2021 Winter CIS200 – Programming Project 4

Release date: March 24, 2021

Due date: April 14, 2021

# Linked list is a perfect data structure for the implementation of stack and queue. Especially for the queue, it avoids the headaches of wrapping-around manipulation for a queue implemented by an array. In this project, you are requested to implement both stack and queue by using a linked list (not an array).

In this project, you have an option to find one partner to form a team. However, at most two persons are allowed in each team to complete this project. In a team setting, remember to provide the names of both team members in the document when you submit the completed project to Canvas.

# **Question 1 (30 points)**

Implement a template class Stack as defined by the following skeleton:

template<class ItemType>

class Stack

{

private:

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

public:

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

Stack(const Stack<ItemType> &x); // copy constructor: implicitly called for a

// deep copy

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

// the nodes of the linked list

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

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

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

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

// from the top to bottom

void Push(ItemType x); // insert x onto the stack

void Pop(ItemType &x); // delete the top element from the stack

// Precondition: the stack is not empty

~Stack(); // Destructor: memory for nodes needs to be deallocated

};

template<class ItemType>

struct NodeType

{

ItemType info;

NodeType\* next;

};

In you main( ) routine, you need to test your class in the following cases:

Stack <int> IntStack;

int x;

IntStack.Pop(x);

IntStack.Push(11);

IntStack.Push(22);

cout << "int length 1 = " << IntStack.length() << endl;

IntStack.Pop(x);

IntStack.Push(33);

cout << "int length 2 = " << IntStack.length() << endl;

cout << “The int stack contains: “ << endl;

IntStack.Print();

IntStack.Push(44);

IntStack.Push(55);

IntStack.Push(66);

if(IntStack.IsFull() == false)

cout << “The int stack is not full !” << endl;

else

cout << “The int stack is full !” << endl;

Stack <int> IntStack2(IntStack);

cout << “The int stack2 contains: “ << endl;

IntStack2.Print();

IntStack2.MakeEmpty();

cout << “The int stack3 contains: “ << endl;

IntStack2.Print();

Stack <float> FloatStack;

float y;

FloatStack.Pop(y);

FloatStack.Push(7.1);

cout << "float length 1 = " << FloatStack.length() << endl;

FloatStack.Push(2.3);

FloatStack.Push(3.1);

cout << "float length 2 = " << FloatStack.length() << endl;

FloatStack.Pop(y);

cout << “The float stack contains: “ << endl;

FloatStack.Print();

Stack <float> FloatStack2 = FloatStack;

cout << “The float stack 2 contains: “ << endl;

FloatStack2.Print();

FloatStack.MakeEmpty();

cout << “The float stack 3 contains: “ << endl;

FloatStack2.Print();

# **Question 2 (30 points)**

# Implement a template class Queue as defined by the following skeleton:

template<class ItemType>

class Queue

{

private:

NodeType<ItemType>\* front; // It points to the front 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 front 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 front of the queue

// Precondition: the queue is not empty

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

};

In you main( ) routine, you need to test your class in the following cases:

Queue<int>IntQueue;

int x;

IntQueue.MakeEmpty();

IntQueue.Dequeue(x);

IntQueue.Enqueue(10);

IntQueue.Enqueue(20);

IntQueue.Enqueue(30);

IntQueue.Enqueue(40);

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

IntQueue.Dequeue(x);

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

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

IntQueue.Print();

if(IntQueue.IsFull() == false)

cout << “The int queue is not full !” << endl;

else

cout << “The int queue is full !” << endl;

Queue<float>FloatQueue;

float y;

FloatQueue.MakeEmpty();

FloatQueue.Dequeue(y);

FloatQueue.Enqueue(7.1);

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

FloatQueue.Enqueue(2.3);

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();