CIS 200 - Lab 0901

1. Problem Statement

Read in integers from a file or the user, and store them in a doubly linked list in ascending order. Also, include a doubly linked sub-list of even integers

2. Requirements

2.1 Assumptions

- Command Line Input
- File Input

2.2 Specifications

- Will Read Integers from File
- Allow User to insert integers
- Allow User to delete integers
- Ask User how to print integers
- Will Output actions to log file

3. Decomposition Diagram

- Program
    - Input
        - File Input Integers
        - User Input Integers
    - Process
        - Fill Linked List with Integers
        - Stich Together Even Sub-List
    - Output
        - Print Linked List Ascending
        - Print Linked List Descending
        - Print Even Linked List Ascending
        - Print Even Linked List Descending

4. Test Strategy

- File
- Valid Data
- Invalid Data

5. Test Plan Version 1

Test Strategy	#	Description	Input	Expected Output	Actual Output	Pass/Fail
File	1	No File				
File	2	Empty File				
File	3	Filled File				
File	4	File Contains Nega				
Valid Data	1	Empty File Print A				
Valid Data	2	Empty File Print D				
Valid Data	3	Empty Even Print A				
Valid Data	4	Empty Even Print D				
Valid Data	5	Full Print Ascend				
Valid Data	6	Full Print Descend				
Valid Data	7	Even Print Ascend				
Valid Data	8	Even Print Descend				
Valid Data	9	Add Even Int				
Valid Data	10	Add Odd Int				
Valid Data	11	Print New List				
Valid Data	12	Delete Even Int				
Valid Data	13	Delete Odd Int				
Valid Data	14	Print New List				
Valid Data	15	Quit				
Valid Data	16	Log Output				
Invalid Data	1	Add Negative Int				

6. Initial Algorithm

1. Create Struct
    a. Name: node
    b. Data:
        i. Integer Data
        ii. Node Previous
        iii. Node Next
        iv. Node Previous Even
        v. Node Previous Even
2. Create Class (For Passing In Linked Lists to Functions)
    a. Name: sortedList
    b. Public:
        i. node head
        ii. node tail
        iii. node evenHead
        iv. node evenTail
3. Create Function
    a. Name: insertIntegers
    b. Parameters: Reference sortedList, Integer Vector Input
    c. Return: VOID
    d. Method:
        i. Declare Holder Nodes For
            1. Prior
            2. Current
            3. Current Even
            4. Next
            5. Temp
        ii. Immediately Set Current to Head
        iii. While Linked List is NOT Empty
            1. Deallocate Data
        iv. Null All Applicable Pointers
        v. For Size of Vector Input
            1. IF Head is NULL
                a. Create Temporary Node
                b. Set Temp Data to Input
                c. Set Head to Temp
            2. ELSE IF Head is LESS than Input
                a. Create Temporary Node
                b. Set Temp Next to Head
                c. Set Temp Data to Input
                d. Set Head Previous to Temp

        e. Set Hea to Temp
    3. ELSE IF Head EQUALS Input
        a. Alert About Duplicate
    4. ELSE
        a. Set Current to HEAD
        b. While Current is Empty AND Current is LESS than Input
            i. Set Previous to Current
            ii. Set Current to Current Next
        c. IF Current is Empty
            i. Create Temporary Node
            ii. Set Temp to Input
            iii. Set Temp Previous to Prior
            iv. Set Prior Next to Temp
        d. ELSE IF Current EQUALS Input
            i. Alert About Duplicate
        e. ELSE
            i. Create Temporary Node
            ii. Set Temp to Input
            iii. Set Temp Next to Current
            iv. Set Temp Previous to Prior
            v. Set Prior Next to Temp
            vi. Set Current Previous to Temp
vi. Reset Current to Head
vii. For Size of Vector Input
    1. IF Current Next is Empty
        a. Set Next Previous to Current
        b. Set Current to Next
viii. Set Tail to Current
ix. Reset Current to Head
x. Reset Temp to Head
xi. While Temp is NOT Empty
    1. IF Temp is EVEN
        a. IF Even Head is Empty
            i. Set Even Head to Temp
            ii. Set Even Tail to Temp
            iii. Set Current to Even Head
        b. ELSE
            i. Set Even Tail to Temp
            ii. Set Next to Temp
            iii. Set Next Even Previous to Current
            iv. Set Current Next Even to Next
            v. Set Current to Next
    2. Set Temp to Next Temp

