Name: Min Sung Cha Student ID: 85408485

Analysis of a few functions and any embedded functions

String.cpp

mscha1@andromeda-71:~/hw/hw5/mscha1

```
#include <iostream>
#include "String.h"

using namespace std;

//CONSTRUCTORS
String::String(const char * s) {
    head = ListNode::stringToList(s);
}

String::String(const String & s){
    head = String::ListNode::copy(s.head);
}

//STATIC METHODS

String::ListNode * String::ListNode::find(char c, ListNode * L){
    return !L || L->info == c ? L : find(c, L->next);
}

String::ListNode * String::ListNode::reverse(ListNode * L) {
    ListNode * result = nullptr;
    for (ListNode * p = L; p != nullptr; p = p->next)
        result = new ListNode(p->info, result);
    return result;
}

String::ListNode * String::ListNode::stringToList(const char * s) {
    return !*s ? 0 : new ListNode(*s, stringToList(s + 1));
}

void String::ListNode::deleteList(ListNode * L){
    if (L != nullptr){
        deleteList(L->next);
        delete L;
    }
}
                                                        27,0-1        Top
```

stringToList: this function recursively converts a c-string into a linksed list by first checking in its base case if the character is null. If not, it creates a new ListNode by recursively calling itself on the remaining characters.

reverse: this function reverses the linked list by creating a new linked list in the reversed order. It loops over the current linked list and creates the new linked list by creating new nodes with the next being preceding values.

deletelist: it deletes all the ListNodes in the linked list by first checking if the node is nullptr. If not it recursively calls itself until reaching the nullptr and then deletes the all the nodes in the linked list.

mscha1@andromeda-71:~/hw/hw5/mscha1

```
int String::ListNode::compare(ListNode * L1, ListNode * L2){
    if (L1 == nullptr && L2 == nullptr)
        return 0;
    else if (L1 == nullptr)
        return -L2-> info;
    else if (L2 == nullptr)
        return L1 -> info;
    else if (L1->info == L2->info)
        return compare (L1->next, L2->next);
    else
        return L1->info - L2->info;
}

String::ListNode * String::ListNode::copy(ListNode * L){
    return !L ? nullptr : new ListNode(L->info, copy(L->next));
}

String::ListNode * String::ListNode::append(ListNode * L1, ListNode * L2){
    return !L1 ? copy(L2) : new ListNode(L1->info, append(L1->next, L2));
}

int String::ListNode::length(ListNode * L){
    return !L ? 0 : 1 + length(L->next);
}

//PUBLIC METHODS
String String::operator = (const String & s){
    head = String::ListNode::copy(s.head);
    return *this;
}

char & String::operator [] (const int index){
    String::ListNode * temp = head;
    if (inBounds(index)){
        for (int i = 0; i < index; i++){
            temp = temp->next;
    }
```

75,5 34%

Type here to search 11:27 PM 2/13/2018

copy: this function recursively copies a linked list into another linked list. In the base case, it checks whether the ListNode is a nullptr. If not, it makes a new list node and recursively calls itself on further list nodes.

Append: it creates a new linked lists with both linked lists passed in as parameters added together. In the base cases it checks whether it has reached the nullptr in the first linked list and copies the second linked list into the new linked list. If not, it recursively calls itself to store all the nodes of the first linked list into the new linked list.

Length: computes the length of the linked list by recursively calling itself until reaching the base case (0) and adding 1 to itself.

mscha1@andromeda-71:~/hw/hw5/mscha1

```
char & String::operator [] (const int index){
    String::ListNode * temp = head;
    if (inBounds(index)){
        for (int i = 0; i < index; i++){
            temp = temp->next;
        }
        return temp->info;
    }
    else
        cerr << "INDEX OUT OF BOUNDS" << endl;
        return temp->info;
}

int String::indexOf(char c){
    int count = 0;
    for (String::ListNode * p = head; p != nullptr; p = p->next){
        if (p->info == c)
            return count;
        ++count;
    }
    return -1;
}

int String::size(){
    return ListNode::length(head);
}

void String::print(ostream & out) {
    for (String::ListNode * p = head; p != nullptr; p = p->next)
        out << p->info;
}

void String::read(istream & in){
    char a[256];
    in.getline(a, 256);
    head = String::ListNode::stringToList(a);
}
```

