Linked Lists and the gdb Debugger
Embedded Design: Enabling Robotics
EECE2160

Michael Brodskiy
Brodskiy.M@Northeastern.edu

March 30, 2023

Date Performed:	March 23, 2023
Partner:	Dylan Powers
Instructor:	Professor Shazli

Abstract

The purpose of this laboratory experiment was to work with classes and familiarize oneself with the linked list advanced data structure, as well as further experience with pointers and memory addresses and their respective operators. By generating a program to interface with a linked list containing a fabricated class, while at the same time avoiding segmentation faults, a stronger grasp of these concepts was created. To avoid segmentation faults, the `gdb` debugger was employed.

KEYWORDS: Linked list, class, pointer, memory address, segmentation fault, `gdb`

1 Equipment

Available equipment included:

- DE1-SoC board
- DE1-SoC Power Cable
- USB-A to USB-B Cable
- Computer
- MobaXTerm SSH Terminal
- USB-to-ethernet Adapter

2 Introduction

In general, a debugger tool is used to inspect the memory of a program in a controlled execution environment, with the common objective of identifying the presence of bugs in a program. In this lab, one goal was to use the `gdb` bugger to debug programs through step-by-step execution and memory inspection. Along with this goal, this lab had a secondary goal to use linked lists as an alternative data structure to store sequence elements, where insertion and deletions have a constant cost. Then, after using both these skills separately, the lab combines them with the purpose of using the `gdb` debugger to explore the execution of a main program using linked lists.

3 Discussion & Analysis

3.1 Pre-lab

```
bash-4.4$ gdb
GNU gdb (GDB) Red Hat Enterprise Linux 8.2-18.el8
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) file person
Reading symbols from person...done.
(gdb) start
Temporary breakpoint 1 at 0x400b2d: file person.cpp, line 26.
Starting program: /Users/Student/mbrodskiy/person

Temporary breakpoint 1, main () at person.cpp:26
26              Person person;
Missing separate debuginfos, use: yum debuginfo-install glibc-2.28-211.el8.x86_64
86_64 libstdc++-8.5.0-16.el8_7.x86_64
(gdb) next
27              person.name = "John";
(gdb) next
28              person.age = 10;
(gdb) next
29              PrintPerson(&person);
(gdb) step
PrintPerson (person=0x7fffffffe5d0) at person.cpp:18
18      cout << person->name << " is " << person->age << " years old\n";
(gdb) print person.name
$1 = "John"
(gdb) next
John is 10 years old
20      }
(gdb) print person.age
$2 = 10
(gdb) next
main () at person.cpp:26
26              Person person;
(gdb) next
31      }
(gdb) next
0x00007ffff7114d85 in __libc_start_main () from /lib64/libc.so.6
(gdb) next
Single stepping until exit from function __libc_start_main,
which has no line number information.
[Inferior 1 (process 2241569) exited normally]
(gdb)
```

Figure 1: gdb output

The gdb commands may be explained as follows:

- `file person` selects the binary called person as the file for analysis
- `start` begins analysis of “person”
- `next` moves to the next point of interest
- `step` enters the function at the current line
- `print` prints the known information for a certain, specified value

