**Lab 6 Program Documentation**

1. **Problem Statement**

We are reading a file and validating the file. We write methods to print the entire list in ascending or descending order based. This list can also be printed depending on if the user wants to print all numbers, or only the even numbers (in ascending or descending order). This program will incorporate the use of doubly linked lists. The main method will serve as a medium to take in the action from the user and to perform the action (by calling set functions) on the list of numbers accordingly. It will also log the action and information accordingly into a log text file.

1. **Requirements**
   1. **Assumptions**

* Input set through data file w/ user input on command line to select action
* Output will be handled by the command line and log text file
  1. **Specifications**
* Struct Node (will contain):
  + Number of integer data type
  + Node pointer next
  + Node pointer previous
  + Node pointer nextEven
  + Node pointer prevEven
* Print functions:
  + Print in ascending numerical order
  + Print in descending numerical order
  + Print in ascending even number order
  + Print in descending even number order
  + Print in log text file using log function
* Other functions:
  + Insert Inter Node function used to insert a number into the list and reattach the pointers accordingly
  + Remove Integer Node function used to remove a number from the list and reattach the pointers accordingly
  + Insert Even Integer Number Node function used to insert an even number into the list and reattach the pointers accordingly
  + Remove Even Integer Node function used to remove an even number from the list and reattach the pointers accordingly
  + Search function to see if list already has the attempted adding value in the list already

1. **Decomposition Diagram**

* Source Code
  + Input
    - Program takes inputs from the integer data file
  + Process
    - Functions listed under specifications can be called by user via command line
    - Program has checks in place to process request and follows them accordingly
  + Output
    - Program will print out the results onto the command line and onto the log file via usage of the print functions

|  |  |
| --- | --- |
| Source Code | |
| Responsibilities:  +printListA(node\* root)  +printListD(node\* root)  +printListAe(node\* root)  +printListDe(node\* root)  +log(string str)  +insert(node\* newNode, node\*&root)  +insertEven(node\* newNode, node\*&root)  +node \*remove(int intList, node\*&root)  +node \*removeEven(int intList, node\*&root)  +has(node\*root, int searchVal) | Collaborators: N/A |

|  |
| --- |
| Source Code |
| printListA(node\* root), printListD(node\* root), printListAe(node\* root), printListDe(node\* root), log(string str), insert(node\* newNode, node\*&root), insertEven(node\* newNode, node\*&root), node \*remove(int intList, node\*&root), node \*removeEven(int intList, node\*&root), has(node\*root, int searchVal) |
| * printListA(node\* root)   + node\* root * printListD(node\* root)   + node\* root * printListAe(node\* root)   + node\* root * printListDe(node\* root)   + node\* root * log(string str)   + string str * insert(node\* newNode, node\*&root)   + node\* newNode, node\*&root * insertEven(node\* newNode, node\*&root)   + node\* newNode, node\*&root * node \*remove(int intList, node\*&root)   + int intList, node\*&root * node \*removeEven(int intList, node\*&root)   + int intList, node\*&root * has(node\*root, int searchVal)   + node\*root, int searchVal |

1. **Test Strategy**

* Valid Data
* Invalid Data

1. **Test Plan Version 1**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| Valid Data | 1 | Test Print Ascending |  |  |  |  |
| Valid Data | 2 | Test Print Ascending Even |  |  |  |  |
| Valid Data | 3 | Test Print Descending |  |  |  |  |
| Valid Data | 4 | Test Print Descending Even |  |  |  |  |
| Valid Data | 5 | Test input method |  |  |  |  |
| Valid Data | 6 | Test remove method |  |  |  |  |
| Invalid Data | 7 | Test wrong input character |  |  |  |  |

1. **Initial Algorithm**

* printListA(node\* root)
  + read through the list of numbers and print from lowest to highest
* printListD(node\* root)
  + read through the list of numbers and print from highest to lowest
* printListAe(node\* root)
  + read through the even numbers from the list of numbers and print the even numbers from lowest to highest
* printListDe(node\* root)
  + read through the even numbers from the list of numbers and print the even numbers from highest to lowest
* log(string str)
  + opens the log text file and inputs whatever string placed in the called function’s parameter when it is called
* insert(node\* newNode, node\*&root)
  + it places the new value into the list based on the ascending order and fixes the pointers accordingly
* insertEven(node\* newNode, node\*&root)
  + it places the new value into the list based on the ascending order and fixes the pointers accordingly so that they match up with the previous and next even numbers
* node \*remove(int intList, node\*&root)
  + it removes the value selected from the list of numbers and fixes the pointers accordingly
* node \*removeEven(int intList, node\*&root)
  + it removes the value selected from the list based on the ascending order and fixes the pointers accordingly so that they match up with the previous and next even numbers
* has(node\*root, int searchVal)
  + it is a boolean that will search for a value and see if it is in the list of given numbers, then it will continue on with the program
* main()
  + it will have a greeting with directions
  + it will open the file to be streamed
  + it will check for the file being open
  + it will check for the file being empty
  + it will read the file and stream it to a list, then log every time it encounters a non positive integer
    - it will also take into account for the even numbers
  + it will have the action selector
  + it will have a goodbye message