106,1 63%

Type here to search 11:27 PM 2/13/2018

mscha1@andromeda-71:~/hw/hw5/mscha1

```
bool String::operator == (const String & s){
    return ListNode::compare(head, s.head) == 0;
}

bool String::operator < (const String & s){
    return ListNode::compare(head, s.head) < 0;
}

String String::operator + (const String & s){
    String temp;
    temp.head = String::ListNode::append(head, s.head);
    return temp;
}

String String::operator += (const String & s){
    head = String::ListNode::append(head, s.head);
    return *this;
}

String String::reverse(){
    String rev;
    rev.head = String::ListNode::reverse(head);
    return rev;
}

String::~String(){
    String::ListNode::deleteList(head);
}

ostream & operator << (ostream & out, String str) {
    str.print(out);
    return out;
}

istream & operator >> (istream & in, String & str){
    str.read(in);
    return in;
```

145,5 99%

Type here to search 11:28 PM 2/13/2018

Operator += : This function utilizes the static compare function defined inside the struct ListNode to create an entirely new ListNode object with both the String objects in s and head added together. Then, the head of this string is assigned to the new ListNode object.

string_test.cpp

mscha1@andromeda-71:~/hw/hw5/mscha1

```cpp
#include <iostream>
#include "String.h"

using namespace std;

void test_constructors_and_print(){
    cout << "-----TESTING CONSTRUCTORS AND PRINT------" << endl;
    String i("Hello World");
    cout << i << endl;

    String first("First");
    String f(first);
    cout << f << endl;
}

void test_assignment(){
    cout << "-----TESTING ASSSIGNMENT-----" << endl;
    String h("Hello World");
    String b("Bye World");
    h = b;
    cout << "Bye World: " << h << endl;

    String you("You");
    String me ("Me");
    me = you;
    cout << "You: " << me << endl;
}

void test_size(){
    cout << "-----TESTING SIZE-----" << endl;
    String h("Hello World");
    String uci("University of California, Irvine");
    cout << "11: " << h.size() << endl;
    cout << "32: " << uci.size() << endl;
}
/*
void test_indexOf(){
```

26,5 Top

Type here to search 11:29 PM 2/13/2018

mscha1@andromeda-71:~/hw/hw5/mscha1

```cpp

void test_equality(){
    String h1("Hello World");
    String h2("Hello World");
    String h3("Bye World");
    cout << "-----TESTING EQUALITY-----" << endl;
    cout << "1: " << (h1 == h2) << endl;
    cout << "0: " << (h1 == h3) << endl;
}

void test_less_than(){
    String h1("Hello World");
    String h2("Hello World");
    String h3("Bye World");
    String a1("b");
    String a2("a");
    cout << "-----TESTING LESS THAN-----" << endl;
    cout << "1: " << (a2 < a1) << endl;
    cout << "1: " << (h3 < h2) << endl;
}

void test_concatenation(){
    cout << "-----TESTING CONCATENATION-----" << endl;
    String s1("HELLO ");
    String s2("WORLD ");
    String s3("BYE ");
    cout << "HELLO WORLD: " << s1 + s2 << endl;
    s3 += s2;
    cout << "BYE WORLD: " << s3 << endl;
}

void test_reverse(){
    cout << "-----TESTING REVERSE----" << endl;
    String s1("Hello");
    String s2("TOMATO IS LOVE");
    cout << "olleH: " << s1.reverse() << endl;
    cout << "EVOL SI OTAMOT: " <<s2.reverse() << endl;
```

