CS1160
LAB 11: STRUCTURES

I. What is a Structure?

Structures are collections of related variables. In contrast, to arrays that contain only elements of the same data type. Structures may contain variables of many different data types.

General Expression:

```
struct   structName {
  data_type   var1;
  ...
};
```

1. To use a structure and store in it values a **structure variable(s) should be declared** from that structure; either, inside the main, inside a user defined function or outside the main as global variable:
   - ► To declare a single variable of the structure

     ```
     struct structName structVarName;
     ```

   - ► To declare a single variable of the structure that has inside it another variable of structure:

     ```
     struct structName structVarName.var1;
     ```

   - ► To declare an array of the structure:

     ```
     struct structName structArray[ size ];
     ```

How an array of structures would looks in the memory:

structArray [ ]				
0	1	2	3	size - 1
structArray [0]. var1	structArray [1]. var1	structArray [2]. var1	structArray [3]. var1	

2. **Accessing the variables**: from main function or user defined functions:
   - ► Single variable of the structure:

     ```
     structName.varName = value;  // the dot is called the member access operator
     ```

   - ► Single variable of the structure that has inside it another variable of structure

     ```
     structArray[index].varName = value;
     ```


Sara Abu Sa'aleek

II. Example

1. Write a C program with the following specifications:
   a. Declare a structure called **EngineeringStudent** that has the following:

      1. Integer variable called studentID to store the id of the student.

      2. String variable (array of characters) of size 30 called studentName to store the student name.

      3. float variable called mathGrade that stores the math grade of the student.

      4. float variable called englishGrade that stores the english grade of the student.

      5. float variable called CSGrade that stores the CS grade of the student.

      6. float variable called studentGPA that stores the gpa of the student.

```c
/* a. Declare the structure */
struct EngineeringStudent {
    int studentID;
    char studentName[30];
    float mathGrade;
    float englishGrade;
    float CSGrade;
    float studentGPA;
};
```

b. In main, declare two variables called **student1** and **student2** of type struct EngineeringStudent and read all of their information (except studentGPA) from the user.

```
struct EngineeringStudent student1, student2;
```

```
/* Read information for student 1 */
printf("Enter information for Student 1\n");

printf("Student ID: ");
scanf("%d", &student1.studentID);
getchar();   /* clear newline from buffer */

printf("Student Name: ");
gets(student1.studentName);

printf("Math Grade: ");
scanf("%f", &student1.mathGrade);

printf("English Grade: ");
scanf("%f", &student1.englishGrade);

printf("CS Grade: ");
scanf("%f", &student1.CSGrade);
```

```
/* Read information for student 2 */
printf("\nEnter information for Student 2\n"

printf("Student ID: ");
scanf("%d", &student2.studentID);
getchar();   /* clear newline from buffer */

printf("Student Name: ");
gets(student2.studentName);

printf("Math Grade: ");
scanf("%f", &student2.mathGrade);

printf("English Grade: ");
scanf("%f", &student2.englishGrade);

printf("CS Grade: ");
scanf("%f", &student2.CSGrade);
```

c. Calculate and print the gpa of each student.

```
/* Calculate GPA */
student1.studentGPA = (student1.mathGrade + student1.englishGrade + student1.CSGrade) / 3;
student2.studentGPA = (student2.mathGrade + student2.englishGrade + student2.CSGrade) / 3;
```

```
printf("\nStudent 1 GPA: %.2f\n", student1.studentGPA);
printf("Student 2 GPA: %.2f\n", student2.studentGPA);
```

d. Create a function called MinGrade that takes a EngineeringStudent struct as an argument and returns the minimum grade among mathGrade, englishGrade, and CSGrade.

```c
/* Function to calculate minimum grade of a student */
float MinGrade(struct EngineeringStudent student) {
    float min = student.mathGrade;
    if (student.englishGrade < min)
        min = student.englishGrade;
    if (student.CSGrade < min)
        min = student.CSGrade;
    return min;
}
```

