Patricia Gavelek, Anders Dahl EECE2160	Embedded Design: Enabling Robotics Lab Assignment 2

Linked Lists and the gdb Debugger

Patricia Gavelek, Anders Dahl
Gavelek.p@husky.neu.edu
dahl.a@husky.neu.edu

Submit date: Feb 7th, 2016
Due Date: Feb 8th, 2016

Abstract

This lab focuses on using gdb as a tool for debugging. It allows for step-by-step execution and memory inspection. Linked list data structures, where insertions and deletions have a constant cost, will also be introduced. A program capable of handling a linked list of people will be produced. The overall goal of this lab is to become familiar with both gdb and linked lists.

Patricia Gavelek, Anders Dahl EECE2160	Embedded Design: Enabling Robotics Lab Assignment 2

Introduction

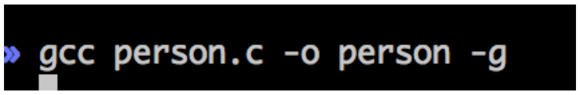
This lab focuses on using gdb as a tool for debugging. It allows for step-by-step execution and memory inspection. Linked list data structures, where insertions and deletions have a constant cost, will also be introduced. Gdb will be used to debug and even break the linked list program. The overall goal of this lab is to become familiar with both gdb and linked lists.

Lab Discussion

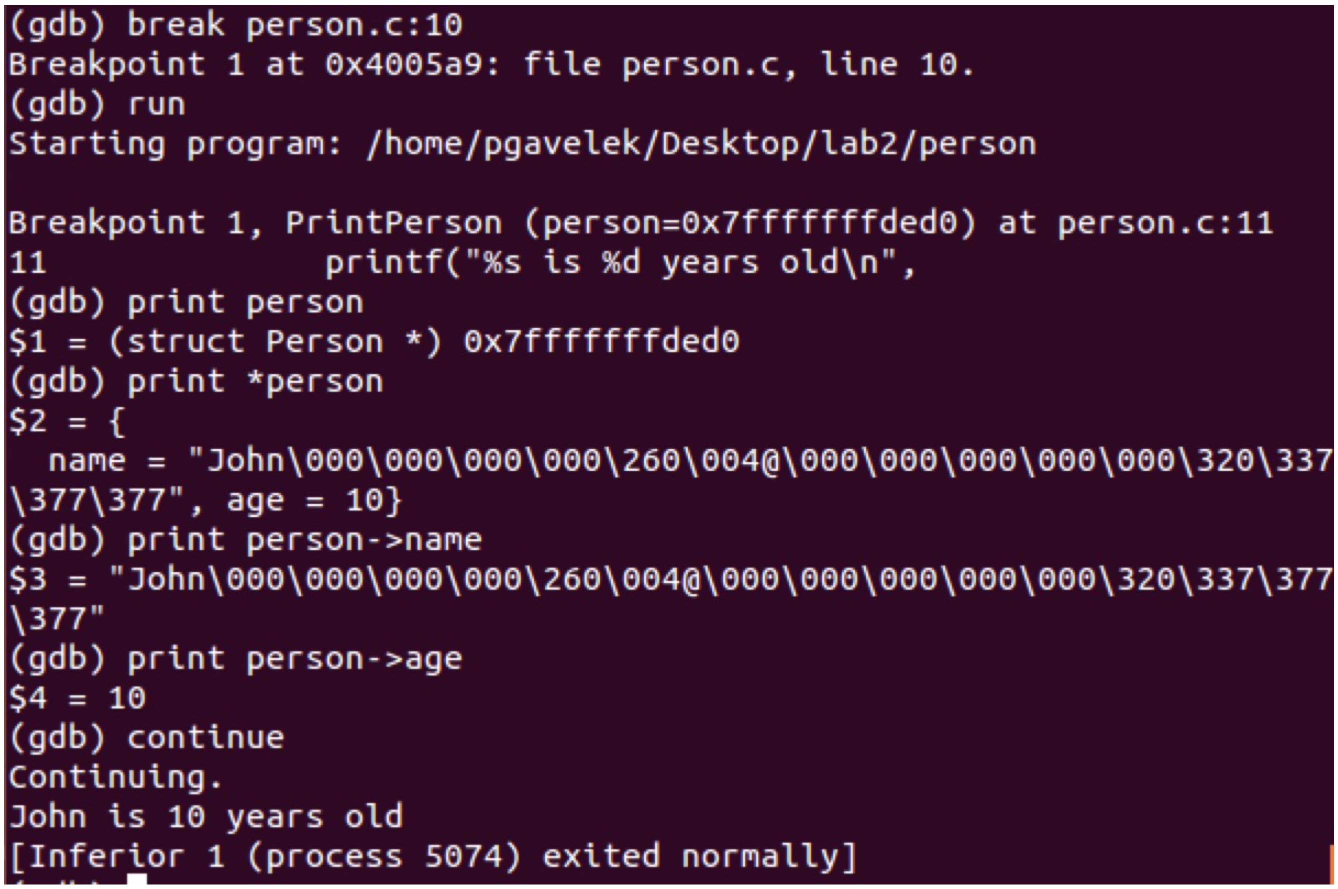
The pre-lab consisted of the menu and the navigational aspects needed in the code. It made sure that the user could to navigate the different functions that were to be implemented. In order to develop the program, the C programming language, the gcc compiler, and the gdb debugger were all used.

Results and Analysis

Assignment 1



```
» gcc person.c -o person -g
```



```
(gdb) break person.c:10
Breakpoint 1 at 0x4005a9: file person.c, line 10.
(gdb) run
Starting program: /home/pgavelek/Desktop/lab2/person

Breakpoint 1, PrintPerson (person=0x7fffffffded0) at person.c:11
11              printf("%s is %d years old\n",
(gdb) print person
$1 = (struct Person *) 0x7fffffffded0
(gdb) print *person
$2 = {
  name = "John\000\000\000\000\260\004@\000\000\000\000\000\320\337
\377\377", age = 10}
(gdb) print person->name
$3 = "John\000\000\000\000\260\004@\000\000\000\000\000\320\337\377
\377"
(gdb) print person->age
$4 = 10
(gdb) continue
Continuing.
John is 10 years old
[Inferior 1 (process 5074) exited normally]
```

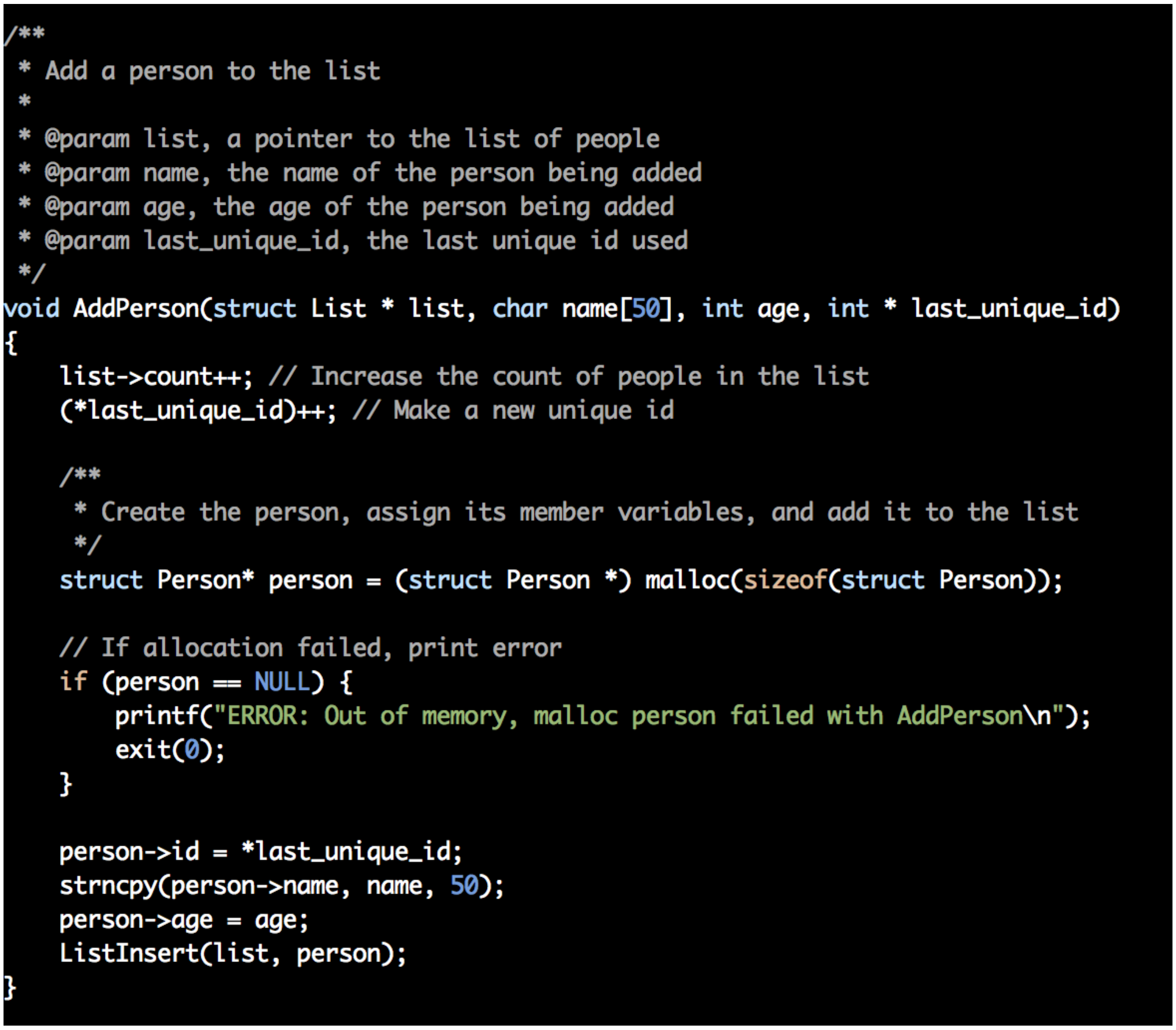
(gdb) print person: Prints the address pointing to a Person structure.

(gdb) print *person: Dereferences the address and prints the actual structure. Its associated values are also shown since they are part of the structure.

