

cout << "The int queue contains: " << endl;

IntQueue.Print();

cout << "int length 3 = " << IntQueue.length() << endl;

IntQueue.Dequeue(x);

cout << "The int queue contains: " << endl;

IntQueue.Print();

cout << "int length 4 = " << IntQueue.length() << endl;

if(IntQueue.IsFull() == false)

cout << "The int queue is not full !" << endl;

else

cout << "The int queue is full !" << endl;

cout <<endl;

Queue<float>FloatQueue;

float y;

FloatQueue.MakeEmpty();

FloatQueue.Dequeue(y);

FloatQueue.Enqueue(7.1);

cout << "The float queue contains: " << endl;

FloatQueue.Print();

cout << "float length 3 = " << FloatQueue.length() << endl;

FloatQueue.Enqueue(2.3);

FloatQueue.Enqueue(2.4);

FloatQueue.Enqueue(2.5);

FloatQueue.Dequeue(y);

cout << "The float queue contains: " << endl;

FloatQueue.Print();

cout << "float length 4 = " << FloatQueue.length() << endl;

FloatQueue.Enqueue(3.1);

FloatQueue.Dequeue(y);

cout << "The float queue contains: " << endl;

FloatQueue.Print();

Queue<float> FloatQueue2 = FloatQueue;

cout << "The float queue 2 contains: " << endl;

FloatQueue2.Print();

FloatQueue.MakeEmpty();

cout << "The float queue 3 contains: " << endl;

FloatQueue2.Print();

}

**queue.h**

#include <cstddef>

#include <iostream>

#define MAX\_ITEM 5

template<class ItemType>

struct NodeType

{

ItemType info;

NodeType\* next;

};

using namespace std;

template<class ItemType>

class Queue

{

private:

NodeType<ItemType>\* head; // It points to the head of a singly-linked list

NodeType<ItemType>\* rear; // It points to the end of a singly-linked list

public:

Queue( ); // default constructor: Queue is created and empty

Queue(const Queue<ItemType> &x); // copy constructor: implicitly called

// for a deep copy

void MakeEmpty(); // Queue is made empty; you should deallocate all

// the nodes of the linked list

bool IsEmpty( ); // test if the queue is empty

bool IsFull( ); // test if the queue is full; assume MAXITEM=5

int length( ); // return the number of elements in the queue

void Print( ); // print the value of all elements in the queue in the sequence

// from the head to rear

void Enqueue(ItemType x); // insert x to the rear of the queue

// Precondition: the queue is not full

void Dequeue(ItemType &x); // delete the element from the head of the queue

// Precondition: the queue is not empty

~Queue(); // Destructor: memory for the dynamic array needs to be deallocated

};

template <class ItemType> Queue<ItemType>::Queue()

{

head = rear = NULL;

}

template <class ItemType> Queue<ItemType>::Queue(const Queue<ItemType> &x)

{

//if the Queue being passed is empty, use the default constructor instead

if(x.rear == NULL)

{

Queue();

return;

}

head = new NodeType<ItemType>;

NodeType<ItemType>\* newNode = head;

for(NodeType<ItemType>\* iter = x.head; iter != NULL; iter = iter->next)

{

newNode->info = iter->info;

if (iter->next != NULL)

{

newNode->next = new NodeType<ItemType>;

newNode = newNode->next;

}

}

newNode->next = NULL;

rear = newNode;

}

template <class ItemType> void Queue<ItemType>::Enqueue(ItemType x)

{

NodeType<ItemType>\* temp = new NodeType<ItemType>;

temp->info = x;

temp->next = NULL;

//if the queue was empty to begin with, the first item added also becomes the head

if(rear == NULL)

{

head = temp;

}

else

{

//the new node will be linked to the current rear of the queue

rear->next = temp;

}

//the new rear of the queue is updated to be the new node

rear = temp;

}

template <class ItemType> void Queue<ItemType>::Dequeue(ItemType &x)

{

//display an error message if the Queue is empty

if(rear == NULL)

{

cout << "Error: could not dequeue. The Queue is empty." << endl;

return;

}

x = head->info;

//create a temp variable to store the location of the next item in queue

NodeType<ItemType>\* temp = head->next;

//delete the head of the queue

delete head;

//set the new head of the queue to the next item in queue

head = temp;

}

template <class ItemType> int Queue<ItemType>::length()

{

int length = 0;

//iterate through the Queue object until it points to NULL

for(NodeType<ItemType>\* iter = head; iter != NULL; iter = iter->next)

{

length++;

}

return length;

}

template <class ItemType> void Queue<ItemType>::Print()

{

//iterate through the Queue object until it points to NULL

for(NodeType<ItemType>\* iter = head; iter != NULL; iter = iter->next)

{

cout << iter->info << endl;

}

}

template <class ItemType> bool Queue<ItemType>::IsEmpty()

{

//returns true if empty, false if not

return rear == NULL;

}

template <class ItemType> bool Queue<ItemType>::IsFull()

{

if(this->length() == MAX\_ITEM)

return true;

return false;

}

template <class ItemType> void Queue<ItemType>::MakeEmpty()

{

for(NodeType<ItemType>\* temp = head; head != NULL; head = temp)

{

temp = head->next;

delete head;

}

}

template <class ItemType> Queue<ItemType>::~Queue()

{

this->MakeEmpty();

}