DREXEL UNIVERSITY
Electrical and Computer Engineering
College of Engineering

ECEC 301
Programming for Engineers

Programming Assignment 4: Employee Database

Professor: James A. Shackleford

Section 062

Yonatan Carver
yac25@drexel.edu

Due Date: December 12, 2017

Yonatan Carver 12.12.2017
ECEC 301 – 062 Project #4

Introduction

The goal of this programming assignment is to create an Employee Database with minimal functionality (adding, deleting, and listing of employees) using the C programming language. The database is considered complete if the following attributes are implemented: (1) add employee entries, (2) delete employee entries, and (3) list employee entries in age ascending order.

Implementation Details

The user is first prompted with a main menu that allows them to complete different actions:

```
          -= MENU =-
[1] Add New Employee
[2] Delete an Employee
[3] List All by Age (Ascending)
[4] Quit
------------------------
Selection:
```

The user will type in a number between 1 and 4 then press <Enter>. Depending on the selection, the following functions are called. Option 1 will prompt the user for information about a new employee to be added to the database, option 2 will prompt the user for the number of an employee to delete from the database, option 3 will list all employees in the database in age ascending order, and option 4 will exit the program.

`int employee_add (struct employee* list)`

If the user selects option 1, "`[1] Add New Employee`" from the main menu, they will be prompted to input the employee's first name (string), last name (string), age (integer), and wage (integer). (Lines 9 – 20)

```
Selection: 1
First Name: <Yoni>
Last Name: <Carver>
Age: <20>
Wage: <35>
```

* Note: The items in brackets (< >) will be the user inputted information

As the user inputs each item of information, it is automatically added to a structure (struct) called "new". This "new" struct represents a new employee entry. Since the base employee struct only has three attributes, name, age, and wage, we must concatenate the first name and last name that was inputted. To do this, we use the `strcat` and `strcopy` functions (from `string.h`). Line 13 shows that the first name is concatenated with the last name (along with a space in the middle). The new full name is then copied to the "new" struct's age member.

In order to add this new entry to the general list of employee entries, we must check to find an empty slot (lines 22 – 24). If the for loop encounters an empty slot of memory, the "new" sctruct is entered and sorted (line 25).

After the user inputs the new employee's information and it is added to as a new entry, the main menu is displayed and the user must specify another action.

```c
int employee_add (struct employee* list) {
        // add employee to record

        char firstname[128], lastname[128];
        struct employee new;

        unsigned int i;

        printf("First Name: ");             // prompt user
        scanf("%s", firstname);             // get user input
        printf("Last Name: ");              // prompt user
        scanf("%s", lastname);              // get user input
        strcat(strcat(firstname, " "), lastname);   // combine first and last name (with space)
        strcpy(new.name, firstname);        // store name as "name" member of struct

        printf("Age: ");            // prompt user
        scanf("%u", &new.age);      // get user input

        printf("Wage: ");           // prompt user
        scanf("%u", &new.wage);     // get user input

        for (i = 0; i < NUM_ELEMENTS; i++)
                if (list[i].age == 0) {     // if there is no entry
                        list[i] = new;      // make the that entry the new entry
                        employee_sort(list);  // sort the list in age ascending order

                        return 0;
                }
        return 1;
}
```

int employee_delete (struct employee* list)

If the user selects option 2, "`[2] Delete an Employee`" from the main menu, they will be prompted to enter the number of an employee to be deleted.

```
Selection: 2
Enter ID # to delete: <0>
** Employee #0 Deleted **
```

* Note: The items in brackets (< >) will be the user inputted information

The process for deleting an employee entry is relatively straightforward. Depending on the user-inputted entry, if it is a valid entry, the for loop on lines 8 – 19 loop over all of the memory in the selected employee structure and set all of the contained values to "0" or NULL (line 11). The program then notifies the user that their employee has been deleted (line 13). If the user inputted number does not correspond to an employee in the database, the program will yell at you and bring up the main menu.

```c
1   int employee_delete (struct employee* list) {
2           unsigned int sel_id, id;
3           unsigned int i;
4
5           printf("Enter ID # to delete: ");
6           scanf("%u", &sel_id);
7
8           for (i = 0, id = 0; i < NUM_ELEMENTS; i++) {
9                   if (list[i].age) {
10                          if (id == sel_id) {
11                                  memset(&list[i], 0, sizeof *list);  // set mem of employee to 0
```

```
                employee_sort(list);                // sort employees
                printf("** Employee #%u Deleted **\n", sel_id);

                return 0;
            }
            id++;
        }
    }
    printf("** Invalid Selection **\n");
    // return to menu
    return 1;

}
```

void employee_sort (struct employee* list)