Before The Main

```c
/* Find and print minimum grade of student1 */
float minGrade1 = MinGrade(student1);
printf("\nMinimum grade of Student 1: %.2f\n", minGrade1);
```

In The Main

e. Create a function called printInfo that takes a EngineeringStudent struct as an argument and prints all the student's information. Use puts() to print the student name.

f. Using printInfo, find and print all information of the student who has the maximum GPA.

```c
/* Function to print student information */
void printInfo(struct EngineeringStudent student) {
    printf("Student ID: %d\n", student.studentID);
    printf("Student Name: ");
    puts(student.studentName);
    printf("Math Grade: %.2f\n", student.mathGrade);
    printf("English Grade: %.2f\n", student.englishGrade);
    printf("CS Grade: %.2f\n", student.CSGrade);
    printf("GPA: %.2f\n", student.studentGPA);
}
```

```c
/* Print student with maximum GPA using printIn
printf("\nStudent with Maximum GPA:\n");
if (student1.studentGPA > student2.studentGPA)
    printInfo(student1);
else
    printInfo(student2);
```

2. Write a C program that does the following (for practice, not for the exam)

   a. Declare a structure called Car with the following members: Price (float), Year (int), Color (String) and Brand (String).

```c
/* a. Declare the structure */
struct Car {
    float price;
    int year;
    char color[20];
    char brand[20];
};
```

   b. In the main function, ask the user to enter the information of 10 cars and save them in an array called CARS.

```c
int main() {
    int SIZE = 10;
    struct Car CARS[SIZE];  // Use SIZE macro for array
    int i;

    /* b. Read information for SIZE cars */
    for (i = 0; i < SIZE; i++) {
        printf("\nEnter information for car %d\n", i + 1);

        printf("Brand: ");
        getchar();          // clear newline from previous input
        gets(CARS[i].brand);

        printf("Color: ");
        gets(CARS[i].color);

        printf("Year: ");
        scanf("%d", &CARS[i].year);

        printf("Price: ");
        scanf("%f", &CARS[i].price);
    }
```

c. Print the color of the car(s) that has (have) a price greater than 15000.

```
/* c. Print colors of cars with price > 15000 */
printf("\nColors of cars with price greater than 15000:\n");
for (i = 0; i < SIZE; i++) {
    if (CARS[i].price > 15000) {
        puts(CARS[i].color);
    }
}
```

d. Write a function called oldestCar that takes the array CARS as an argument and returns the oldest car among all cars.

```
/* d. Function to find the oldest car */
struct Car oldestCar(struct Car cars[]) {
    int i;
    struct Car oldest = cars[0];

    for (i = 1; i < SIZE; i++) {
        if (cars[i].year < oldest.year) {
            oldest = cars[i];
        }
    }
    return oldest;
}
```

e. In the main, call the function oldestCar and print all information of the oldest car.

```
/* e. Call function and print oldest car information */
struct Car oldest = oldestCar(CARS);

printf("\nOldest Car Information:\n");

printf("Brand: ");
puts(oldest.brand);

printf("Color: ");
puts(oldest.color);

printf("Year: %d\n", oldest.year);
printf("Price: %.2f\n", oldest.price);
```

Task

Product Inventory Management with Functions

1. Declare a structure called Product with the following members:

- productID (int) — Product ID
- productName (char array of size 30) — Name of the product
- price (float) — Price of the product
- quantity (int) — Number of items in stock
- totalValue (float) — Total value of the product (price * quantity)

2. In main, declare two variables: product1 and product2 of type Product. Read all information except totalValue from the user. Use gets() for the product names.

3. Create the following functions:

- **float calculateTotalValue(Product p) —** calculates and returns the total value of a product (price * quantity).
- **void printProductInfo(Product p)** — prints all product information, using puts() for the product name.
- **Product productWithHigherValue(Product p1, Product p2**) — returns the product with the higher total value.
- **Product productWithLowerPrice(Product p1, Product p2)** — returns the product with the lower price.

4. In main, use these functions to:

- Calculate and print the total value of each product.
- Print all information of the product with the higher total value.
- Print the name and price of the product with the lower price.