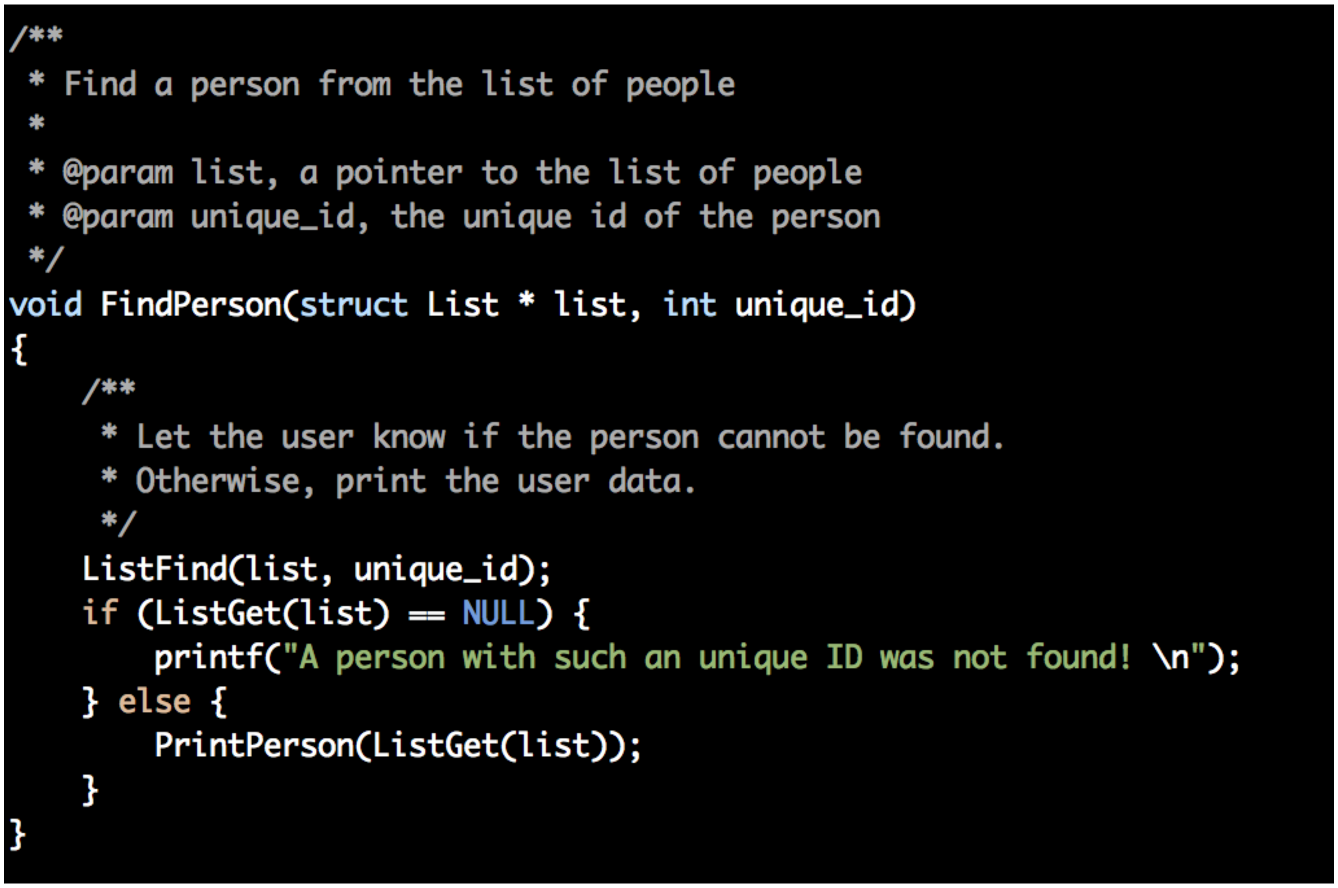
(**gdb) print person→name:** Dereferences the address and prints the value of the structure's name field. → name is a shorthand for *person.name

(gdb) print person→age: Deferences the address and prints the value of the structure's age field. → age is a shorthand for *person.age

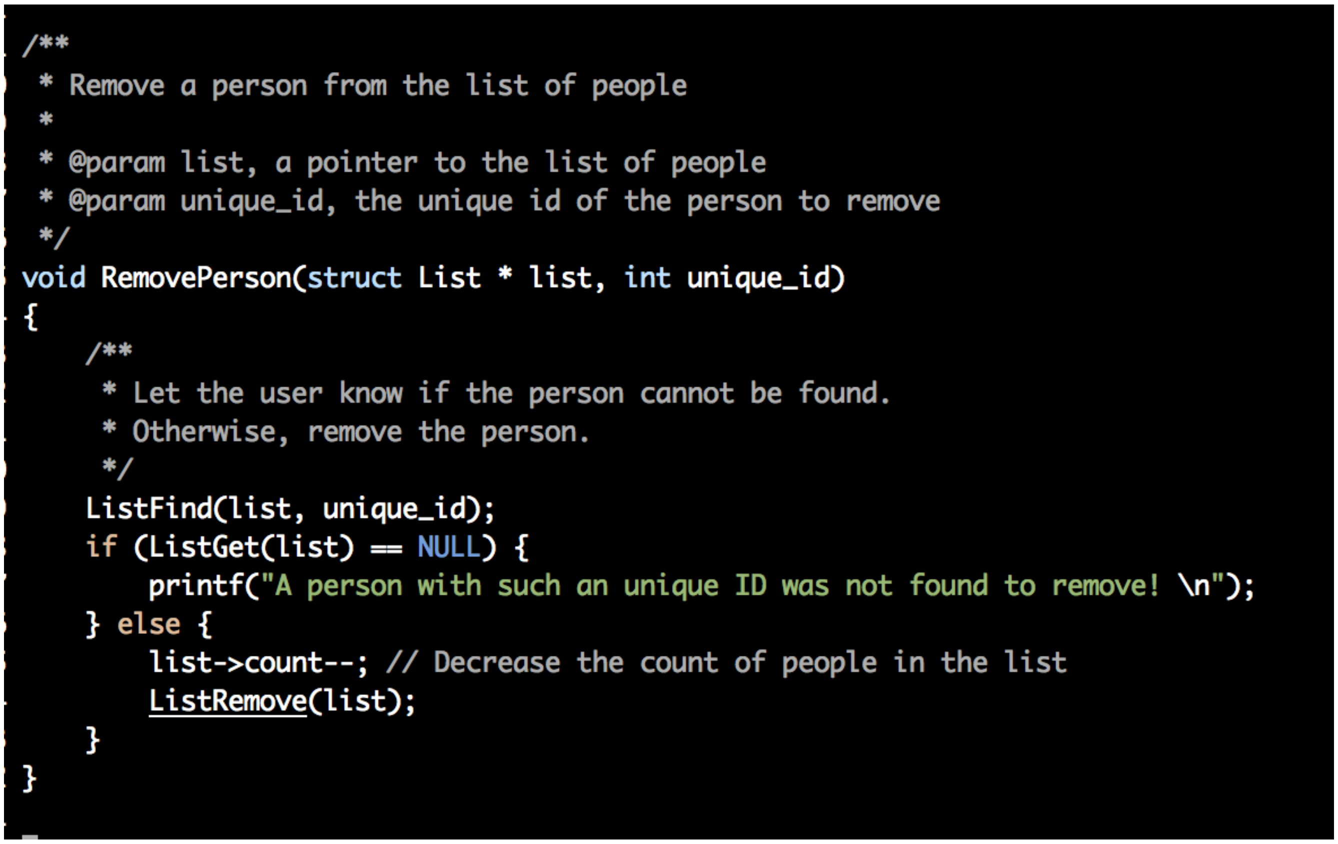
Assignment 2 – Code (All of the new code, given code not included):



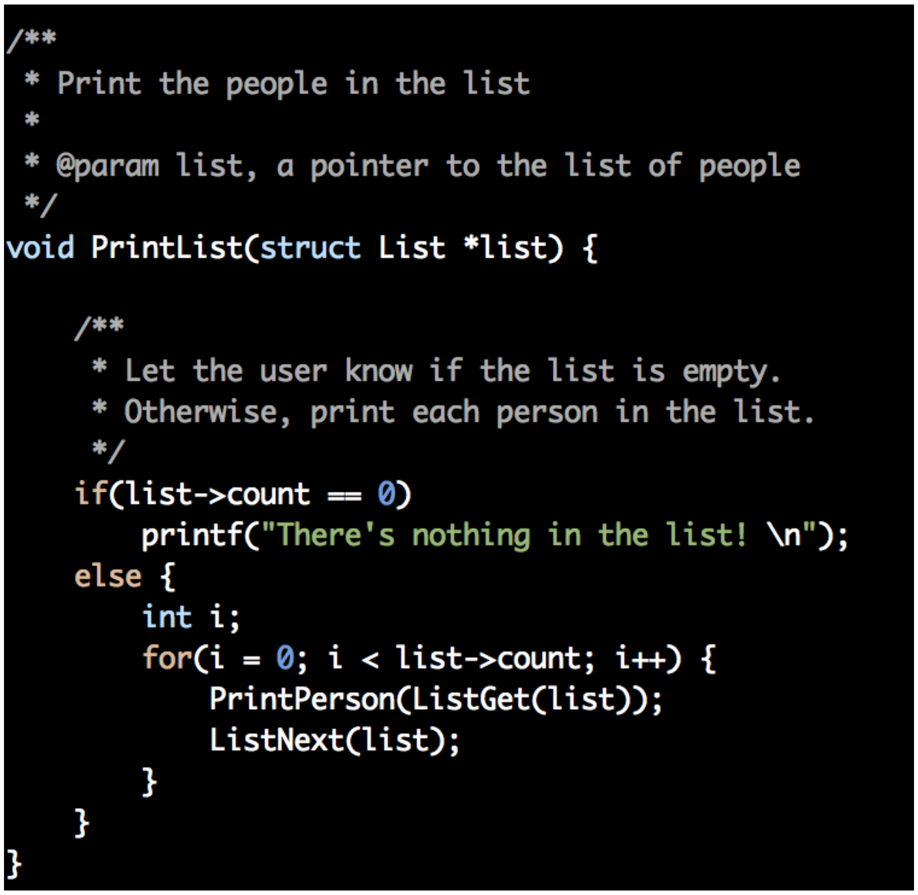
```
/**
 * Add a person to the list
 *
 * @param list, a pointer to the list of people
 * @param name, the name of the person being added
 * @param age, the age of the person being added
 * @param last_unique_id, the last unique id used
 */
void AddPerson(struct List * list, char name[50], int age, int * last_unique_id)
{
    list->count++; // Increase the count of people in the list
    (*last_unique_id)++; // Make a new unique id

    /**
     * Create the person, assign its member variables, and add it to the list
     */
    struct Person* person = (struct Person *) malloc(sizeof(struct Person));

    // If allocation failed, print error
    if (person == NULL) {
        printf("ERROR: Out of memory, malloc person failed with AddPerson\n");
        exit(0);
    }

    person->id = *last_unique_id;
    strncpy(person->name, name, 50);
    person->age = age;
    ListInsert(list, person);
}
```



