WEEK5 - STRUCTURES

30 October 2024 17:48

The struct keyword enables you to define a collection of variable of various type called a structure that you can treat as a single unit. A general way to declare a struct id as follow:

Struct name
{
Type filed name;
Type filed name;
…
}

Ex.
Struct player
{
Char name[30];
Int goal_scored;
}
declared a structure called player, not a variable but a type
Variable name within player are all called members or fields.

Possible to declare variable with type player
Struct player player1;
Player1 is instance of structure

```c
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:

```c
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:

```c
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:

```c
Player player1;
```

A member of a structure can be referred by writing the variable name followed by a period, followed by the member of the variable name.

Player player2;

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

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 = "60"
};
```

YOU CAN ACESS STRUCTURE MEMBERS ALSO WITH A ->

printf("%s", cardPtr->suit); // displays Hearts

SELF REFERENTIAL STRUCTURES

A struct type may not contain a variable of it own struct type. But it may contain a pointer to that struct type.

```
struct employee {
   char firstName[20];
   char lastName[20];
   int age;
   double hourlySalary;
};
```

```
struct employee {
   char firstName[20];
   char lastName[20];
   unsigned int age;
   double hourlySalary;
   struct employee *managerPtr; // pointer
};
```

A structure containing a member that's a pointer to the same struct type is a selfreferential structure.

OPERATION THAT CAN BE PERFORMED ON STRUCTURES:

You can perform the following operations on structs:

- assigning one struct variable to another of the *same* type (Section 10.7)—for a pointer member, this copies only the address stored in the pointer,
- taking the address (&) of a struct variable (Section 10.4),
- accessing a struct variable's members (Section 10.4),
- using the sizeof operator to determine a struct variable's size, and
- zero initializing a struct variable in its definition, as in

```
struct card myCard = {};
```

) Assigning a structure of one type to one of a different type is a compilation error.

Comparing structure objects is not allowed

struct card myCard = {"Three", "Hearts"};
If there are fewer initializers than members, the remaining members are automatically initialized to 0 or NULL (for pointer members).

STRUCTURES AS MEMBERS OF A STRUCTURE

In some case, a member of a structure can also be a structure. For example, in the structure Player, the name can be defined as another structure which contains first_name and last_name. Similarly, one may one to add date_of_birth which contains specific detail for day, month and year. In such cases, we can define name or date_of_birth as a struct. There are two ways to deal with this:

- We can define two more structure for name and date_of_birth then declare the corresponding members in the structure Player:

```
typedef struct Player Player;
typedef struct Date Date;
typedef struct Name Name;
typedef struct DOB DOB;

struct DOB
{
        int day;
        int month;
        int year;
};

struct Name
{
        char firstName[30];
        char lastName[30];
};
```

```
struct Player //Structure type definition
{
 Name name;   //the member name is an instance of the structure Name
 int goal_scored;
 char position[5];
 float price;
 DOB date_of_birth;  //the member date_of_birth is an instance of the structure DOB
};
```

```
Player player1; //Declare a new instance of Player

strcpy(player1.name.firstName, "Cristiano");
strcpy(player1.name.lastName, "Ronaldo");
player1.goal_scored = 500;
player1.price = 100;
player1.dob.day = 5;
player1.dob.month = 2;
player1.dob.year = 1985;
```

NOTE writing this way `player1.name.firstName = "Cristiano"` in this case is not acceptable in C. However. you can write this way if the `firstName` is declared as follows: `char *firstName;` .

You could define the Date and Name structure within the Player structure definition.
Similar to variable you can pass structures as arguments to a function and you can also return a structure from a function.

```
typedef struct Player Player;
typedef struct Date Date;
typedef struct Name Name;
typedef struct DOB DOB;

struct DOB
{
	int day;
	int month;
	int year;
};

struct Name
{
	char firstName[30];
	char lastName[30];
};


struct Player //Structure type definition
{
 Name name;   //the member name is an instance of the structure Name
 int goal_scored;
 char position[5];
 float price;
 DOB date_of_birth;  //the member date_of_birth is an instance of the structure DOB
};

/*function prototype*/
int countGoal(Player p1, Player p2);

int main()
{
	Player p1, p2;
	int goals = 0;

	/* fill data for p1 */
	strcpy(p1.name.firstName, "Cristiano");
	strcpy(p1.name.lastName, "Ronaldo");
```

```
    p1.goal_scored = 500;

    /*fill data for p2 */
    strcpy(p2.name.firstName, "Lionel");
    strcpy(p2.name.lastName, "Messi");
    p2.goal_scored = 500;

    /* call the function to calculate the total of goals */
    goals = countGoal(p1, p2);

    return 0;
}

int countGoal(Player p1, Player p2)
{
    return p1.goal_scored + p2.goal_scored;
}
```

- The above program use the pass-by-value mechanism. The two instance of the struct Player - p1 and p2 - are copied and passed as arguments for the function countGoal. The function then does the calculation and return the result.

PASSING STRUCTURE VS ARRAY

Unlike arrays, structures are passed by value—a copy of the entire structure is passed.
This requires the execution-time overhead of making a copy of each data item in the structure and storing it on the computer's function call stack.
Passing large objects such as structures by using pointers to constant data obtains the performance of pass-by-reference and the security of pass-by-value.
In this case, the program copies only the address at which the structure is stored—typically four or eight bytes.
If memory is low and execution efficiency is a concern, use pointers.
If memory is in abundance and efficiency is not a major concern, pass data by value to enforce the principle of least privilege.
Some systems do not enforce const well, so pass-byvalue is still the best way to prevent data from being modified.

The function will access the original structure directly through the pointer. More often than not, structures are passed to a function using a pointer, just for these reasons of efficiency.

```
int countGoalWithPointers(Player *p1, Player *p2)
{
    return p1->goal_scored + p2->goal_scored;
}


int main()
{
    Player p1, p2;
    int goals = 0;

    /* fill data for p1 */
    strcpy(p1.name.firstName, "Cristiano");
    strcpy(p1.name.lastName, "Ronaldo");
    p1.goal_scored = 500;

    /*fill data for p2 */
    strcpy(p2.name.firstName, "Lionel");
    strcpy(p2.name.lastName, "Messi");
    p2.goal_scored = 500;

    /* call the function to calculate the total of goals */
    goals = countGoalWithPointers(&p1, &p2);

    return 0;
}
```

A structure can be seen as a type which is similar to basic types e.g. int and float. Therefore a function can return a struct variable.

```
Player addPlayer()
{
    Player p;
    printf("First name:\n");
```

```
    scanf("%s", p.name.firstName);

    printf("Last name:\n");
    scanf("%s", p.name.lastName);

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

    printf("Preferred position:\n");
    scanf("%s", p.position);

    printf("Market price:\n");
    scanf("%f", &p.price);

    printf("Date of birth:\n");
    scanf("%d/%d/%d",&p.dob.day,&p.dob.month,&p.dob.year);

    return p;
}
```