To:
Tenouk

C LAB WORKSHEET 14
C Structures, struct Part 1

1. Understanding, creating and using C structure data type.
2. Tutorial reference that should be used together with this worksheet is: C/C++ type specifiers – struct, union, typedef.

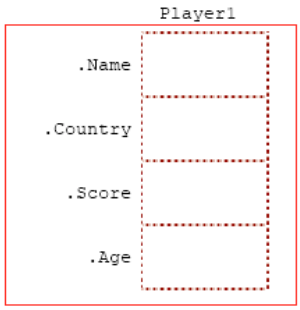
In an Array worksheet we had four game score from four different players, which we read and printed in reverse order. Instead of defining Score1, Score2, Score3 and Score4, we defined one array that held all four scores, that is Score[4]. We grouped them in an array because all items were of the **same data type** and had a similar meaning, they were all game scores.
However what about the group of data items that are related but have **different data types**?
For example, suppose that along with the game scores, we want to store the name of each player, the country from which they come and their ages. Here, the score is a float, the player and the country are character strings and the age is an integer. We would like to store these four items together as one unit and to handle and process them together as a group. In order to do this we can use **structure data type**. The keyword use to define this data type is **struct**. For our needs, we can define the structure as shown below.

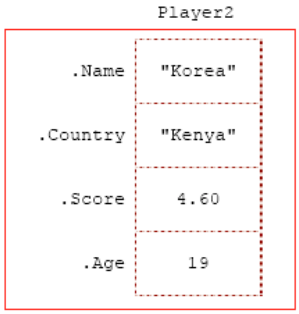
```
struct Participant
{
    char Name[20];
    char Country[25];
    float Score;
    int Age;
};
```

struct is a keyword and the Participant is a name or tag that we provided. All that we are doing is defining a structure called struct Participant. We are not creating a variable instead we are creating a **new data type** that includes or aggregate other C and C++ data types. You can think of this as a structure template from which structure variables may be defined. Any variable declared as being of this type will have these four parts that are called **members of the structure**.
Later in the **C++** worksheet of this series we will add more features to structure templates and call them **classes**. **Variables** defined as classes will be called **objects**. Just as variables are created from data types or structure variables are created from structure definitions, **objects are created from classes**.
With arrays, we need to specify the subscript or the index to access one item inside the array.
To access one item in a structure, we need to specify the **name of the member**. The order in which the structure members are defined is not important. The following code shows how structure variables are defined and initialized.

```
struct Participant Player1, Player2 = {"Korea", "Kenya", 4.6, 19};
```

The following Figures try to show the defining two structure variables.





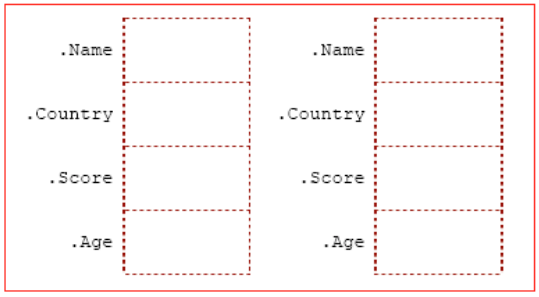
Player1 is defined and has four members. Player2 is also defined and its four members are initialized. Here, the order is important. They should be initialized in the same order that the structure members are defined as shown in the above Figure. In database terminology, a member of a structure variable is called a **field**, a structure variable with its data is called a **database record** and a collection of related records is called a **file**.
With arrays, we specify the subscript or an index enclosed in brackets to access one element.
With structures, we must follow the variable name by a **period**, followed by the member name to access one member. For example, Player2.Score is equal to 4.60. The following code:

```
strcpy(Player1.Country, Player2.Country);
```

will copy the country (value) of Player2 as the country of Player1. Additionally, while Player2.Country is equal to the string “Kenya”, Player2.Country[3] is equal to the character ‘y’. We can always write an array of players like the following.

```
struct Participant Player[2];
```

This will create an array called Player[ ] with only 2 slots, where each slot is a structure of type struct Participant. See the following Figure that try to depict an array of structure.