Listing 1: Menu Printing Code

```
1 #include <iostream>
2 #include <string>
```

```
3  using namespace std;
4
5  // Linked List Management Code
6  struct Person
7  {
8          // Unique identifier for the person
9          int id;
10         // Information about person
11         string name;
12         int age;
13         // Pointer to next person in list
14         Person *next;
15 };
16 struct List
17 {
18         // First person in the list. A value equal to NULL
               indicates that the
19         // list is empty.
20         Person *head;
21         // Current person in the list. A value equal to
               NULL indicates a
22         // past-the-end position.
23         Person *current;
24         // Pointer to the element appearing before 'current
               '. It can be NULL if
25         // 'current' is NULL, or if 'current' is the first
               element in the list.
26         Person *previous;
27         // Number of persons in the list
28         int count;
29 };
30
31 // Give an initial value to all the fields in the list.
32 void ListInitialize(List *list)
33 {
34         list->head = NULL;
35         list->current = NULL;
36         list->previous = NULL;
37         list->count = 0;
38 }
39 // Move the current position in the list one element
       forward. If last element
40 // is exceeded, the current position is set to a special
       past-the-end value.
41 void ListNext(List *list)
42 {
```

```
43          if (list->current)
44          {
45                  list->previous = list->current;
46                  list->current = list->current->next;
47          }
48  }
49  // Move the current position to the first element in the
        list.
50  void ListHead(List *list)
51  {
52          list->previous = NULL;
53          list->current = list->head;
54  }
55  // Get the element at the current position, or NULL if the
        current position is
56  // past-the-end.
57  Person *ListGet(List *list)
58  {
59          return list->current;
60  }
61  // Set the current position to the person with the given id
        . If no person
62  // exists with that id, the current position is set to past
        -the-end.
63  void ListFind(List *list, int id)
64  {
65          ListHead(list);
66          while (list->current && list->current->id != id)
67                  ListNext(list);
68  }
69  // Insert a person before the element at the current
        position in the list. If
70  // the current position is past-the-end, the person is
        inserted at the end of
71  // the list. The new person is made the new current element
         in the list.
72  void ListInsert(List *list, Person *person)
73  {
74          // Set 'next' pointer of current element
75          person->next = list->current;
76          // Set 'next' pointer of previous element. Treat
                the special case where
77          // the current element was the head of the list.
78          if (list->current == list->head)
79                  list->head = person;
80          else
```

```
81                  list->previous->next = person;
82          // Set the current element to the new person
83          list->current = person;
84  }
85  // Remove the current element in the list. The new current element will be the
86  // element that appeared right after the removed element.
87  void ListRemove(List *list)
88  {
89          // Ignore if current element is past-the-end
90          if (!list->current)
91                  return;
92          // Remove element. Consider special case where the current element is
93          // in the head of the list.
94          if (list->current == list->head)
95                  list->head = list->current->next;
96          else
97                  list->previous->next = list->current->next;
98          // Free element, but save pointer to next element first.
99          Person *next = list->current->next;
100         delete list->current;
101         // Set new current element
102         list->current = next;
103 }
104 void PrintPerson(Person *person)
105 {
106         cout << "Person with ID: " << person->id << endl;
107         cout << "\tName: " << person->name << endl;
108         cout << "\tAge: " << person->age << endl << endl;;
109 }
110
111 /** main function: Will create and process a linked list
112  */
113 int main() {
114         List list;                              // Create the main list
115         ListInitialize(&list);                  // Initialize the list
116 //*************** PUT THE REST OF YOUR CODE HERE ******************
117
118         string options[] = {"Add a person", "Find a person", "Remove a person", "Print the list", "Exit"};
119
```

```
120         int choice = 0;
121 
122         do {
123 
124                 for (int i = 0; i < 5; i++) {
125 
126                         cout << (i + 1) << ". " << options[
                                i] << endl;
127 
128                 }
129 
130                 cout << "Select an option: ";
131                 cin >> choice;
132                 cout << "You selected: ";
133 
134                 if (choice == 1) {
135 
136                         cout << "\"" << options[choice - 1]
                                << "\"" << endl;
137 
138                 }
139 
140                 else if (choice == 2) {
141 
142                         cout << "\"" << options[choice - 1]
                                << "\"" << endl;
143 
144                 }
145 
146                 else if (choice == 3) {
147 
148                         cout << "\"" << options[choice - 1]
                                << "\"" << endl;
149 
150                 }
151 
152                 else if (choice == 4) {
153 
154                         cout << "\"" << options[choice - 1]
                                << "\"" << endl;
155 
156                 }
157 
158                 else if (choice == 5) {
159 
160                         cout << "\"" << options[choice - 1]
```

```
                                << "\"" << endl;
161
162                 }
163
164                 else {
165
166                         cout << "Error. Invalid option. Try
                                again." << endl << endl;
167
168                 }
169
170         } while (choice < 1 || choice > 5);
171
172
173
174
175
176
177
178
179 } //end main
```

```
bash-4.4$ ./personList
1. Add a person
2. Find a person
3. Remove a person
4. Print the list
5. Exit
Select an option: 0
You selected: Error. Invalid option. Try again.

1. Add a person
2. Find a person
3. Remove a person
4. Print the list
5. Exit
Select an option: 4
You selected: "Print the list"
bash-4.4$
```

Figure 2: Sample menu output

3.2 Assignment 1

The goal of Assignment 1 was to load a program `person` on gbd and set a breakpoint at the beginning of the function `PrintPerson`. In Figure 11, the output is provided for the commands (gdb) `print person`, (gdb) `*person`, (gdb) `print->name`, and (gdb) `print->age`.

```
bash-4.4$ gdb person
GNU gdb (GDB) Red Hat Enterprise Linux 8.2-18.el8
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from person...done.
(gdb) break person.cpp:14
Breakpoint 1 at 0x400ae2: file person.cpp, line 18.
(gdb) run
Starting program: /Users/Student/mbrodskiy/Documents/Lab 7/person

Breakpoint 1, PrintPerson (person=0x7fffffffe570) at person.cpp:18
18      cout << person->name << " is " << person->age << " years old\n";
Missing separate debuginfos, use: yum debuginfo-install glibc-2.28-211.el8.x86_64
86_64 libstdc++-8.5.0-16.el8_7.x86_64
(gdb) print person
$1 = (Person *) 0x7fffffffe570
(gdb) print *person
$2 = {name = "John", age = 10}
(gdb) print person->name
$3 = "John"
(gdb) print person->age
$4 = 10
(gdb)
```