4. Create Function
    a. Name: printList
    b. Parameters: Reference sortedList, Character Choice
    c. Return: VOID
    d. Method:
        i. Declare Holder Nodes for
            1. Current
        ii. IF Choice is A
            1. Set Current to Head
            2. IF Current is Empty
                a. Alert List Is Empty
            3. ELSE
                a. Alert Listing Ascending Integers
                b. While Current is NOT Empty
                    i. Print Current
                    ii. Move Current to Next
        iii. IF Choice is D
            1. Set Current to Tail
            2. IF Current is Empty
                a. Alert List is Empty
            3. ELSE
                a. Alert Listing Descending Integers
                b. While Current is NOT Empty
                    i. Print Current
                    ii. Move Current to Previous
5. Create Function
    a. Name: printEvenList
    b. Parameters: Reference sortedList, Character Choice
    c. Return: VOID
    d. Method:
        i. Declare Holder Nodes for
            1. Current
        ii. IF Choice is A
            1. Set Current to Even Head
            2. IF Current is Empty
                a. Alert List Is Empty
            3. ELSE
                a. Alert Listing Ascending Integers
                b. While Current is NOT Empty
                    i. Print Current
                    ii. Move Current to Next Even
        iii. IF Choice is D
            1. Set Current to Even Tail

2. IF Current is Empty
    a. Alert List is Empty
3. ELSE
    a. Alert Listing Descending Integers
    b. While Current is NOT Empty
        i. Print Current
        ii. Move Current to Previous Even

6. Create MAIN
    a. Begin Menu Loop
        i. Print Menu
        ii. Begin Selection Phase
            1. 0 to Quit
            2. 1 to Insert Integers
                a. 0 to Exit
                b. Only Allow Positive Ints
                c. Log Action
            3. 2 to Remove Integers
                a. 0 to Exit
                b. Only Allow Positive Ints
                c. Log Action
            4. 3 to Print All Integers
                a. A to print Ascending
                b. D to print Descending
            5. 4 to Print Even Integers
                a. A to print Ascending
                b. D to print Descending
        iii. Quitting LOG THAT

7. Test Plan Version 2

Test Strategy	#	Description	Input	Expected Output	Actual Output	Pass/Fail
File	1	No File	NULL	“NO FILE”		
File	2	Empty File	integer.dat	“Empty File”		
File	3	Filled File	integer.dat	No Error		
File	4	File Contains Nega	integer.dat	“Negative Attempt”		
Valid Data	1	Empty File Print A	integer.dat	Empty		
Valid Data	2	Empty File Print D	integer.dat	Empty		

Valid Data	3	Empty Even Print A	integer.dat	Empty		
Valid Data	4	Empty Even Print D	integer.dat	Empty		
Valid Data	5	Full Print Ascend	Mixed List	Mixed List		
Valid Data	6	Full Print Descend	Mixed List	Mixed List		
Valid Data	7	Even Print Ascend	Even List	Even List		
Valid Data	8	Even Print Descend	Even List	Even List		
Valid Data	9	Add Even Int	880	Mixed List + 880		
Valid Data	10	Add Odd Int	363	Mixed List + 363		
Valid Data	11	Print New List	NULL	^^^		
Valid Data	12	Delete Even Int	52	Mixed List - 52		
Valid Data	13	Delete Odd Int	101	Mixed List - 101		
Valid Data	14	Print New List	NULL	^^^		
Valid Data	15	Quit	NULL	Quit		
Valid Data	16	Log Output	log.txt	All Actions Above		
Invalid Data	1	Add Negative Int	-27	Error		