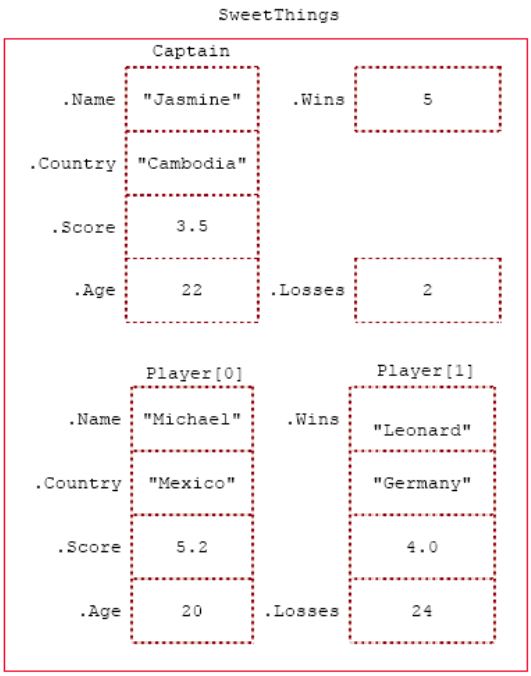
To read in the name for the player in **slot 0**, the statement will look like this:

```
// scanf("%s, &Player[0].Name);
scanf_s("%s, &Player[0].Name, 20);
```

The & is optional here because as you have learned previously, array names are actual addresses anyway. We can also create a structure based on other structures. Consider the the following code example.

```
struct Team       // defining a new structure
{
    struct Participant Captain;        // struct Team contains a structure
    int Wins;
    int Losses;
    struct Participant Player[2];      // struct Team has an array of structures
};

// defining and initializing a variable
struct Team SweetThings =
{
    {"Jasmine", "Cambodia", 3.5, 22},
    5,
    2,
    {{"Michael", "Mexico", 5.2, 20},
    {"Leonard", "Germany", 4.0, 24}}    // need the outer braces
};

// just defining a variable
struct Team RowdyBabies;
```

In the following Figure, you can see what the variable SweetThings looks like, a structure of structures.



If we want to copy the contents of SweetThings to RowdyBabies, all we need to do is:

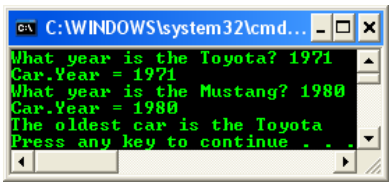
```
RowdyBabies = SweetThings;
```

However, for the sake of obtaining experience in handling structures, let us do the same thing the long way.

```
// copy the Captain values
strcpy(RowdyBabies.Captain.Name,   SweetThings.Captain.Name);
strcpy(RowdyBabies.Captain.Country,   SweetThings.Captain.Country);

RowdyBabies.Captain.Age       = SweetThings.Captain.Age;
RowdyBabies.Captain.Score     = SweetThings.Captain.Score;

// copy the Player values
for(i = 0; i <= 1; ++i)
{
    strcpy(RowdyBabies.Player[i].Name,  SweetThings.Player[i].Name);
    strcpy(RowdyBabies.Player[i]. Country,  SweetThings.Player[i].Country);

    RowdyBabies.Player[i].Age     = SweetThings.Player[i].Age;
    RowdyBabies.Player[i].Score = SweetThings.Player[i].Score;
}

// Copy the Wins and Losses
RowdyBabies.Wins       = SweetThings.Wins;
RowdyBabies.Losses     = SweetThings.Losses;
```