Figure 3: Output given by gdb for specified commands

In the output shown it can be seen that the first command (gdb) `print person` prints the value of the person variable which is a pointer to a `Person` object. The value of `person` is `0x7fffffffe570`. The next command, (gdb) `print *person`, prints the contents of the `Person` object that `person` is pointing to. The output shows that there is a name attribute of "John" and an age attribute of 10. As for the third command, (gdb) `print person->name` the program is requesting the value of the name attribute of the `Person` object which is John as shown. Lastly, (gdb) `print person->age` prints the value of the age attribute of the `Person` object. This is why 10 is out after this command is displayed.

3.3 Assignment 2

The goal of Assignment 2 was to start with the implementation of a main program to test the linked list, with a similar structure to the one used to test the dynamically

growing array in the previous lab. To do this, a program was written that includes all the type definitions and functions presented in class to manage the `List` and `Person` data structures, together with an implementation of the main program. The complete code of the program is included in the entire program, included in the Appendix below.

3.4 Assignment 3

The purpose of Assignment 3 was to run the program written in Assignment 2 several times in order to test the behavior of each menu option. For each option, the output of the program is shown below.

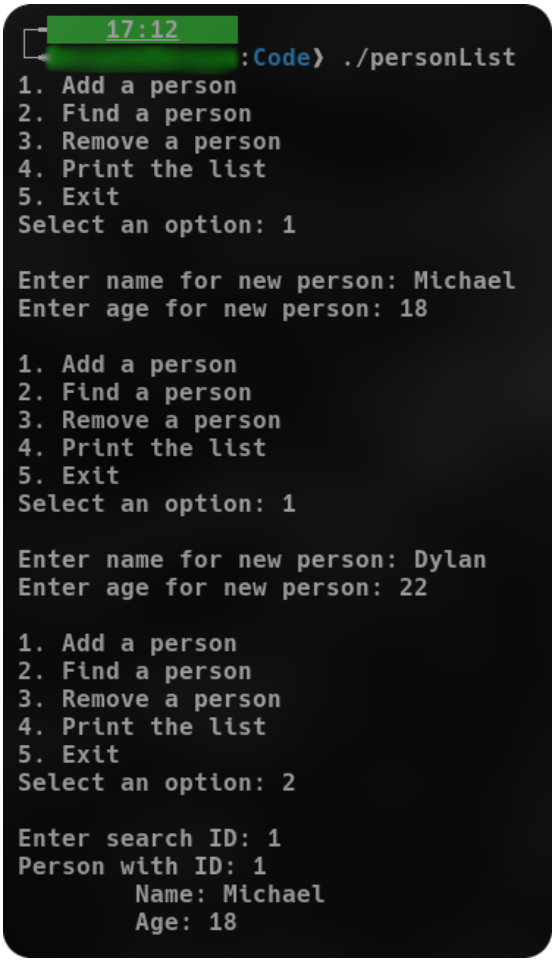

Figure 4: Shows menu options 1 and 2

Figure 4 shows the first menu option twice. Option 1 is selected and the name Michael and the age 18 are entered and stored. Then, Option 1 is selected again

and the name Dylan and the age 22 are entered and stored. Finally, Option 2, Find a person, is selected and the search ID of 1 is entered. With this information, the program outputted the name Michael and the age 18. This output tells the user that the name Michael and the age 18 were successfully added and that it is possible to locate the name and age of a person with an id. It makes sense that search id 1 returned Michael and not Dylan, because Michael was entered first.

