CS351 Project 3 - LinkedList & struct

Sean Poston & Logan Emel

4/28/2020

repl.it link:
https://repl.it/@seanposton4/CS351Project3Struct

YouTube link:
https://youtu.be/WfYp1m_JSqw

Abstract

Tasked to create a program that input 20 pieces of data into a file, be able to load them, search through the list, along with deleting a record on the list, is to some a very difficult task, which proved to be the case to some degree. However today, presented along with this report, is a completed program that does just that! Continue reading for an explanation to how each of the pieces work!

Introduction

This project proved to be challenging and educational in the end. In C language, data structures are much harder than other languages because there is no default library that holds the data structures and their respective functions. In this project, the simple task of populating a LinkedList and a file with records of twenty different people was difficult due to the aforementioned reasons.

The project also required three functions to be used with the file and the LinkedList: load, search, and remove. The load function was used to print out, in a formatted manner, the LinkedList with the current records. The search function was used to search the LinkedList for a specific data value and return the full record of the found record, if it existed. Finally the remove function was meant to delete a node from the LinkedList and the same line from the output file. There will be more details on these later on in the report.

Working as a group made this project more educational and easier in the end. It was educational in the respect that the group members traded ideas and helped each other with problems and difficult areas. The group aspect proved to make the project easier because, as well as helping each other, the group members could divide and conquer the project as a whole. One member did the menu function, the LinkedList base operations, and the file initialization. The other member did the LinkedList functions required in the assignment. Both members worked on this report to make it as in-depth as possible.

The rest of this report will be specifics regarding the different code areas of the project. This should break down into three areas: the LinkedList, the output file, and the four main functions. The last section will take the longest, and it will be the most thoroughly covered section in this report. This group thanks you for reading.

Main Contexts

How about starting with the beginning, the introductory parts of the code!

```c
/*
Authors: Sean Poston & Logan Emel
Purpose: Create a program that manipulates LinkedLists and files.
Date: 4/28/2020
*/
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
//Fun Time Fun Zone Below

struct personNode {
    int ID;
    char lastName[20];
    char firstName[20];
    int age;
    struct personNode *nextPtr;
}; //initialize struct

FILE *fPtr;
```

Here we have, all the different headers needed for the code, along with the struct personNode that was received to work with. FILE *fPtr; defining fPtr to the file that will soon be accessed.

```c
//Define PersonNode Pointer Type
typedef struct personNode PersonNode;
typedef PersonNode *PersonNodePtr;
//struct personNode personNode;

//Function Prototypes
bool initalizeFile(PersonNodePtr *sPtr, FILE *fileOut);
void load(PersonNodePtr currentPtr);
void search(PersonNodePtr currentPtr, int searchInput);
void destroy(PersonNodePtr currentPtr, FILE *fileOut); //remove and delete are both keywords in C coolcoolcool
int getNumOfNodes(PersonNodePtr currentPtr);
int printMenu(void);
int printSearchMenu(void);
```

Lots of functions, oh my! First two lines assign pointers to the aforementioned structure. Beginning with initializeFile(), it creates the file being worked with, where the task was to create a file with default data to begin, which was concluded, having an array of information would be easiest, shown here:

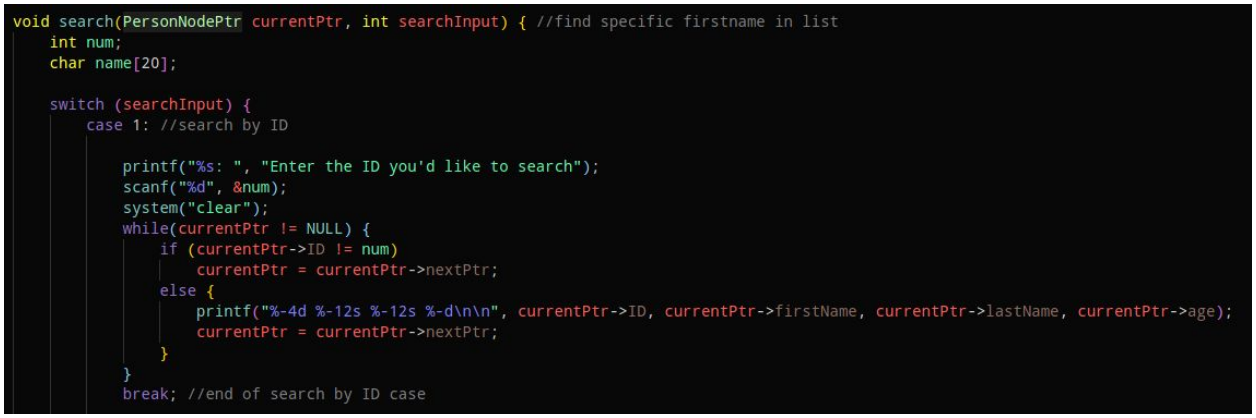
```c
bool initalizeFile(PersonNodePtr *sPtr, FILE *fPtr) { //inputs to file and initialized LinkedList
    //input function
    //arrays for automatic input (makes things less cluttered)
    char inlast[20][20] = {"Smith", "Smith", "Smith", "Smith", "Smith", "Smith", "Smith", "Smith", "Brown", "Brown", "Brown", "
    char infirst[20][20] = {"Liam", "Noah", "William", "James", "Oliver", "Benjamin", "Elijah", "Lucas", "Mason", "Logan", "Emma", "Oliv
    int inage[20] = {25, 26, 27, 60, 35, 45, 10, 35, 20, 19, 29, 39, 49, 59, 18, 17, 12, 10, 8, 15};

    //actual writing function

    for (int i = 0; i < 20; i++) {
        PersonNodePtr newPtr;
        PersonNodePtr previousPtr;
        PersonNodePtr currentPtr;
        newPtr = malloc(sizeof(PersonNode));

        if(newPtr != NULL) {
            newPtr->ID = (i + 1);
            strcpy(newPtr->firstName, &infirst[i - 1][20]);
            strcpy(newPtr->lastName, &inlast[i - 1][20]);
            newPtr->age = inage[i];
            newPtr->nextPtr = NULL;

            if (i < 19)
                fprintf(fPtr, "%d, %s, %s, %d\n", newPtr->ID, newPtr->firstName, newPtr->lastName, newPtr->age);
            else
                fprintf(fPtr, "%d, %s, %s, %d", newPtr->ID, newPtr->firstName, newPtr->lastName, newPtr->age); //don't print newline on

            previousPtr = NULL;
            currentPtr = *sPtr;

            if (previousPtr == NULL) {
                newPtr->nextPtr = *sPtr;
                *sPtr = newPtr;
            }
            else {
                previousPtr->nextPtr = newPtr;
                newPtr->nextPtr = currentPtr;
            }
        }
        else {
            puts("LinkedList already initialized");
            return false;
        }
    }
        return true;
}
```

The arrays put in demo information, and the for loop proceeds to both print the values to a file, along with adding them to a linked list!
Moving on to load(), this loads the file for printing, as shown:

```c
void load(PersonNodePtr currentPtr) { //print out the linkedlist of the file and show average age
    int ageSum = 0;
    int maxID = 0; //used to calculate data entries
    int nodes = getNumOfNodes(currentPtr);


    printf("%-4s %-12s %-12s %-s\n", "ID", "First Name", "Last Name", "Age");
    while(currentPtr != NULL) {
    //set list item as active account to print
        if (currentPtr->ID > maxID)
            maxID = currentPtr->ID;
        ageSum += currentPtr->age; //add to age sum
        printf("%-4d %-12s %-12s %-d\n", currentPtr->ID, currentPtr->firstName, currentPtr->lastName, currentPtr->age);
        currentPtr = currentPtr->nextPtr;
    }

    printf("\nThe average age is: %.2f\n\n", (float)ageSum / (float)nodes);
}
```

It defines variables that also find the sum and maximum ID number of the values. It scrolls through the list using the linked list to display all the information for the user, before printing the average at the bottom.

Up next, search(), which is very large. There are four cases of search, each being able to search for their respective data values (ID, First Name, Last Name, Age). For the sake of saving space, and because they're nearly identical, only one case will be shown in this screenshot:



```c
void search(PersonNodePtr currentPtr, int searchInput) { //find specific firstname in list
    int num;
    char name[20];

    switch (searchInput) {
        case 1: //search by ID

            printf("%s: ", "Enter the ID you'd like to search");
            scanf("%d", &num);
            system("clear");
            while(currentPtr != NULL) {
                if (currentPtr->ID != num)
                    currentPtr = currentPtr->nextPtr;
                else {
                    printf("%-4d %-12s %-12s %-d\n\n", currentPtr->ID, currentPtr->firstName, currentPtr->lastName, currentPtr->age);
                    currentPtr = currentPtr->nextPtr;
                }
            }
            break; //end of search by ID case
```