8. Code

[source.cpp]

```
//Program Name: THIS PROGRAM MADE ME WANT TO DIE.
//Programmer Name: Arthur Aigeltinger IV
//Description: 4 lines screwed this up for more hours than I care to count.
//Date Created: 11/19/18

#include <cctype>
#include <fstream>
#include <iostream>
#include <ostream>
#include <vector>

//Dooubly Linked List Node
struct node
{
	int data;	//Contains Integer from File or User Input
```

```
	node* prev;  //Pointer to previous node
	node* next;  //Pointer to next node
	node* evenPrev;     //Pointer to previous even node
	node* evenNext;     //Pointer to next even node
};

//Class to contain Double Linked List for parameter passing
class sortedList
{
public:
	node* head = NULL;
	node* tail = NULL;
	node* evenHead = NULL;
	node* evenTail = NULL;
};

//Function Prototypes
void insertIntegers(sortedList&, std::vector<int>); //Fill Linked List with Integers
from Buffer Vector
void printList(sortedList&, char);                              //Print Linked List
in Ascending or Descending Order
void printEvenList(sortedList&, char);                    //Print Even Sub-List in
Ascending or Descending Order

int main()
{
	//Declerations
	std::vector<int> integerInputs;        //Vector of Integers for Linked List
	std::ifstream input("integer.dat");    //Input Stream Adding Initial Integers
	std::ofstream log("log.txt");          //Output Stream for Logging Actions
Performed
	char option = ' ';                           //Holder Variable for
Ascending/Descending Option
	int choice = 1;                                    //Holder Variable for Menu
Options
	int tempInput = NULL;                        //Holder Variable for filling
Vector

	log << "Program Starting" << std::endl;

	sortedList list;

	//Initial Prompt
	std::cout << "Integers being read from 'integer.dat'" << std::endl <<
std::endl;

	if (!input.is_open())     //Is It Open?
	{
```

```
		std::cout << "NO FILE FOUND" << std::endl;
		log << "NO FILE FOUND" << std::endl;
	}
	else if (input.peek() == std::ifstream::traits_type::eof())	//Is It Empty
	{
		std::cout << "File Was Empty" << std::endl;
		log << "Initial File Was Empty" << std::endl;
	}
	else	//READ THAT SHIZ
	{
		//Check for negative numbers

		while (!input.eof())
		{
			input >> tempInput;

			if (tempInput <= 0)
			{
				std::cout << "ATTEMPTED TO READ NEGATIVE FROM FILE" <<
std::endl;
				log << "File Contained Negative Integer" << std::endl;
			}
			else
			{
			integerInputs.push_back(tempInput);
			}
		}
		insertIntegers(list, integerInputs);	//Fill Linked List Initially
	}

	//Begin Menu
	do
	{
		std::cout << "Please Select an Option" << std::endl;
		std::cout << "-----------------------" << std::endl;
		std::cout << "(1) Manually Add Integers" << std::endl;
			//TODO
		std::cout << "(2) Manually Delete Integer" << std::endl;
		//TODO
		std::cout << "(3) Print Full Linked List" << std::endl;
		std::cout << "(4) Print Even Linked List" << std::endl;
			//TODO
		std::cout << "(0) QUIT" << std::endl << std::endl;

		std::cout << "Choice: ";
		std::cin >> choice;

		switch (choice)
```

```
		{
		case 0:
			std::cout << "Quitting Program" << std::endl;
			break;
		case 1:		//Adding Integers
			std::cout << "You May Only Add POSITIVE Integers" << std::endl;
			std::cout << "Enter '0' to EXIT" << std::endl << std::endl;
			log << "Began Manually Entering Integers" << std::endl;

			do
			{
				std::cout << "Integer: ";
				std::cin >> tempInput;

				if (tempInput < 0)
				{
					std::cout << "INVALID INPUT" << std::endl;
					log << "Tried Invalid Integer: " << tempInput <<
std::endl;
				}
				else if (tempInput > 0)
				{
					integerInputs.push_back(tempInput);
					log << "Valid Integer Added: " << tempInput <<
std::endl;
				}
			} while (tempInput != 0);

			log << "Exited Manual Entry" << std::endl;
			insertIntegers(list, integerInputs);
			break;
		case 2:
			std::cout << "You are removing POSITIVE Integers" << std::endl;
			std::cout << "Enter '0' to EXIT" << std::endl << std::endl;
			log << "Began Manually Deleting Integers" << std::endl;

			do
			{
				std::cout << "Integer: ";
				std::cin >> tempInput;

				if (tempInput < 0)
				{
					std::cout << "INVALID INPUT" << std::endl;
					log << "Tried Invalid Integer: " << tempInput <<
std::endl;
				}
				else if (tempInput > 0)
```

```
                        {
                                for (int i = 0; i < integerInputs.size(); i++)
                                {
                                        if (integerInputs.at(i) == tempInput)
                                        {

integerInputs.erase(integerInputs.begin() + i);
                                                integerInputs.shrink_to_fit();
                                                log << "Valid Integer Deleted: " <<
tempInput << std::endl;
                                        }
                                }

                        }
                } while (tempInput != 0);

                insertIntegers(list, integerInputs);
                break;
        case 3:

                do
                {
                        std::cout << "Print In (A)scending / (D)escending Order?"
<< std::endl;
                        std::cin >> option;
                        option = toupper(option); //Uppercase

                        switch (option)
                        {
                        case 'A':
                                printList(list, 'A');
                                log << "Printed All Integers In Ascending Order" <<
tempInput << std::endl;
                                break;
                        case 'D':
                                printList(list, 'D');
                                log << "Printed All Integers In Descending Order" <<
tempInput << std::endl;
                                option = 'A';
                                break;
                        default:
                                std::cout << "INVALID OPTION" << std::endl;
                                break;
                        }
                } while (option != 'A');

                break;
        case 4:
```

```
                    do
                    {
                           std::cout << "Print In (A)scending / (D)escending Order?"
<< std::endl;
