2021 Fall CIS200 – Programming Assignment 3

Submitted By: Hira Sharif

Machine type: pc machine

Compiler type: Visual Studio

Date: Nov 18, 2021

# **Question 1 (40 points)**

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

#include<iostream>

using namespace std;

template<class ItemType>

struct NodeType

{

ItemType info;

NodeType\* next;

};

template<class ItemType>

class Stack

{

private:

NodeType<ItemType>\* topPtr;

// It points to a singly-linked list

public:

Stack()

{

topPtr = NULL;

}// default constructor: Stack is created and empty

Stack(const Stack<ItemType>& x)

{

topPtr = NULL;

if (x.topPtr == NULL)

return;

topPtr = new NodeType<ItemType>;

topPtr->info = x.topPtr->info;

topPtr->next = NULL;

NodeType<ItemType>\* t1 = topPtr;

NodeType<ItemType>\* t2 = x.topPtr;

while (t2->next != NULL)

{

t1->next = new NodeType<ItemType>;

t1->next->info = t2->next->info;

t1->next->next = NULL;

t1 = t1->next;

t2 = t2->next;

}

}// copy constructor: implicitly called for a

// deep copy

void MakeEmpty()

{

topPtr = NULL;

}// Stack is made empty; you should deallocate all the nodes of the linked list

bool IsEmpty()

{

return (topPtr == NULL);

}// test if the stack is empty

bool IsFull()

{

return (length() == 5);

}// test if the stack is full; assume MAXITEM=5

int length()

{

int l = 0;

NodeType<ItemType>\* temp = topPtr;

while (temp != NULL)

{

l++;

temp = temp->next;

}

return l;

}// return the number of elements in the stack

void Print()

{

NodeType<ItemType>\* t = topPtr;

while (t != NULL)

{

cout << t->info << endl;

t = t->next;

}

}// print the value of all elements in the stack in the sequence

// from the top to bottom

void Push(ItemType x)

{

if (topPtr == NULL)

{

topPtr = new NodeType<ItemType>;

topPtr->next = NULL;

topPtr->info = x;

}

else if (!IsFull())

{

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

t->next = topPtr;

t->info = x;

topPtr = t;

}

}// insert x onto the stack

void Pop(ItemType& x)

{

if (!IsEmpty())

{

x = topPtr->info;

NodeType<ItemType>\* t = topPtr;

topPtr = topPtr->next;

delete t;

}

}// delete the top element from the stack

// Precondition: the stack is not empty

~Stack()

{

if (!IsEmpty())

{

while (topPtr != NULL)

{

NodeType<ItemType>\* t = topPtr;

topPtr = topPtr->next;

delete t;

}

}

}// Destructor: memory for nodes needs to be deallocated

};

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

int main()

{

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();

return 0;

}

**DISCRIPTION:**

The program describe the stack implementation in linked list. It creates a structure for list node and a class for the stack implementation using different constructors and functions. Then main function first implement all the functions for integer data type, by push and pop values and displaying length and elements of linked list on the screen, and then for float data type.

|  |  |  |  |
| --- | --- | --- | --- |
| functions | Reason for test | Expected result | Actual result |
| stack() | Initialize TopPtr by Null. | It should Initialize TopPtr by Null. | It initializes TopPtr by Null.. |
| stack(x) | Copies the values of list x into stack TopPtr. | It should copy the values of list x into stack TopPtr. | It copies the values of list x into stack TopPtr. |
| makeEmplty() | Makes the list empty. | It should make the list empty by deleting all nodes. | It makes the list empty by deleting all nodes. |
| IsEmpty() | Check if list is empty. | It should check if length if the list is 0. | It checks if length if the list is 0. |
| IsFull() | Check if list is full | It should check if length of the list is maximum or not. | It checks if length of the list is maximum or not. |
| length() | Returns the length of the list. | It should add the length each time it reads value from list and return it. | It returns the length of the list. |
| push(x) | Pushes new node into the stack. | It should add new element into the stack. | It pushes new node into the stack. |
| pop(x) | Delete a node from the stack | It should delete last node entered in the stack. | It deletes a node from the stack |
| ~stack() | Deallocate all memory for the stack list. | Deallocate all memory for the stack list by calling MakeEmpty function. | It deallocate all memory for the stack list. |

**INCLUSION:**

This program shows stack implementation for linked list using function pointers.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

# **Question 2 (40 points)**

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

#include<iostream>

using namespace std;

template<class ItemType>

struct NodeType

{

ItemType info;

NodeType<ItemType>\* next;

};

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()

{

front = NULL;

rear = NULL;

