Spencer Turenne

Criterion C

Techniques:

Class:

I am using a linked list. This linked list uses a class called Node to create the list itself. This allows me to easily add, remove, and edit files as well as change the order or find one or two files as stated in my success criteria. The class I created was the IBIO class. This uses the IBIO input and output methods to print and receive input from the user.

If/Then Statements:

This is used often in the program itself. Most notably it is used to show or not show who the loan officer was as the loan officer is only to be shown if my client represented the bank. Therefore an if/then statement is used to determine if the bank was represented. An example is when a new file is added to the database, as seen below. It checks to see if he is representing 3,4, or above (although 4 is the highest) as both of these numbers correspond to representing the bank in some way. If the bank is represented then it asks for the loan officer's name. If not then the loan officer is not needed.

```
temp.Lawyer = out.input("Please enter the lawyer's name:"); //gets info
if (temp.Representing > 2) { //if rep bank gets loan officer
  temp.LoanOfficer = out.input("Please enter the loan officer's name:");
}
return head;
```

Select/Case (Switch):

These are utilized quite a bit within the program. The best examples are when there are menus for the user to enter a choice. I then use this statement to see what choice the user enters.

```
do { //while input != 7
    Out(head);
    ib.output(Divider()); //outputs the divider (----------)
    ib.output("Menu 1:");
    ib.output("    1)Add a File");
    ib.output("    2)See all the categories of one file");
    ib.output("    3)Sort");
    ib.output("    4)Search");
    ib.output("    5)Edit a File");
    ib.output("    6)Delete a file");
    ib.output("    7)Save and quit");
    input = ib.inputInt("Choice:"); //gets choice

    switch (input) { //switch for user choice from menu one
        case 1: //add
            AddFile(head); //calls addFile method and sends node head
            break;
        case 2: //see all
            input = ib.inputInt("Please enter the file number you want to see all of:"); //gets file number
            SeeAll(head, input); //sends the node head and what file number
            ib.input("Press enter to continue:"); //pauses
            input = -1; //resets input (incase it is 7 and it would stop loop)
            break;
        case 3: //sort
            ib.output(Divider());
            ib.output("Menu 2:");
            ib.output("    1)Buyer");
            ib.output("    2)Seller");
            ib.output("    3)Date");
            ib.output("    4)Address");
            ib.output("    5)Representing");
            ib.output("    6)Email");
            ib.output("    7)Phone");
            ib.output("    8)Buyers Social");
            ib.output("    9)Sellers Social");
            ib.output("    10)File Location");
            ib.output("    11)Lawyer"); //outputs menu
            input = ib.inputInt("What would you like to sort by:"); //gets how user wants to sort
            head = Sort(head, input); //calls sort method and sends head and input (input = how user wants to sort)
            input = -1; //resets input
            break;
        case 4: //search
            ib.output(Divider());
```

Sorting:

This is used in order to fulfill the success criteria. This sorting technique is bubble sorting. For more on sorting see the algorithms. This is an example.

```
        temp.Address = sort.Address;
        temp.Representing = sort.Representing;
        temp.Email = sort.Email;
        temp.Phone = sort.Phone;
        temp.SocialBuyer = sort.SocialBuyer;
        temp.SocialSeller= sort.SocialSeller;
        temp.Location= sort.Location;
        temp.Lawyer = sort.Lawyer;
        temp.LoanOfficer = sort.LoanOfficer; //sets the temp (blank)to the current node

        sort.Buyer = sort.next.Buyer;
        sort.next.Buyer = temp.Buyer;
        sort.Seller = sort.next.Seller;
        sort.next.Seller = temp.Seller;
        sort.Date = sort.next.Date;
        sort.next.Date = temp.Date;
        sort.Representing = sort.next.Representing;
        sort.next.Representing = temp.Representing;
        sort.Email = sort.next.Email;
        sort.next.Email = temp.Email;
        sort.Phone = sort.next.Phone;
        sort.next.Phone = temp.Phone;
        sort.SocialBuyer = sort.next.SocialBuyer;
        sort.next.SocialBuyer = temp.SocialBuyer;
        sort.SocialSeller = sort.next.SocialSeller;
        sort.next.SocialSeller = temp.SocialSeller;
        sort.Location = sort.next.Location;
        sort.next.Location = temp.Location;
        sort.Lawyer = sort.next.Lawyer;
        sort.next.Lawyer = temp.Lawyer;
        sort.LoanOfficer = sort.next.LoanOfficer;
        sort.next.LoanOfficer = temp.LoanOfficer;
```