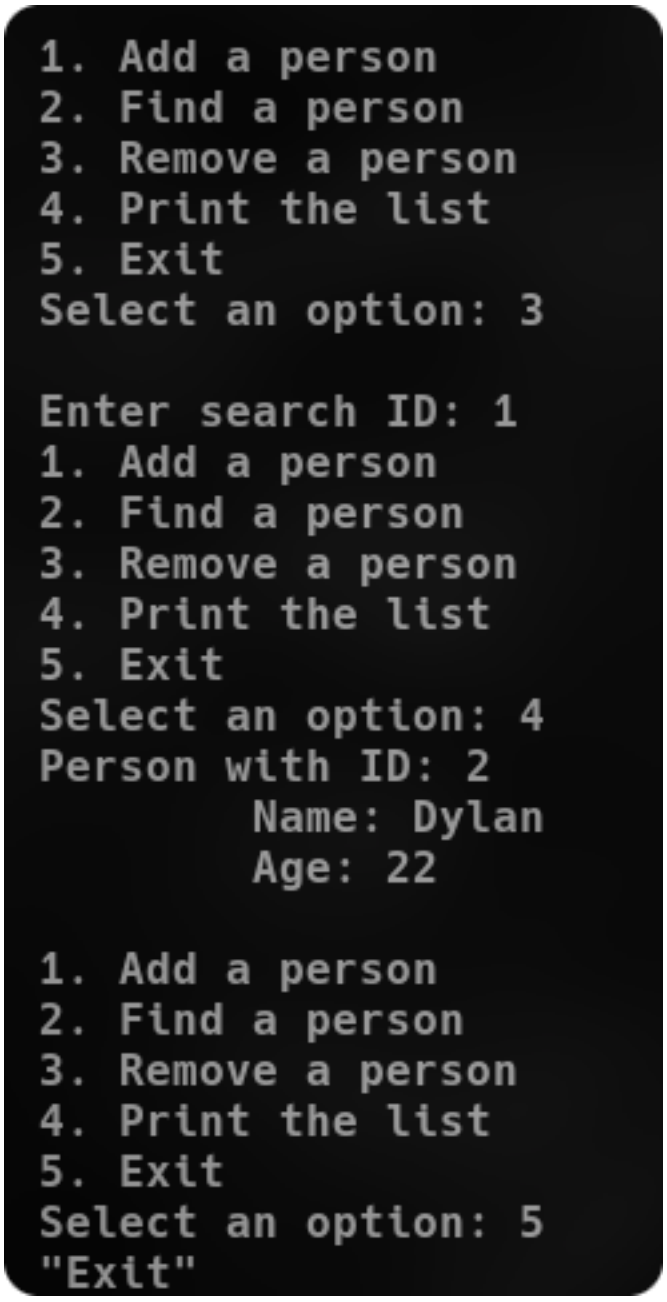

Figure 5: Shows menu option 3, 4, and 5

In Figure 5, Option 3 is shown first. When option 3 was selected, the search id of the person that was to be removed was entered. As can be seen the search id 1 was entered. Next for Option 4, Print the list, this was selected after option 3. Thus, the

list should print with only Dylan because Michael was removed, which is what the program returned. Lastly, Option 5, Exit, was selected and the program outputted the message “Exit” and the menu was not displayed again.

3.5 Assignment 4

The goal of Assignment 4 was to run the linked list main program developed in the first part of the lab and set a breakpoint at the end of the loop in function `main()`, right before the menu is printed for the second time. When the program was run and the menu option that allows the user to insert a new element was selected, the commands (gdb) `print list`, (gdb) `print list.head`, and (gdb) `print list.head->next` were run. The full sequence ran on gdb as well as the output of the commands are shown in Figures 6 and 7.