}// default constructor: Queue is created and empty

Queue(const Queue<ItemType>& x)

{

if (x.front == NULL)

{

front = NULL;

rear = NULL;

}

else

{

NodeType<ItemType>\* a = new NodeType<ItemType>();

a->info = x.front->info;

a->next = NULL;

front = a;

rear = a;

NodeType<ItemType>\* b = x.front->next;

while (b != NULL)

{

a = new NodeType<ItemType>();

a->info = b->info;

a->next = NULL;

rear->next = a;

rear = rear->next;

b = b->next;

}

}

}// copy constructor: implicitly called

// for a deep copy

void MakeEmpty()

{

NodeType<ItemType>\* e;

while (front!=NULL)

{

e = front;

front = front->next;

delete e;

}

rear = NULL;

}// Queue is made empty; you should deallocate all

// the nodes of the linked list

bool IsEmpty()

{

return(front == NULL);

}// test if the queue is empty

bool IsFull()

{

return(length() == 5);

}// test if the queue is full; assume MAXITEM=5

int length()

{

if (front == NULL)

return 0;

else

{

int l = 0;

NodeType<ItemType>\* t = front;

while (t != NULL)

{

l++;

t = t->next;

}

return l;

}

}// return the number of elements in the queue

void Print()

{

if (IsEmpty())

return;

else

{

NodeType<ItemType>\* t1 = front;

while (t1 != NULL)

{

cout << t1->info << endl;

t1 = t1->next;

}

}

}// print the value of all elements in the queue in the sequence

// from the front to rear

void Enqueue(ItemType x)

{

if (IsFull())

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

else

{

NodeType<ItemType>\* n = new NodeType<ItemType>();

n->info = x;

n->next = NULL;

if (IsEmpty())

{

front = n;

rear = n;

}

else

{

rear->next = n;

rear = rear->next;

}

}

}// insert x to the rear of the queue

// Precondition: the queue is not full

void Dequeue(ItemType& x)

{

if (IsEmpty())

cout << "queue is empty! " << endl;

else

{

NodeType<ItemType>\* t = front;

x = front->info;

front = front->next;

if (front == NULL)

rear = NULL;

delete t;

}

}// delete the element from the front of the queue

// Precondition: the queue is not empty

~Queue()

{

MakeEmpty();

}// Destructor: memory for the dynamic array needs to be deallocated

};

int main()

{

Queue<int>IntQueue;

//integer type queue

int x;

IntQueue.MakeEmpty();

//dequeue empty queue

IntQueue.Dequeue(x);

//enqueue new values

IntQueue.Enqueue(10);

IntQueue.Enqueue(20);

IntQueue.Enqueue(30);

IntQueue.Enqueue(40);

//print values

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;

// float type queue

Queue<float>FloatQueue;

float y;

FloatQueue.MakeEmpty();

//dequeue empty queue

FloatQueue.Dequeue(y);

//enqueue new value

FloatQueue.Enqueue(7.1);

//print values

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();

}

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**DISCRIPTION:**

The program describe the queue implementation in linked list. It creates a structure for list node and a class for the queue implementation using different constructors and functions. Then main function first implement all the functions for integer data type, by enqueue and dequeue values and displaying length and elements of linked list on the screen, and then for float data type.

|  |  |  |  |
| --- | --- | --- | --- |
| functions | Reason for test | Expected result | Actual result |
| queue() | Initialize front and rear by Null. | It should Initialize front and rear by Null. | It initializes front and rear by Null. |
| queue(x) | Copies the values of list x into the queue and make it rear. | It should copy the values of list x into queue and make it rear. | It copies the values of list x into the queue. |
| makeEmplty() | Makes the list empty. | It should make the list empty by deleting all nodes. | It makes the list empty by deleting all nodes. |
| IsEmpty() | Check if list is empty. | It should check if the front is NULL. | It checks if the front is NULL. |
| IsFull() | Check if list is full | It should check if length of the list is maximum or not. | It checks if length of the list is maximum or not. |
| length() | Returns the length of the list. | It should add the length each time it reads value from list and return it. | It returns the length of the list. |
| Enqueue(x) | adds new node into the queue. | It should add new element into the queue. | It pushes new node into the queue. |
| Dequeue(x) | Delete a node from the queue | It should delete first node entered in the queue. | It deletes a node from the queue. |
| ~stack() | Deallocate all memory for the queue list. | Deallocate all memory for the queue list by deleting each value. | It deallocate all memory for the queue list. |

**INCLUSION:**

This program shows stack implementation for linked list using function pointers.