Structures

Workshop 5 (out of 10 marks - 6% of your final grade)

In this workshop, you will code a C-language program with an object of structure type.

LEARNING OUTCOMES

Upon successful completion of this workshop, you will have demonstrated the abilities:

- to store data of different types using a structure type
- to declare an object of structure type
- to access the members of an object of structure type
- to describe to your instructor what you have learned in completing this workshop

SUBMISSION POLICY

The "in-lab" section is to be completed during your assigned lab section. It is to be completed and submitted by the end of the workshop period. If you attend the lab period and cannot complete the in-lab portion of the workshop during that period, ask your instructor for permission to complete the in-lab portion after the period. If you do not attend the workshop, you can submit the “in-lab” section along with your “at-home” section (with a penalty; see below). The “at-home” portion of the lab is due on the day that is two days before your next scheduled workshop (23:59).

All your work (all the files you create or modify) must contain your name, Seneca email and student number.

You are responsible to back up your work regularly.

Late submission penalties:

- In-lab portion submitted late, with at-home portion: 0 for in-lab. Maximum of 70/70 for at-home and reflection
- If any of in-lab, at-home or reflection portions is missing the mark will be zero.

IN-LAB: (30%)

Download or clone workshop 5 (**WS05**) from https://github.com/Seneca-144100/IPC-Workshops

Write your code for this segment in the **emp_inlab.c** file provided with the Visual Studio template project inside the **in-lab** folder.

In this workshop segment, you will implement, Add and Display employee data functionality using C structs and arrays.

OVERVIEW

The **emp_inlab.c** **template file** has the following already implemented instructions:

- Display a menu list as shown in the following inside a do-while loop construct:
  1. Display Employee Information
  2. Add Employee
  0. Exit
- Capture user input for the above options. Store to an variable named “option”
- Ability to iterate multiple menu choices (until 0 is entered) with required selection construct to direct process flow to the required logic/functionality (using switch). Refer to the comments for each case to identify the functionality required.
- Display an error message for invalid menu option selections in the case
- Initial output information for menu options 1 and 2 is provided with the relevant formatting.

You are required to complete the following.

- Define the number of employees **SIZE** to be 2 using the # directive and inserting it between the library directive and the main function definition (you will increase this value later)
- Declare a struct to represent employee data, which has the following information
  - An identification number stored in an
  - Age stored in an
  - Salary stored in a

- Define an array of Employee 's named **emp** that can hold **SIZE** elements. Initialize all of the elements and their member variables to be 0. (Hint: You can assign **{0}** to **emp** at its definition.)

IMPLEMENT EXIT PROGRAM FUNCTIONALITY IN CASE 0

Print the exiting message.

```
> Exiting Employee Data Program. Good Bye!!! <
```

IMPLEMENT DISPLAY FUNCTIONALITY IN CASE 1

- Display the elements of the **emp** array. Only display the **valid** employee data ("*Employee ID, Employee Age* and *Employee Salary*"). Do not display any other data. **Hint:** If the identification number is positive, then it is **valid** employee data. Use the following formatting in a statement:

  `%6d%9d%11.2lf` ←→ (*Employee ID, Employee Age* and *Employee Salary*)

- After completing the display, enter a newline using a statement.

IMPLEMENT ADD EMPLOYEE FUNCTIONALITY IN CASE 2

- Keep track of the number of valid employees using an variable.
- Check if the **emp** array is full.
- If the array is full, display the following error message.

  ```
  > ERROR!!! Maximum Number of Employees Reached <
  ```

- If the array is not full, accept new employee data ("*Employee ID, Employee Age* and *Employee Salary*") as per the program output listed below and store that data in an empty element of the **emp** array. Increment the number of valid employees as required.

PROGRAM COMPLETION

Your program is complete if your output matches the following output. Red numbers show the user's input.

G T O

D

```
               D   111
                    34
               N       78980.88

                    D

                                      2

               D   112
                    41
               N       65000

                    D

                                      2

 MM M            I                   M

                    D

                                      1

    D        B      N G MT

                    D

                                      0

                               B
```

IN_LAB SUBMISSION:

To test and demonstrate execution of your program use the same data as the output example above or any information needed.

If not on matrix already, upload your **emp_inlab.c** to your matrix account. Compile and run your code and make sure everything works properly.

Then run the following script from your account: (replace profname.proflastname with your professors Seneca userid)

and follow the instructions.

Please Note

- A successful submission does not guarantee full credit for this workshop.
- If the professor is not satisfied with your implementation, your professor may ask you to resubmit. Resubmissions will attract a penalty.

AT_HOME: (30%)

Copy all of the in-lab code into the "**emp_athome.c**" file, which you will find in the **at-home** folder. Implement the following cases to **emp_athome.c** file.

- Change the value of **SIZE** to 4.
- Expand the menu list (printing) to include options 3 & 4 after option 2 with the following
  - Update Employee Salary
  - Remove Employee
- Create two switch-cases for option 3 & 4 after case 2. Do not forget to include statements at the end of each case.

IMPLEMENT UPDATE EMPLOYEE DATA FUNCTIONALITY IN CASE 3

- Display the following initially

```
> Update Employee Salary <
> ====================== <
```

- Prompt the user for the employee identification number using a loop. While the number is not found in the employee array, keep prompting the user.
- Once the number is found, display the current salary for the employee with that identification number and prompt the user to input the new salary. Replace the old salary with the input value.

IMPLEMENT REMOVE EMPLOYEE FUNCTIONALITY IN CASE 4

- Display the following initially

```
> Remove Employee <
> =============== <
```

- Copy and paste from case 3 the   loop that prompts the user for the employee identification number.
- Once the number is found, display the following message and change the employee identification number to indicate an empty slot.

```
> Employee [REPLACE WITH EMP ID] will be removed <
```

- Decrement the valid employee count by one

PROGRAM COMPLETION

Your program is complete, if your output matches the following output. Red numbers show the user's input.

```
          G T      O

                    D

   P               N
   M


                                      5

 MM M  D                 O

                    D

   P               N
   M


                                      2



               D    111
                    34
               N       78980.88

                    D

   P               N
   M


                                      2



               D    112
```

41
N 65000
D
P N
M
2

D 113
53
N 120345.78
D
P N
M
2

D 114
25
N 46780
D
P N
M
2

MM M I M
D
P N
M
1

D B N G MT

D

P N
M

3

P D

D 112
O
I N 99999.99

D

P N
M

1

D B N G MT

D

P N
M

4

M

D 112

D

P N
M

1

D B N G MT

D

P N
M

0

B

AT-HOME REFLECTION (40%)

Please provide brief answers to the following questions in a text file named **reflect.txt.**

1) In two or three sentences discuss the advantages of using an array of structs versus parallel arrays when dealing with related data
2) The preferred method of declaring a struct data type is in a header file. Briefly explain why. Hint: this is explained in the notes.

AT_HOME SUBMISSION:

To test and demonstrate execution of your program use the same data as the output example above.

If not on matrix already, upload your **emp_athome.c** **and** **reflect.txt** to your matrix account. Compile and run your code and make sure everything works properly.

Then run the following script from your account: (replace profname.proflastname with your professors Seneca userid)

and follow the instructions.

Please Note

- A successful submission does not guarantee full credit for this workshop.
- If the professor is not satisfied with your implementation, your professor may ask you to resubmit. Resubmissions will attract a penalty.