Figure 6: First half of full sequence ran on gdb and output of commands

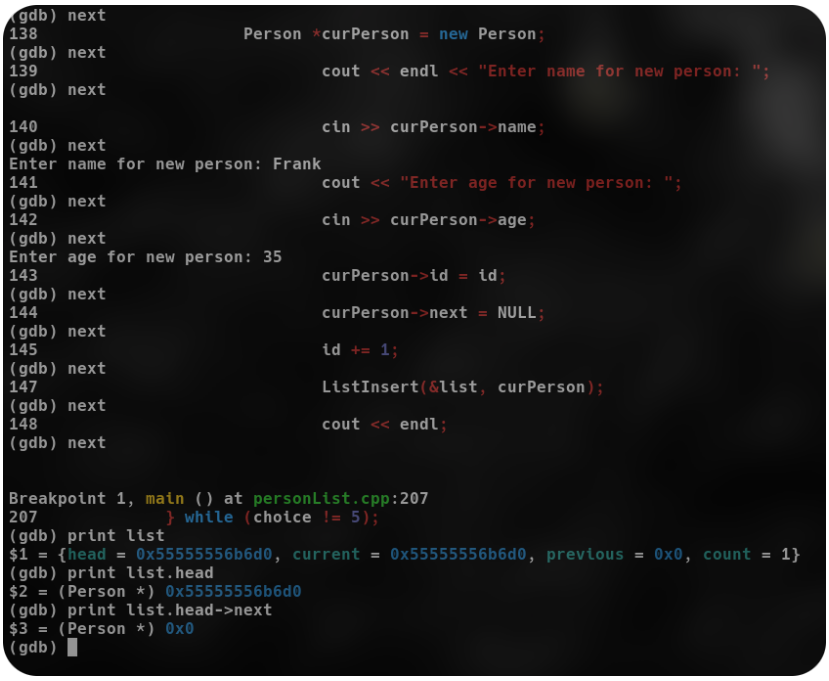

Figure 7: Second half of full sequence ran on gdb and output of commands

Figure 6 is most of the sequence leading up to running the commands which is shown in Figure 7. The first of the specified commands run was the `print list` command. This command printed the current `Person` object of the linked list. The next command that was run was print `list.head`. This command outputs the contents of the head field of the starting address. Lastly the print `list.head->next` command was run, which outputs the next field of the object pointed to by the head. In the case displayed above, the next field was `NULL` which is why the output was `0x0`.

3.6 Assignment 5

The goal of Assignment 5 was to modify any position of the code in the linked list to strategically make it perform an invalid memory operation that makes the program produce a segmentation fault or to use a real intermediate unstable version of the code that produced a program crash during the development of previous assignments. The program written to do this was run on gdb and the sequence of gdb commands that were run to infer the value of the variable causing the problem is shown in Figure 8.

Figure 8: Sequence of gdb commands to infer the value of the variable causing the problem

As can be seen in Figure 8, the (gdb) `next` command was run until the program received signal SIGSEGV and there was a Segmentation fault. This fault was intentionally produced by permitting the `ListGet()` function to attempt to access an nonexistent object.

3.7 Extra Credit

The extra credit section relied on the creation of a method to sort the linked list in two ways: by person name or by age. In our case, this was done by copying the `Person` objects over to two parallel arrays, which were then sorted. The existing linked list was then wiped. Subsequently, the now-sorted arrays were inserted back into the linked list. The linked lists were sorted in descending order, as a sorting key was not provided. Below are images depicting a test case of the sorting program.

```
1. Add a person
2. Find a person
3. Remove a person
4. Print the list
5. Sort the list
6. Exit
Select an option: 1

Enter name for new person: Michael
Enter age for new person: 18

1. Add a person
2. Find a person
3. Remove a person
4. Print the list
5. Sort the list
6. Exit
Select an option: 1

Enter name for new person: Dylan
Enter age for new person: 22

1. Add a person
2. Find a person
3. Remove a person
4. Print the list
5. Sort the list
6. Exit
Select an option: 1

Enter name for new person: Phil
Enter age for new person: 31
```