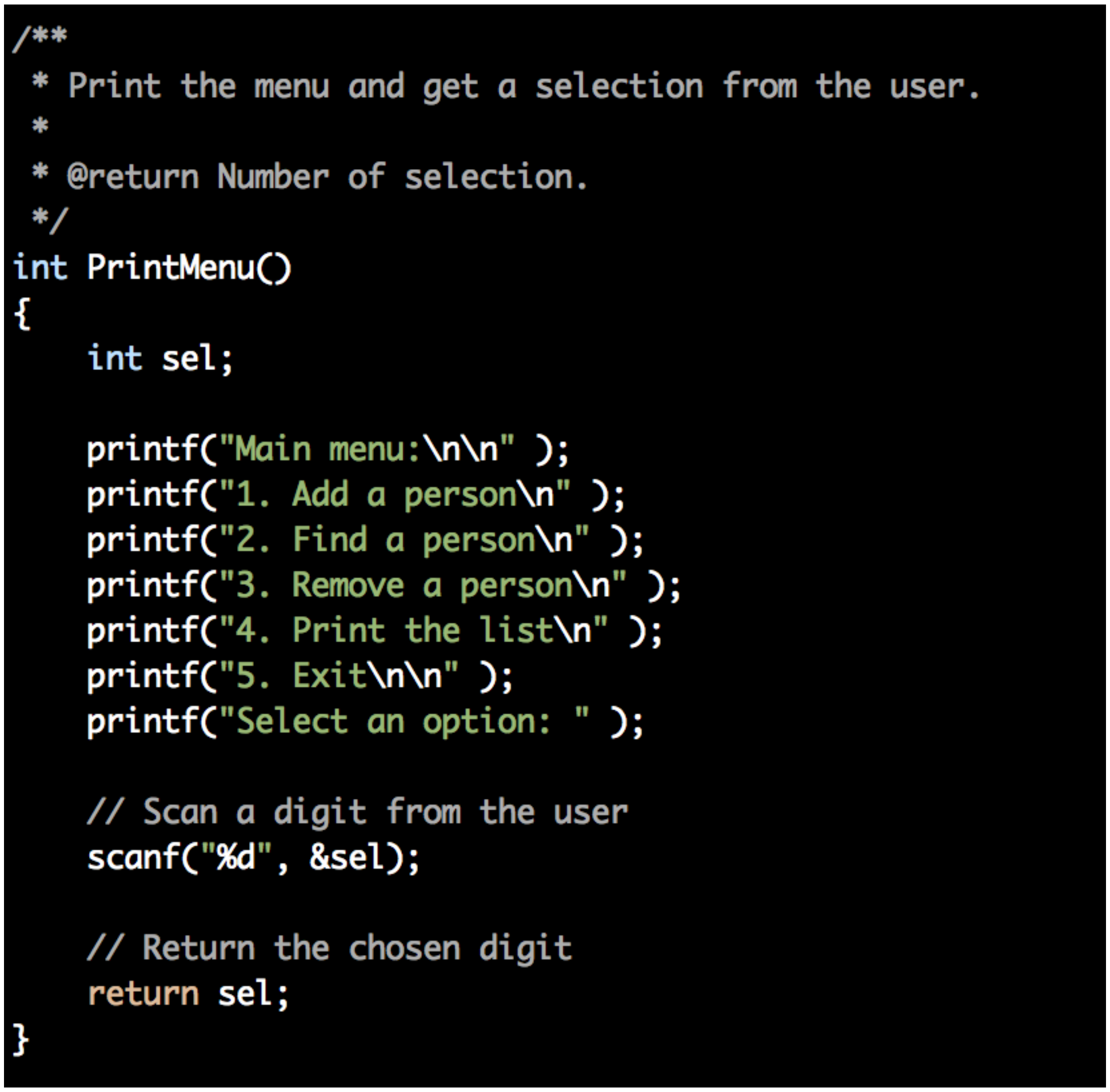
```
/**
 * Find a person from the list of people
 *
 * @param list, a pointer to the list of people
 * @param unique_id, the unique id of the person
 */
void FindPerson(struct List * list, int unique_id)
{
    /**
     * Let the user know if the person cannot be found.
     * Otherwise, print the user data.
     */
    ListFind(list, unique_id);
    if (ListGet(list) == NULL) {
        printf("A person with such an unique ID was not found! \n");
    } else {
        PrintPerson(ListGet(list));
    }
}
```



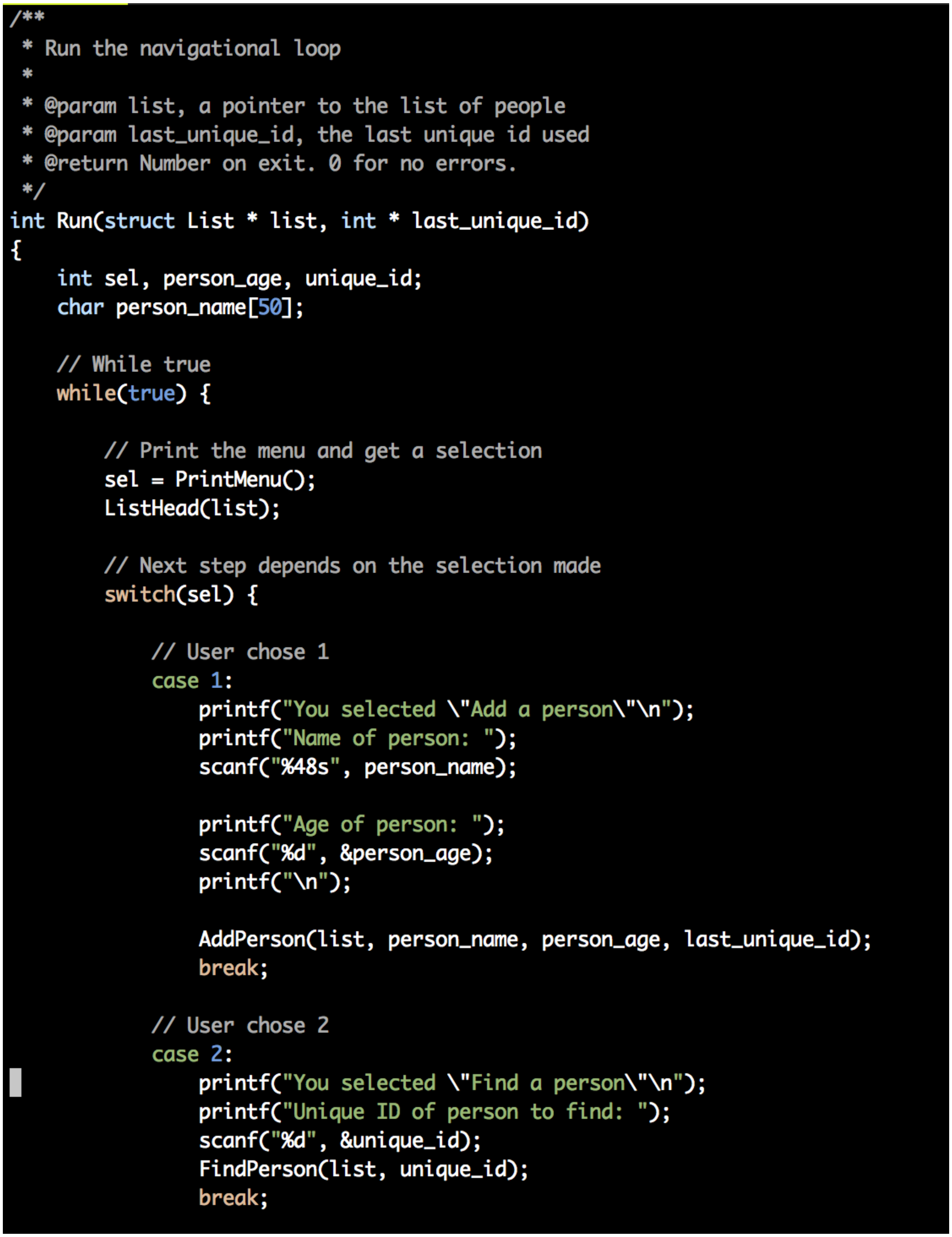
```
/**
 * Remove a person from the list of people
 *
 * @param list, a pointer to the list of people
 * @param unique_id, the unique id of the person to remove
 */
void RemovePerson(struct List * list, int unique_id)
{
    /**
     * Let the user know if the person cannot be found.
     * Otherwise, remove the person.
     */
    ListFind(list, unique_id);
    if (ListGet(list) == NULL) {
        printf("A person with such an unique ID was not found to remove! \n");
    } else {
        list->count--; // Decrease the count of people in the list
        ListRemove(list);
    }
}
```



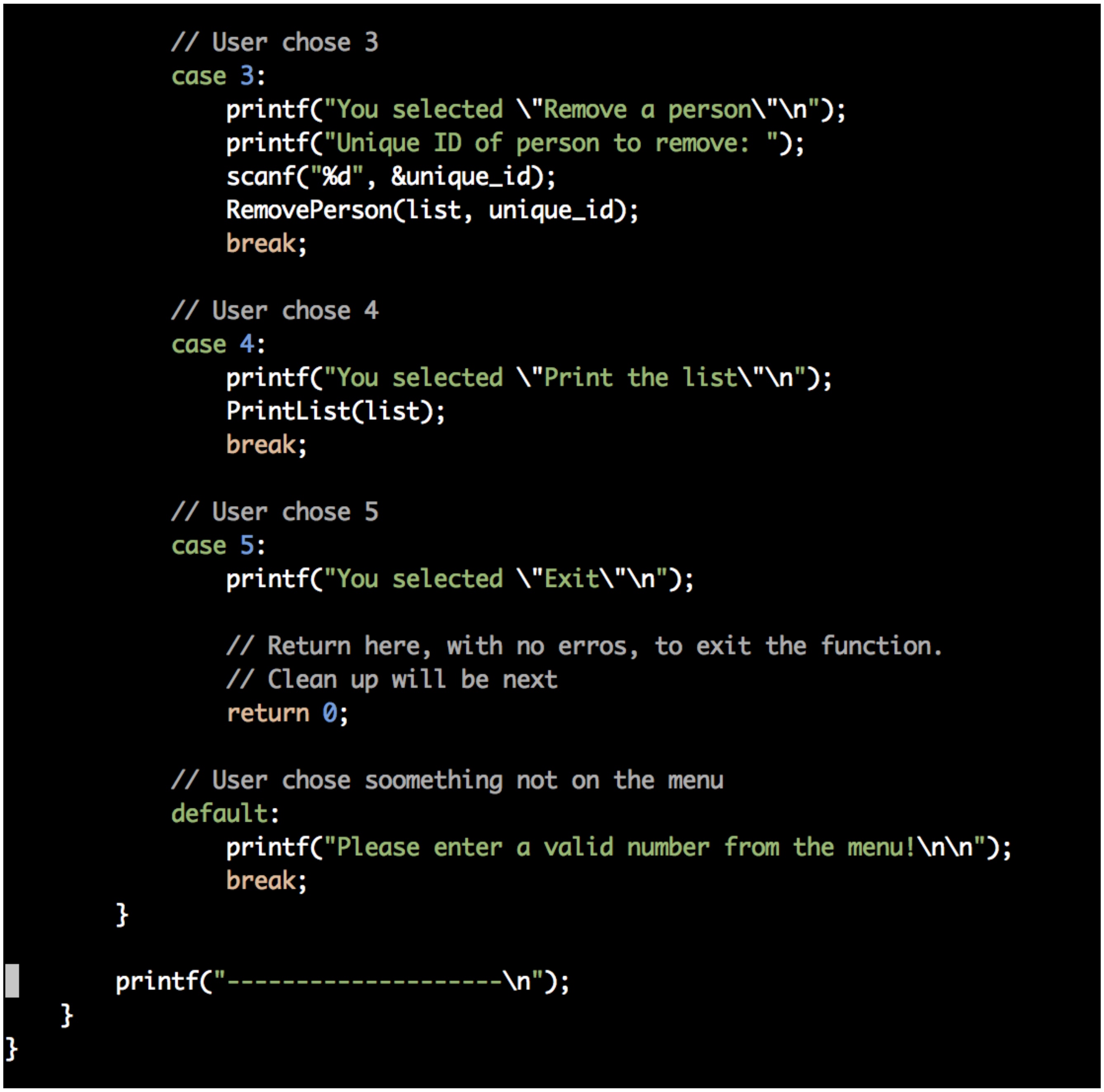
```
/**
 * Print the people in the list
 *
 * @param list, a pointer to the list of people
 */
void PrintList(struct List *list) {

    /**
     * Let the user know if the list is empty.
     * Otherwise, print each person in the list.
     */
    if(list->count == 0)
        printf("There's nothing in the list! \n");
    else {
        int i;
        for(i = 0; i < list->count; i++) {
            PrintPerson(ListGet(list));
            ListNext(list);
        }
    }
}
```



