1. **Program Statement**

This program reads in numbers from a file and write them on the screen in two orders according to user’s choice. It also allows user to manipulate data in the file along with displaying sub linked list, which is the only even integer numbers

1. **Requirements**
   1. **Assumptions**
      1. User types in the file name with .txt at the end
   2. **Specifications**
      1. Functions for delete, insert, display, find, ascend, descend, displaying sub list.
      2. Struct with next and previous nodes inside along with int data
      3. User input of a file name
      4. Check if file exists
2. **Decomposition Diagram**

|  |  |  |
| --- | --- | --- |
| **Main** | | |
| **Input** | **Process** | **Output** |
| File name | Check if file exists | If it exists, output it exists. Else say it does not |
| Menu selection of all the functions and manipulation | According to a specific menu selection, function call is made |  |
| Quit |  | Thank you message |

1. **Test Strategy**
   1. **Valid Data**
   2. **Invalid Data**
2. **Test Plan Version 1**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| Valid | 1 | User types in correct file name that exists |  |  |  |  |
| Valid | 2 | User types in correct menu selection in bounds |  |  |  |  |
| Invalid | 1 | User types in incorrect file name that does not exist |  |  |  |  |
| Invalid | 2 | User types in incorrect menu selection that is not in bounds |  |  |  |  |

1. **Initial Algorithm**
   1. In main() class
      1. Make a struct outside of main
         1. Have variables of data, struct node to point to next, struct node to point to previous
      2. Make an instance of class DoublyLinkedList
      3. Do this while keep looping variable is 0
         1. Ask for file name and store it in a variable
         2. Open the file and if it doesn’t open, output file does not exist and loop until found
         3. If file is empty, say it is empty
         4. Else set keep looping variable to 1
      4. If file is open, do this while not end of file
         1. Store the next number in a variable
         2. Check to see if the variable is greater than 0.
         3. If it is, then put it in the object and print out the variable’s value
         4. If it is negative, say this is negative
      5. While true, display the menu
      6. Using switch case, call particular function using the cases
   2. Class *DoublyLinkedList*
      1. Make the same struct again
      2. *PutIn* class
         1. Have nodes pointing to variable s and temp
         2. Using linked lists, check to see if next position is null
         3. If it is null, then set temp’s last value to null and head’s next value to current value of temp
         4. Else set variable s to head’s next value
            1. And while s’s next value is not null, set s to its next value and set its next value to temp
            2. Set temp’s previous value to 5
      3. *TakeOut* class
         1. If head’s data is equal to the value passed in
            1. Set temp to head and head to head’s next
            2. Head’s previous should be null
            3. Print out number is deleted
         2. While a variable’s next’s next is not null
            1. If variable’s next’s data equals the passed in value

Set temp to variable’s next, variable’s next value equals null.

* + - * 1. If passed in value is less than 0

Print out negative numbers not allowed

* + - * 1. Else , print out number is not in the list
    1. *Find* class
       1. While variable x is not null
          1. If a variable’s data is the value passed in
          2. Return the counter
          3. Else set counter to 1 and the variable’s next value to variable
    2. *Display* class
       1. If head is null, say the list is empty
       2. Ask user which direction they want to move
          1. If ascending, call function ascending
          2. If descending, call function descend
    3. *Ascend* class
       1. Check to see if list is empty
       2. If not, using a for loop, go though the list and print out the data in ascending order
    4. *Descend* class
       1. Check to see if list is empty
       2. If not, using a for loop, go through the list and print out the data in descending order

1. **Test Plan Version 2**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| Valid | 1 | User types in correct file name that exists | “Numbers.txt” | File found |  |  |
| Valid | 2 | User types in correct menu selection in bounds | 5 | Quit, thank you message |  |  |
| Invalid | 1 | User types in incorrect file name that does not exist | “Number.txt” | File does not exist |  |  |
| Invalid | 2 | User types in incorrect menu selection that is not in bounds | 0 | Invalid menu selection |  |  |

1. **Code**

**CIS200\_Lab\_8.cpp**

#include "DoublyLinkedList.h"

#include "stdafx.h"

#include<iostream>

#include<fstream>

#include <string>

using namespace std;

struct node

{

int data;

struct node \*next;

struct node \*prev;

}\*head, \*tail;

int main()

{

DoublyLinkedList myList;

int choice, number;

string fileName;

ifstream inFile;

int keepLooping = 0;

while (keepLooping == 0) {

cout << "Enter file name" << endl;

cin >> fileName;

inFile.open(fileName);

if (!inFile.is\_open()) {

cout << "The file does not exist\nTry again" << endl;

}

else if (inFile.peek() == ifstream::traits\_type::eof()) {

cout << "the file is empty" << endl;

inFile.close();

}

else {

keepLooping = 1;

}

}

if (inFile.is\_open())

{

int c;

while (!inFile.eof())

{

inFile >> c;

if (c >= 0)

myList.PutIn(c);

cout << c << endl;

if (c<0) {

cout << "This is negative" << endl;