You can see that, although the first solution is only one statement, the expanded solution illustrates how to handle individual members of structures.
Now let us take a simple structure and use it to illustrate how structures are used with functions. We will do this by writing the entire program. In the following program example, we define struct Mobile. It is simply made up of two members: one is called Make[ ] and the other is Year. The structure is defined outside any function, so it is available globally to all functions.
main() defines Car1 and Car2 of type struct Mobile. Then main() calls the function FindYear() by passing one character string with a value of “Ford”. Execution then goes to that function.
In FindYear(), the character string is called Name[ ]. A local variable of type struct Mobile is defined with the name of Car. FindYear() asks for the year of the car given by Name[ ] and reads in the year member of Car. The make member of Car is assigned the string in Name[ ]. Then FindYear () returns the name and the year of the car as type struct Mobile.
This structure is assigned to Car1 in main(). Similarly, the string “Mustang” is passed to FindYear (), which returns that string and its year back to main() as a struct Mobile. This new structure is assigned to Car2.
Now main() calls the function PrintOldest(). Instead of passing a string, main() passes two structures. Also, instead of assigning the function to a variable like Car1 and Car2, main() doesn’t assign PrintOldest() to anything. Hence, there is no return in PrintOldest() and its return data type is void.
PrintOldest() makes a copy of these two structures into Auto1 and Auto2. If this function had changed the contents of these structures, the changes would not be reflected in Car1 and Car2 in main(). This is like scalars and unlike arrays. The function proceeds to find the oldest of the two cars and prints its make. Note that there is no return statement here.

```
#include <stdio.h>
#include <string.h>

// Global definition
struct Mobile
{
    char Make[20];
    int Year;
};

// function prototype with structure return type
struct Mobile FindYear(char Name[ ]);
// function prototype with structures arguments passed
void PrintOldest(struct Mobile Auto1, struct Mobile Auto2);

void main(void)
{
    struct Mobile Car1, Car2;

    Car1 = FindYear("Toyota");
    Car2 = FindYear("Mustang");
    PrintOldest(Car1, Car2);
}

// this function receives one string that is the make of a car,
// reads in its year and returns them both as "struct Mobile"
struct Mobile FindYear(char Name[ ])
{
    struct Mobile Car;
    printf("What year is the %s? ", Name);
    // scanf("%d", &Car.Year);
    scanf_s("%d", &Car.Year, 1);
    printf("Car.Year = %d\n", Car.Year);
    // strcpy(Car.Make, Name);
    strcpy_s(Car.Make, sizeof(Car.Make), Name);
    return Car;
}

// this function receives two structures of type "struct Mobile"
// and prints the year of the oldest car. It returns nothing.
void PrintOldest(struct Mobile Auto1, struct Mobile Auto2)
{
    if(Auto1.Year < Auto2.Year)
        printf("The oldest car is the %s\n", Auto1.Make);
    else if(Auto1.Year > Auto2.Year)
        printf("The oldest car is the %s\n", Auto2.Make);
    else
        printf("They are both the same age!\n");
```



}

struct Data Type Practices

1. Let us introduce two new functions intended for strings. You may enter “You’re making me blue” and then enter “All my loving”. Show the output and answer the questions.

```
#include <stdio.h>

void main()
{
  char a[25], b[25];

  puts("Name your favorite song title: ");
  // gets(a);
  gets_s(a, 25);
  puts("\nName another song title:");
  // scanf("%s", &b);
  scanf_s("%s", &b, 25);
  printf("\na = %s, b = %s\n", a, b);
}
```

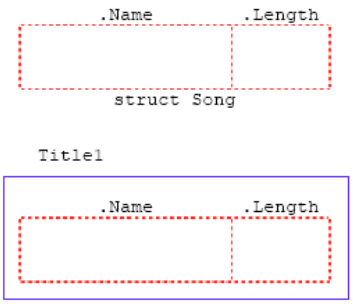
```
Name your favorite song title:
Wind of Change

Name another song title:
Seek and Destroy

a = Wind of Change, b = Seek
Press any key to continue . . .
```

   a. Was the scanf_s() able to read the entire string or only the first word?
   b. Was the gets_s() able to read the entire string or only the first word?
   c. Did the gets_s() stop reading at the first space or the first return that was entered?
   d. Did the puts() add a ‘\n’ at the end of the string that is printed?
   e. Which function would you prefer to read in a string, scanf_s() or gets_s ()?

a. Only the first word.
b. The entire string.
c. The first return that was entered.
d. Yes it did.
e. Of course gets()/gets_s().