                           std::cin >> option;
                           option = toupper(option); //Uppercase

                           switch (option)
                           {
                           case 'A':
                                  printEvenList(list, 'A');
                                  log << "Printed Even Integers In Ascending Order" <<
tempInput << std::endl;
                                  break;
                           case 'D':
                                  printEvenList(list, 'D');
                                  log << "Printed Even Integers In Descending Order"
<< tempInput << std::endl;
                                  option = 'A';
                                  break;
                           default:
                                  std::cout << "INVALID OPTION" << std::endl;
                                  break;
                           }
                    } while (option != 'A');

                    break;
             default:
                    std::cout << "INVALID MENU OPTION" << std::endl;
                    break;
             }


      } while (choice != 0);

      log << "Program Ending" << std::endl;
      log.close();

      system("pause");
      return 0;
}

//Description: Take Vector of Integers and Insert them in order into a doubly linked
list
//Pre-condition: Filled Vector and compatible linked list
//Post-Condition: Filled Linked List
void insertIntegers(sortedList& list, std::vector<int> input)
```

```
{
	node* prior;
	node* current;
	node* currentEven;
	node* next;
	node* temp;

	//ERASING FILLED LINK LIST
	current = list.head;

	while (current != NULL)
	{
		next = current->next;
		free(current);
		current = next;
	}

	list.head = NULL;
	list.tail = NULL;
	list.evenHead = NULL;
	list.evenTail = NULL;
	//END ERASING

	//BEGIN FILLING LINKED LIST
	for (int i = 0; i < input.size(); i++)
	{
		if (list.head == NULL)    //If Empty, Create Head
		{
			temp = new node;
			temp->data = input.at(i);
			temp->next = NULL;
			temp->prev = NULL;
			list.head = temp;
		}
		else if (list.head->data > input.at(i))//Insert One Greater
		{
			temp = new node;
			temp->next = list.head;
			temp->prev = NULL;
			temp->data = input.at(i);

			list.head->prev = temp;
			list.head = temp;

		}
		else if (list.head->data == input.at(i))
		{
			std::cout << "Attempted to insert a duplicate " << input.at(i) <<
```

```
"." << std::endl;
		}
		else
		{
			prior = NULL;
			current = list.head;

			while (current != NULL && (current->data < input.at(i)))
			{
				prior = current;
				current = current->next;
			}
			if (current == NULL)
			{
				temp = new node;
				temp->data = input.at(i);
				temp->next = NULL;
				temp->prev = prior;
				prior->next = temp;
			}
			else if (current->data == input.at(i))
			{
				std::cout << "Attempted to insert a duplicate " <<
input.at(i) << "." << std::endl;
			}
			else
			{
				temp = new node;
				temp->data = input.at(i);
				temp->next = current;
				temp->prev = prior;
				prior->next = temp;
				current->prev = temp;
			}
		}
	}
	//END FILLING LINKED LIST

	//BEGIN LINKING TAIL
	current = list.head;	//Reset Current to Head

	//Loop to define PREV pointers
	for (int i = 0; i < input.size(); i++)
	{
		if (current->next != NULL)
		{
		current->next->prev = current;
		current = current->next;
```

```
		}
	}

	list.tail = current;	//Define Tail as End
	//END LINKING TAIL

	//BEGIN LINKING EVEN HEAD SUBLIST
	current = list.head;
	temp = list.head;

	while (temp != NULL)
	{
		if (temp->data % 2 == 0)
		{
			if (list.evenHead == NULL)
			{
				list.evenHead = temp;
				list.evenTail = temp;
				current = list.evenHead;
			}
			else
			{
				list.evenTail = temp;
				next = temp;
				next->evenPrev = current;
				current->evenNext = next;

				current = next;
				next = NULL;
			}
		}
		temp = temp->next;
	}

	//Cover up the evidence of the murder...
	if (list.evenHead != NULL)
	{
		list.evenHead->evenPrev = NULL;
		list.evenTail->evenNext = NULL;
	}
}

//Description: Take in filled linked list an control character to print it in one of
two ways
//Pre-condition: Sorted Linked List, choice character
//Post-Condition: List Printed in Ascending or Descending Order
void printList(sortedList& list, char choice)
{
```

```
//Declare and Set Current Marker
node * current;

//If Good, Print all Integers
if (choice == 'A')
{
	//PRINT ASCENDING
	current = list.head;

	if (current == NULL)
	{
		std::cout << "List is empty" << std::endl;
	}
	else
	{
		std::cout << "List of Ascending Integers  : ";

		while (current != NULL)
		{
			std::cout << current->data;
			current = current->next;
			std::cout << " ";
		}
	}

	//Formatting
	std::cout << std::endl << std::endl;
}

if (choice == 'D')
{
	//PRINT DESCENDING
	current = list.tail;

	if (current == NULL)
	{
		std::cout << "List is empty" << std::endl;
	}
	else
	{
		std::cout << "List of Descending Integers : ";

		while (current != NULL)
		{
			std::cout << current->data;
			current = current->prev;
			std::cout << " ";
		}
```

```
		}

		//Formatting
		std::cout << std::endl << std::endl;
	}
}

//Description: Take in filled linked list an control character to print it in one of
two ways
//Pre-condition: Sorted Linked List, choice character
//Post-Condition: Even List Printed in Ascending or Descending Order
void printEvenList(sortedList& list, char choice)
{
	//Declare and Set Current Marker
	node * current;

	//If Good, Print all Integers
	if (choice == 'A')
	{
		//PRINT ASCENDING
		current = list.evenHead;

		if (current == NULL)
		{
			std::cout << "List is empty" << std::endl;
		}
		else
		{
			std::cout << "List of Even Ascending Integers  : ";

			while (current != NULL)
			{
				std::cout << current->data;
				current = current->evenNext;
				std::cout << " ";
			}
		}

		//Formatting
		std::cout << std::endl << std::endl;
	}

	if (choice == 'D')
	{
		//PRINT DESCENDING
		current = list.evenTail;

		if (current == NULL)
```

```
		{
			std::cout << "List is empty" << std::endl;
		}
		else
		{
			std::cout << "List of Even Descending Integers : ";

			while (current != NULL)
			{
				std::cout << current->data;
				current = current->evenPrev;
				std::cout << " ";
			}
		}

		//Formatting
		std::cout << std::endl << std::endl;
	}
}
```