}

}cout << "The list is created";

}

while (1)

{

cout << "Welcome, human!" << endl;

cout << "1.New Number" << endl;

cout << "2.Delete" << endl;

cout << "3.Display List" << endl;

cout << "4.Display even list" << endl;

cout << "5.Exit" << endl;

cout << "Enter your choice : ";

cin >> choice;

switch (choice)

{

case 1:

{

cout << "Enter the new number: ";

cin >> number;

int k = myList.find(number);

if (k == 1)

{

if (number >= 1)

{

myList.PutIn(number);

}

else

cout << "Invalid number" << endl;

}

else

cout << "Value is already in the list " << endl;

cout << endl;

break;

}

case 2:

if (head == NULL)

{

cout << "The list empty" << endl;

break;

}

cout << "Enter the number to delete: ";

cin >> number;

myList.TakeOut(number);

cout << endl;

break;

case 3:

{

myList.display(myList);

cout << endl;

break;

}

case 4:

{

myList.displaySubList(myList);

cout << endl;

break;

}

case 5:

cout << endl << "Thank you and see you later, user" << endl;

exit(1);

default:

cout << "Please enter a valid choice" << endl;

}

}

return 0;

}

**DoublyLinkedList.cpp**

#include "DoublyLinkedList.h"

DoublyLinkedList::DoublyLinkedList()

{

head->next = NULL;

}

void DoublyLinkedList::PutIn(int value)

{

node \*s, \*temp;

temp = new node;

temp->data = value;

temp->next = NULL;

if (head->next == NULL)

{

temp->prev = NULL;

head->next = temp;

}

else {

s = head->next;

while (s->next != NULL)

{

s = s->next;

s->next = temp;

temp->prev = s;

}

}

count++;

}

void DoublyLinkedList::TakeOut(int value)

{

node \*temp, \*z;

if (head->data == value)

{

temp = head;

head = head->next;

head->prev = NULL;

cout << "Number Deleted" << endl;

free(temp);

count--;

}

z = head;

while (z->next->next != NULL)

{

if (z->next->data == value)

{

temp = z->next;

z->next = temp->next;

temp->next->prev = z;

cout << "Number Deleted" << endl;

free(temp);

count--;

}

z = z->next;

}

if (z->next->data == value)

{

temp = z->next;

free(temp);

z->next = NULL;

cout << "Number Deleted" << endl;

count--;

}

if (value < 0) {

cout << endl;

cout << "NO NEGATIVE NUMBERS IN THE LIST , THERFORE YOU CAN NOT DELETE A NEGATIVE NUMBER" << endl;

}

else

cout << endl;

cout << "THE NUMBER " << value << " IS NOT PRESENT IN THE LIST " << endl;

}

int DoublyLinkedList::find(int value)

{

node \*z = head;

int k = 1;

while (z != NULL)

{

if (value == (z->data))

{

k = 0;

return k;

}

else

{

k = 1;

z = z->next;

}

}

return k;

}

void DoublyLinkedList::display(DoublyLinkedList object)

{

node \*temp;

if (head == NULL)

{

cout << "The List is empty" << endl;

return;

}

char order;

cout << "In which Direction you want to display list(A/D)" << endl;

cin >> order;

if (order == 'A')

{

object.Ascend();

}

if (order == 'D')

{

object.Descend();

}

temp = head;

cout << "The List is :" << endl;

while (temp != NULL)

{

cout << temp->data << " ";

temp = temp->next;

}

}

void DoublyLinkedList::Ascend()

{

node \*temp, \*z;

int value;

if (head == tail && head == NULL)

{

cout << "The List is empty " << endl;

return;

}

z = head;

for (int i = 0; i < count; i++)

{

temp = z->next;

while (temp != NULL)

{

if (z->data > temp->data)

{

value = z->data;

z->data = temp->data;

temp->data = value;

}

temp = temp->next;

}

z = z->next;

}

}

void DoublyLinkedList::Descend()

{

node \*temp, \*z;

int value;

if (head == tail && head == NULL)

{

cout << "The List is empty " << endl;

return;

}

z = head;

for (int i = 0; i < count; i++)

{

temp = z->next;

while (temp != NULL)

{

if (z->data < temp->data)

{

value = z->data;

z->data = temp->data;

temp->data = value;

}

temp = temp->next;

}

z = z->next;

}

}

void DoublyLinkedList::displaySubList(DoublyLinkedList object)

{

node \*z;

if (head == NULL)

{

cout << " The List is empty" << endl;

return;

}

char order;

cout << "In what order you want to display the Sublist(A/D)" << endl;

cin >> order;

if (order == 'A') {

object.Ascend();

z = head;

cout << "The List is :" << endl;

while (z != NULL)

{

if (z->data % 2 == 0) {

cout << z->data << " ";

}

z = z->next;

}

cout << endl;

}

if (order == 'D') {

object.Descend();

z = head;

cout << "The List is :" << endl;

while (z != NULL)

