Author 1:	Jonathan Mabery
Email 1:	jmabery@uco.edu
Author 2:	Andrew Aprile
Email 2:	aaprile@uco.edu
Course:	ADS. CMSC3613. CRN10040
Assignment	Project 1. p01

3. Project Progress. Everything here is also in the file ProjectPlan.docx

1. Project Objectives

Implement a program that contains a record of people using a list structure. Use either contiguous or linked list implementation. Insert a database from a text file. Text file has a list of last names, first names, and ID numbers.

2. Task Breakdown

Task #	Description	Status	Assigned To
1	Understand class structure and data model	Done	Both
2	Implement core functionalities	Undergoing	Both
2-1	Insert records in alphabetical order	Done	Jonathan Mabery
2-2	Print the contents of the record	Done	Andrew Aprile
2-3	Support search functionality	Done	Both
3	Error handling and input validation	Done	Andrew Aprile
4	Test application	Done	Both
5	Finalize code and documentation	Done	Both

3. Timeline

Make a tentative timeline to manage the project progress.

Date	Task	Notes
9/13/25	Start Project. Setup GitHub	Done
9/15/25	Complete task 1	Done
9/16/25	Complete task 2	Done
9/20/25	Complete task 3	Done
9/20/25	Testing and debugging	Done
9/21/25	Submit project, code, report, documentation	Done

4. Technical Details

- **Language:** C++ (Standard: e.g., C++11 or C++14)
- **Development Environment:** [e.g., Windows 11, Visual Studio Code, g++, etc.]
- **Key Concepts Used:**

- OOP (Classes/Objects)
- File I/O
- List
- Sorting
- String matching and comparison
- Efficient manipulation of a customized data type

5. File Structure

```
/p01_group17.zip
│
├──/Code
        ├── List.h
        ├── utility.h
        ├── main.cpp
        ├── utility.cpp
        ├── makefile
        ├── data.txt
        ├── people.txt
        ├── smallpeople.txt
        ├── input_simulate.txt
├── p01_report.docx
├── p01-report.pdf
├── ProjectPlan.docx
├── ProjectPlan.pdf
└── README.md
```

6. Testing Plan

- **Manual test cases**
  - Create test cases
    - User selects invalid menu option.
      - Solution:
      - Display error message.
      - Display menu options again
    - User selects “Import List from File”. Then user inputs a file name that doesn’t exist
      - Solution:
      - Display message that says “Added 0 entries”
      - Display menu options again
    - User wants to display or search a record before adding entries
      - Solution:
      - Display a message saying that the record is empty
      -
      ```
      2
      Record is empty! Enter a file to continue.
      ```
    - When the user selects the search list menu option and doesn’t enter the option for id or first last name.
      - Solution:
      - Display error message and ask again for input
      -
      ```
      3
      Do you want to search by ID or by First and Last names?
      Enter "id" or "fl" : aa
      Invalid input!.
      Enter "id" or "fl" :
      ```
  - Input scenarios with expected vs actual output

- **Test in a standard environment**
  - Set up GitHub Actions
  - Compile the code
  - Optional: test cases
    - Duplicate entries
    - Invalid user input

7. Documentation

- **README.md** with:
  - Project description
  - How to compile and run
  - Usage instructions
- Code comments for clarity

8. Completion Checklist

Item	Done (Y / N)
All features implemented	Y
Code compiles without errors	Y
Optional: all test cases pass	Y
Proper documentation	Y
Code is commented	Y
Submitted on time	Y

4. Discussion Questions

Q1:Which version of the list implementations is better: contiguous or linked? Please explain.

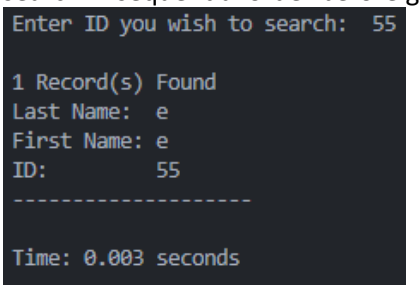
Contiguous is better when you have a known number of records, want to access them quickly, and don't care about reserving a lot of memory.

Linked is better when you don't know the number of records beforehand, don't care about quick access, and want to preserve memory (memory is only used when a new record is created).

We used the contiguous implementation.

With our code, we started the clock when the user entered an ID or first and last name, then displayed the amount of time it took to find the record once found.

As shown here, we imported the data.txt file that the instructor provided and then searched for the last record with ID=55. Even though it is at the end of the list, it was found in 0.003 seconds. With a linked implementation, it would have to search in sequential order before going to the end of the list.



Q2: How to understand the function: void traverse(void (*visit) (List_entry &))?

This sends the address of the visit function to the list traverse function. The traverse function then calls the visit function to print out the data in the list that called it.
In main, we called the .traverse function and passed the function pointer “visit”.

```
107             record_list.traverse(visit);
```

List.h then iterated through the a loop using the record list counter, and calls the visit function for each pass

```
for (int i = 0; i < count; i++)
   (*visit)(entry[i]);
```

The visit function is declared in utility.h

```
13    void visit(Personal_record &r);
```

And defined in utility.cpp

```
12    void visit(Personal_record &r){
13       cout << left << setw(12) << "Last Name: " << r.last_name << endl;
14       cout << left << setw(12) << "First Name: " << r.first_name << endl;
15       cout << left << setw(12) << "ID: " << r.ID << endl;
16       cout << "--------------------" << endl;
17    }
```

Q3: What’s the search algorithm you used in Item 3? Briefly explain why it’s efficient.