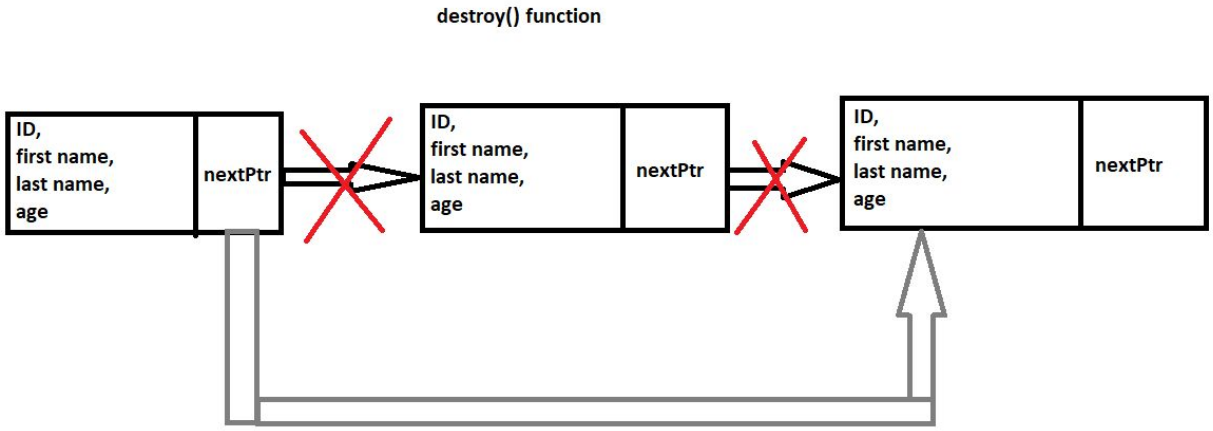
This will take the ID number from the user, and then search the linked list until it finds the record that has that ID/name/etc, depending on the initial option the user selected to search.

Continuing forward, destroy():

```c
void destroy(PersonNodePtr currentPtr, FILE *fileOut) { //removes a node from the LinkedList and a line from the file https://www.c
    PersonNodePtr temp;
    PersonNodePtr head;
    head = currentPtr;

    int nodes = getNumOfNodes(currentPtr); //get rows
    int idNum;

    printf("%s: ", "Enter the ID number to remove");
    scanf("%d", &idNum);
    idNum++;

    while(currentPtr != NULL) {
        if (currentPtr->ID == idNum) {
            temp = currentPtr->nextPtr;
            currentPtr->nextPtr = temp->nextPtr;
            free(temp);
            break;
        }
        else
            currentPtr = currentPtr->nextPtr;
    }// end of while loop

    reallocateID(head);

    // int row = 0; //block of code to remove line from the file https://www.geeksforgeeks.org/c-program-count-number-lines-file/
    // char c;
    // for (c = getc(fileOut); c != EOF; c = getc(fileOut)) {
    //  if (c == '\n')
    //      row++;
    //  if (row == idNum - 1) {

    //  }
    // }
}
```

Repl wouldn't allow for the name remove for the function, so destroy was the closest thing. There was a lot of trouble getting this portion to work. This essentially asks for the user to input the ID number of the record to remove, scrolls through to find that ID, and removes it from the list. This portion of the project remains incomplete due to time and difficulty. It's halfway complete because it flawlessly removes the node from the LinkedList, reallocates the ID

numbers to fill the missing gap, and prints it out perfectly. However, it does not remove the line from the file. Here is a diagram showing the method's functionality:



The middle node, which is temp in this case, is left to be freed into nonexistence. Other languages like Java have garbage collection, but in a lower language like C, it must be freed by the programmer with *free(temp)*.

Next, a reallocate function:

```
void reallocateID(PersonNodePtr currentPtr) { //reset the IDs after removing a node
    for (int i = getNumOfNodes(currentPtr); currentPtr != NULL; i--) {
        currentPtr->ID = i;
        currentPtr = currentPtr->nextPtr;
    }
}
```

Essentially this just reconfigures the linked list when a record gets deleted.
And the last of the functional functions, getNumOfNodes:

```
int getNumOfNodes(PersonNodePtr currentPtr) { //used to get number of getNumOfNodes in the linked list
    int nodes = 0;

    while(currentPtr != NULL) {
        nodes++;
        currentPtr = currentPtr->nextPtr;
    }
    return nodes;
}
```

This essentially, just figures out how many entries are in the list.