Figure 9: Extra Credit Test Run, part 1

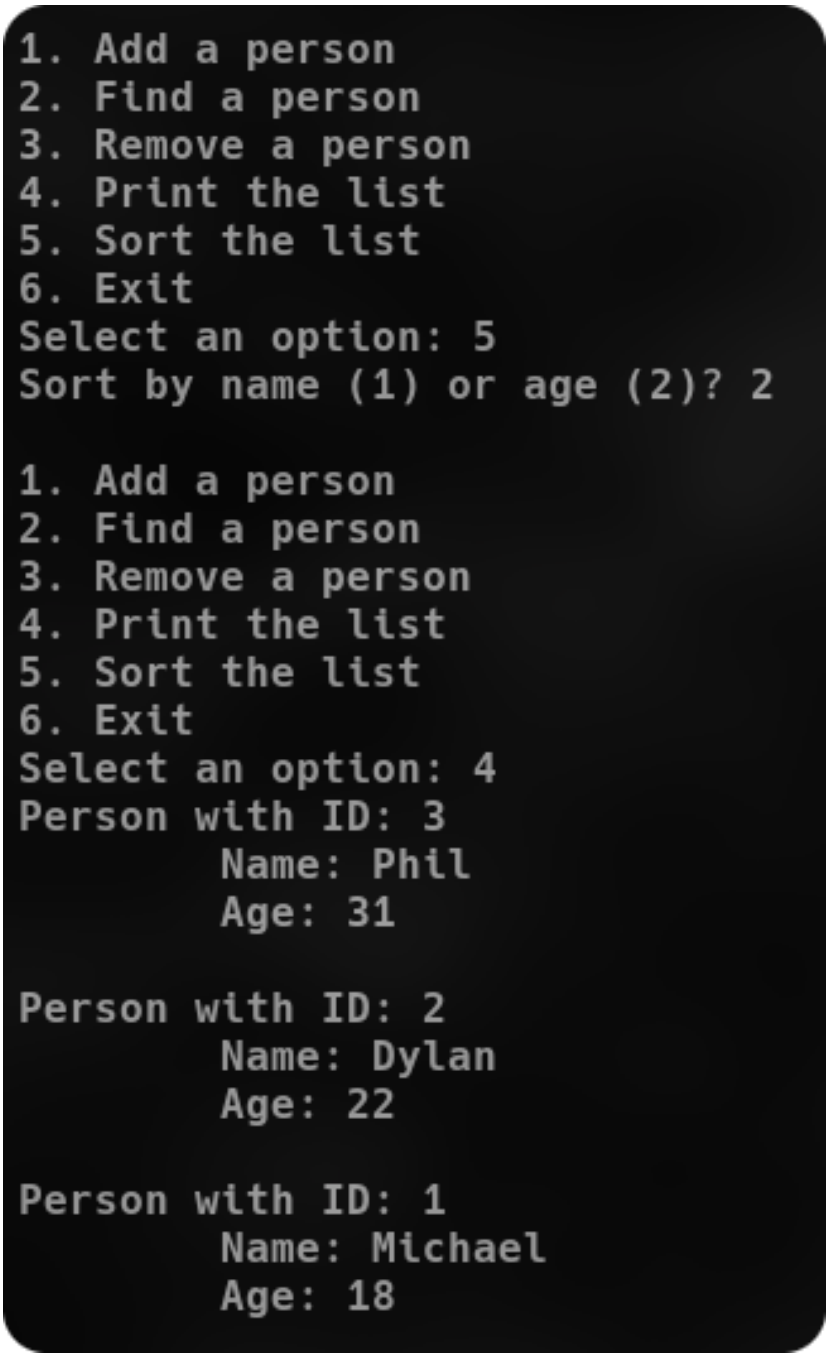

Figure 10: Extra Credit Test Run, part 2

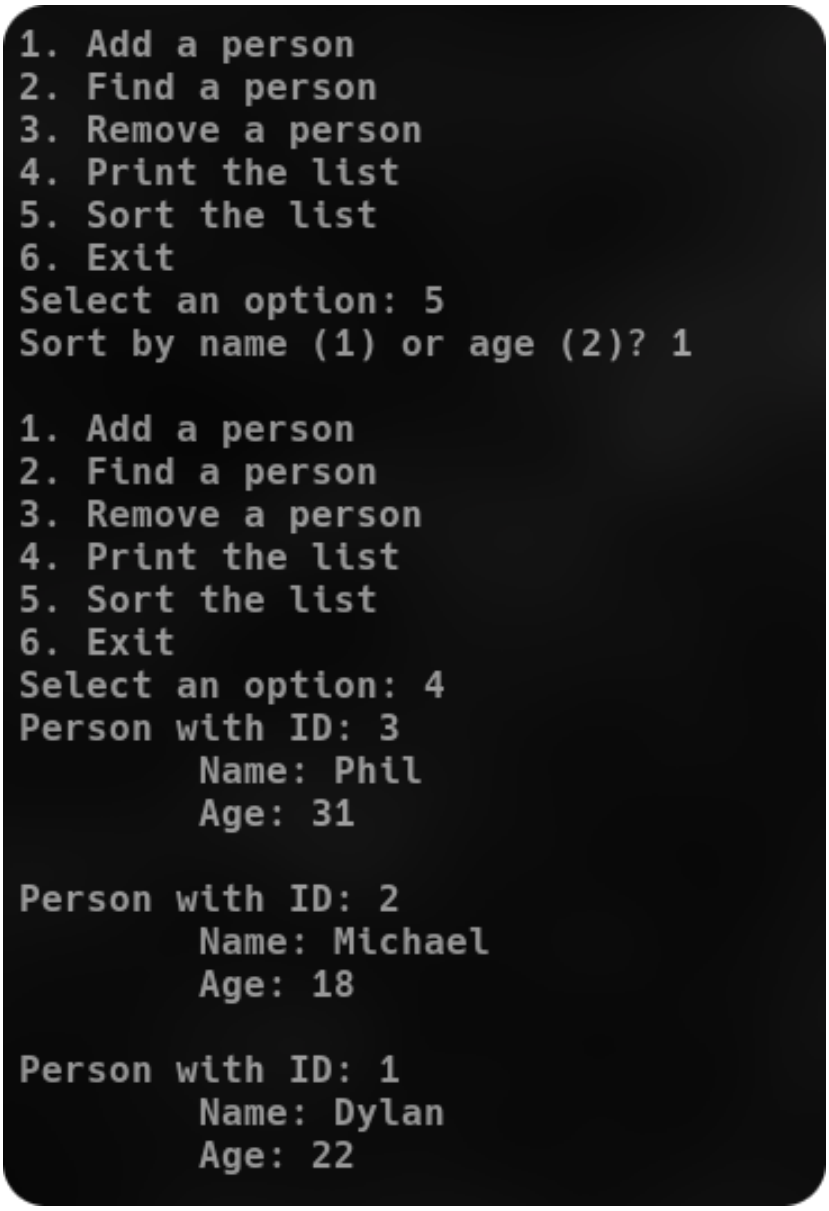

Figure 11: Extra Credit Test Run, part 3

4 Conclusion

Overall, due to the heavy reliance on knowledge of pointers, memory address references, linked lists, class structures, and the ability to interface with the aforementioned, the lab provided an effectively advanced lesson regarding these concepts.

Especially in terms of linked lists, working with them in an actual example allowed for a deeper understanding.