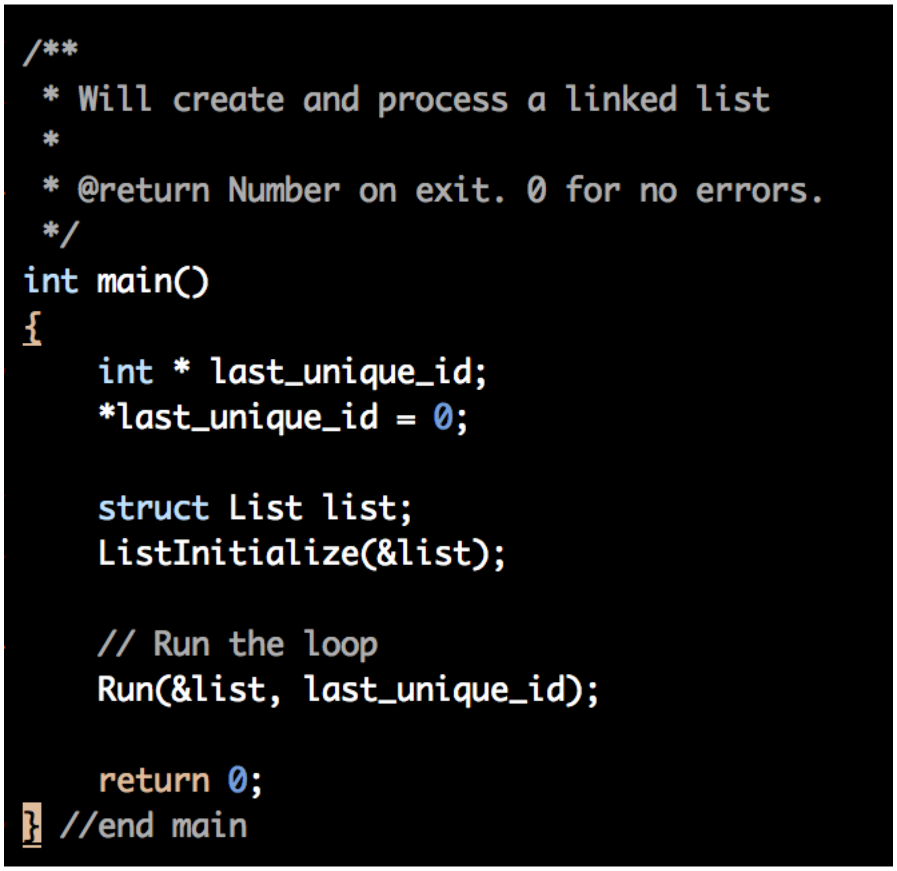
```
/**
 * Print the menu and get a selection from the user.
 *
 * @return Number of selection.
 */
int PrintMenu()
{
    int sel;

    printf("Main menu:\n\n" );
    printf("1. Add a person\n" );
    printf("2. Find a person\n" );
    printf("3. Remove a person\n" );
    printf("4. Print the list\n" );
    printf("5. Exit\n\n" );
    printf("Select an option: " );

    // Scan a digit from the user
    scanf("%d", &sel);

    // Return the chosen digit
    return sel;
}
```



```c
/**
 * Run the navigational loop
 *
 * @param list, a pointer to the list of people
 * @param last_unique_id, the last unique id used
 * @return Number on exit. 0 for no errors.
 */
int Run(struct List * list, int * last_unique_id)
{
    int sel, person_age, unique_id;
    char person_name[50];

    // While true
    while(true) {

        // Print the menu and get a selection
        sel = PrintMenu();
        ListHead(list);

        // Next step depends on the selection made
        switch(sel) {

            // User chose 1
            case 1:
                printf("You selected \"Add a person\"\n");
                printf("Name of person: ");
                scanf("%48s", person_name);

                printf("Age of person: ");
                scanf("%d", &person_age);
                printf("\n");

                AddPerson(list, person_name, person_age, last_unique_id);
                break;

            // User chose 2
            case 2:
                printf("You selected \"Find a person\"\n");
                printf("Unique ID of person to find: ");
                scanf("%d", &unique_id);
                FindPerson(list, unique_id);
                break;
```

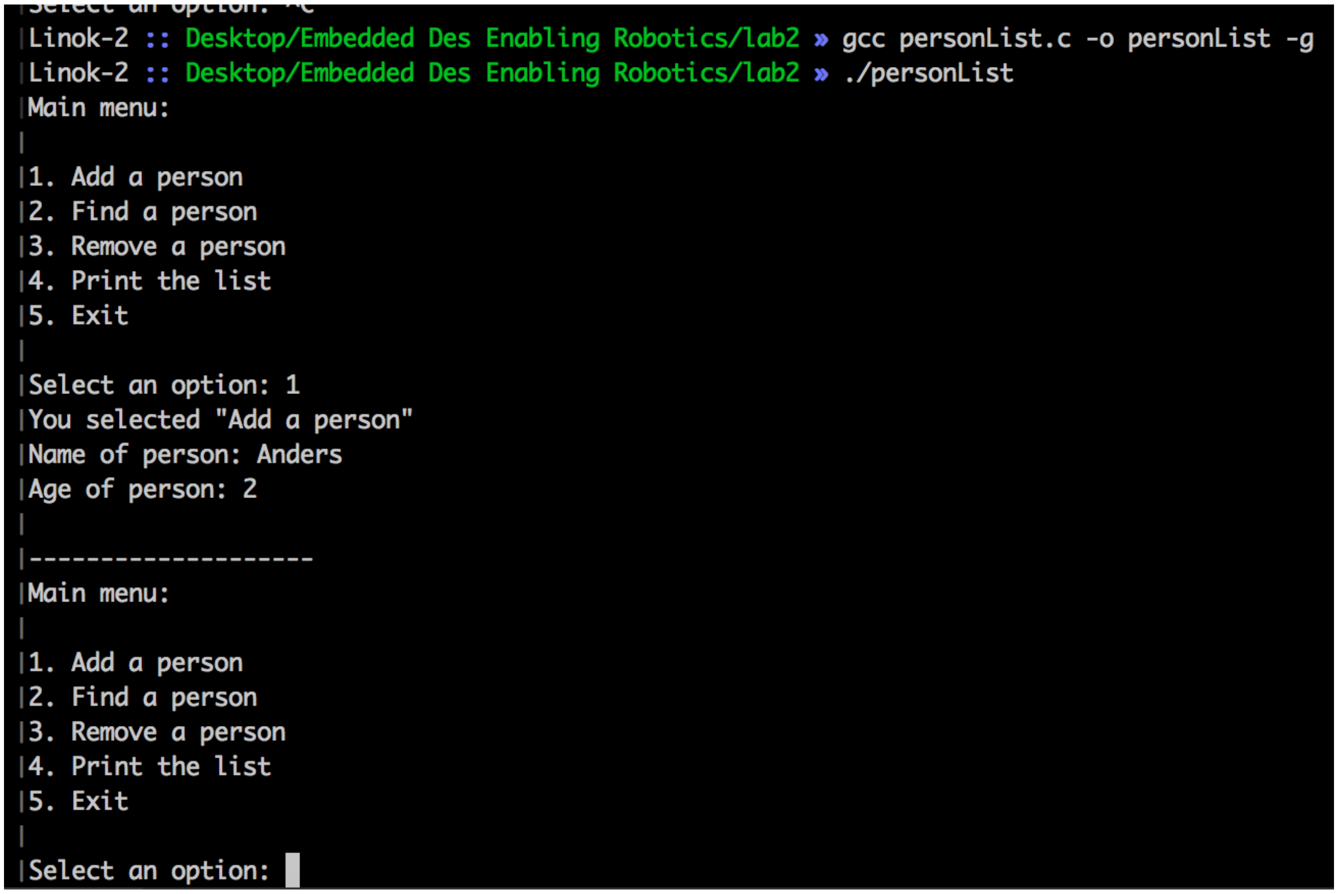


```
            // User chose 3
            case 3:
                printf("You selected \"Remove a person\"\n");
                printf("Unique ID of person to remove: ");
                scanf("%d", &unique_id);
                RemovePerson(list, unique_id);
                break;

            // User chose 4
            case 4:
                printf("You selected \"Print the list\"\n");
                PrintList(list);
                break;

            // User chose 5
            case 5:
                printf("You selected \"Exit\"\n");

                // Return here, with no erros, to exit the function.
                // Clean up will be next
                return 0;

            // User chose soomething not on the menu
            default:
                printf("Please enter a valid number from the menu!\n\n");
                break;
        }

        printf("--------------------\n");
    }
}
```



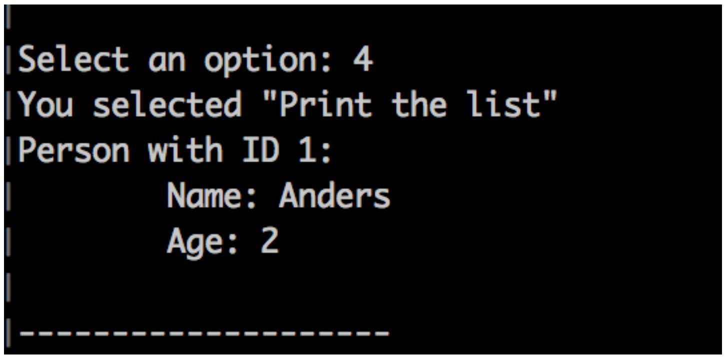
```
/**
 * Will create and process a linked list
 *
 * @return Number on exit. 0 for no errors.
 */
int main()
{
    int * last_unique_id;
    *last_unique_id = 0;

    struct List list;
    ListInitialize(&list);

    // Run the loop
    Run(&list, last_unique_id);

    return 0;
} //end main
```

Assignment 3 – Running of the Code:



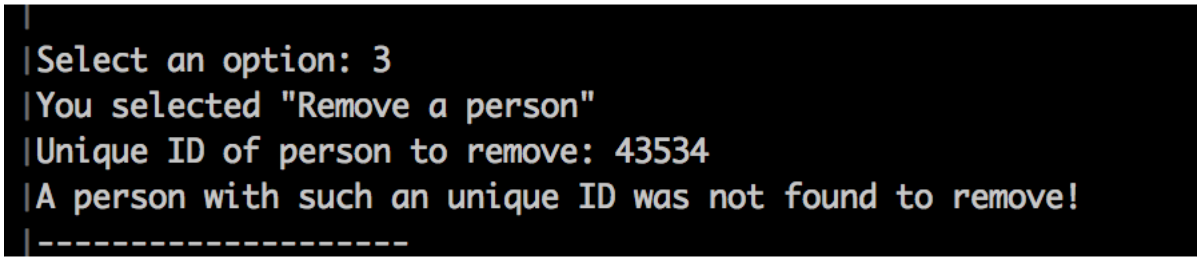
```
|Linok-2 :: Desktop/Embedded Des Enabling Robotics/lab2 » gcc personList.c -o personList -g
|Linok-2 :: Desktop/Embedded Des Enabling Robotics/lab2 » ./personList
|Main menu:
|
|1. Add a person
|2. Find a person
|3. Remove a person
|4. Print the list
|5. Exit
|
|Select an option: 1
|You selected "Add a person"
|Name of person: Anders
|Age of person: 2
|
|--------------------
|Main menu:
|
|1. Add a person
|2. Find a person
|3. Remove a person
|4. Print the list
|5. Exit
|
|Select an option:
```

Option 1: Here the correct results are shown. We added a person, and it gets added to the program's list of people. We would have to print the list in order to see it, however.