And finally, the last two functions, the two menu print functions:

```c
int printMenu(void) {
    int input;
    puts("----------------------------");
    printf("%s","1 to print.\n2 to search.\n3 to remove.\n0 to exit.\n");
    puts("----------------------------");
    printf("%s", "\nEnter your choice: ");
    scanf("%d", &input);

    return input;
}

int printSearchMenu(void) {
    int input;
    puts("----------------------------");
    printf("%s","1 to search by ID.\n2 to search by first name.\n3 to search by last name.\n4 to search by age.\n0 to return.\n");
    puts("----------------------------");
    printf("%s", "\nEnter your choice: ");
    scanf("%d", &input);

    return input;
}
```

Both of these are pretty self explanatory, they print the menus called earlier in the Main() function.

Speaking of Main():

```c
int main(void) {
    system("clear");
    FILE *fileOut; //initalize fileOutPointer

    if ((fileOut = fopen("person.csv", "w")) != NULL) {
        puts("File has been opened correctly.\n");
    } else { puts("File not opened correctly. Exiting."); exit(0); }

    PersonNodePtr startPtr = NULL; //initalize starting pointer for LinkedList

    if (initalizeFile(&startPtr, fileOut)) { //check if file is initalized okay
        puts("File has been intialized.\n");
    } else { puts("File not initialized. Exiting."); exit(0); }

    int input = printMenu();
    int searchInput = -1;

    while (input != 0) {
        switch (input) {

            case 1: //load function
                system("clear");
                load(startPtr);
                break; // end of load case

            case 2: //search function
                system("clear");
                searchInput = printSearchMenu();

                while (searchInput != 0) {//search inner switch loop start
                    if (searchInput > 0 && searchInput < 5)
                        search(startPtr, searchInput);
                    else
                        puts("Invalid input");

                    searchInput = printSearchMenu();
                }//end of search loop
                system("clear");
            break; //end of search case

            case 3: //remove function
                system("clear");
                destroy(startPtr, fileOut);
                break; //end of remove case

            default:
                system("clear");
                puts("Invalid input");
        }
```

```
        }
        input = printMenu();
    }
    system("clear");
    puts("Exited.");
    return 0;
}
```

A quick summary: it creates the file, begins the linked list, calls the initialize function, presents the user with the main menu, and reads the user's input to call the function needed to get the task completed.

Conclusion

This project was more difficult than it seemed at first glance. Doing this project was excellent for solidifying knowledge of LinkedList and file manipulation in C. These concepts are very important in the programming world, and they should be treated as such in the classroom. They have many applications that go beyond their basic uses as data structures. Knowing how to manipulate a LinkedList and a file are some of the most basic operations in programming, and C is the best language for learning this. For this reason, this project is one of the most invaluable that we have done during this semester.

References

- https://www.codesdope.com/blog/article/deletion-of-a-give-node-from-a-linked-list-in-c/ (for destroy function)
- https://www.geeksforgeeks.org/c-program-count-number-lines-file/ (for the beginning of file deletion)