We used a simple linear search algorithm. We made a for loop, limiting the loop to the size of the record, and searched for each element in the record.
This is really the only way to do it if the list is unsorted as in the case of ID.

```
145             for (int i=0; i<record_list.size(); ++i){
146                if((record_list.retrieve(i, retrieve_record)) == success && retrieve_record.ID == search_ID){
147                      found = true;
148                      matching_records.insert(0, retrieve_record);
149                      break;
150                }
151             }
```

For searching based on last name, we could have implemented a binary search. This is because we inserted each element into the list in alphabetical order.
If the list was much larger, then using a binary search would be something that would be better.

Q4: Can you directly use the comparison operators (e.g., >, <, >=, etc.) to compare instances of Personal_record? If not directly supported, can you provide a solution to make it possible? Briefly explain your design.

Our Personal_record design does not allow the use of those operators but it would be straightforward to make it so. We would have to implement operator overloading. This allows us to directly compare characteristics of our record without using dot notation.
For example, if we made two Personal_record ojects named ‘a’ and ‘b’, assigned the number 5 to a, and 6 to b, we can use the less than symbol to ask the program if a is less than b.
If those numbers were employee numbers representing the order in which they were hired, we could then see which employee was hired first.

Personal_record a(“Joe”, “Smith”, 5);
Personal_record b(“Mary”, “Jane”, 6);

```
if(a<b){
        cout << “Joe was hired before Mary” << endl;
} else{
        cout << “Mary was hired before Joe” << endl;
}
```

We would write the code in our Personal_record structure

```
bool operator<(const Personal_record& other) const {
        return ID < other.ID;
}
```

5. Screenshots of the test run

Make and Execute

```
Laptop@Andrew MINGW64 ~/Desktop/Visual Studio Code Programs/3. Fall 2025 ADS Class/Assignments/Project_1_Github/p01-ads/Code (main)
$ make
rm -f *.o *.exe
g++ -c -g main.cpp
g++ -c -g utility.cpp
g++ -o p01 main.o utility.o

Laptop@Andrew MINGW64 ~/Desktop/Visual Studio Code Programs/3. Fall 2025 ADS Class/Assignments/Project_1_Github/p01-ads/Code (main)
$ ./p01.exe

**********
Menu:
1. Import List from File
2. Display List
3. Search List
x. Exit
**********
```

Enter ‘1’ and press enter

Enter ‘data.txt’ and press enter

The data.txt file has two duplicates. For each one, it prints out the line “Duplicate record found. Discarding.

```
1

Enter Data File Name:
data.txt
Duplicate record found. Discarding.
Duplicate record found. Discarding.
Added 7 entries

**********
Menu:
1. Import List from File
2. Display List
3. Search List
x. Exit
**********
```

Enter ‘2’ and press enter.

This displays the record in alphabetical order with each record separated by dashes -.

It also prints the number of records in our list.

```
2
----Display Record----
Last Name:  a
First Name: a
ID:         11
--------------------
Last Name:  bb
First Name: bbbb
ID:         22
--------------------
Last Name:  c
First Name: c
ID:         31
--------------------
Last Name:  c
First Name: e
ID:         35
--------------------
Last Name:  cc
First Name: c
ID:         33
--------------------
Last Name:  dd
First Name: ddd
ID:         44
--------------------
Last Name:  e
First Name: e
ID:         55
--------------------
Number of records: 7
```

Type ‘3’ and press enter
Type ‘id’ or ‘fl’ to search by ID or Name
If you select id, enter id and press enter

```
3
Do you want to search by ID or by First and Last names?
Enter "id" or "fl" : id
Enter ID you wish to search:  55

1 Record(s) Found
Last Name:  e
First Name: e
ID:         55
--------------------

Time: 0.002 seconds

Do you want to search again? (y/n):
```

Type ‘y’ if you want to search again
Type ‘fl’ then type ‘c cc’. This finds a record

```
Do you want to search again? (y/n):  y

Do you want to search by ID or by First and Last names?
Enter "id" or "fl" : fl
Enter First, then Space, then Last. I.E. "First Last": c cc

1 Record(s) Found
Last Name:  cc
First Name: c
ID:         33
--------------------

Time: 0.004 seconds

Do you want to search again? (y/n):
```

Type ‘n’ to exit search. This gets you back to the main menu.

```
Do you want to search again? (y/n):  n


************************
Menu:
1. Import List from File
2. Display List
3. Search List
x. Exit
************************
```

Type ‘x’ to exit the program.

```
  x

Laptop@Andrew MINGW64 ~/Desktop/Visual Studio Code Programs/3. Fall 2025 ADS Class/Assignments/Project_1_Github/p01-ads/Code (main)
$
```