1. **Test Plan Version 2**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| Valid Data | 1 | Test Print Ascending | ‘A’ |  |  |  |
| Valid Data | 2 | Test Print Ascending Even | ‘a’ |  |  |  |
| Valid Data | 3 | Test Print Descending | ‘D’ |  |  |  |
| Valid Data | 4 | Test Print Descending Even | ‘d’ |  |  |  |
| Valid Data | 5 | Test input method | ‘I’, 46 |  |  |  |
| Valid Data | 6 | Test remove method | ‘R’, 46 |  |  |  |
| Invalid Data | 7 | Test wrong input character | ‘U’ |  |  |  |
| Invalid Data | 8 | Test remove number that isn’t there | ‘R’, 3 |  |  |  |

1. **Code**
   1. **Source Code**

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

struct node {

int num;

node \*next = nullptr;

node \*prev = nullptr;

node \*nextEven = nullptr;

node \*prevEven = nullptr;

};

void printListA(node\* root) { //print list ascending

if (root != nullptr) {

cout << root->num << endl;

printListA(root->next);

}

}

void printListD(node\* root) { //changes nullptr and root->next for reading purposes, print list descending

if (root != nullptr) {

printListD(root->next);

cout << root->num << endl;

}

}

void printListAe(node\* root) { //print list ascending evens

if (root != nullptr) {

cout << root->num << endl;

printListAe(root->nextEven);

}

}

void printListDe(node\* root) { //print even list descending

if (root != nullptr) {

printListDe(root->nextEven);

cout << root->num << endl;

}

}

void log(string str) { //print onto console & into log.txt file

ofstream logFile;

logFile.open("log.txt");

cout << str;

logFile << str;

logFile.close();

}

void insert(node\* newNode, node \*& root)

{

if (root == nullptr) { //if list is empty

root = newNode;

}

else { //if list is not empty

if (root->num > newNode->num) { // if inserting to the left of root

newNode->next = root;

root->prev = newNode;

root = newNode;

}

else { // if inserting to the right of root

node\* search = root; // search will be newNode's prev

while (search->next != nullptr && search->next->num < newNode->num) {

search = search->next;

}

// search is now set properly

if (search->next != nullptr) { // not at the end?

newNode->next = search->next;

search->next->prev = newNode;

}

search->next = newNode;

newNode->prev = search;

}

}

}

void insertEven(node\* newNode, node \*& root)

{

if (root == nullptr) { //if list is empty

root = newNode;

}

else { //if list is not empty

if (root->num > newNode->num) { // if inserting to the left of root

newNode->nextEven = root;

root->prevEven = newNode;

root = newNode;

}

else { // if inserting to the right of root

node\* search = root; // search will be newNode's prev

while (search->nextEven != nullptr && search->nextEven->num < newNode->num) {

search = search->nextEven;

}

// search is now set properly

if (search->nextEven != nullptr) { // not at the end?

newNode->nextEven = search->nextEven;

search->nextEven->prevEven = newNode;

}

search->nextEven = newNode;

newNode->prevEven = search;

}

}

}

node\* remove(int intList, node\*&root) {

node \*search = root;

while ((search != nullptr) && (search->num != intList)) {

search = search->next;

}

if (search == nullptr) {

cout << "couldn't find your value :(\n";

return nullptr;

}

else {

if (search->prev == nullptr) { // nothing before s, root == search

if (search->next == nullptr) { // nothing after s

root = nullptr;

}

else { // stuff after s

search->next->prev = nullptr;

root = search->next;

}

}

else { // stuff before s

if (search->next == nullptr) { // nothing after s, its at the end of the list

search->prev->next = nullptr;

}

else {

search->prev->next = search->next;

search->next->prev = search->prev;

}

}

return search;

}

}

node\* removeEven(int intList, node\*&root) {

node \*search = root;

while ((search != nullptr) && (search->num != intList)) {

search = search->nextEven;

}

if (search == nullptr) {

cout << "couldn't find your value :(\n";

return nullptr;

}

else {

if (search->prevEven == nullptr) { // nothing before s, root == search

if (search->nextEven == nullptr) { // nothing after s

root = nullptr;

}

else { // stuff after s

search->nextEven->prevEven = nullptr;

root = search->nextEven;

