```
// File Name: StringList.cpp
//
// Author : Archana Sridhar
// Net ID : a_s1316
// Date: 04/10/2019
// Assignment Number: 5
// CS5301 Spring 2019
// Instructor: Dr.Jill Seaman
//
// C++ program that manipulates a list of strings  including the implementation
//             of the StringList member functions
#include<iostream>
#include<iomanip>
#include "StringList.h"
using namespace std;
//*************************************************************************
// StringList: creates an empty list (constructor)
//*************************************************************************

StringList::StringList()
{
    head = NULL;
}

//*************************************************************************
// StringList: deallocates all the nodes in StringList (destructor)
//*************************************************************************

StringList::~StringList()
{
    StringNode *p = head;
    StringNode *n;

    while (p != NULL)
    {
        n = p->next;
        delete p;
        p = n;
    }

}

//*************************************************************************
// count: returns the total number of nodes in the list
// returns the number of nodes in the list
//*************************************************************************

int StringList::count()
{
    StringNode *p = head;
    int count = 0;
    while (p != NULL)
    {
        p = p->next;
        count++;
    }
    return count;
}

//*************************************************************************
// display: displays the list of strings to the screen,
//          one string per line.
//*************************************************************************
```

```
void StringList::display()
{
    StringNode *p = head;
    while (p != NULL)
    {
        cout << p->data << endl;
        p = p->next;
    }
}
//*************************************************************************
// push_back: adds a new node containing the string to the end of the list
// item : the string that needs to be added to the node
//*************************************************************************

void StringList::push_back(string item)
{
    StringNode *newNode = new StringNode;
    StringNode *p = head;

    newNode->data = item;
    newNode->next = NULL;

    if(head==NULL)
        p = head;

    else
    {
        while(p->next)
            p=p->next;
        p->next=newNode;
    }
}
//*************************************************************************
// push_front : adds a new node containing the string to the front of the list
// item  : the string that needs to be added to the node
//*************************************************************************

void StringList::push_front(string item)
{
    StringNode *newNode = new StringNode;
    StringNode *p = head;

    newNode->data = item;

    if(head == NULL)
    {
        head = newNode;
        newNode->next = nullptr;
    }
    else
    {
        head = newNode;
        newNode->next = p;
    }
}

//*************************************************************************
// pop_front : removes the first node if the list is not empty
//*************************************************************************

void StringList::pop_front()
{
    if (head != NULL)
    {
        StringNode *p = head;
```

```
        head = head->next;
        delete p;
    }
}

//*************************************************************************
// remove: removed the node that m (argument) points to in the list
//*************************************************************************

void StringList::remove(StringNode *m)
{
    StringNode *p = head;
    StringNode *n = NULL;
    while(m != NULL && p != m)
    {
        n = p;
        p = p->next;
    }
    if(head == NULL)
    {
        return;
    }
    if(m)
    {
        if(m==head)
        {
            head = m->next;
            delete m;
        }
        else
        {
            n->next = p->next;
            m->next = NULL;
            delete m;
        }
    }
}
//*************************************************************************
// maximum: returns the string that would come last in alphabetical ordering.
// returns pointer to the node that comes last in alphabetical order
//*************************************************************************

StringList::StringNode * StringList:: maximum()
{
    StringNode *p = head;
    StringNode *maximum = p;
    string max = p->data;
    while(p != NULL)
    {
        if(p->data > max)
        {
            max = p->data;
            maximum = p;
        }
        p = p->next;
    }
    return maximum;
}
//*************************************************************************
// sort : sorts the strings in the linked list and displays in alphabetical
//        order
// returns the sorted list of strings
//*************************************************************************

void StringList::sort()
```

```
{

    StringNode *newHead= NULL;
    StringNode *p = NULL;

    while (head != NULL)
    {
        p = maximum();
        StringNode *n = new StringNode;
        n->data = p->data;
        n->next = newHead;
        newHead = n;
        remove(p);
    }
    head = newHead;
}
```