**Project 1: Insertion Sort (Stacks) (C++)**

**CSC 323-32 Rafael Carmilema**

**Due Date: (10/20/2015)**

**Algorithm steps**:

**Step 0:** Main() function (must be in Main.cpp)

* 1. Check the command-line arguments
  2. Process Input (Parsing of the file)
  3. Create LinkedList

**Step 1:**Data Structure

* 1. LinkedList class (with a dummy node)  
     - Set dummy node to -9999  
     - Include constructors/destructors/all needed methods  
     - Insert function that takes in a single data creates a new node with that data and inserts into the list by insertion sort.
  2. Node struct  
     - Must be nested within LinkedList class  
     - Each node contains two components : data (integer), next (Node pointer)

**Step 2:**Print()  
-specific function for outputting the linkedlist, overloading << operator is a better choice!

MAIN

#include <iostream>

#include <fstream>

#include "LinkedList.h"

using namespace std;

int main(int argc, char\*\* argv) {

string ra;

LinkedList test;

if ( argc <2 ) // argc should be 2 for correct execution

// We print argv[0] assuming it is the program name

cout<<"usage: "<< argv[0] <<" <filename>\n";

else {

// We assume argv[1] is a filename to open

ifstream the\_file ( argv[1] );

// Always check to see if file opening succeeded

if ( !the\_file.is\_open() )

cout<<"Could not open file\n";

else {

int x;

// the\_file.get ( x ) returns false if the end of the file

// is reached or an error occurs

ofstream the\_outputfile(argv[2]);

while ( the\_file>>x){

test.AddNode(x);

test.PrintTest(the\_outputfile);

test.Print();

//cout<<"#"<<i<<" "<< x<<endl;

}

the\_file.close();

the\_outputfile.close();

//test.Print();

//ofstream the\_outputfile(argv[2]);

//the\_outputfile<<"RAFAEL carmilema asmal"<<endl;

//the\_outputfile.close();

}

// the\_file is closed implicitly here

}

}

CLASS LINKEDLIST

#include <iostream>

#include <cstdlib>

#include <string>

#include "LinkedList.h"

using namespace std;

LinkedList::LinkedList() {

head=NULL;

dummy=NULL;

//temp=NULL;

}

void LinkedList::AddNode(int addData){

nodePtr n = new node;

n->next = NULL;

n->data = addData;

if(head!=NULL){

if(Search(addData)){

return;

}

else{

if((head->data)>(n->data)){

n->next=head;

head=n;

}

else{

nodePtr curr= head;

nodePtr temp=head;

while(curr!=NULL){

if((curr->data)<(n->data)){

temp=curr;

curr = curr->next;

}

else{

n->next=curr;

temp->next=n;

break;

}

}

if(curr==NULL)

temp->next=n;

}

}

}

else{

head = n;

}

}

bool LinkedList::Search(int value){

nodePtr foundNode = head;

while(foundNode!=NULL){

if(foundNode->data==value){

return true;

}

foundNode = foundNode->next;

}

return false;

}

/\*void LinkedList::DeleteNode(int delData){

nodePtr delPtr =NULL;

temp = head;

curr = head;

while(curr!=NULL && curr->data != delData){

temp=curr;

curr= curr->next;

}

if(curr==NULL){

cout<<"The number was not found"<<endl;

delete delPtr;

}

else{

delPtr = curr;

curr= curr->next;

temp->next= curr;

delete delPtr;

cout<<"The number "<<delData<<" was found and delete "<<endl;

}

}\*/

void LinkedList::Print(){

nodePtr curr=head;

while(curr!=NULL){

if(curr==head){

if(curr->next!=NULL)

cout<<"ListHead -->"<<"("<<curr->data<<","<<curr->next->data<<")--> ";

else

cout<<"ListHead -->"<<"("<<curr->data<<","<<"-1"<<")--> "<<endl;

}

else{

if(curr->next!=NULL){

cout<<" ("<<curr->data<<","<<curr->next->data<<")-->";

}

else{

cout<<" ("<<curr->data<<","<<"-1"<<")"<<endl;

}

}

curr=curr->next;

}

}

void LinkedList::PrintTest(std::ofstream &output){

nodePtr curr=head;

string test="";

int count=0;

while(curr!=NULL&&count!=10){

if(curr==head){

if(curr->next!=NULL){

output<<"ListHead -->"<<"("<<curr->data<<","<<curr->next->data<<")--> ";

}

else

output<<"ListHead -->"<<"("<<curr->data<<","<<"-1"<<")--> "<<endl;

}

else{

if(curr->next!=NULL&&count!=9){

output<<" ("<<curr->data<<","<<curr->next->data<<")-->";

}

else{

output<<endl;

}

}

curr=curr->next;

count++;