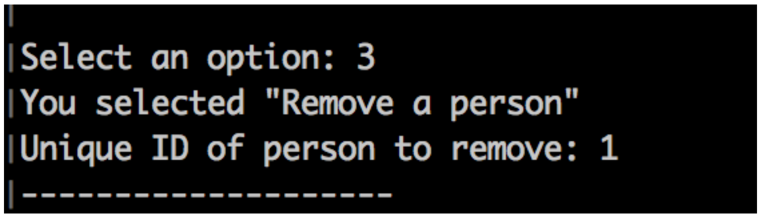


```
|
|Select an option: 4
|You selected "Print the list"
|Person with ID 1:
|        Name: Anders
|        Age: 2
|
|--------------------
```

Option 4: We print the list, and as expected, the added person shows up.

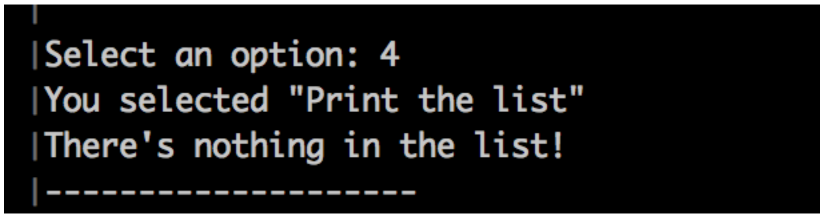


```
|
|Select an option: 3
|You selected "Remove a person"
|Unique ID of person to remove: 43534
|A person with such an unique ID was not found to remove!
|--------------------
```



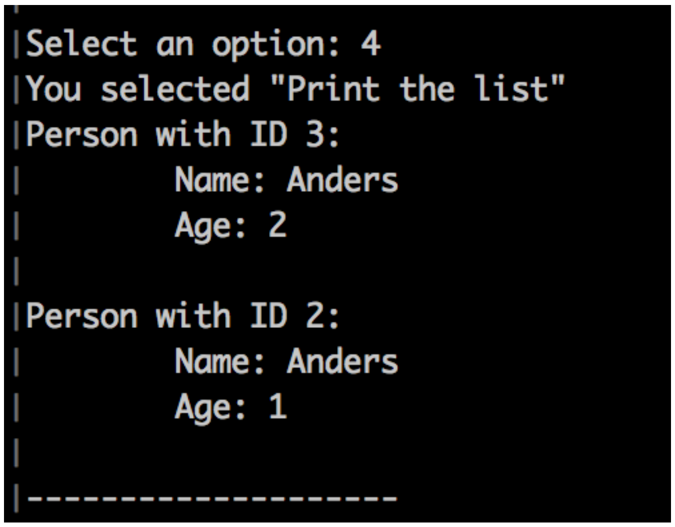
```
|
|Select an option: 3
|You selected "Remove a person"
|Unique ID of person to remove: 1
|--------------------
```

Option 3: If we try to remove a person who does not exist, we get an error. Otherwise, it removes the person.

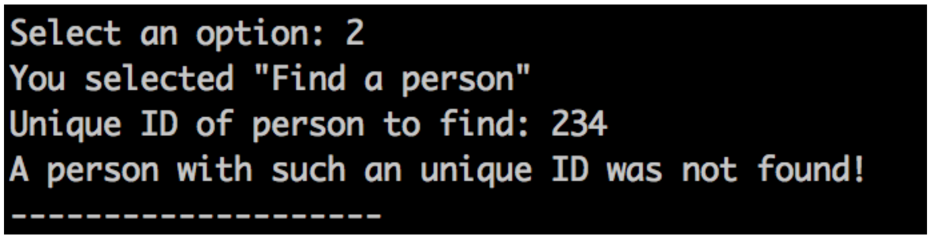


```
|
|Select an option: 4
|You selected "Print the list"
|There's nothing in the list!
|--------------------
```

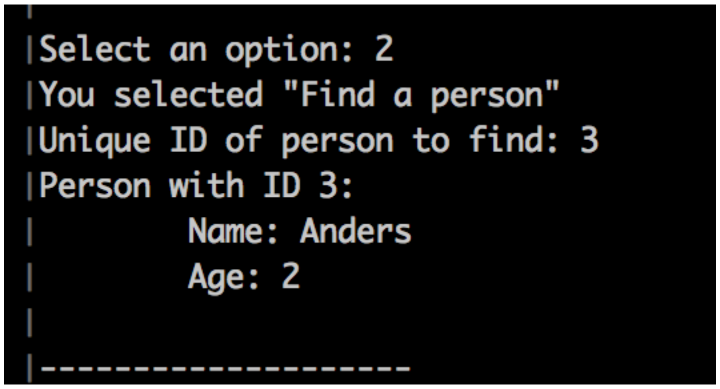
Option 1: Now, when the list is printed, it says the list is empty as expected.



```
|Select an option: 4
|You selected "Print the list"
|Person with ID 3:
|        Name: Anders
|        Age: 2
|
|Person with ID 2:
|        Name: Anders
|        Age: 1
|
|--------------------
```



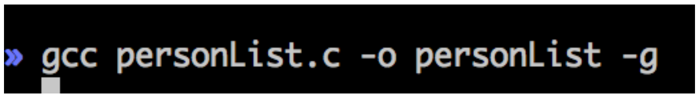
```
Select an option: 2
You selected "Find a person"
Unique ID of person to find: 234
A person with such an unique ID was not found!
--------------------
```



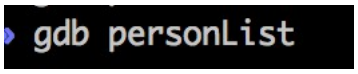
```
|Select an option: 2
|You selected "Find a person"
|Unique ID of person to find: 3
|Person with ID 3:
|        Name: Anders
|        Age: 2
|
|--------------------
```

Option 2: If we try to find a person who does not exist, we get an error. Otherwise, it finds the person.

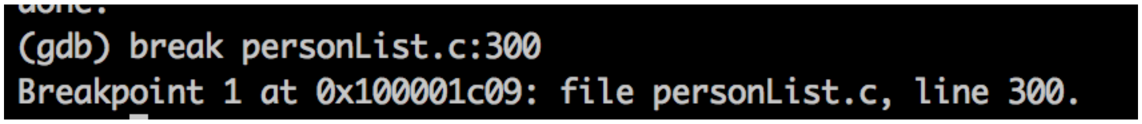
Assignment 4:



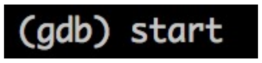
```
» gcc personList.c -o personList -g
```



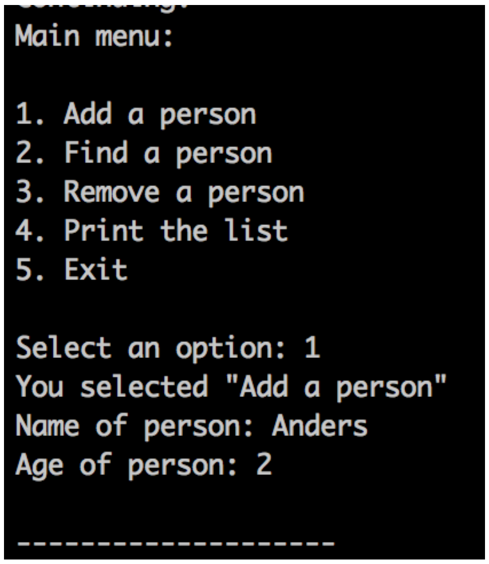
```
› gdb personList
```



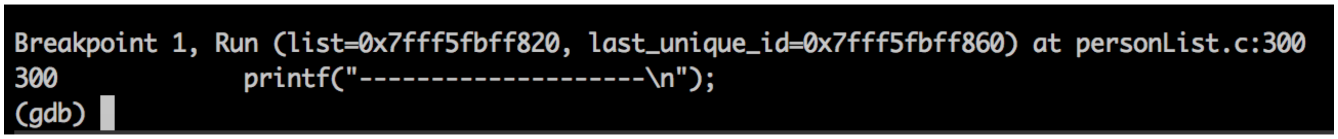
```
(gdb) break personList.c:300
Breakpoint 1 at 0x100001c09: file personList.c, line 300.
```



```
(gdb) start
```

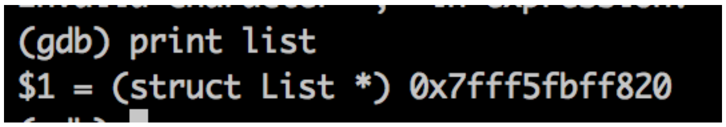


```
Main menu:

1. Add a person
2. Find a person
3. Remove a person
4. Print the list
5. Exit

Select an option: 1
You selected "Add a person"
Name of person: Anders
Age of person: 2


--------------------
```



```
Breakpoint 1, Run (list=0x7fff5fbff820, last_unique_id=0x7fff5fbff860) at personList.c:300
300             printf("-------------------\n");
(gdb)
```

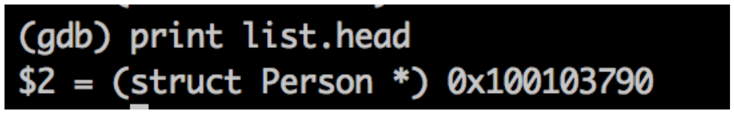
(gdb) print list



```
(gdb) print list
$1 = (struct List *) 0x7fff5fbff820
```

The command outputs the entire address stored in the pointer called list. It also describes what type of data the pointer points to, a List struct.

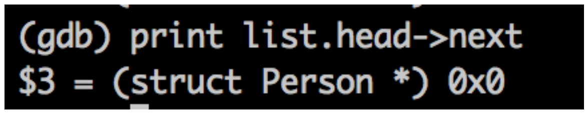
(gdb) print list.head



```
(gdb) print list.head
$2 = (struct Person *) 0x100103790
```

The command outputs the entire address of the head of the linked list. It also describes what type of data the pointer points to, a Person struct.