Team Member List

- Sean Poston
- Logan Emel

Source Code

```c
/*
Authors: Sean Poston & Logan Emel
Purpose: Create a program that manipulates LinkedLists and files.
Date: 4/28/2020
*/
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
//Fun Time Fun Zone Below

struct personNode {
   int ID;
   char lastName[20];
   char firstName[20];
   int age;
   struct personNode *nextPtr;
}; //initialize struct

FILE *fPtr;

//Define PersonNode Pointer Type
typedef struct personNode PersonNode;
typedef PersonNode *PersonNodePtr;
//struct personNode personNode;

//Function Prototypes
bool initalizeFile(PersonNodePtr *sPtr, FILE *fileOut);
void load(PersonNodePtr currentPtr);
void search(PersonNodePtr currentPtr, int searchInput);
void destroy(PersonNodePtr currentPtr, FILE *fileOut); //remove and delete
are both keywords in C coolcoolcool
int getNumOfNodes(PersonNodePtr currentPtr);
int printMenu(void);
int printSearchMenu(void);
```

```c
void reallocateID(PersonNodePtr currentPtr);

int main(void) {
   system("clear");
   FILE *fileOut; //initalize fileOutPointer

   if ((fileOut = fopen("person.csv", "w")) != NULL) {
       puts("File has been opened correctly.\n");
   } else { puts("File not opened correctly. Exiting."); exit(0); }
    PersonNodePtr startPtr = NULL; //initalize starting pointer for
LinkedList



   if (initalizeFile(&startPtr, fileOut)) { //check if file is initalized
okay
       puts("File has been intialized.\n");
   } else { puts("File not initialized. Exiting."); exit(0); }

   int input = printMenu();
   int searchInput = -1;

   while (input != 0) {
       switch (input) {

           case 1: //load function
               system("clear");
               load(startPtr);
               break; //end of load case

           case 2: //search function
               system("clear");
               searchInput = printSearchMenu();

               while (searchInput != 0) {//search inner switch loop start
                   if (searchInput > 0 && searchInput < 5)
                       search(startPtr, searchInput);
                   else
```

```
                    puts("Invalid input");

                searchInput = printSearchMenu();
            }//end of search loop
            system("clear");
        break; //end of search case

        case 3: //remove function
            system("clear");
            destroy(startPtr, fileOut);
            break; //end of remove case

        default:
            system("clear");
            puts("Invalid input");
    }
    input = printMenu();
  }
  system("clear");
  puts("Exited.");
  return 0;
}

bool initalizeFile(PersonNodePtr *sPtr, FILE *fPtr) { //inputs to file and
initialized LinkedList
  //input function
  //arrays for automatic input (makes things less cluttered)
  char inlast[20][20] = {"Smith", "Smith", "Smith", "Smith", "Smith",
"Smith", "Smith", "Smith", "Smith", "Brown", "Brown", "Brown", "Brown",
"Brown", "Brown", "Brown", "Brown", "Brown", "Brown", "Brown"};
  char infirst[20][20] = {"Liam", "Noah", "William", "James", "Oliver",
"Benjamin", "Elijah", "Lucas", "Mason", "Logan", "Emma", "Olivia", "Ava",
"Isabella", "Sophia", "Charlotte", "Mia", "Amelia", "Harper", "Evelyn"};
  int inage[20] = {25, 26, 27, 60, 35, 45, 10, 35, 20, 19, 29, 39, 49,
59, 18, 17, 12, 10, 8, 15};

  //actual writing function
```

```
   for (int i = 0; i < 20; i++) {
       PersonNodePtr newPtr;
       PersonNodePtr previousPtr;
       PersonNodePtr currentPtr;
       newPtr = malloc(sizeof(PersonNode));

       if(newPtr != NULL) {
           newPtr->ID = (i + 1);
           strcpy(newPtr->firstName, &infirst[i - 1][20]);
           strcpy(newPtr->lastName, &inlast[i - 1][20]);
           newPtr->age = inage[i];
           newPtr->nextPtr = NULL;

           if (i < 19)
               fprintf(fPtr, "%d, %s, %s, %d\n", newPtr->ID,
newPtr->firstName, newPtr->lastName, newPtr->age);
           else
               fprintf(fPtr, "%d, %s, %s, %d", newPtr->ID,
newPtr->firstName, newPtr->lastName, newPtr->age); //don't print newline
on last line

           previousPtr = NULL;
           currentPtr = *sPtr;

           if (previousPtr == NULL) {
               newPtr->nextPtr = *sPtr;
               *sPtr = newPtr;
           }
           else {
               previousPtr->nextPtr = newPtr;
               newPtr->nextPtr = currentPtr;
           }
       }
       else {
           puts("LinkedList already initialized");
           return false;
```

```
        }
    }
        return true;
}

void load(PersonNodePtr currentPtr) { //print out the linkedlist of the
file and show average age
    int ageSum = 0;
    int maxID = 0; //used to calculate data entries
    int nodes = getNumOfNodes(currentPtr);


    printf("%-4s %-12s %-12s %-s\n", "ID", "First Name", "Last Name",
"Age");
    while(currentPtr != NULL) {
    //set list item as active account to print
        if (currentPtr->ID > maxID)
            maxID = currentPtr->ID;
        ageSum += currentPtr->age; //add to age sum