}

}

else { // stuff before s

if (search->nextEven == nullptr) { // nothing after s, its at the end of the list

search->prevEven->nextEven = nullptr;

}

else {

search->prevEven->nextEven = search->nextEven;

search->nextEven->prevEven = search->prevEven;

}

}

return search;

}

}

bool has(node \* root, int searchVal) { //search the list and see if the value is already in the list

node \* search = root;

while (search == nullptr) {

if (search->num == searchVal) return true;

search = search->next;

}

return false;

}

int main() {

int intList;

node \*root = nullptr;

node \*evenRoot = nullptr;

ifstream myReadFile;

myReadFile.open("integer.dat");

if (!myReadFile.is\_open())

{

log("Could not open file.\n");

return -1;

}

if (myReadFile.peek() == -1) { //check file empty

log("File is empty\n");

return -1;

}

while (myReadFile >> intList) {

if (intList <= 0) {

log("Encountered non positive integer! Continuing anyway\n");

break;

}

node \* newNode = new node;

newNode->num = intList;

insert(newNode, root);

if (intList % 2 == 0) {

insertEven(newNode, evenRoot);

}

}

char action = 0;

while (action != 'Q') {

log("How to use:\n 'A': Print in Ascending Format\n 'a': Print evens in ascending format\n 'D': Print in Descending Format\n 'd': Print evens in descending format\n 'I': Insert Integer\n 'R': Remove Integer\n 'Q': Quit Program\n"); // finish with directions

cin >> action;

switch (action)

{

case 'A': //ascending

log("Printed in Ascending:\n");

printListA(root);

break;

case 'a': // asc evens

log("Printed in Ascending Evens:\n");

printListAe(evenRoot);

break;

case 'D': //descending

log("Printed in Descending:\n");

printListD(root);

break;

case 'd': // desc evens

log("Printed in Descending Evens:\n");

printListDe(evenRoot);

break;

case 'I': { //insert

log("Input integer into list\n");

int inputInt;

log("What integer would you like to insert into the list?\n");

cin >> inputInt;

if (inputInt <= 0) {

log("Encountered non positive integer! Continuing anyway\n");

}

if (has(root, inputInt)) {

log("Value already is in list.\n");

}

node \* newNode = new node;

newNode->num = inputInt;

insert(newNode, root);

if (inputInt % 2 == 0) {

insertEven(newNode, evenRoot);

}

break;

}

case 'R': //remove

log("Remove integer from list\n");

int removeInt;

log("Which value would you like to remove?\n");

cin >> removeInt;

remove(removeInt, root);

if (removeInt % 2 == 0)

{

removeEven(removeInt, evenRoot);

}

break;

default:

log("Please reread the directions and try again\n");

break;

}

}

cout << "Thank you for using this application! Hope to see you again!" << endl;

return 0;

}

1. **Updated Algorithm**

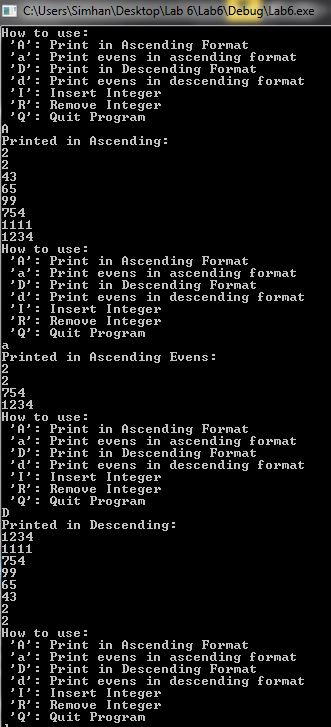
* printListA(node\* root)
  + read through the list of numbers and print from lowest to highest using the pointers that connect each number value to the next
* printListD(node\* root)
  + read through the list of numbers and print from highest to lowest using the pointers that connect each number value to the next
* printListAe(node\* root)
  + read through the even numbers from the list of numbers and print the even numbers from lowest to highest using the pointers that connect each number value to the next
* printListDe(node\* root)
  + read through the even numbers from the list of numbers and print the even numbers from highest to lowest using the pointers that connect each number value to the next
* log(string str)
  + opens the log text file and inputs whatever string placed in the called function’s parameter when it is called
  + it then read into the log text file whatever is placed in the string parameter
  + it then closes the log file
* insert(node\* newNode, node\*&root)
  + it places the new value into the list based on the ascending order and fixes the pointers accordingly
  + it will check if the list is empty
  + it will check to insert to the left of the root node first
  + then it will check to insert to the right of the root node
  + then it will also use the search node to check its status in relation to the end of the list of numbers
  + then it will make sure the rest of the pointers are reconnected accordingly to the list (regular pointers)
* insertEven(node\* newNode, node\*&root)
  + it places the new value into the list based on the ascending order and fixes the pointers accordingly so that they match up with the previous and next even numbers
  + it will check if the list is empty
  + it will check to insert to the left of the root node first
  + then it will check to insert to the right of the root node
  + then it will also use the search node to check its status in relation to the end of the list of numbers
  + then it will make sure the rest of the pointers are reconnected accordingly to the list (even pointers)
* node \*remove(int intList, node\*&root)
  + it removes the value selected from the list of numbers and fixes the pointers accordingly
  + then it will make sure the rest of the pointers are reconnected accordingly to the list (regular pointers)
* node \*removeEven(int intList, node\*&root)
  + it removes the value selected from the list based on the ascending order and fixes the pointers accordingly so that they match up with the previous and next even numbers
  + then it will make sure the rest of the pointers are reconnected accordingly to the list (even pointers)
* has(node\*root, int searchVal)
  + it is a boolean that will search for a value and see if it is in the list of given numbers
  + it will return true if the number is found in the list
  + then it will continue on with the program
* main()
  + it will have a greeting with directions
  + it will open the file to be streamed
  + it will check for the file being open
  + it will check for the file being empty
  + it will read the file and stream it to a list, then log every time it encounters a non-positive integer
    - it will also take into account for the even numbers
  + it will have the action selector
  + it will have a goodbye message

