**Homework 8 – CS60 Linnell**

**Arjun Kohli**

**W1579330**

**Problem 1:**

#include <iostream>

using namespace std;

int sequence(int n) {

if (n == 0 || n == 1) {

return 1;

} else {

return sequence(n-2)+n-1;

}

}

int main() {

cout << sequence(7) << endl;

}

**Output:**

13

**Problems 2&4:**

**Header File (node.h):**

#ifndef NODE\_H

#define NODE\_H

#include <iostream>

class node

{

public:

// pre: none

// post: creates a new node with given data and link values

node(const int & initdata, node \* initlink);

// pre: none

// post: creates a new node with defaults: data=0 and link=nullptr

node();

// pre: none

// post: returns data value stored in this node

int data() const;

// pre: none

// post: returns link value stored in this node

node \* link();

// pre: none

// post: returns CONSTANT link value stored in this node

const node \* link() const;

// pre: none

// post: sets the data field to given value

void set\_data(const int & newdata);

// pre: none

// post: sets the link field to given value

void set\_link(node \* newlink);

private:

int data\_;

node \* link\_;

// invariants:

// data\_ holds the data value stored in this node

// link\_ holds the address of the next node

};

void list\_reverse(node \* &head\_ptr, node \* &tail\_ptr);

void list\_clear(node \* &head\_ptr, node \* &tail\_ptr);

//I overloaded this so I could output my linked list when testing my functions in main

std::ostream & operator <<(std::ostream & os, const node \* head);

#endif // NODE\_H

**Class File (node.cpp):**

#include "node.h"

node::node(){

data\_ = 0;

link\_ = nullptr;

}

node::node(const int& initdata, node \* initlink)

{

data\_ = initdata;

link\_ = initlink;

}

int node::data() const

{

return data\_;

}

node \* node::link()

{

return link\_;

}

//returns constant node pointer

const node \* node::link() const//This is a const function - I promise the function doesn't change anything - On a const node\*, you can only call const functions

{

return link\_;

}

void node::set\_data(const int & newdata)

{

data\_ = newdata;

}

void node::set\_link(node \* newlink)

{

link\_ = newlink;

}

//////////////////////////////////////////////////////////////////

void list\_clear(node \* &head\_ptr, node \* &tail\_ptr) {

node \*p = head\_ptr;

while(p != nullptr) {

head\_ptr = p->link();

delete p;

p = head\_ptr;

}

}

void list\_reverse(node \* &head\_ptr, node \* &tail\_ptr) {

node \*p = nullptr;

node \*c = head\_ptr;

node \*n = nullptr;

while (c!=nullptr) {

n = c->link();

c->set\_link(p);

p = c;

c = n;

}

head\_ptr = p;

}

//I overloaded this so I could output my linked list when testing my functions in main

std::ostream & operator <<(std::ostream & os, const node \* head){

for(const node \*p = head; p!= nullptr; p=p->link()){

os << p->data() << " ";

}

return os;

}

**Main File (main.cpp):**

#include "node.h"

#include <iostream>

using namespace std;

int main(){

node \*head = new node(4, nullptr);

node \*tail(head);

head = new node(8, head);

head = new node(1, head);

head = new node(5, head);

head = new node(1, head);

head = new node(3, head);

cout << head << endl;

list\_reverse(head, tail);

cout << head << endl;

list\_clear(head, tail);

cout << head << endl;

return 0;

}

**Output:**

3 1 5 1 8 4

4 8 1 5 1 3

//Prints empty line because list is clear

**Problem 3:**

#include <iostream>

using namespace std;

void reverse(int a[], int first, int last) {

if (first >= last) {

return ;

} else {

int temp = a[first];

a[first] = a[last];

a[last] = temp;

reverse(a, first+1, last-1);

}

}

int main() {

int a[] = {1, 2, 3, 4, 5};

for (int i = 0; i < 5; i++) {

cout << a[i] << " ";

}

cout << endl;

reverse(a, 0, 4);

for (int i = 0; i < 5; i++) {

cout << a[i] << " ";

}

cout << endl;

}

**Output:**

1 2 3 4 5

5 4 3 2 1

**Problem 5:**

#include <iostream>

using namespace std;

void merge(int arr[], int start, int mid, int end) {

int start2 = mid+1;

if (arr[mid] <= arr[start2]) {

return ;

} else {

while (start <= mid && start2<=end) {

if (arr[start] <= arr[start2]) {

start++;

} else {

int value = arr[start2];

int index = start2;

while (index != start) {

arr[index] = arr[index-1];

index--;

}

arr[start] = value;

merge(arr, start+1, start2, end);

}

}

}

}

void mergesort(int arr[], int start, int end) {

if (start < end) {

return ;

} else {

int mid = 1 + (end-1)/2;

int q1 = mid/2;

int q2 = mid + mid/2;

merge(arr, start, q1, end);

merge(arr, mid, q2, end);

merge(arr, start, mid, end);

}

}

int main() {

int a[] = {3, 9, 12, 2, 10, 17};

for (int i = 0; i < 6; i++) {

cout << a[i] << " ";

}

cout << endl;

merge(a, 0, 2, 5);

for (int i = 0; i < 6; i++) {

cout << a[i] << " ";

}

cout << endl;

mergesort(a, 0, 5);

for (int i = 0; i < 6; i++) {

cout << a[i] << " ";

}

cout << endl;

}

**Output:**

3 9 12 2 10 17

2 3 9 10 12 17

2 3 9 10 12 17