2. Firstly, enter “Ooowee Babe” and then 2.40 for the following program. Show the output and answer the questions.

```
#include <stdio.h>

void main()
{
  // definition of structure
  struct Song
  {
      char Name[25];
      float Length;
  };
  // end of definition

  // variable declaration
  struct Song Title1;

  printf("The size of Title1 structure variable is %d bytes.\n", sizeof(Title1));
  puts("Name your favorite song title: ");
  gets_s(Title1.Name, 25);
  puts("How long is it?");
  scanf_s("%f", &Title1.Length);
  printf("\nYour song is ");
  puts(Title1.Name);
  printf("And it is %.2f min. long.\n", Title1.Length);
}
```

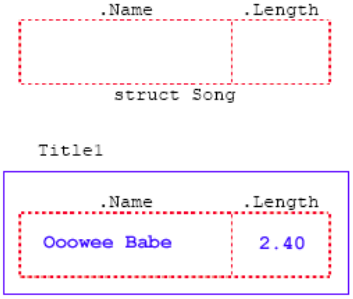



```
---------------------------------------------------------------
The size of Title1 structure variable is 32 bytes.
Name your favorite song title:
Ooowee Babe
How long is it?
2.40

Your song is Ooowee Babe
And it is 2.40 min. long.
Press any key to continue . . .
```

Just as the int and float are data types, struct Song is a new data type that we have defined.

   a. What are the two known types used to define struct Song?
   b. struct Song is defined with two members. What are their names?
   c. Do the words char or float by themselves create new variables or allocate space in memory?
   d. Since struct Song is also a new data type, do you think that it creates in itself a new variable?
   e. Is a semicolon used at the end of the structure definition?
   f. Since the definition of the structure doesn’t create a new variable, what is the name of the variable declared using the struct Song data type?
   g. Title1 is a new variable that takes up space in memory. How many parts does it have? What are their names?
   h. Show the contents of each member inside the box Title1.
   i. An array is **a collection of many items of the same data type**, such as int or char. Similarly, a structure data type is a collection of many items. Do they have to be of the same data type?

a. char and float.
b. Name and Length.
c. They allocate space in memory.
d. Yes it creates in itself a new variable.
e. Yes.
f. Title1.
g. It has two parts named name and length.



h. ...

j. When accessing a slot in an array, a set of brackets is used, such as a[2] = 0;. When we want to access a member of a structure, what do we use?
k. How would you have assigned the Length member of Title1 to 0?
l. How would you have assigned the name member of Title1 to “Mr. Moonlighting”?

i. No. structure can have different data types.
j. We use a dot operator and the structure's member name.
k. Title1.Length = 0;
l. Title1.Name = “Mr. Moonlighting”.

3. In the following program example enter “Riders on the Storm” and 3.10 for the sample input data. Show the output and answer the questions.

```
#include <stdio.h>
#include <string.h>

void main()
{
  struct Song
  {
      char Name[25];
      float Length;
  };

  struct Song Title1, Title2;

  strcpy_s(Title2.Name, 25, "My Teardrops");
  Title2.Length = 2.35f;
  puts("Name your favorite song:");
  gets_s(Title1.Name, 25);
  puts("How long is it?");
  scanf_s("%f", &Title1.Length);
  printf("\nMy song is %s\n Your song is ", Title2.Name);
  puts(Title1.Name);
  printf("Yours is %.2f min. longer \n", Title1.Length - Title2.Length);
}
```

a. Can you print out both members of Title2 without specifying the member names as shown below? Does this work?

   printf("%s %.2f\n", Title2);

b. Can you assign Title2 to Title1 without specifying their members as shown below? Does this work?

   Title1 = Title2;

c. Does the following code work? Why or why not?

   Title1.Name = Title2.Name

```
Name your favorite song:
Riders on the Storm
How long is it?
3.10

My song is My Teardrops
 Your song is Riders on the Storm
Yours is 0.75 min. longer
Press any key to continue . . .
```

a. No, we can't and this doesn't work. We need the name of the member.
b. Yes we can and it does work provided that they both have same structure as in this example. Here we assign the whole structure.
c. It doesn't work. We cannot assign the value of Title2.Name directly to Title1.Name but we can copy the value as the following code:

   strcpy_s(Title1.Name, 25, Title2.Name);