5 Appendix

Listing 2: Complete Source Code

```
#include <iostream>
#include <cstdlib>
#include <string>
using namespace std;

// Linked List Management Code
struct Person
{
        // Unique identifier for the person
        int id;
        // Information about person
        string name;
        int age;
        // Pointer to next person in list
        Person *next;
};
struct List
{
        // First person in the list. A value equal to NULL indicates that the
        // list is empty.
        Person *head;
        // Current person in the list. A value equal to NULL indicates a
        // past-the-end position.
        Person *current;
        // Pointer to the element appearing before 'current'. It can be NULL if
        // 'current' is NULL, or if 'current' is the first element in the list.
        Person *previous;
        // Number of persons in the list
        int count;
};

// Give an initial value to all the fields in the list.
void ListInitialize(List *list)
{
        list->head = NULL;
```

```
36          list->current = NULL;
37          list->previous = NULL;
38          list->count = 0;
39  }
40  // Move the current position in the list one element
        forward. If last element
41  // is exceeded, the current position is set to a special
        past-the-end value.
42  void ListNext(List *list)
43  {
44          if (list->current)
45          {
46                  list->previous = list->current;
47                  list->current = list->current->next;
48          }
49  }
50  // Move the current position to the first element in the
        list.
51  void ListHead(List *list)
52  {
53          list->previous = NULL;
54          list->current = list->head;
55  }
56  // Get the element at the current position, or NULL if the
        current position is
57  // past-the-end.
58  Person *ListGet(List *list)
59  {
60          return list->current;
61  }
62  // Set the current position to the person with the given id
        . If no person
63  // exists with that id, the current position is set to past
        -the-end.
64  void ListFind(List *list, int id)
65  {
66          ListHead(list);
67          while (list->current && list->current->id != id)
68                  ListNext(list);
69  }
70  // Insert a person before the element at the current
        position in the list. If
71  // the current position is past-the-end, the person is
        inserted at the end of
72  // the list. The new person is made the new current element
         in the list.
```

```
73  void ListInsert(List *list, Person *person)
74  {
75          // Set 'next' pointer of current element
76          person->next = list->current;
77          // Set 'next' pointer of previous element. Treat
                the special case where
78          // the current element was the head of the list.
79          if (list->current == list->head)
80                  list->head = person;
81          else
82                  list->previous->next = person;
83          // Set the current element to the new person
84          list->current = person;
85      list->count += 1;
86  }
87  // Remove the current element in the list. The new current
        element will be the
88  // element that appeared right after the removed element.
89  void ListRemove(List *list)
90  {
91          // Ignore if current element is past-the-end
92          if (!list->current)
93                  return;
94          // Remove element. Consider special case where the
                current element is
95          // in the head of the list.
96          if (list->current == list->head)
97                  list->head = list->current->next;
98          else
99                  list->previous->next = list->current->next;
100         // Free element, but save pointer to next element
                first.
101         Person *next = list->current->next;
102         delete list->current;
103         // Set new current element
104         list->current = next;
105     list->count -= 1;
106 }
107 void PrintPerson(Person *person)
108 {
109         cout << "Person with ID: " << person->id << endl;
110         cout << "\tName: " << person->name << endl;
111         cout << "\tAge: " << person->age << endl << endl;;
112 }
113
114 /** main function: Will create and process a linked list
```

```
115  */
116 int main() {
117         List list;                                  // Create
                the main list
118         ListInitialize(&list);                      //
                Initialize the list
119 //*************** PUT THE REST OF YOUR CODE HERE
    ******************
120
121         string options[] = {"Add a person", "Find a person"
                , "Remove a person", "Print the list", "Sort the
                 list", "Exit"};
122
123         int choice = 0;
124         int id = 1;
125
126         while (choice != 6) {
127
128                 for (int i = 0; i < 6; i++) {
129
130                         cout << (i + 1) << ". " << options[
                            i] << endl;
131
132                 }
133
134                 cout << "Select an option: ";
135                 cin >> choice;
136
137                 if (choice == 1) {
138
139                 Person *curPerson = new Person;
140                         cout << endl << "Enter name for new
                             person: ";
141                         cin >> curPerson->name;