One of the important requirements of this program is to store and display all employee data in age ascending order. While this function is not explicitly called by the user, it is used “under-the-hood” to constantly make sure that the employee database is sorted correctly. This specific sorting algorithm is referred to as “Bubble Sort”. Simply, bubble sorting is the process by which the algorithm steps through each member in a list and compares adjacent items (line 7). If these two items are not in the correct order (in this case, age ascending order), it swaps them. The algorithm continues this process until the entire database is in the correct order (lines 8 – 10).

```
1   void employee_sort (struct employee* list) {
2           unsigned int i, j;
3           struct employee tmp;
4
5           for (i = 0; i < (NUM_ELEMENTS - 1); i++)
6                   for (j = 0; j < (NUM_ELEMENTS - 1); j++)
7                           if (list[j].age > list[j+1].age) {
8                                   tmp = list[j];
9                                   list[j] = list[j+1];
10                                  list[j+1] = tmp;
11                          }
12  }
```

void employee_print (struct employee* e)
void employee_print_all (struct employee* list)

If the user selects option 3, “`[3] List All by Age (Ascending)`” from the main menu, a list of all employees in the database will be displayed. The employee number is the number the user may input to delete an employee from the list.

```
Selection: 3
--------------
Employee #0
Name: Morty Smith
Age: 14
Wage: 137
--------------
Employee #1
Name: Yoni Carver
Age: 20
Wage: 35
--------------
```

```
Employee #2
Name: James Shackleford
Age: 400
Wage: 120
--------------
```

There are two functions involved in printing each employee's information: employee_print and employee_print_all. employee_print is a simple function that accepts an employee structure and prints each member (name, age, and wage). Likewise, employee_print_all iterates over multiple employee structures and on each structure calls the employee_print function to print the employee name, age, and wage.

```
void employee_print (struct employee* e)
{
    printf ("Name: %s\n", e->name);
    printf (" Age: %u\n", e->age);
    printf ("Wage: %u\n", e->wage);
}


void employee_print_all (struct employee* list) {
        // print all employee info
        unsigned int i, id;

        printf("--------------\n");
        for (i = 0, id = 0; i < NUM_ELEMENTS; i++)
                if (list[i].age) {
                        printf("Employee #%u\n", id);
                        employee_print(list + i);    // print data for each employee
                        printf("--------------\n");
                        id++;                        // next person in list
                }
}
```

Yonatan Carver 12.12.2017
ECEC 301 – 062 Project #4

Testing Results

Adding Employees

```
       -= MENU =-
[1] Add New Employee
[2] Delete an Employee
[3] List All by Age (Ascending)
[4] Quit
------------------------
Selection: 1
First Name: Yoni
Last Name: Carver
Age: 20
Wage: 35
       -= MENU =-
[1] Add New Employee
[2] Delete an Employee
[3] List All by Age (Ascending)
[4] Quit
------------------------
Selection: 1
First Name: Morty
Last Name: Smith
Age: 14
Wage: 137
       -= MENU =-
[1] Add New Employee
[2] Delete an Employee
[3] List All by Age (Ascending)
[4] Quit
------------------------
Selection: 1
First Name: James
Last Name: Shackleford
Age: 400
Wage: 120
```

Listing Employees

```
       -= MENU =-
[1] Add New Employee
[2] Delete an Employee
[3] List All by Age (Ascending)
[4] Quit
------------------------
Selection: 3
--------------
Employee #0
Name: Morty Smith
 Age: 14
Wage: 137
--------------
Employee #1
Name: Yoni Carver
 Age: 20
Wage: 35
--------------
Employee #2
Name: James Shackleford
 Age: 400
Wage: 120
--------------
```

Deleting Employees

```
       -= MENU =-
[1] Add New Employee
[2] Delete an Employee
[3] List All by Age (Ascending)
[4] Quit
------------------------
Selection: 2
Enter ID # to delete: 0
** Employee #0 Deleted **
       -= MENU =-
[1] Add New Employee
[2] Delete an Employee
[3] List All by Age (Ascending)
[4] Quit
------------------------
Selection: 2
Enter ID # to delete: 2
** Invalid Selection **
```

Exiting the Program

```
       -= MENU =-
[1] Add New Employee
[2] Delete an Employee
[3] List All by Age (Ascending)
[4] Quit
------------------------
Selection: 4
yonicarver@YAC-LENOVO:~$
```