{

if (z->data % 2 == 0) {

cout << z->data << " ";

}

z = z->next;

}

cout << endl;

}

}

**DoublyLinkedList.h**

#pragma once

#include<iostream>

#include<fstream>

#include <string>

#include "stdafx.h"

using namespace std;

struct node

{

int data;

struct node \*next;

struct node \*prev;

}\*head, \*tail;

class DoublyLinkedList

{

public:

DoublyLinkedList();

void PutIn(int value);

void TakeOut(int value);

int find(int value);

void display(DoublyLinkedList object);

void Ascend();

void Descend();

void displaySubList(DoublyLinkedList object);

int count = 0;

};

1. **Updated Algorithm**
2. In main() class
   * 1. Make a struct outside of main
        1. Have variables of data, struct node to point to next, struct node to point to previous
     2. Make an instance of class DoublyLinkedList
     3. Do this while keep looping variable is 0
        1. Ask for file name and store it in a variable
        2. Open the file and if it doesn’t open, output file does not exist and loop until found
        3. If file is empty, say it is empty
        4. Else set keep looping variable to 1
     4. If file is open, do this while not end of file
        1. Store the next number in a variable
        2. Check to see if the variable is greater than 0.
        3. If it is, then put it in the object and print out the variable’s value
        4. If it is negative, say this is negative
     5. While true, display the menu
     6. Using switch case, call particular function using the cases
3. Class *DoublyLinkedList*
   * 1. Make the same struct again
     2. *PutIn* function
        1. Have nodes pointing to variable s and temp
        2. Using linked lists, check to see if next position is null
        3. If it is null, then set temp’s last value to null and head’s next value to current value of temp
        4. Else set variable s to head’s next value
           1. And while s’s next value is not null, set s to its next value and set its next value to temp
           2. Set temp’s previous value to 5
     3. *TakeOut* function
        1. If head’s data is equal to the value passed in
           1. Set temp to head and head to head’s next
           2. Head’s previous should be null
           3. Print out number is deleted
        2. While a variable’s next’s next is not null
           1. If variable’s next’s data equals the passed in value

Set temp to variable’s next, variable’s next value equals null.

* + - * 1. If passed in value is less than 0

Print out negative numbers not allowed

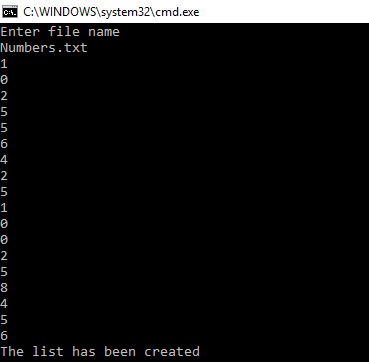
* + - * 1. Else , print out number is not in the list
    1. *Find* function
       1. While variable x is not null
          1. If a variable’s data is the value passed in
          2. Return the counter
          3. Else set counter to 1 and the variable’s next value to variable
    2. *Display* function
       1. If head is null, say the list is empty
       2. Ask user which direction they want to move
          1. If ascending, call function ascending
          2. If descending, call function descend
    3. *Ascend* function
       1. Check to see if list is empty
       2. If not, using a for loop, go though the list and print out the data in ascending order
    4. *Descend* function
       1. Check to see if list is empty
       2. If not, using a for loop, go through the list and print out the data in descending order
    5. *displaySunList* function
       1. Check to see if list is empty
       2. If is not, then ask user what order do they want to display
       3. If ascending
          1. Call the method ascent to make this list in ascending order
          2. Check each value in the list and module is with 2. If it equals 0, then print that value out
       4. If descending
          1. Call the method descend to make this list in descending order
          2. Check each value in the list and module it with 2. If it equals 0, then print the value out

1. **Test Plan Version 3**

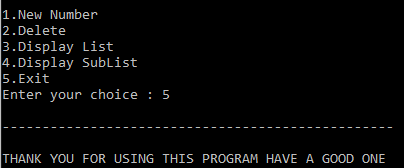
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| Valid | 1 | User types in correct file name that exists | “Numbers.txt” | File found | Reads in the file | Pass |
| Valid | 2 | User types in correct menu selection in bounds | 5 | Quit, thank you message | Thank you message | Pass |
| Invalid | 1 | User types in incorrect file name that does not exist | “Number.txt” | File does not exist | File does not exist. Try again | Pass |
| Invalid | 2 | User types in incorrect menu selection that is not in bounds | 0 | Invalid menu selection | Please enter a valid choice | Pass |

1. **Screenshots**

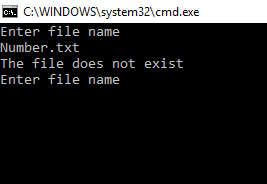
Valid Test Case 1:



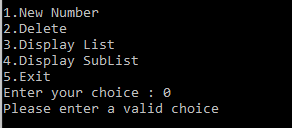
Valid Test Case 2:



Invalid Test Case 1:



Invalid Test Case 2:



1. **Status**