

# Struct

☰ Tags	c
🕒 Created time	@October 10, 2024 11:54 AM
# Lecture No.	9
☑ Reviewed	<input type="checkbox"/>

## Introduction

Suppose you may want to write a program to handle football players data. Each player has some information such as player name, date of birth, position, goals scored, etc. It is possible to setup arrays for each type but limits you. You would need all individual arrays to be synchronised. C provides a better way to handle this with the concepts of `structs`.

## Declare a Struct

The struct keyword enables you to define a collection of variables called a structure that you can treat as a single unit. You can define it as follows:

```
struct <struct_name>
{
    <type> <field_name>;
    <type> <field_name>;
    ...
};

// Football player example:
struct Player
```

```
{
    char name[30];
    int goal_scored;
};
```

- This example declares a structure type called `Player`.
- Note that this isn't a variable name; it's a new type.
- This type name is referred to as a *structure tag* or a *tag name*.
- The naming of the structure tag follows the same rules as for a variable name, which you should be familiar with by now.
- The variable names within the `Player` structure, `name` and `goal_scored`, are called members or fields.

You can then declare a variable using this structure as follows:

```
struct Player player1;
```

It is also possible to create an instance of a structure at the same time when it is declared as follows:

```
struct Player
{
    char name[30];
    int goal_scored;
} player1;
```

- The following is to initialise the value for each field of an instance:

```
struct Player
{
    char name[30];
    int goal_scored;
    char position[2];
}
```

```
float price;  
} player1 = {"Cristiano Ronaldo", 500, "ST", 100};
```

or you can declare the struct `Player` first and create an instance later as follows:

```
struct Player player1 = {"Cristiano Ronaldo", 500, "ST", 100};
```

Note that, in case you don't want to keep re-using the keyword `struct` everytime, you can use `typedef` as follows:

```
typedef struct Player Player;
```

- This defines `Player` to be the equivalent of `struct Player`.

Then you can define a variable of type `Player` like this:

```
Player player1;
```

## Accessing Structure Methods

```
Player player2;  
  
player2.goal_scored = 500;  
player2.price = 100;
```

- The `.` between the structure variable name and the member name is called the member election operator.
- Structure members are the same as variables of the same type. You can set their values and use them in expressions in the same way as ordinary variables.

You can also create and initialise values for an instance of a struct in a more organised way as follows.

```
Player player3 = {  
    .name = "Kante", .position = "CM", .goal_scored = 10, .price  
};
```

## Example

**?** Write a program to manage a list of football players. The program accepts a player detail input from keyboard and prints it out.

```
#include<stdio.h>  
#include<stdlib.h>  
  
typedef struct Player Player; //Define Player as a type name  
  
struct Player //Structure type definition  
{  
    //Define struct members  
    char name[30];  
    int goal_scored;  
    char position[2];  
    float price;  
};  
  
int main(int argc, char*argv[])  
{  
  
    Player player1; //Declare a variable type Player. This vari
```

```

//Input a player
printf("Player name:\n");
scanf("%s", player1.name); //Get an input from keyboard, store it in player1.name

printf("Preferred position?\n");
scanf("%s", player1.position);

printf("Market price:\n");
scanf("%f", &player1.price); //We need & in this case because we are passing the address of player1.price

printf("Goal scored:\n");
scanf("%d", &player1.goal_scored);

//Print a player by accessing to each members of the struct
printf("-----\n");
printf("You have input the following player:\n");
printf("Name: %s\n", player1.name);
printf("Position: %s\n", player1.position);
printf("Goal scored: %d\n", player1.goal_scored);
printf("Market price: %.2f\n", player1.price);

return 0;
}

```

## Arrays of Structures

We want to write a program that can accept multiple inputs and print all. To deal with such requirements, we can use arrays of structures.

```

Player players[50];

```

```

#include<stdio.h>
#include<stdlib.h>

typedef struct Player Player; //Define Player as a type name

struct Player //Structure type definition
{
    char name[30];
    int goal_scored;
    char position[5];
    float price;
};

int main(int argc, char*argv[])
{
    Player player[50]; //Declare a variable type Player. This va

    //Input a player
    int pcount = 0; //to count the player
    int selection; //This is to store the decision of the user :
    do
    {
        printf("Player name:\n");
        scanf("%s", player[pcount].name); //Get an input from ke

        printf("Preferred position:\n");
        scanf("%s", player[pcount].position);

        printf("Market price:\n");
        scanf("%f", &player[pcount].price); //We need & in this

        printf("Goal scored:\n");
        scanf("%d", &player[pcount].goal_scored);
    }
}

```

```

        ++pcount; //Increase the player count variable by 1;

        printf("Do you want to add more player (1 = Y, 0 = N)?\n");
        scanf("%d", &selection);

        if(selection == 0)
            break;

    } while (pcount < 50);

    //Print all players:
    printf("-----\n");
    printf("You have input the following players:\n");

    for(int i = 0; i < pcount; ++i)
    {
        printf("Player %d:\n", i+1); //Just a signal
        printf("Name: %s\n", player[i].name);
        printf("Position: %s\n", player[i].position);
        printf("Goal scored: %d\n", player[i].goal_scored);
        printf("Market price: %.2f\n", player[i].price);
        printf("\n");
    }

    return 0;
}

```

In this program the do, while loop will keep asking the user for input until there are 50 player structures in the array.

What does fgets do?