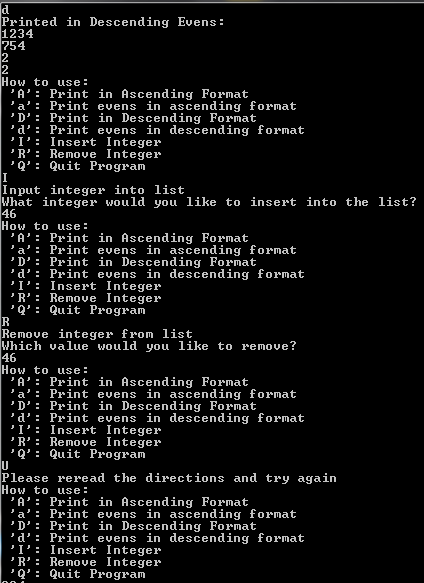
1. **Test Plan Version 3**

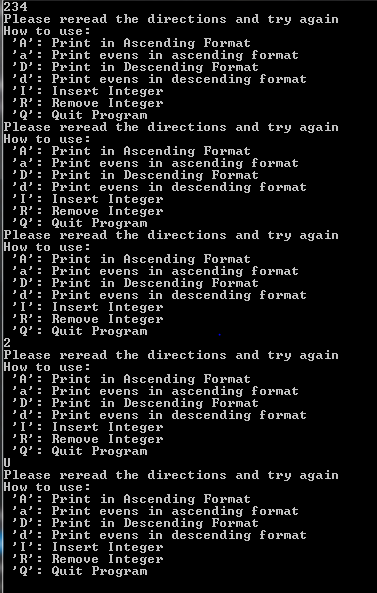
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | | Actual Output | Pass/Fail |
| Valid Data | 1 | Test Print Ascending | ‘A’ | | Printed in Ascending:  2  2  43  65  99  754  1111  1234 |  | Pass |
| Valid Data | 2 | Test Print Ascending Even | ‘a’ | | Printed in Ascending Evens:  2  2  754  1234 |  | Pass |
| Valid Data | 3 | Test Print Descending | ‘D’ | | Printed in Descending:  1234  1111  754  99  65  43  2  2 |  | Pass |
| Valid Data | 4 | Test Print Descending Even | ‘d’ | | Printed in Descending Evens:  1234  754  2  2 |  | Pass |
| Valid Data | 5 | Test input method | ‘I’, 46 | | “What integer would you like to insert into the list?” |  | Pass |
| Valid Data | 6 | Test remove method | ‘R’, 46 | | “Which value would you like to remove?” |  | Pass |
| Invalid Data | 7 | Test wrong input character | ‘U’ | | “Please reread the direction and try again” |  | Pass |
| Invalid Data | 8 | Test a number input | 234 | | It read it as three input values and prints out the directions again 3 times |  | Fail |
| Invalid Data | 9 | Test a number input | 2 | | It read the single value and printed out the directions again once |  | Pass |

1. **Screenshots**

All test cases screenshots listed below







1. **Error Log**
   1. Need to double check the log text file error output. It worked earlier but needs some fine tuning in the code.
   2. It reads each integer in an integer input (in the action sequence) as a separate action entry, and reposts the directions again accordingly. That needs fine tuning.
2. **Status**
   1. The final status of my program was that it worked fully. The only failure was due to the integer input in the action sequence, but even then it still continues with the program as though it was an invalid input. The integer entry did not affect the log text file or the program as a whole.