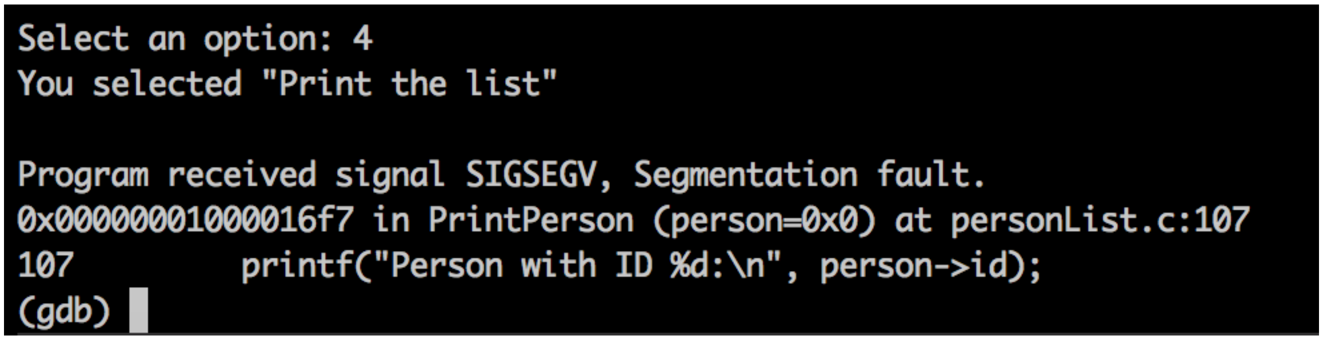
(gdb) print list.head->next



```
(gdb) print list.head->next
$3 = (struct Person *) 0x0
```

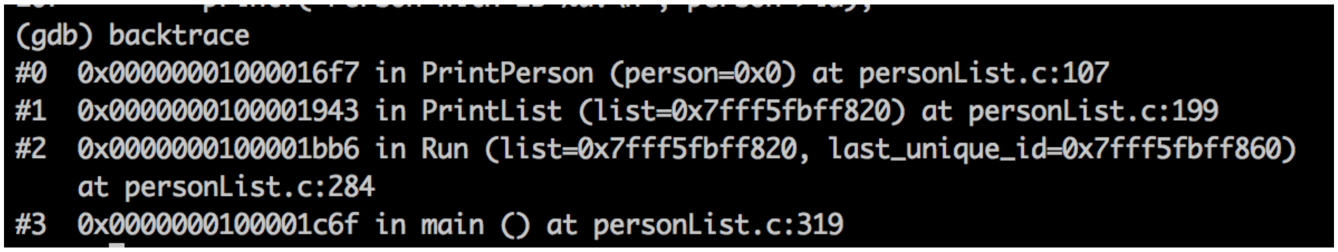
The command outputs the person after the head of the linked list. Since we only added one person, there is nothing after the head. The pointer points to NULL, 0x0, or nothing. It also describes what type of data the pointer should point to if a pointer existed, a person Struct.

Assignment 5:



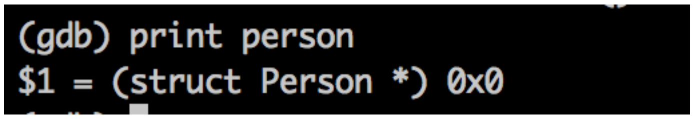
```
Select an option: 4
You selected "Print the list"

Program received signal SIGSEGV, Segmentation fault.
0x00000001000016f7 in PrintPerson (person=0x0) at personList.c:107
107         printf("Person with ID %d:\n", person->id);
(gdb) 
```

A segmentation fault here in indicates that the program was trying to read or write to an illegal memory location.



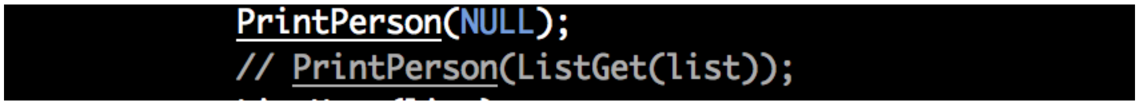
```
(gdb) backtrace
#0  0x00000001000016f7 in PrintPerson (person=0x0) at personList.c:107
#1  0x0000000100001943 in PrintList (list=0x7fff5fbff820) at personList.c:199
#2  0x0000000100001bb6 in Run (list=0x7fff5fbff820, last_unique_id=0x7fff5fbff860)
    at personList.c:284
#3  0x0000000100001c6f in main () at personList.c:319
```

The backtrace (a trace going backwards) shows that the underlying issue occurs at personList.c:107. The person with address 0x0 seems to be having issues, obviously.



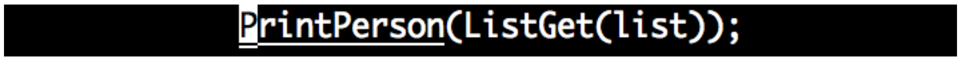
```
(gdb) print person
$1 = (struct Person *) 0x0
```

Print the person at this variable. Notice how the address is NULL.



```
            PrintPerson(NULL);
            // PrintPerson(ListGet(list));
```

After investigating the code at personList.c:107 the issue presents itself. We remove the line causing the error.

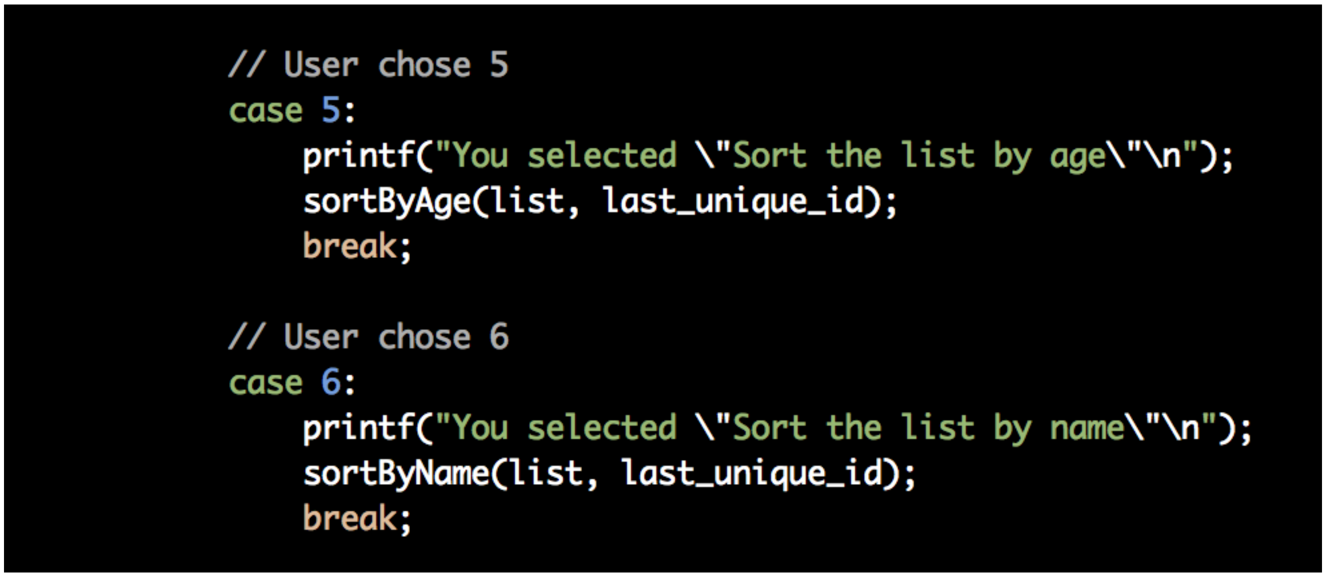


```
            PrintPerson(ListGet(list));
```

EXTRA CREDIT:

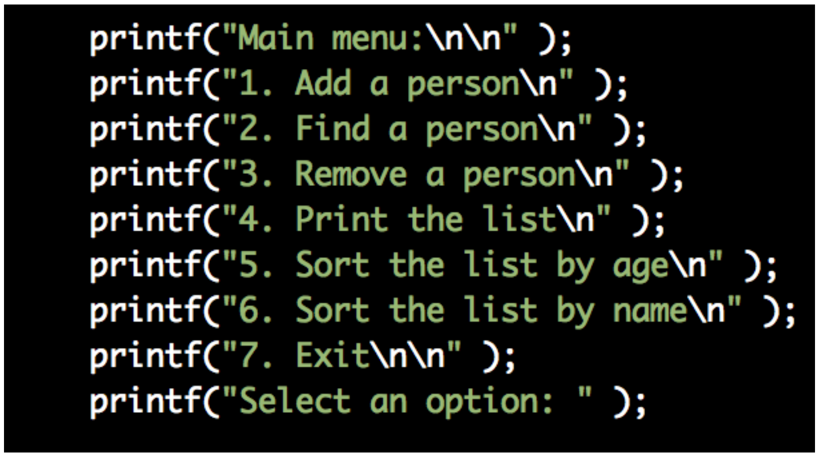
Extra Credit – Code:

In Run:

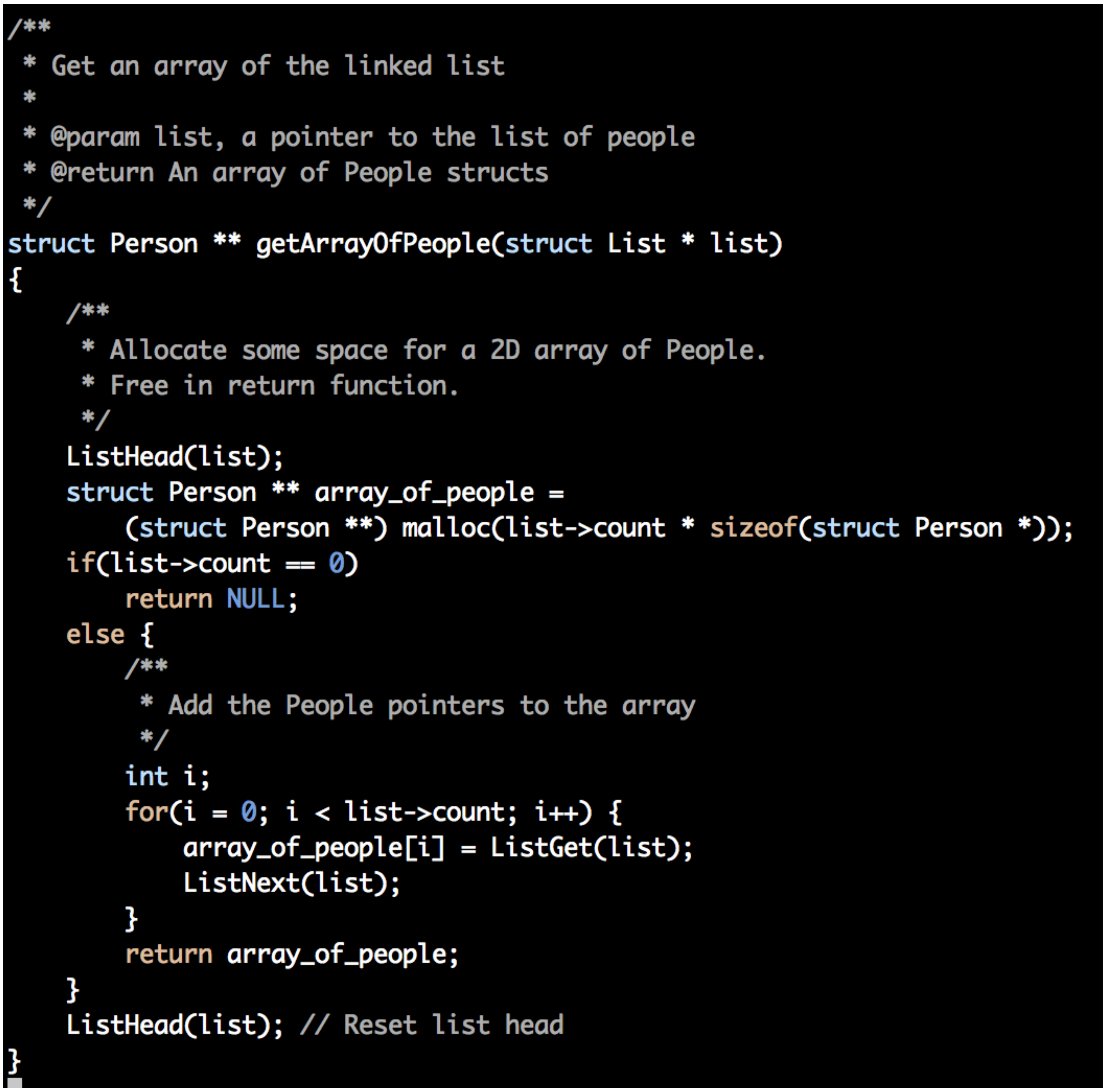


```
        // User chose 5
        case 5:
            printf("You selected \"Sort the list by age\"\n");
            sortByAge(list, last_unique_id);
            break;

        // User chose 6
        case 6:
            printf("You selected \"Sort the list by name\"\n");
            sortByName(list, last_unique_id);
            break;
```

