**HOMEWORK 8 – Linked Lists, Stacks, and Queues**

Answer the following questions. **THIS ASSIGNMENT WILL BE DUE AT A DATE TO BE ANNOUNCED IN CLASS.** (points as noted – 100 points total)

1. Explain the difference between appending and inserting a node. When is one preferred over the other? (10 points)

The difference between appending and inserting a node is appending a node means adding a new node to the end of the list, like python and lists. While inserting a node is adding a new node to a specific place in the linked list. Appending a node is preferred when you just need a new node at the end of the list, this is easier to code, but limits where the node can be, i.e., only at the end. However, inserting a new node allows the user to put the node in a specific position which depending on the requirements of the problems is a lot more useful.

1. What are the two steps involved in deleting a node from a linked list? Why can’t we just use the delete operator to remove the node from memory? (10 points)

The Two steps involved in deleting a node from a linked list is to first find the node in the actual linked list and then to reconstruct the links without the node you want removed. The “third step would be to actually delete the node.” You can’t just use the delete operator to remove the node from memory, because even though the node will be deleted there is also a node pointing to that node and the node you want deleted is pointing to the next node. Just using the delete operator will mess up the linked list.

1. What is the principal difference between stacks and queues? Included at some point in your answer should be the terms “FIFO” and “LIFO” (to wit, the meaning of those acronyms should be included, as well). (10 points)

N/A



4. Given a linked list with the following node and pointer initializations:

struct Cnode {

double value;

struct Cnode\* next;

};

struct Cnode\* first = NULL;

Assume that the list contains the values 30.41, 47.49, 74.75, and 85.86.

1. Pictorially indicate the linked list, assuming that, in the list’s final state, first points to the first node. (10 points)



1. Write the code to create a pointer to Cnode called thisNode which is to be initialized to point to whatever first points. (10 points)

struct Cnode \*thisNode = first;

1. Write the code to advance thisNode to point to the node containing 74.75. (10 points)

while (thisNode != NULL && thisNode->value != 74.75) {

thisNode = thisNode->next;

}

5. Given the linked list declaration from Problem 4, with first pointing to the first node. Analyze (as thoroughly as you deem appropriate!) the effect of the following code. Be certain to include a description of the code’s true purpose in your analysis. (20 points)

struct Cnode \*nodePtr = first, \*prevNodePtr = NULL;

double temp;

if(nodePtr−>next == NULL)

cout << "Can’t perform this operation − only one node in list!\n";

else {

prevNodePtr = nodePtr;

nodePtr = nodePtr−>next;

do {

temp = nodePtr−>value;

nodePtr−>value = prevNodePtr−>value;

prevNodePtr−>value = temp;

prevNodePtr = nodePtr;

nodePtr = nodePtr−>next;

} while(nodePtr != NULL);

}

The codes true purpose is to swap values of the nodes that are next/adjacent to it and by the time it reaches the end the linked list will be reversed.

struct Cnode \*nodePtr = first, \*prevNodePtr = NULL;

double temp;

These line of code declares two points and points the first to the first node and the second to NULL. It also declares a variable temp with a double type

if(nodePtr−>next == NULL)

cout << "Can’t perform this operation − only one node in list!\n";

else

These lines of code check if there is at least one node after the first node. If there isn’t and there is only one node it gives you the error message, “Can’t perform this operation – only one node in list!”. However if these is more than one node, it runs:

prevNodePtr = nodePtr;

nodePtr = nodePtr−>next;

Which initializes the prevNodePtr to the nodePtr and advances the nodePtr to the next node.

After that is runs a loop:

do {

temp = nodePtr−>value;

nodePtr−>value = prevNodePtr−>value;

prevNodePtr−>value = temp;

prevNodePtr = nodePtr;

nodePtr = nodePtr−>next;

} while(nodePtr != NULL);

This loops runs until the end of the linked list or (the nodePtr is equal to NULL) and in the loop it first sets the temp variable with the value of the current node.

Then with these two lines:

nodePtr−>value = prevNodePtr−>value;

prevNodePtr−>value = temp;

it switches the current node with the value of the previous node using the variable. Then with these two lines:

prevNodePtr = nodePtr;

nodePtr = nodePtr−>next;

it advances the previous node pointer to the current node and then points nodePtr to the next node.

The loops then checks if it is a the end of the linked list. If it is the code stops. If not the code repeats.

6. Given the linked list from Problem 4. Assume that the following additional definitions are made:

struct Cnode \*nodePtr = first, \*prevNodePtr = NULL;

Write the code to move the last node to the front of the list. Do NOT forget to make sure that the pointer in the last node in the resulting list is “grounded” to NULL. (20 points)

#include <iostream>

using namespace std;

struct Cnode {

double value;

struct Cnode\* next;

};

struct Cnode\* first = NULL;

struct Cnode \*nodePtr = first, \*prevNodePtr = NULL;

int main()

{

while (nodePtr->next != NULL) {

prevNodePtr = nodePtr;

nodePtr = nodePtr->next;

}

prevNodePtr->next = NULL;

nodePtr->next = first;

first = nodePtr;

return 0;

}