9. Updated Algorithm

1. Create Struct
    a. Name: node
    b. Data:
        i. Integer Data
        ii. Node Previous
        iii. Node Next
        iv. Node Previous Even
        v. Node Previous Even
2. Create Class (For Passing In Linked Lists to Functions)
    a. Name: sortedList
    b. Public:
        i. node head
        ii. node tail
        iii. node evenHead
        iv. node evenTail
3. Create Function
    a. Name: insertIntegers
    b. Parameters: Reference sortedList, Integer Vector Input
    c. Return: VOID
    d. Method:
        i. Declare Holder Nodes For
            1. Prior

2. Current
3. Current Even
4. Next
5. Temp

ii. Immediately Set Current to Head
iii. While Linked List is NOT Empty
    1. Deallocate Data
iv. Null All Applicable Pointers
v. For Size of Vector Input
    1. IF Head is NULL
        a. Create Temporary Node
        b. Set Temp Data to Input
        c. Set Head to Temp
    2. ELSE IF Head is LESS than Input
        a. Create Temporary Node
        b. Set Temp Next to Head
        c. Set Temp Data to Input
        d. Set Head Previous to Temp
        e. Set Hea to Temp
    3. ELSE IF Head EQUALS Input
        a. Alert About Duplicate
    4. ELSE
        a. Set Current to HEAD
        b. While Current is Empty AND Current is LESS than Input
            i. Set Previous to Current
            ii. Set Current to Current Next
        c. IF Current is Empty
            i. Create Temporary Node
            ii. Set Temp to Input
            iii. Set Temp Previous to Prior
            iv. Set Prior Next to Temp
        d. ELSE IF Current EQUALS Input
            i. Alert About Duplicate
        e. ELSE
            i. Create Temporary Node
            ii. Set Temp to Input
            iii. Set Temp Next to Current
            iv. Set Temp Previous to Prior
            v. Set Prior Next to Temp
            vi. Set Current Previous to Temp
vi. Reset Current to Head
vii. For Size of Vector Input
    1. IF Current Next is Empty

            a. Set Next Previous to Current
            b. Set Current to Next
        viii. Set Tail to Current
        ix. Reset Current to Head
        x. Reset Temp to Head
        xi. While Temp is NOT Empty
            1. IF Temp is EVEN
                a. IF Even Head is Empty
                    i. Set Even Head to Temp
                    ii. Set Even Tail to Temp
                    iii. Set Current to Even Head
                b. ELSE
                    i. Set Even Tail to Temp
                    ii. Set Next to Temp
                    iii. Set Next Even Previous to Current
                    iv. Set Current Next Even to Next
                    v. Set Current to Next
            2. Set Temp to Next Temp
        xii. Set Even Previous of Even Head to NULL
        xiii. Set Even Next of Even Tail to NULL
4. Create Function
    a. Name: printList
    b. Parameters: Reference sortedList, Character Choice
    c. Return: VOID
    d. Method:
        i. Declare Holder Nodes for
            1. Current
        ii. IF Choice is A
            1. Set Current to Head
            2. IF Current is Empty
                a. Alert List Is Empty
            3. ELSE
                a. Alert Listing Ascending Integers
                b. While Current is NOT Empty
                    i. Print Current
                    ii. Move Current to Next
        iii. IF Choice is D
            1. Set Current to Tail
            2. IF Current is Empty
                a. Alert List is Empty
            3. ELSE
                a. Alert Listing Descending Integers
                b. While Current is NOT Empty

                i. Print Current
                ii. Move Current to Previous

5. Create Function
    a. Name: printEvenList
    b. Parameters: Reference sortedList, Character Choice
    c. Return: VOID
    d. Method:
        i. Declare Holder Nodes for
            1. Current
        ii. IF Choice is A
            1. Set Current to Even Head
            2. IF Current is Empty
                a. Alert List Is Empty
            3. ELSE
                a. Alert Listing Ascending Integers
                b. While Current is NOT Empty
                    i. Print Current
                    ii. Move Current to Next Even
        iii. IF Choice is D
            1. Set Current to Even Tail
            2. IF Current is Empty
                a. Alert List is Empty
            3. ELSE
                a. Alert Listing Descending Integers
                b. While Current is NOT Empty
                    i. Print Current
                    ii. Move Current to Previous Even
6. Create MAIN
    a. Begin Menu Loop
        i. Print Menu
        ii. Begin Selection Phase
            1. 0 to Quit
            2. 1 to Insert Integers
                a. 0 to Exit
                b. Only Allow Positive Ints
                c. Log Action
            3. 2 to Remove Integers
                a. 0 to Exit
                b. Only Allow Positive Ints
                c. Log Action
            4. 3 to Print All Integers
                a. A to print Ascending
                b. D to print Descending

