

Assignment 2

Unit test scripts of code:

Test 1 ID:		Test index() function with valid index
<i>Preconditions:</i>		A Rope instance with "Hello, world!" is created.
<i>Input Data:</i>		Index 7
<i>Expected Result:</i>		Function should return "w".
<i>Postconditions:</i>		The Rope remains unchanged.

Code Execution:

```
w
C:\Users\erbab\Desktop\
```

Test 2 ID:		Concatenation and Indexing			
<i>Preconditions:</i>		Two Rope instances are created with "Hello, " and "world!".			
<i>Input Data:</i>		Concatenate the two Rope instances and perform index operations.			
<i>Expected Result:</i>		The concatenated Rope should have a length of 13. Index 0 should be 'H', index 7 should be 'w', and index 13 should be '!'.			
<i>Postconditions:</i>		The original Rope instances remain unchanged.			

Code Execution:

```
Hw!
C:\Users\erbab\Desktop\
```

Test 3 ID:		Split Operation			
<i>Preconditions:</i>		A Rope instance `concatenated` with content "Hello, world!" is created.			
<i>Input Data:</i>		Split concatenated` at index 7.			
<i>Expected Result:</i>		The left Rope should have a length of 7 and index 0 should be 'H'. The right Rope should have a length of 6 and index 0 should be 'w'.			
<i>Postconditions:</i>		The original concatenated Rope instance remains unchanged.			

Code Execution:

```
Hw
C:\Users\erbab\Desktop\
```

Test 4 ID:		Insert Operation			
<i>Preconditions:</i>		A Rope instance, left, with the content "Hello, world!" is created.			
<i>Input Data:</i>		Insert " beautiful" at index 2 in left.			
<i>Expected Result:</i>		After inserting " beautiful" at index 2, the Rope should have a length of 24. Index 0 should be 'H', index 2 should be ' ', and index 11 should be 'b'.			
<i>Postconditions:</i>		The original `left` Rope instance remains unchanged.			

Code Execution:

```
H b
C:\Users\erbab\Desktop\
```

Test 5 ID:		Remove Operation			
<i>Preconditions:</i>		A Rope instance is created with content "Hello, beautiful world!".			
<i>Input Data:</i>		Remove 10 characters starting from index 2.			
<i>Expected Result:</i>		After removal, the Rope should have a length of 14. Index 0 should be 'H', index 2 should be 'e', and index 3 should be 'a'.			
<i>Postconditions:</i>		The original Rope instance remains unchanged.			

Code Execution:

```
Hea
C:\Users\erbab\Desktop\
```

Test 6 ID:		Subrope Functionality		
<i>Preconditions:</i>		A Rope instance is created with the content "Hello, world!".		
<i>Input Data:</i>		Get the subrope from index 2 to 9.		
<i>Expected Result:</i>		The subrope should have a length of 7. Index 0 should be 'l', index 3 should be 'o', and index 6 should be ','.		
<i>Postconditions:</i>		The original Rope instance remains unchanged.		

Code Execution:

```
lo,
C:\Users\erbab\Desktop\
```

A listing of the source code is below (if needed):

A listing of the source code:

```
//Samuel Barker
//00100768
//sbarker1@my.athens.edu
//CS 417 Assignment 2 Source code
//All tests are included in this file, along with the class.

#include <memory>
#include <string>
#include <iostream>

class Rope {
private:
    struct Node {
        int weight;
        std::string str;
        std::shared_ptr<Node> left;
        std::shared_ptr<Node> right;

        Node(const std::string& s) : weight(s.length()), str(s), left(nullptr),
right(nullptr) {}
        Node() : weight(0), str(""), left(nullptr), right(nullptr) {}
    };

    std::shared_ptr<Node> root;

public:
    Rope() : root(std::make_shared<Node>()) {}
    Rope(const std::string& s) : root(std::make_shared<Node>(s)) {}

    char index(int i) {
        return index(root, i);
    }

    Rope concat(const Rope& s2) {
        Rope newRope;
        newRope.root = std::make_shared<Node>();
        newRope.root->left = root;
        newRope.root->right = s2.root;
        newRope.root->weight = root->weight;

        return newRope;
    }

    void split(int i, Rope& left, Rope& right) {
        split(root, i, left.root, right.root);
    }

    void insert(int i, const std::string& s) {
        Rope left, right;
        split(i, left, right);

        Rope inserted(s);
        *this = left.concat(inserted).concat(right);
    }
}
```

```

void remove(int start, int length) {
    Rope left, middle, right;
    split(start, left, middle);
    middle.split(length, middle, right);

    *this = left.concat(right);
}

Rope subrope(int i, int j) {
    Rope result;
    subrope(root, i, j, result.root);
    return result;
}

private:
char index(const std::shared_ptr<Node>& node, int i) {
    if (node->weight < 1)
        return index(node->right, i - node->weight);
    if (node->left)
        return index(node->left, i);
    return node->str[i];
}

void split(const std::shared_ptr<Node>& node, int i, std::shared_ptr<Node>&
left, std::shared_ptr<Node>& right) {
    if (!node)
        return;

    if (i < node->weight) {
        right = node;
        split(node->left, i, left, right->left);
    }
    else {
        left = node;
        split(node->right, i - node->weight, left->right, right);
    }

    updateWeight(left);
    updateWeight(right);
}

void updateWeight(const std::shared_ptr<Node>& node) {
    if (node)
        node->weight = (node->left ? node->left->weight : 0) + (node->right ?
node->right->weight : 0) + node->str.length();
}

void subrope(const std::shared_ptr<Node>& node, int i, int j,
std::shared_ptr<Node>& result) {
    if (!node)
        return;

    if (i < node->weight) {
        subrope(node->left, i, j, result);
        if (j > node->weight) {

```

```

        result = node;
        subrope(node->right, i - node->weight, j - node->weight, result-
>right);
    }
}
else {
    subrope(node->right, i - node->weight, j - node->weight, result);
}
}
};

```

```

int main() {
    Rope rope1("Hello, ");
    Rope rope2("world!");

    Rope concatenated = rope1.concat(rope2);
    Rope left, right;
    concatenated.split(7, left, right);

    std::cout << right.index(0); //Output: w

    return 0;
}

```

//BELOW IS THE REST OF TESTS, THEY ARE COMMENTED OUT TO ALLOW
//SINGLE TEST ABOVE TO RUN WITH NO OVERLOADS.

```

/*
int main() {
    Rope rope1("Hello, ");
    Rope rope2("world!");

    // Concatenation and index test
    Rope concatenated = rope1.concat(rope2);
    std::cout << concatenated.index(0);
    std::cout << concatenated.index(7);
    std::cout << concatenated.index(13);
    //Output: Hw!

    // Split test
    Rope left, right;
    concatenated.split(7, left, right);
    std::cout << left.index(0);
    std::cout << right.index(0);
    //Output: Hw

    // Insert test
    left.insert(2, " beautiful");
    std::cout << left.index(0);
    std::cout << left.index(2);
    std::cout << left.index(11);
    //Output: H b

    // Remove test
    left.remove(2, 10);
}
*/

```

```
std::cout << left.index(0);
std::cout << left.index(2);
std::cout << left.index(3);
//Output: Hea

// Subrope test
Rope sub = concatenated.subrope(2, 9);
std::cout << sub.index(0);
std::cout << sub.index(3);
std::cout << sub.index(6);
//Output: lo,

return 0;
}

*/
```