        printf("%-4d %-12s %-12s %-d\n", currentPtr->ID,
currentPtr->firstName, currentPtr->lastName, currentPtr->age);
        currentPtr = currentPtr->nextPtr;
    }

    printf("\nThe average age is: %.2f\n\n", (float)ageSum / (float)nodes);
}

void search(PersonNodePtr currentPtr, int searchInput) { //find specific
firstname in list
    int num;
    char name[20];

    switch (searchInput) {
        case 1: //search by ID

            printf("%s: ", "Enter the ID you'd like to search");
            scanf("%d", &num);
```

```
            system("clear");
            while(currentPtr != NULL) {
                if (currentPtr->ID != num)
                    currentPtr = currentPtr->nextPtr;
                else {
                    printf("%-4d %-12s %-12s %-d\n\n", currentPtr->ID,
currentPtr->firstName, currentPtr->lastName, currentPtr->age);
                    currentPtr = currentPtr->nextPtr;
                }
            }
            break; //end of search by ID case

        case 2: //search by first name
            printf("%s: ", "Enter the first name you'd like to search");
            scanf("%s", name);
            system("clear");
            while(currentPtr != NULL) {
                if (strcmp(currentPtr->firstName, name) != 0)
                    currentPtr = currentPtr->nextPtr;
                else {
                    printf("%-4d %-12s %-12s %-d\n\n", currentPtr->ID,
currentPtr->firstName, currentPtr->lastName, currentPtr->age);
                    currentPtr = currentPtr->nextPtr;
                }
            }
            break; //end of search by first name case

        case 3: //search by last name
            printf("%s: ", "Enter the last name you'd like to search");
            scanf("%s", name);
            system("clear");
            while(currentPtr != NULL) {
                if (strcmp(currentPtr->lastName, name) != 0)
                    currentPtr = currentPtr->nextPtr;
                else {
                    printf("%-4d %-12s %-12s %-d\n\n", currentPtr->ID,
currentPtr->firstName, currentPtr->lastName, currentPtr->age);
```

```
                    currentPtr = currentPtr->nextPtr;
                }
            }
            break; //end of search by last name case

        case 4: //search by age
            printf("%s: ", "Enter the age you'd like to search");
            scanf("%d", &num);
            system("clear");
            while(currentPtr != NULL) {
                if (currentPtr->age != num)
                    currentPtr = currentPtr->nextPtr;
                else {
                    printf("%-4d %-12s %-12s %-d\n\n", currentPtr->ID,
currentPtr->firstName, currentPtr->lastName, currentPtr->age);
                    currentPtr = currentPtr->nextPtr;
                }
            }
            break; //end of search by age case
    }
}

void destroy(PersonNodePtr currentPtr, FILE *fileOut) { //removes a node
from the LinkedList and a line from the file
https://www.codesdope.com/blog/article/deletion-of-a-give-node-from-a-link
ed-list-in-c/
    PersonNodePtr temp;
    PersonNodePtr head;
    head = currentPtr;

    int nodes = getNumOfNodes(currentPtr); //get rows
    int idNum;

    printf("%s: ", "Enter the ID number to remove");
    scanf("%d", &idNum);
    idNum++;
```

```
   while(currentPtr != NULL) {
       if (currentPtr->ID == idNum) {
           temp = currentPtr->nextPtr;
           currentPtr->nextPtr = temp->nextPtr;
           free(temp);
           break;
       }
       else
           currentPtr = currentPtr->nextPtr;
   }// end of while loop

   reallocateID(head);

   // int row = 0; //block of code to remove line from the file
https://www.geeksforgeeks.org/c-program-count-number-lines-file/
   // char c;
   // for (c = getc(fileOut); c != EOF; c = getc(fileOut)) {
   //  if (c == '\n')
   //      row++;
   //  if (row == idNum - 1) {

   //  }
   // }
}

void reallocateID(PersonNodePtr currentPtr) { //reset the IDs after
removing a node
   for (int i = getNumOfNodes(currentPtr); currentPtr != NULL; i--) {
       currentPtr->ID = i;
       currentPtr = currentPtr->nextPtr;
   }
}

int getNumOfNodes(PersonNodePtr currentPtr) { //used to get number of
getNumOfNodes in the linked list
   int nodes = 0;
```

```
    while(currentPtr != NULL) {
        nodes++;
        currentPtr = currentPtr->nextPtr;
    }
    return nodes;
}

int printMenu(void) {
    int input;
    puts("-----------------------------");
    printf("%s","1 to print.\n2 to search.\n3 to remove.\n0 to exit.\n");
    puts("-----------------------------");
    printf("%s", "\nEnter your choice: ");
    scanf("%d", &input);

    return input;
}

int printSearchMenu(void) {
    int input;
    puts("-----------------------------");
    printf("%s","1 to search by ID.\n2 to search by first name.\n3 to
search by last name.\n4 to search by age.\n0 to return.\n");
    puts("-----------------------------");
    printf("%s", "\nEnter your choice: ");
    scanf("%d", &input);

    return input;
}
```