5. 4 to Print Even Integers
    a. A to print Ascending
    b. D to print Descending

iii. Quitting LOG THAT

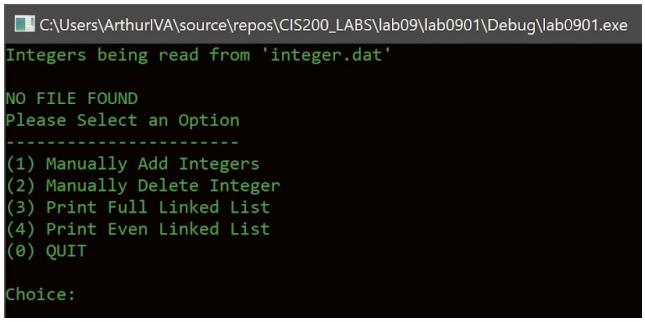
10. Test Plan Version 3

Test Strategy	#	Description	Input	Expected Output	Actual Output	Pass /Fail
File	1	No File	NULL	“NO FILE”	“NO FILE”	Pass
File	2	Empty File	integer.dat	“Empty File”	“Empty File”	Pass
File	3	Filled File	integer.dat	No Error	No Error	Pass
File	4	File Contains Nega	integer.dat	“Negative Attempt”	“Negative Attempt”	Pass
Valid Data	1	Empty File Print A	integer.dat	Empty	Empty	Pass
Valid Data	2	Empty File Print D	integer.dat	Empty	Empty	Pass
Valid Data	3	Empty Even Print A	integer.dat	Empty	Empty	Pass
Valid Data	4	Empty Even Print D	integer.dat	Empty	Empty	Pass
Valid Data	5	Full Print Ascend	Mixed List	Mixed List	Mixed List	Pass
Valid Data	6	Full Print Descend	Mixed List	Mixed List	Mixed List	Pass
Valid Data	7	Even Print Ascend	Even List	Even List	Even List	Pass
Valid Data	8	Even Print Descend	Even List	Even List	Even List	Pass
Valid Data	9	Add Even Int	880	Mixed List + 880	Mixed List + 880	Pass
Valid Data	10	Add Odd Int	363	Mixed List + 363	Mixed List + 363	Pass
Valid Data	11	Print New List	NULL	^^^	^^^	Pass
Valid Data	12	Delete Even Int	52	Mixed List - 52	Mixed List - 52	Pass
Valid Data	13	Delete Odd Int	101	Mixed List - 101	Mixed List - 101	Pass
Valid Data	14	Print New List	NULL	^^^	^^^	Pass
Valid Data	15	Quit	NULL	Quit	Quit	Pass

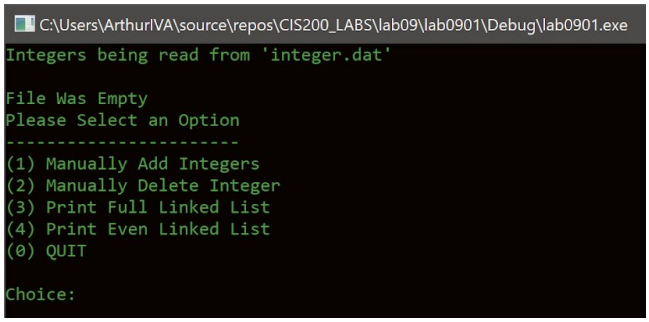
Valid Data	16	Log Output	log.txt	All Actions Above	Check Snippets	Pass
Invalid Data	1	Add Negative Int	-27	Error	Error	Pass