}

}

HEADER

#ifndef LINKEDLIST\_H\_

#define LINKEDLIST\_H\_

#include <fstream>

class LinkedList{

private:

typedef struct node{

int data;

node\* next;

}\* nodePtr;

nodePtr head;

nodePtr dummy;

//nodePtr curr;

//nodePtr temp;

public: // functions go here prototypes

LinkedList();

void AddNode(int addData);

void DeleteNode(int delData);// finds the val and deletes it if its in the list

void Print();

void PrintTest(std::ofstream &output);

bool Search(int value);

};

#endif

**OUTPUT:**

**ListHead -->(91,-1)-->**

**ListHead -->(91,322)-->**

**ListHead -->(91,322)--> (322,702)-->**

**ListHead -->(91,322)--> (322,702)--> (702,9999)-->**

**ListHead -->(12,91)--> (91,322)--> (322,702)--> (702,9999)-->**

**ListHead -->(12,91)--> (91,133)--> (133,322)--> (322,702)--> (702,9999)-->**

**ListHead -->(12,91)--> (91,133)--> (133,322)--> (322,702)--> (702,4152)--> (4152,9999)-->**

**ListHead -->(12,91)--> (91,133)--> (133,213)--> (213,322)--> (322,702)--> (702,4152)--> (4152,9999)-->**

**ListHead -->(12,23)--> (23,91)--> (91,133)--> (133,213)--> (213,322)--> (322,702)--> (702,4152)--> (4152,9999)-->**

**ListHead -->(12,23)--> (23,91)--> (91,133)--> (133,213)--> (213,322)--> (322,702)--> (702,1724)--> (1724,4152)--> (4152,9999)-->**

**ListHead -->(12,23)--> (23,91)--> (91,133)--> (133,213)--> (213,322)--> (322,702)--> (702,825)--> (825,1724)--> (1724,4152)-->**

**ListHead -->(12,23)--> (23,91)--> (91,127)--> (127,133)--> (133,213)--> (213,322)--> (322,702)--> (702,825)--> (825,1724)-->**

**ListHead -->(12,23)--> (23,91)--> (91,127)--> (127,133)--> (133,213)--> (213,322)--> (322,538)--> (538,702)--> (702,825)-->**

**ListHead -->(12,23)--> (23,29)--> (29,91)--> (91,127)--> (127,133)--> (133,213)--> (213,322)--> (322,538)--> (538,702)-->**

**ListHead -->(12,23)--> (23,29)--> (29,91)--> (91,127)--> (127,133)--> (133,213)--> (213,322)--> (322,538)--> (538,702)-->**

**ListHead -->(12,23)--> (23,29)--> (29,91)--> (91,92)--> (92,127)--> (127,133)--> (133,213)--> (213,322)--> (322,538)-->**

**ListHead -->(12,23)--> (23,29)--> (29,91)--> (91,92)--> (92,93)--> (93,127)--> (127,133)--> (133,213)--> (213,322)-->**

**ListHead -->(12,23)--> (23,29)--> (29,91)--> (91,92)--> (92,93)--> (93,127)--> (127,133)--> (133,213)--> (213,322)-->**

**ListHead -->(12,23)--> (23,29)--> (29,91)--> (91,92)--> (92,93)--> (93,127)--> (127,133)--> (133,213)--> (213,322)-->**

**ListHead -->(12,23)--> (23,29)--> (29,91)--> (91,92)--> (92,93)--> (93,127)--> (127,133)--> (133,213)--> (213,322)-->**

**ListHead -->(12,23)--> (23,29)--> (29,91)--> (91,92)--> (92,93)--> (93,127)--> (127,133)--> (133,213)--> (213,322)-->**

**ListHead -->(12,23)--> (23,29)--> (29,91)--> (91,92)--> (92,93)--> (93,127)--> (127,133)--> (133,213)--> (213,322)-->**

**ListHead -->(12,23)--> (23,29)--> (29,91)--> (91,92)--> (92,93)--> (93,127)--> (127,133)--> (133,213)--> (213,322)-->**

**ListHead -->(12,23)--> (23,29)--> (29,91)--> (91,92)--> (92,93)--> (93,127)--> (127,133)--> (133,213)--> (213,322)-->**

**ListHead -->(12,23)--> (23,29)--> (29,91)--> (91,92)--> (92,93)--> (93,127)--> (127,133)--> (133,213)--> (213,322)-->**

**ListHead -->(12,23)--> (23,29)--> (29,91)--> (91,92)--> (92,93)--> (93,127)--> (127,133)--> (133,213)--> (213,322)-->**

**ListHead -->(12,16)--> (16,23)--> (23,29)--> (29,91)--> (91,92)--> (92,93)--> (93,127)--> (127,133)--> (133,213)-->**

**ListHead -->(12,16)--> (16,23)--> (23,29)--> (29,91)--> (91,92)--> (92,93)--> (93,127)--> (127,133)--> (133,213)-->**