36,0-1 48%

Type here to search 11:29 PM 2/13/2018

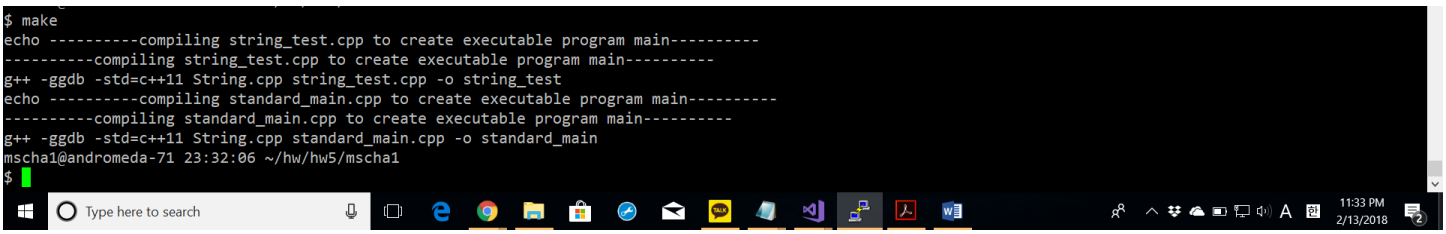
mscha1@andromeda-71:~/hw/hw5/mscha1

```
    cout << "olleH: " << s1.reverse() << endl;
    cout << "EVOL SI OTAMOT: " <<s2.reverse() << endl;
}

void test_index_operator(){
    cout << "-----TESTING INDEX OPERATOR-----" << endl;
    String h("Hello World");
    cout << "l: " << h[2] << endl;
    cout << "d: " << h[10] << endl;
    cout << "ERROR: " << h[15] << endl;
}

void test_indexOf(){
    cout << "-----TESTING INDEXOF----" << endl;
    String h("Hello World");
    cout << "2: " << h.indexOf('l') << endl;
    cout << "-1: " << h.indexOf('z') << endl;
}

void test_read(){
    cout << "-----TESTING READ-----" << endl;
    String s;
    cin >> s;
    cout << s << endl;
}

int main(){
    test_constructors_and_print();
    test_assignment();
    test_size();
    test_equality();
    test_less_than();
    test_concatenation();
    test_reverse();
    test_index_operator();
    test_indexOf();
    test_read();
                                                                    71,5          97%
```

Type here to search

11:30 PM
2/13/2018

Screenshot of successful compiling with make command



My make command compiles both of the main functions at the same time.

Screenshots of successful execution with valgrind command

valgrind string_test.cpp

```
mscha1@andromeda-71:~/hw/hw5/mscha1
0: 0
-----TESTING LESS THAN-----
1: 1
1: 1
-----TESTING CONCATENATION-----
HELLO WORLD: HELLO WORLD
BYE WORLD: BYE WORLD
-----TESTING REVERSE----
olleH: olleH
EVOL SI OTAMOT: EVOL SI OTAMOT
-----TESTING INDEX OPERATOR-----
l: l
d: d
INDEX OUT OF BOUNDS
ERROR: H
-----TESTING INDEXOF----
2: 2
-1: -1
-----TESTING READ-----
min
min
==15584==
==15584== HEAP SUMMARY:
==15584==     in use at exit: 72,976 bytes in 18 blocks
==15584==   total heap usage: 330 allocs, 312 frees, 77,968 bytes allocated
==15584==
==15584== LEAK SUMMARY:
==15584==    definitely lost: 48 bytes in 3 blocks
==15584==    indirectly lost: 224 bytes in 14 blocks
==15584==      possibly lost: 0 bytes in 0 blocks
==15584==    still reachable: 72,704 bytes in 1 blocks
==15584==         suppressed: 0 bytes in 0 blocks
==15584== Rerun with --leak-check=full to see details of leaked memory
==15584==
==15584== For counts of detected and suppressed errors, rerun with: -v
==15584== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
mscha1@andromeda-71 23:34:30 ~/hw/hw5/mscha1
$
```

valgrind standard_main.cpp

```
mscha1@andromeda-71:~/hw/hw5/mscha1
==15594== Memcheck, a memory error detector
==15594== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==15594== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==15594== Command: standard_main
==15594==
+: FirstSecond
+=: FirstSecond
indexOf(char): 4
LT: 0
<<:
<<: Fourth
==: 0
[]: i
<<: First First
[]: i
INDEX OUT OF BOUNDS
[]: F
Enter a test string: min
min
0
0
==15594==
==15594== HEAP SUMMARY:
==15594==     in use at exit: 72,960 bytes in 17 blocks
==15594==   total heap usage: 88 allocs, 71 frees, 74,096 bytes allocated
==15594==
==15594== LEAK SUMMARY:
==15594==    definitely lost: 32 bytes in 2 blocks
==15594==    indirectly lost: 224 bytes in 14 blocks
==15594==      possibly lost: 0 bytes in 0 blocks
==15594==    still reachable: 72,704 bytes in 1 blocks
==15594==         suppressed: 0 bytes in 0 blocks
==15594== Rerun with --leak-check=full to see details of leaked memory
==15594==
==15594== For counts of detected and suppressed errors, rerun with: -v
==15594== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
mscha1@andromeda-71 23:35:03 ~/hw/hw5/mscha1
$
```