Sushil Khanal

Section 1

Project 4

**ORDERING AND MERGING TWO LISTS USING LINKED LISTS**

**Introduction**

A linked list is a linear data structure where each element is a separate object. Each element (we will call it a node) of a list is comprising of two items - the data and a reference to the next node. The last node has a reference to null. The entry point into a linked list is called the head of the list.

The objective of this project is to implement an ordered list abstract data type in a class with a linked list and use that class in a program that orders two group of integers, each group in its own list, and then merge them into one ordered list; without any errors.

**Data structure**

This program uses three data structure, .h file, .cpp file and int. All the headers are stored in .h file and the mains are stored in .cpp file. Int item is defined at first in the data structures Node.

**Functions**

The program uses nine functions.

1. Merge()
2. make\_empty()
3. length()
4. insert()
5. merge\_list()
6. ostream& operator<<()
7. get\_node()
8. input\_file()
9. read\_file()

**The Main Program**

After the program has been successfully compiled, it asks for two input files. The data from the input files are merged together and, in the output,, the integers are displayed in an ascending order.

**Code**

Merge.cpp

#include <iostream>

#include <cstdlib>

#include <cassert>

#include "merge.h"

using namespace std;

merge::~merge()

{

Node\* temp;

while (first != NULL) {

temp = first;

first = first->next;

delete temp;

}

}

void merge::make\_empty()

{

Node\* temp;

while (first != NULL) {

temp = first;

first = first->next;

delete temp;

}

}

int merge::length()

{

item count = 1;

Node\* temp;

temp = first;

while (temp->next != NULL) {

temp = temp->next;

count++;

}

return count;

}

void merge::insert(item entry)

{

Node\* prev;

if (first == NULL || entry < first->data) {

first = get\_node(entry, first);

}

else {

prev = first;

while (prev->next != NULL && prev->next->data < entry)

prev = prev->next;

prev->next = get\_node(entry, prev->next);

}

}

void merge::merge\_list(const merge& list1, const merge& list2)

{

Node\* last;

Node\* n1;

Node\* n2;

n1 = list1.first;

n2 = list2.first;

if (n1 != NULL && n2 != NULL) {

if (n1->data < n2->data) {

first = get\_node(n1->data, NULL);

n1 = n1->next;

}

else if (n2->data < n1->data) {

first = get\_node(n2->data, NULL);

n2 = n2->next;

}

else {

first = get\_node(n1->data, NULL);

n1 = n1->next;

n2 = n2->next;

}

last = first;

}

while (n1 != NULL && n2 != NULL) {

if (n1->data < n2->data) {

last->next = get\_node(n1->data, NULL);

n1 = n1->next;

}

else if (n2->data < n1->data) {

last->next = get\_node(n2->data, NULL);

n2 = n2->next;

}

else {

last->next = get\_node(n2->data, NULL);

n1 = n1->next;

n2 = n2->next;

}

last = last->next;

}

while (n1 != NULL) {

last->next = get\_node(n1->data, NULL);

n1 = n1->next;

last = last->next;

}

while (n2 != NULL) {

last->next = get\_node(n2->data, NULL);

n2 = n2->next;

last = last->next;

}

}

ostream& operator<<(ostream& out\_s, const merge& list)

{

merge::Node\* n;

n = list.first;

cout << n->data << " ";

while (n->next != NULL) {

n = n->next;

out\_s << n->data << " ";

}

return out\_s;

}

merge::Node\* merge::get\_node(const item& entry, Node\* n)

{

Node\* temp;

temp = new Node;

temp->data = entry;

temp->next = n;

return temp;

}

Merge.h

#include <iostream>

#ifndef MERGE\_H

#define MERGE\_H

using namespace std;

class merge {

public:

typedef int item;

merge() { first = NULL; } //constructor

~merge(); //destructor

void insert(item entry); //modification member function

void make\_empty(); //This function clears the list.

void merge\_list(const merge& list1, const merge& list2);

int length(); //constant member function

friend std::ostream& operator<<(std::ostream& out\_s, const merge& list); //friend function

private:

struct Node {

item data;

Node\* next;

};

Node\* first;

Node\* get\_node(const item& entry, Node\* n);

};

#endif

Project4.cpp

#include <iostream>

#include <fstream>

#include <cstdlib>

#include "merge.h"

#include "merge.cpp"

using namespace std;

void input\_file(ifstream &infile);

void read\_file(ifstream& infile, merge& list);

int main()

{

merge list1,list2,list;

ifstream infile;

input\_file(infile);

read\_file(infile,list1);

cout<<"There are "<<list1.length()<<" values in the first list:\n";

cout<<endl<<list1<<endl<<endl;

infile.close();

input\_file(infile);

read\_file(infile,list2);

cout<<"There are "<<list2.length()<<" values in the second list:\n";

cout<<endl<<list2<<endl<<endl;

infile.close();

cout<<endl;

list.merge\_list(list1,list2);

cout<<"There are "<<list.length()<<" values in the merge list:\n";

cout<<endl<<list<<endl<<endl;

return 0;

}

void input\_file(ifstream& infile)

{

const int FILESIZE = 20;

char input\_filename[FILESIZE+1];

cout<<"Enter input filename: ";

cin>>input\_filename;

infile.open(input\_filename);

if(!infile)

{

cout<<"Cannot read the input filename.\n";

return;

}

}

void read\_file(ifstream& infile, merge& list)

{

int x;

while(infile>>x)

{

list.insert(x);

}

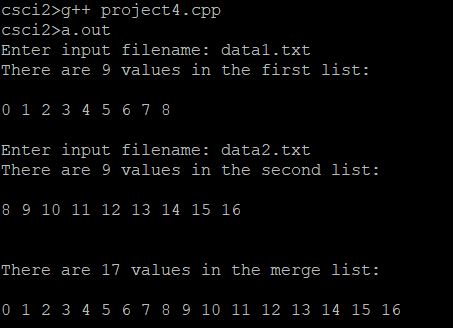
}

**User Document**

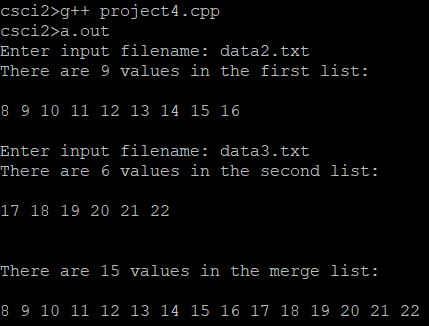
After completing the code, the program was compiled in putty by using the command “g++ project4.cpp”. Then we type a.out. The program prompts for an input file. When one input file is entered the program reads and displays the data stored in the data file, and asks us to enter another input file and does the similar thing, along with properly arranged merged list of those two input files.

**Tests**

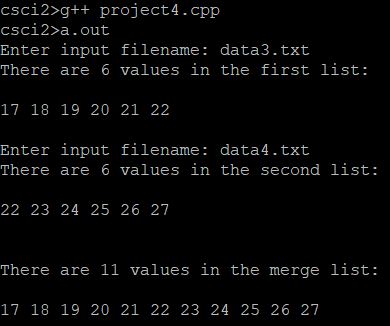
Test 1

****

Test 2



Test 3



**Conclusion**

We were able to implement an ordered list abstract data type with linked list and use that class in a program that orders two group of integers, each group in its own list and then merge them into one ordered list without any errors. I was very relived when the program ran successfully. I had been working on this for quite sometime and I hadn’t able to get the desired output.