Files:

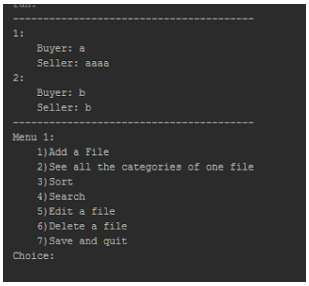
Because the database is saved to a file it is import for the data to be read from a file. It also saves to a file at the end. When the program starts it reads from the file and creates the database using the information within.

Linked List:

The database uses a linked list structure to keep the files of my client. I created a Node class and it has all the information the client needs for a file. Then I used this class to create a variable of a Node and make the list itself.

Output:

Using the IBIO output a very simple and easy to understand display is created. This makes it easy for my client to use the database itself. I indent where necessary and have a divider to mark the end and beginning of sections.



(See next page)

Outside Sources:

IBIO:

I utilized the IBIO input and output methods (found online) in order to make my program match the IB guidelines. This also was a better alternative to the other method I have utilized in the past as it leads to less errors and code breaking.

Mock Files from Client:

I used mock files from my client to help simulate a real experience within my program while testing. This is not used in the final product, however it is a mock of what will be put in eventually.

Algorithms:

Delete:

I used two if/then conditional statements to determine how a node should be deleted. It checks to see if it is not the head node, this contains an else in case it is the head. Then it checks to see if it was the end of the list. Then it deletes the node accordingly.

```
if (temp.prev != null) { //if its not the head
    temp.prev.next = temp.next; //deletes by making p
} else { //if it is the head
    head = head.next; //moves head to head.next
    head.prev = null; //deletes the original head
}
if (temp.next != null) { //if it is not the end
    temp.next.prev = temp.prev; //deletes it again
}
```

Sorting:

Sorting was included in my database. I utilized a bubble sort algorithm. This algorithm sorts the list in the was specified by the user. This one part of the sorting has two loops. One, the first, goes while swapDone which only sets swapDone to false. Then after the 2nd loop it makes the sort node equal the head. The 2nd loop moves sort along and compares it to the next node in the list. If it needs to be sorted (in this case if buyer is not greater than the next buyer) then it sorts it.

```
while (swapDone) {

        swapDone = false; //sets it to false
    while (sort != null) { //while the node is not null

        switch (how) { //switch for how they want to sort
            case 1: //buyer
                if (sort.next != null && sort.Buyer.compareTo(sort.next.Buyer) > 0) {
                    temp.Buyer = sort.Buyer;
```

Search:

The search algorithm is used for searching by many categories. Therefore it compares the individual variable corresponding to each item to “” (or nothing) as this is the base state. It it is not 0 (meaning there is something) then it also checks to see if the variable and the current node at that item are not equal. If both conditions are met it needs to be deleted from the searched list. It then checks all the nodes in the list and deletes them when needed. It also checks each node to every item the user searched by using the same method.

```
if (b.compareTo("") != 0 && b.compareTo(head.Buyer) != 0) {
    if (head.prev != null) {
        head.prev.next = head.next;
        head.next.prev = head.prev;
        head = head.next;
    } else {
        head = head.next;
        head.prev = null;
    }
    used = true;
}
```