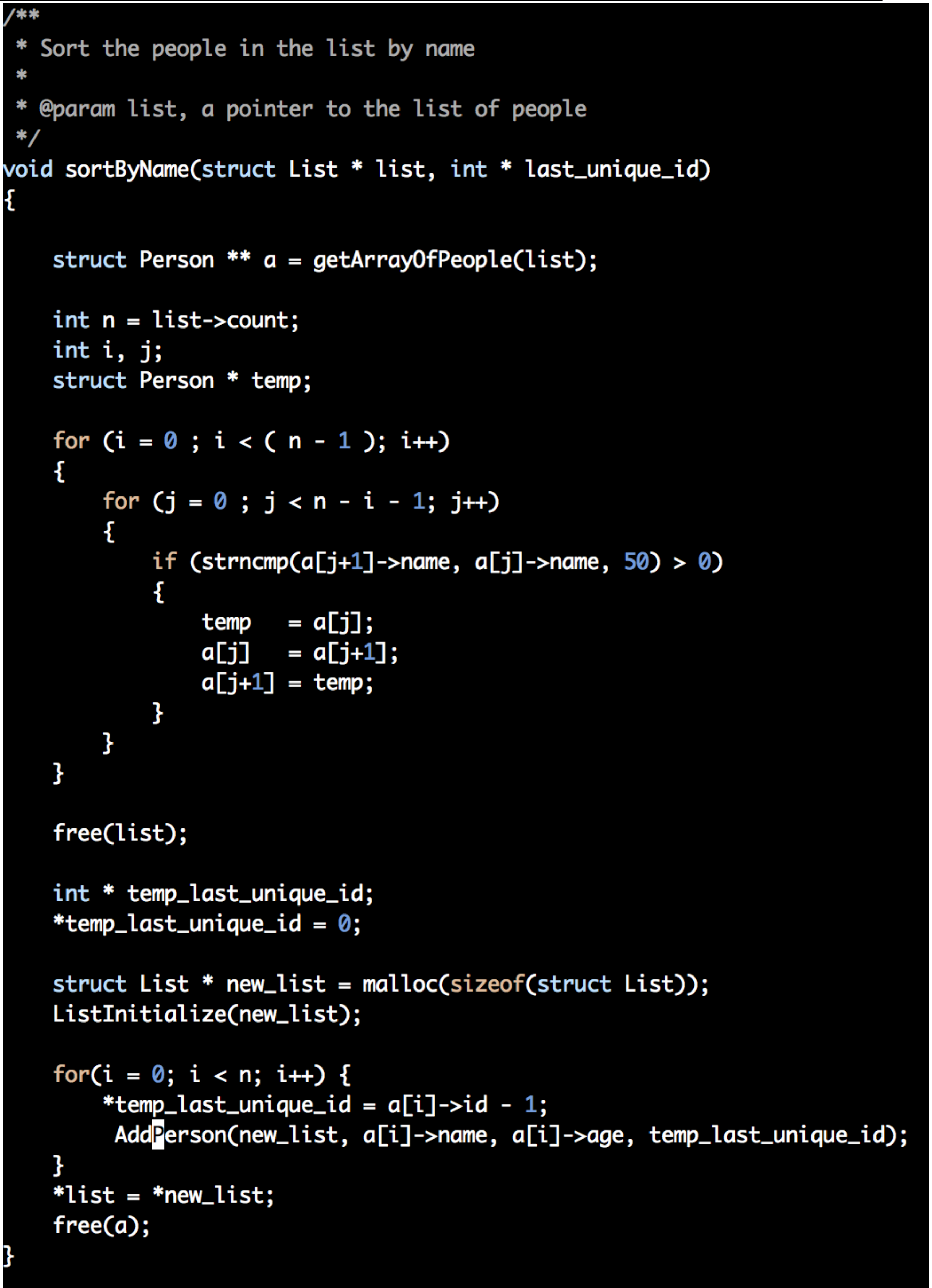
In PrintMenu:



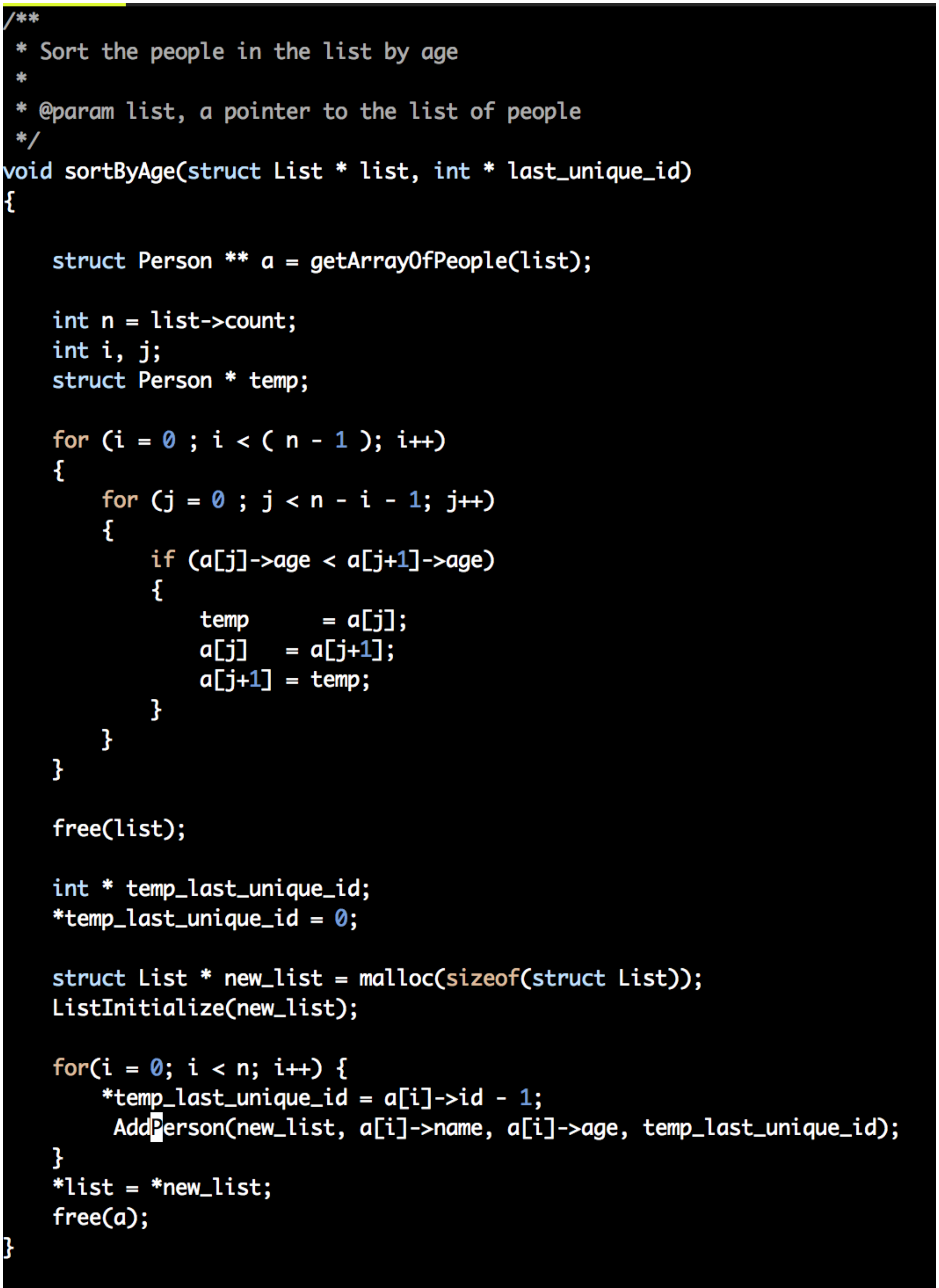
```
    printf("Main menu:\n\n" );
    printf("1. Add a person\n" );
    printf("2. Find a person\n" );
    printf("3. Remove a person\n" );
    printf("4. Print the list\n" );
    printf("5. Sort the list by age\n" );
    printf("6. Sort the list by name\n" );
    printf("7. Exit\n\n" );
    printf("Select an option: " );
```



```
/**
 * Get an array of the linked list
 *
 * @param list, a pointer to the list of people
 * @return An array of People structs
 */
struct Person ** getArrayOfPeople(struct List * list)
{
    /**
     * Allocate some space for a 2D array of People.
     * Free in return function.
     */
    ListHead(list);
    struct Person ** array_of_people =
        (struct Person **) malloc(list->count * sizeof(struct Person *));
    if(list->count == 0)
        return NULL;
    else {
        /**
         * Add the People pointers to the array
         */
        int i;
        for(i = 0; i < list->count; i++) {
            array_of_people[i] = ListGet(list);
            ListNext(list);
        }
        return array_of_people;
    }
    ListHead(list); // Reset list head
}
```

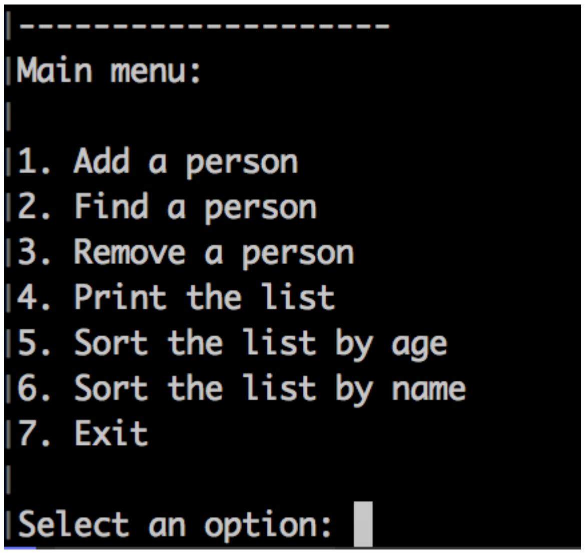


```c
/**
 * Sort the people in the list by name
 *
 * @param list, a pointer to the list of people
 */
void sortByName(struct List * list, int * last_unique_id)
{

    struct Person ** a = getArrayOfPeople(list);

    int n = list->count;
    int i, j;
    struct Person * temp;

    for (i = 0 ; i < ( n - 1 ); i++)
    {
        for (j = 0 ; j < n - i - 1; j++)
        {
            if (strncmp(a[j+1]->name, a[j]->name, 50) > 0)
            {
                temp   = a[j];
                a[j]   = a[j+1];
                a[j+1] = temp;
            }
        }
    }

    free(list);

    int * temp_last_unique_id;
    *temp_last_unique_id = 0;

    struct List * new_list = malloc(sizeof(struct List));
    ListInitialize(new_list);

    for(i = 0; i < n; i++) {
        *temp_last_unique_id = a[i]->id - 1;
         AddPerson(new_list, a[i]->name, a[i]->age, temp_last_unique_id);
    }
    *list = *new_list;
    free(a);
}
```