11. Screenshots

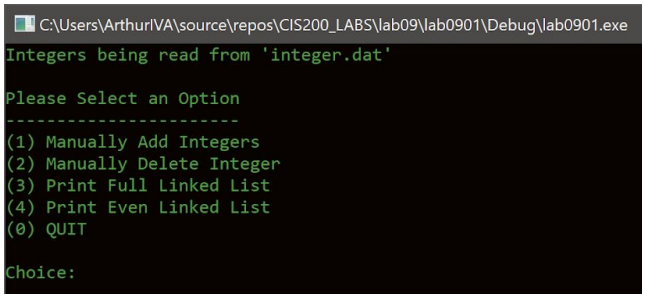
File 1



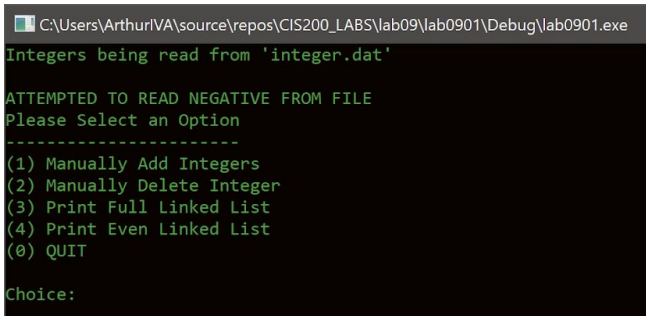
File 2



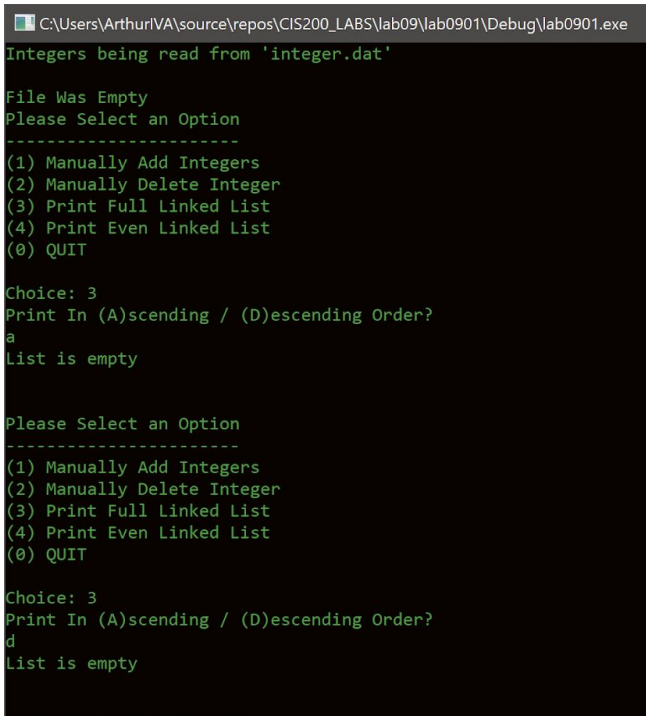
File 3



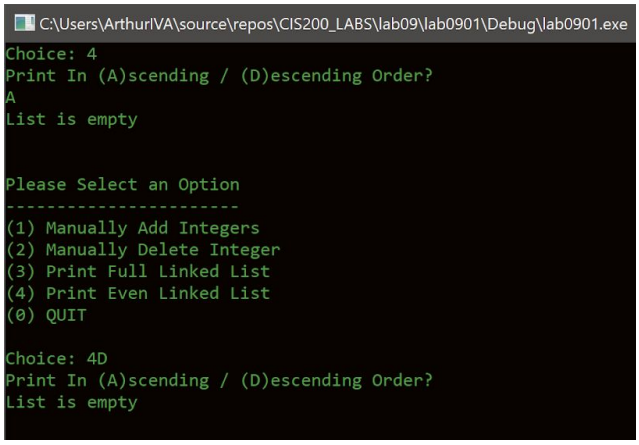
File 4



Valid 1 & 2



Valid 3 & 4



Valid 5 & 6

```
C:\Users\ArthurlVA\source\repos\CIS200_LABS\lab09\lab0901\Debug\lab0901.exe
Choice: 3
Print In (A)scending / (D)escending Order?
a
List of Ascending Integers  : 52 76 98 101 330 666 1007

Please Select an Option
----------------------
(1) Manually Add Integers
(2) Manually Delete Integer
(3) Print Full Linked List
(4) Print Even Linked List
(0) QUIT

Choice: 3
Print In (A)scending / (D)escending Order?
d
List of Descending Integers : 1007 666 330 101 98 76 52
```

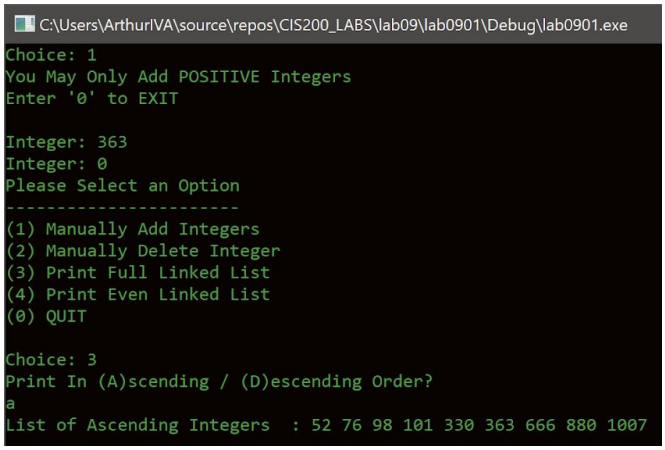
Valid 7 & 8

```
C:\Users\ArthurlVA\source\repos\CIS200_LABS\lab09\lab0901\Debug\lab0901.exe
Choice: 4
Print In (A)scending / (D)escending Order?
a
List of Even Ascending Integers  : 52 76 98 330 666

Please Select an Option
----------------------
(1) Manually Add Integers
(2) Manually Delete Integer
(3) Print Full Linked List
(4) Print Even Linked List
(0) QUIT

Choice: 4
Print In (A)scending / (D)escending Order?
D
List of Even Descending Integers : 666 330 98 76 52
```