142                         cout << "Enter age for new person:
                            ";
143                         cin >> curPerson->age;
144                         curPerson->id = id;
145                         curPerson->next = NULL;
146                         id += 1;
147
148                         ListInsert(&list, curPerson);
149                         cout << endl;
150
151                 }
152
```

```
153                 else if (choice == 2) {
154 
155                         int searchID;
156                         cout << endl << "Enter search ID: "
                                ;
157                         cin >> searchID;
158                         ListFind(&list, searchID);
159                         PrintPerson(ListGet(&list));
160                         cout << endl;
161 
162                 }
163 
164                 else if (choice == 3) {
165 
166                         int searchID;
167                         cout << endl << "Enter search ID: "
                                ;
168                         cin >> searchID;
169                         ListFind(&list, searchID);
170 
171                         if (ListGet(&list) == NULL) {
172 
173                                 cout << "No Person with ID
                                        #" << searchID << endl;
174 
175                         } else {
176 
177                                 ListRemove(&list);
178 
179                         }
180 
181                 }
182 
183                 else if (choice == 4) {
184 
185                         ListHead(&list);
186 
187                         for (int i = list.count - 1; i >=
                                0; i--) {
188 
189                                 PrintPerson(ListGet(&list))
                                        ;
190                                 ListNext(&list);
191 
192                         }
193 
```

```
194                 }
195
196                 else if (choice == 5) {
197
198             int sortParam;
199             string people[list.count];
200             int ages[list.count];
201
202             cout << "Sort by name (1) or age (2)? ";
203             cin >> sortParam;
204
205             ListHead(&list);
206
207             for (int i = 0; i < list.count; i++) {
208
209                 if (ListGet(&list) != NULL) {
210
211                     people[i] = ListGet(&list)->name;
212                     ages[i] = ListGet(&list)->age;
213
214                 }
215
216                 ListNext(&list);
217
218             }
219
220             ListHead(&list);
221
222             if (sortParam == 1) {
223
224                 int smallest_index;
225
226                     for (int i = 0; i < list.count; i++) {
227
228                             smallest_index = i;
229
230                             for (int j = i + 1; j < list.
                                    count; j++) {
231
232                                 if (people[j] < people[
                                        smallest_index])
                                        smallest_index = j;
233
234                             }
235
```

```
236                         swap(people[i], people[
                                smallest_index]);
237                     swap(ages[i], ages[smallest_index]);
238 
239                     }
240 
241 
242             } else if (sortParam == 2) {
243 
244                 int smallest_index;
245 
246                     for (int i = 0; i < list.count; i++) {
247 
248                             smallest_index = i;
249 
250                             for (int j = i + 1; j < list.
                                    count; j++) {
251 
252                                     if (ages[j] < ages[
                                            smallest_index])
                                            smallest_index = j;
253 
254                             }
255 
256                             swap(people[i], people[
                                    smallest_index]);
257                     swap(ages[i], ages[smallest_index]);
258 
259                     }
260 
261             } else {
262 
263                 cout << "Invalid Option!" << endl;
264                 break;
265 
266             }
267 
268             while (ListGet(&list) != NULL) {
269 
270                 ListRemove(&list);
271                 ListNext(&list);
272 
273             }
274 
275             ListHead(&list);
276             ListRemove(&list);
```

```
277             id = 1;
278
279             for (int i = 0; i < (sizeof(people)/sizeof(*
                    people)); i++) {
280
281                 Person *curPerson = new Person;
282                 curPerson->name = people[i];
283                 curPerson->age = ages[i];
284                 curPerson->id = id;
285                 ListInsert(&list, curPerson);
286                 id++;
287
288             }
289
290             cout << endl;
291
292         }
293
294         else if (choice == 6) {
295
296                     cout << "\"" << options[choice - 1]
                            << "\"" << endl;
297
298                 }
299
300                 else {
301
302                     cout << "Error. Invalid option. Try
                            again." << endl << endl;
303
304                 }
305
306         }
307
308 } //end main
```