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.

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

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

3 Discussion & Analysis

3.1 Assignment 1

3.2 Assignment 2

3.3 Assignment 3

3.4 Assignment 4

3.5 Assignment 5

3.6 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 1: Extra Credit Test Run, part 1

```
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: 5
Sort by name (1) or age (2)? 2

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: 4
Person with ID: 3
        Name: Phil
        Age: 31

Person with ID: 2
        Name: Dylan
        Age: 22

Person with ID: 1
        Name: Michael
        Age: 18
```

Figure 2: Extra Credit Test Run, part 2

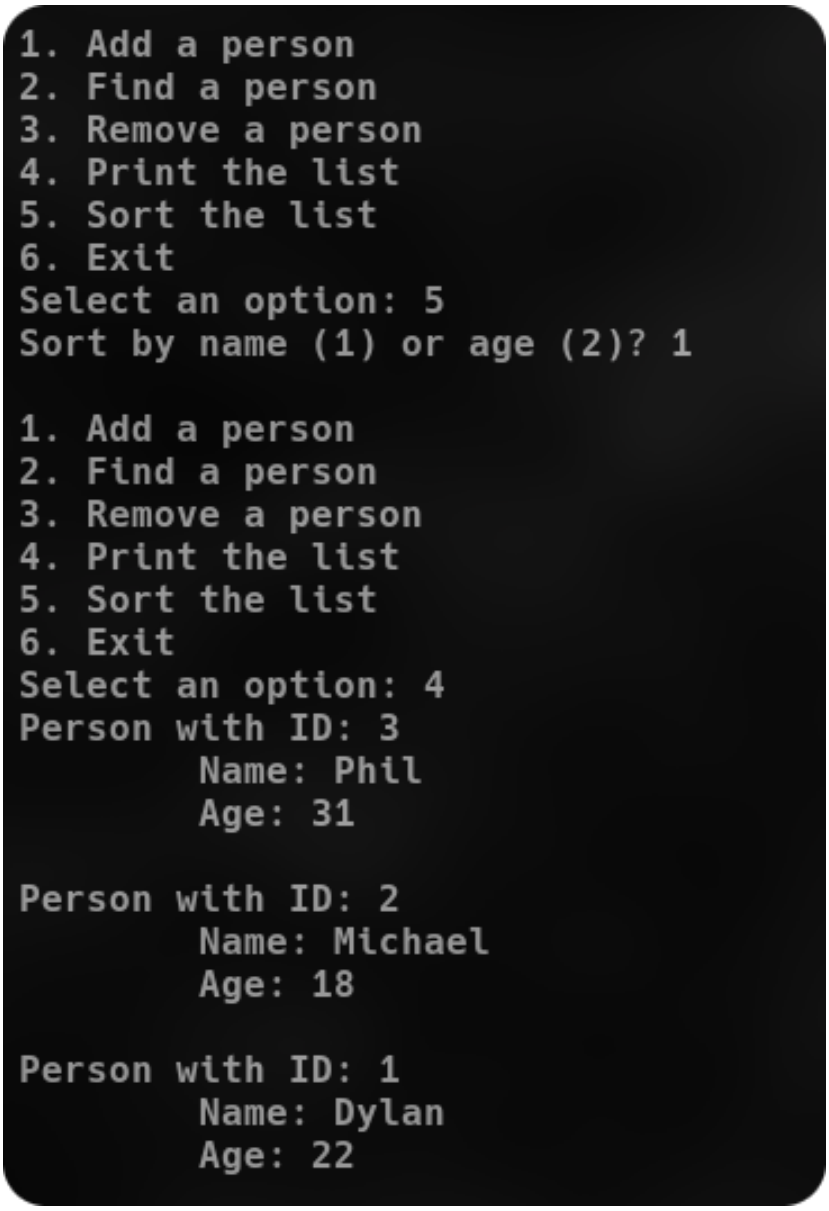

Figure 3: Extra Credit Test Run, part 3

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 1: 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
```

```
 */
int main() {
        List list;                                  // Create the main list
        ListInitialize(&list);                      // Initialize the list
//*************** PUT THE REST OF YOUR CODE HERE ****************

        string options[] = {"Add a person", "Find a person", "Remove a person", "Print the list", "Sort the list", "Exit"};

        int choice = 0;
        int id = 1;

        while (choice != 6) {

                for (int i = 0; i < 6; i++) {

                        cout << (i + 1) << ". " << options[i] << endl;

                }

                cout << "Select an option: ";
                cin >> choice;

                if (choice == 1) {

                Person *curPerson = new Person;
                        cout << endl << "Enter name for new person: ";
                        cin >> curPerson->name;
                        cout << "Enter age for new person: ";
                        cin >> curPerson->age;
                        curPerson->id = id;
                        curPerson->next = NULL;
                        id += 1;

                        ListInsert(&list, curPerson);
                        cout << endl;

                }

```

```
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
```

```
                        swap(people[i], people[
                            smallest_index]);
                    swap(ages[i], ages[smallest_index]);

                    }


            } else if (sortParam == 2) {

                int smallest_index;

                    for (int i = 0; i < list.count; i++) {

                            smallest_index = i;

                            for (int j = i + 1; j < list.
                                count; j++) {

                                    if (ages[j] < ages[
                                        smallest_index])
                                        smallest_index = j;

                            }

                            swap(people[i], people[
                                smallest_index]);
                    swap(ages[i], ages[smallest_index]);

                    }

            } else {

                cout << "Invalid Option!" << endl;
                break;

            }

            while (ListGet(&list) != NULL) {

                ListRemove(&list);
                ListNext(&list);

            }

            ListHead(&list);
            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
```