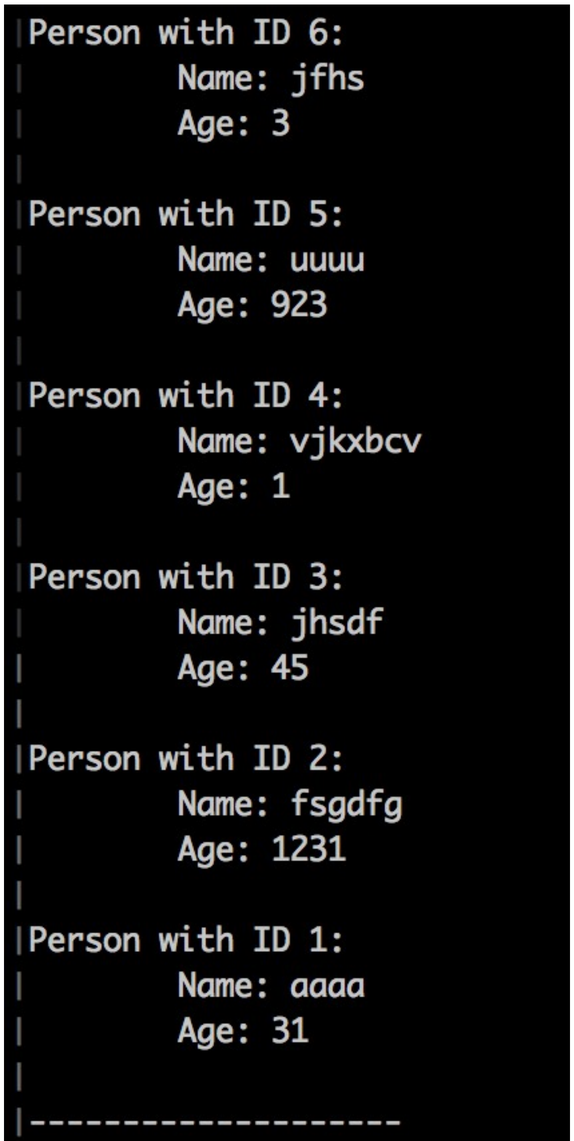


```
/**
 * Sort the people in the list by age
 *
 * @param list, a pointer to the list of people
 */
void sortByAge(struct List * list, int * last_unique_id)
{

    struct Person ** a = getArrayOfPeople(list);

    int n = list->count;
    int i, j;
    struct Person * temp;

    for (i = 0 ; i < ( n - 1 ); i++)
    {
        for (j = 0 ; j < n - i - 1; j++)
        {
            if (a[j]->age < a[j+1]->age)
            {
                temp      = a[j];
                a[j]    = a[j+1];
                a[j+1] = temp;
            }
        }
    }

    free(list);

    int * temp_last_unique_id;
    *temp_last_unique_id = 0;

    struct List * new_list = malloc(sizeof(struct List));
    ListInitialize(new_list);

    for(i = 0; i < n; i++) {
        *temp_last_unique_id = a[i]->id - 1;
         AddPerson(new_list, a[i]->name, a[i]->age, temp_last_unique_id);
    }
    *list = *new_list;
    free(a);
}
```

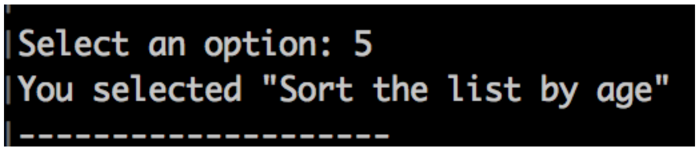
Extra Credit – Running the Code:



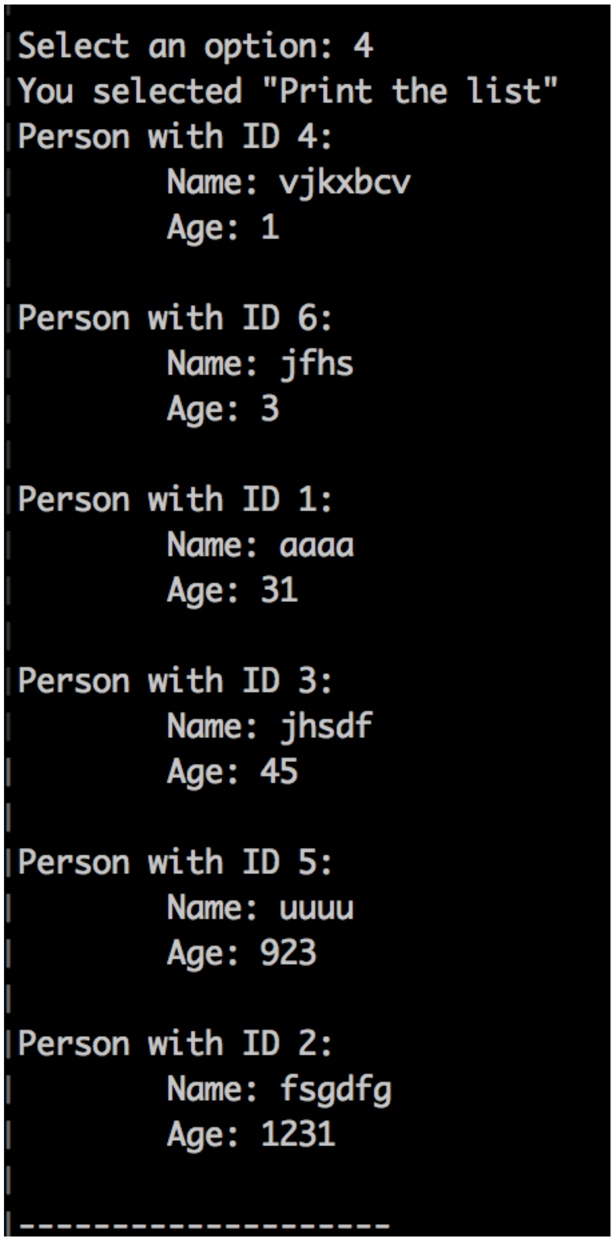
```
--------------------
Main menu:

1. Add a person
2. Find a person
3. Remove a person
4. Print the list
5. Sort the list by age
6. Sort the list by name
7. Exit

Select an option:
```



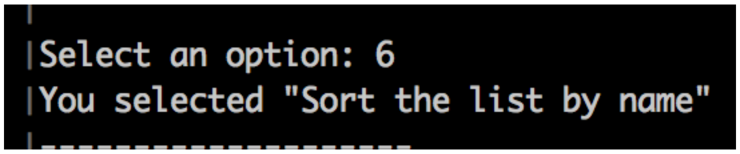
```
Person with ID 6:
        Name: jfhs
        Age: 3

Person with ID 5:
        Name: uuuu
        Age: 923

Person with ID 4:
        Name: vjkxbcv
        Age: 1

Person with ID 3:
        Name: jhsdf
        Age: 45

Person with ID 2:
        Name: fsgdfg
        Age: 1231

Person with ID 1:
        Name: aaaa
        Age: 31

--------------------
```



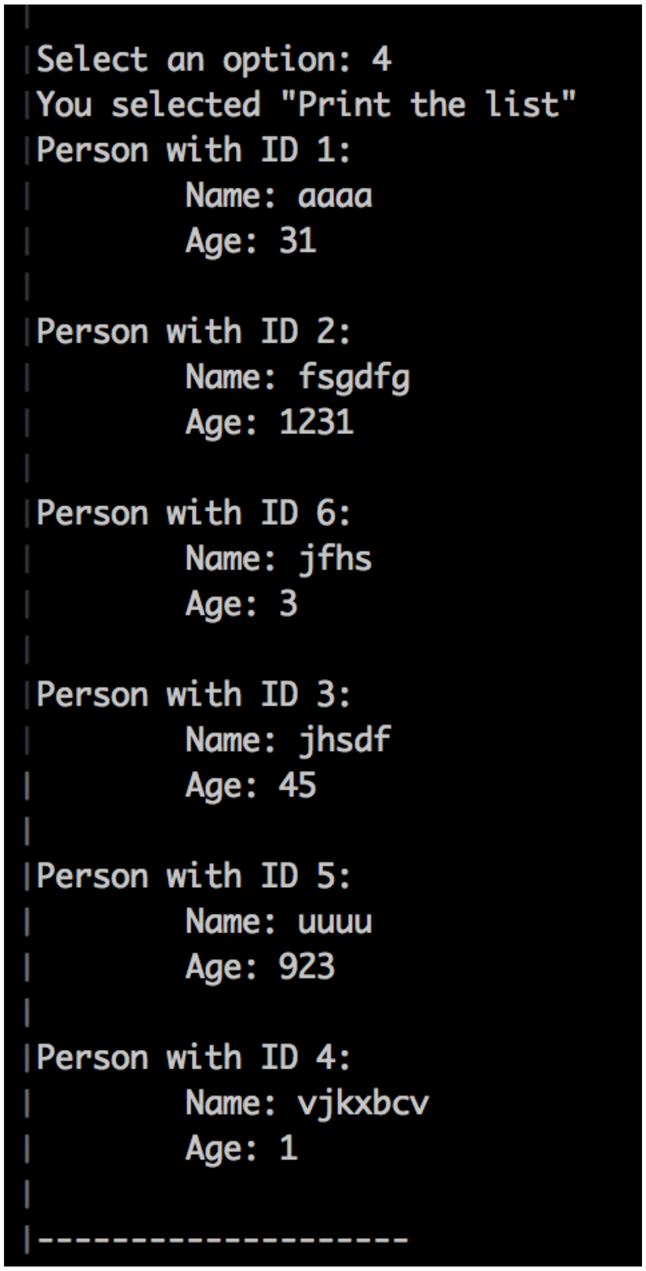
```
Select an option: 5
You selected "Sort the list by age"
--------------------
```



```
Select an option: 4
You selected "Print the list"
Person with ID 4:
        Name: vjkxbcv
        Age: 1

Person with ID 6:
        Name: jfhs
        Age: 3

Person with ID 1:
        Name: aaaa
        Age: 31

Person with ID 3:
        Name: jhsdf
        Age: 45

Person with ID 5:
        Name: uuuu
        Age: 923

Person with ID 2:
        Name: fsgdfg
        Age: 1231

--------------------
```



```
|
|Select an option: 6
|You selected "Sort the list by name"
|--------------------
```



```
|
|Select an option: 4
|You selected "Print the list"
|Person with ID 1:
|        Name: aaaa
|        Age: 31
|
|Person with ID 2:
|        Name: fsgdfg
|        Age: 1231
|
|Person with ID 6:
|        Name: jfhs
|        Age: 3
|
|Person with ID 3:
|        Name: jhsdf
|        Age: 45
|
|Person with ID 5:
|        Name: uuuu
|        Age: 923
|
|Person with ID 4:
|        Name: vjkxbcv
|        Age: 1
|
|--------------------
```