ECE-C301 Programming for Engineers
Instructor: James A. Shackleford
Programming Assignment 4

1 The Big Idea

In this assignment you will be writing a simple program that manages employees. Your program will be able to perform the following simple actions:

- **Add an Employee Record** – This action adds an employee record. Each employee record consists of the employee's name, age, and hourly wage.
- **Delete an Employee Record** – This action deletes an existing employee record. Attempting to delete an invalid or non-existent record must return an error message without crashing the program.
- **List All Employees in order of Ascending Age** – This action lists all of the employee records from youngest to oldest.
- **Quit** – This action quits the program.

2 The Employee Record

An employee record is simply a `struct employee`, which has the following form:

```
struct employee {
    char name[128];
    unsigned int age;
    unsigned int wage;
};
```

This structure can be found in the provided header file `employee.h`.

Your program will be primarily concerned with maintaining an array of `struct employee` entries. This can be seen in the primary program logic provided in `menu.c`. Because we will be using an array of `struct employee` entries, the number of employees we can store will be limited to the number of elements in the array. Consequently, we will be unable to *actually* delete employee entries–the entire array of employee entires must be allocated throughout the duration of the program's execution. We will learn how to overcome this limitation in the future. However, for now, we must agree on a method to invalidate an employee entry–such invalidated entries will be said to be "deleted". For this assignment, we will mark an employee entry as invalidated (i.e. "deleted") if its first byte is `NULL` (i.e. `'\0'`). As we will discuss later, we will keep our array sorted such that all valid entires are first in the array with invalid entries following. In other words, much like a C string, our `struct employee` array will be `NULL` terminated.

3 Getting Ready: Basic Program Organization

The program structure itself is provided to you in `menu.c`. As you can see, it is a fairly simple program that fits entirely on a single 8.5×11" page:

```c
#include <stdio.h>
#include <string.h>
#include "employee.h"

#define MAX 100
#define OPTION_ADD  1
#define OPTION_DEL  2
#define OPTION_LIST 3
#define OPTION_QUIT 4

unsigned int print_menu ()
{
    unsigned int action;

    printf("       -= MENU =-\n");
    printf("[1] Add New Employee\n");
    printf("[2] Delete an Employee\n");
    printf("[3] List All by Age (Acending)\n");
    printf("[4] Quit\n");
    printf("------------------------\n");
    printf("Selection: ");

    scanf("%u", &action);

    return action;
}

int main (unsigned int argc, char** argv)
{
    struct employee list[MAX];

    unsigned int running = 1;

    /* Set all bits in the employee array to zero */
    memset (list, 0, MAX*sizeof(struct employee));

    while (running) {
        switch (print_menu()) {
            case OPTION_ADD:
                employee_add(list);
                break;
            case OPTION_DEL:
                employee_delete(list);
                break;
            case OPTION_LIST:
                employee_print_all(list);
                break;
            case OPTION_QUIT:
                running = 0;
                break;
        };
    }
    return 0;
}
```

As you can see, the program can be so sort because it depends heavily on external code and data structures declared in the header `employee.h`. As you may have guessed, the implementations for these functions reside in the provided `employee.c`, which contains empty functions. It is the goal of this assignment for you to fill in these empty functions. When you are successful, you must be able to compile and run the program as follows:

```
$ gcc -o project4 menu.c employee.c
$ ./project4
```

4 Adding Employees

The process of adding employees is fairly straight forward. If the user chooses to add a new employee, they must be prompted (in this order) for the following information: First Name, Last Name, Age, and Wage. After this new entry is added to your array of `struct employee` entires, you must sort your array in order of ascending age (we will talk about this in the next section. Here is what your program must look like when adding employees:

```
       -= MENU =-
[1] Add New Employee
[2] Delete an Employee
[3] List All by Age (Acending)
[4] Quit
------------------------
Selection: 1
First Name: Robert
Last Name: Baratheon
Age: 54
Wage: 999
       -= MENU =-
[1] Add New Employee
[2] Delete an Employee
[3] List All by Age (Acending)
[4] Quit
------------------------
Selection: 1
First Name: Donald
Last Name: Trump
Age: 70
Wage: 50
       -= MENU =-
[1] Add New Employee
[2] Delete an Employee
[3] List All by Age (Acending)
[4] Quit
------------------------
Selection: 1
First Name: Mickey
Last Name: Mouse
Age: 88
Wage: 12
```

Note: Since you are reading the First and Last names of the employee separately, you will need to concatenate them in order to store the full name as a single string in the provided `struct employee` data structure.

5 Sorting

You will need to keep your array of `struct employee` entires sorted at all times. This means that entires must be ordered in ascending age (youngest first, oldest last) with all invalidated entries following the oldest employee. Invalidated entires do not need to be stored in any particular order amongst themselves, but they must all be stored after the sorted valid entries.

To help you in this task, here is an example implementation of a very simple sorting algorithm called "Bubble Sort" being used to sort a simple array of integers:

```c
#include <stdio.h>

int main (int argc, char** argv)
{
    int i, j, n;
    int array[100];
    int tmp;

    printf("Enter number of elements\n");
    scanf("%d", &n);

    printf("Enter %d integers\n", n);
    for (i=0; i<n; i++)
        scanf("%d", &array[i]);

    for (i=0; i<(n-1); i++)
        for (j=0; j<n-i-1; j++)
            if (array[j] > array[j+1]) {
                tmp = array[j];
                array[j]   = array[j+1];
                array[j+1] = tmp;
            }

    printf("Sorted list:\n");
    for (i=0; i<n; i++)
        printf("%d\n", array[i]);

    return 0;
}
```

You must sort your array of `struct employee` entires after any addition or deletion operation. This will require that you modify the provided sorting code in order to work the more complex `struct employee` data structure instead of just simple integers.

6 Listing All Employees by Ascending Age

If the user chooses to list all employees, they must be displayed in order of ascending age. For example, given the employees we added earlier, your program must provide output in the following format:

```
       -= MENU =-
[1] Add New Employee
[2] Delete an Employee
[3] List All by Age (Acending)
[4] Quit
------------------------
Selection: 3
--------------
Employee #0
Name: Robert Baratheon
 Age: 54
Wage: 999
--------------
Employee #1
Name: Donald Trump
 Age: 70
Wage: 50
--------------
Employee #2
Name: Mickey Mouse
 Age: 88
Wage: 12
--------------
Num Employees: 3
```

7 Deleting an Employee

The process of deleting an employee is fairly tricky! Be very careful and thoughtful when you go about doing this! Given the interactions we have had with the program thus far, deletion must work as follows:

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

       -= MENU =-
[1] Add New Employee
[2] Delete an Employee
[3] List All by Age (Acending)
[4] Quit
------------------------
Selection: 3
--------------
Employee #0
Name: Robert Baratheon
 Age: 54
Wage: 999
--------------
Employee #1
Name: Mickey Mouse
 Age: 88
Wage: 12
--------------
Num Employees: 2
```

Note that your program must be smart enough to not attempt the deletion of a non-existent employee. For example, when trying to delete a non-existent employee 56, your program must provide the following output:

```
      -= MENU =-
[1] Add New Employee
[2] Delete an Employee
[3] List All by Age (Acending)
[4] Quit
------------------------
Selection: 2
Enter ID # to delete: 56
** Invalid Selection**

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

8 Deliverables

Your deliverables for this project are a working program that *you* wrote – no copying from your friends or lab mates, please! Also, please submit a short report detailing the project, your implementation, and your testing results including screen shots (similar to what I have provided for you here).

Submit your report in PDF format.

Submit your complete source code as a zip file.