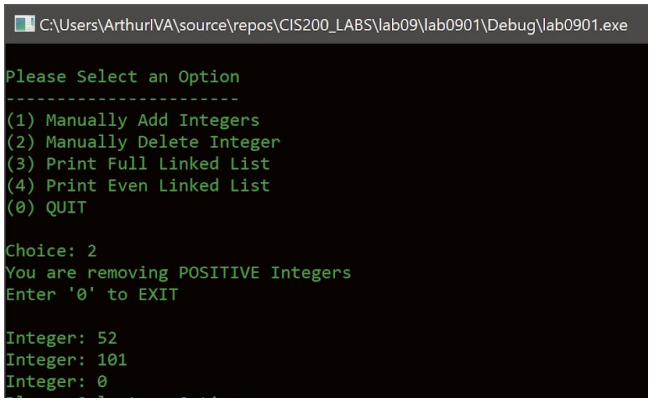
Valid 9 & 10

```
C:\Users\ArthurlVA\source\repos\CIS200_LABS\lab09\lab0901\Debug\lab0901.exe

Choice: 1
You May Only Add POSITIVE Integers
Enter '0' to EXIT

Integer: 880
Integer: 0
Please Select an Option
----------------------
(1) Manually Add Integers
(2) Manually Delete Integer
(3) Print Full Linked List
(4) Print Even Linked List
(0) QUIT

Choice: 1
You May Only Add POSITIVE Integers
Enter '0' to EXIT

Integer: 363
Integer: 0
Please Select an Option
```

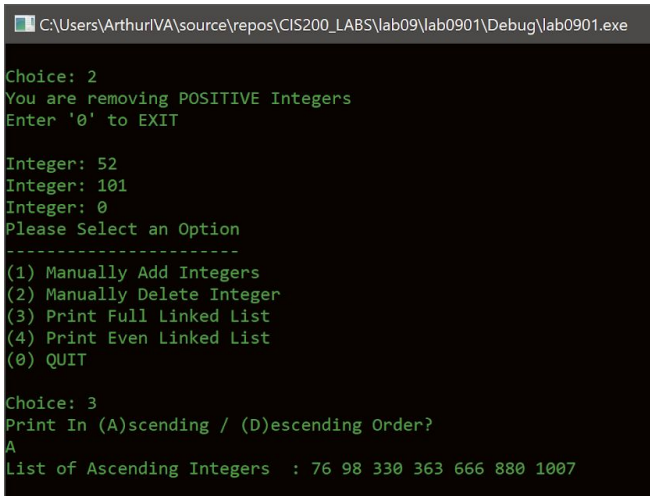
Valid 11



Valid 12 & 13



Valid 14



Valid 15

```
C:\Users\ArthurlVA\source\repos\CIS200_LABS\lab09\lab0901\Debug\lab0901.exe
You are removing POSITIVE Integers
Enter '0' to EXIT

Integer: 52
Integer: 101
Integer: 0
Please Select an Option
----------------------
(1) Manually Add Integers
(2) Manually Delete Integer
(3) Print Full Linked List
(4) Print Even Linked List
(0) QUIT

Choice: 3
Print In (A)scending / (D)escending Order?
A
List of Ascending Integers  : 76 98 330 363 666 880 1007

Please Select an Option
----------------------
(1) Manually Add Integers
(2) Manually Delete Integer
(3) Print Full Linked List
(4) Print Even Linked List
(0) QUIT

Choice: 0
Quitting Program
Press any key to continue . . .
```

Valid 16

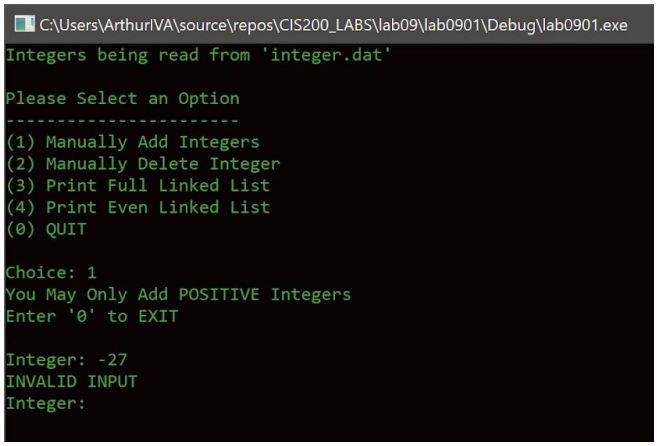
log.txt - Notepad

File Edit Format View Help

```
Program Starting
Printed All Integers In Ascending Order
Printed All Integers In Descending Order
Printed Even Integers In Ascending Order
Printed Even Integers In Descending Order
Began Manually Entering Integers
Valid Integer Added: 880
Exited Manual Entry
Began Manually Entering Integers
Valid Integer Added: 363
Exited Manual Entry
Printed All Integers In Ascending Order
Began Manually Deleting Integers
Valid Integer Deleted: 52
Valid Integer Deleted: 101
Printed All Integers In Ascending Order
Program Ending
```

Windows (Ln 16, Col 4 100%

Invalid 1



12. Error Log

Error Type (Logic/Runtime)	Cause of Error	Solution to Error
Logic	Missing certain checks when linking together all of the even integers with the evenNext and evenPrev pointers within node*	Was advised by another student on which checks were missing and subsequently added them.

13. Status

Program is operational with current specifications and assumptions.