Structures

Akash Phukan

Roll No: 05, MCA 1st SEM

07-11-2024

Contents

What is a Structure?

- The structure in C is a user-defined data type that can be used to group heterogeneous items (of possibly different types) into a single type.
- The struct keyword is used to define the structure in the C programming language.
- The items in the structure are called its member and they can be of any valid data type.(Such as, int, char, float etc)
- C allows Nesting of structures. They can be done in two ways:
  - By separate nested structure. (creating instances)
  - By embedded nested structure. (structure in other structure)

Why do we need structure?

- Structure is a user-defined data in C language will allows us to combine data of different types together.
- Structure helps to construct a complex data type which is more meaningful.
- C structures can be used to store huge data.

Syntax
Structure Declaration

- We have to declare structure in C before using it in our program.
- In structure declaration, we specify its member variables along with their datatype.
- We can use the struct keyword to declare the structure in C using the following syntax:

Syntax:

```
struct structureName {
dataType memberName1;
dataType memberName1;
....
};
```

Syntax

Structure Definition

- To use structure in our program, we have to define its instance. We can do that by creating variables of the structure type.
- We can define structure variables using two methods.

Syntax

Structure Definition

1. Structure Variable Declaration with Structure Template

```
struct StructureName {
dataType memberName1;
dataType memberName1;
....
....
}variable1, varaible2, ...;
```

2. Structure Variable Declaration after Structure Template

```
// structure declared beforehand
struct structureName variable1, variable2, .......;
```

Syntax

Access Structure Members

- We can access structure members by using the (.) dot operator.

Syntax

structureName.member1;
strcutureName.member2;

Real World Example

- For this example, let us consider a Garage.
- In the garage we want to store 3 cars(say).

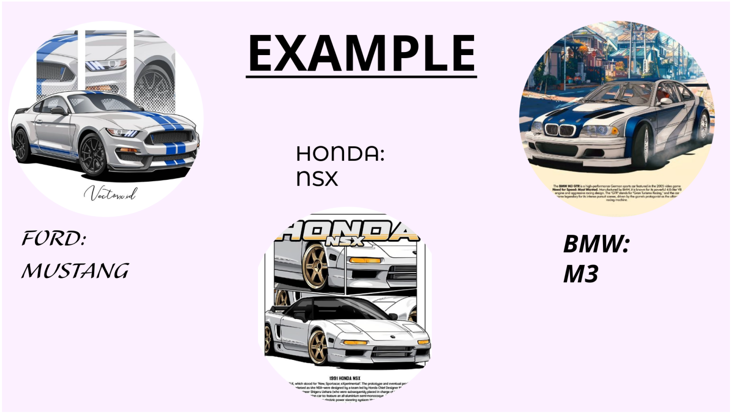


Figure: Different cars that we want to store in the garage

Each car will be an entity in the garage. And each car will have different characteristics.
To better understand this, take a look at the following figures:

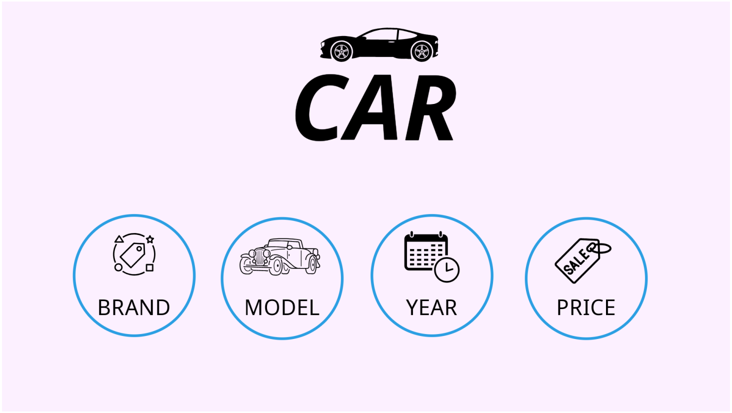


Figure: A car and its characteristics

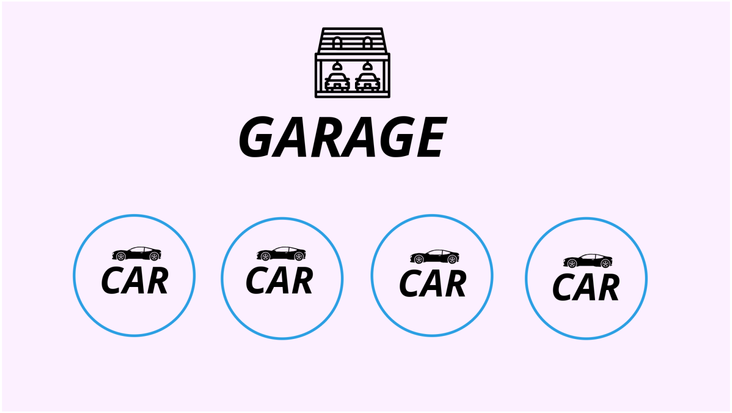
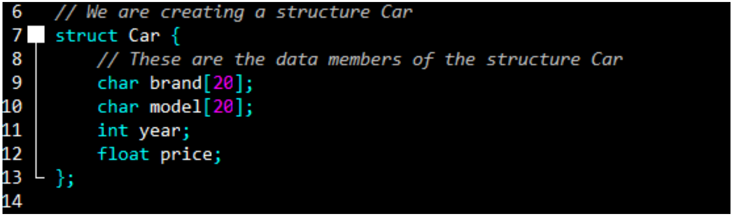


Figure: A Garage containing Cars

Example in code

- Let us see how the above real life example would look like when translated it into code.

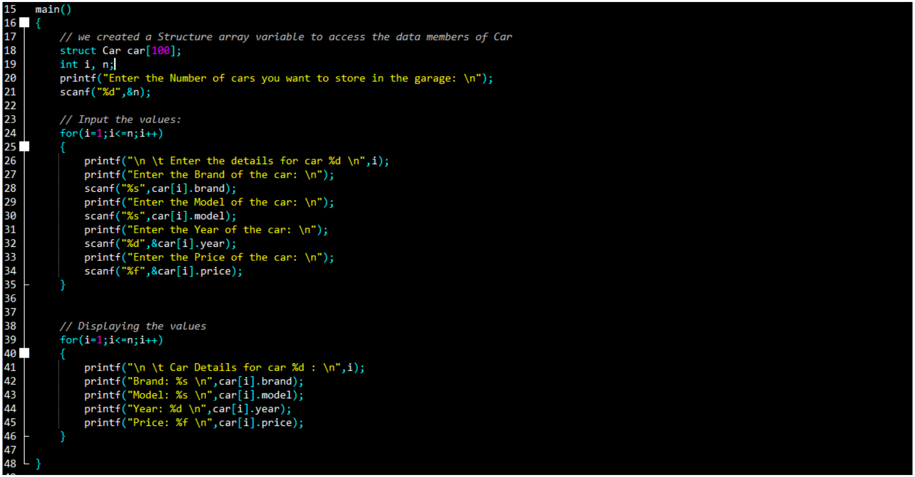
Here we are creating a Structure Car, which will have heterogeneous data members.
In other words Structure Declaration.



```
// We are creating a structure Car
struct Car {
    // These are the data members of the structure Car
    char brand[20];
    char model[20];
    int year;
    float price;
};
```

Figure: Structure for a Car

In the main function, we will be creating an array variable for our use-defined data structure Car. Through the use of this array variable we can access different members of the structure.
In computing terms, its called Structure Definition and Access Structure Members.

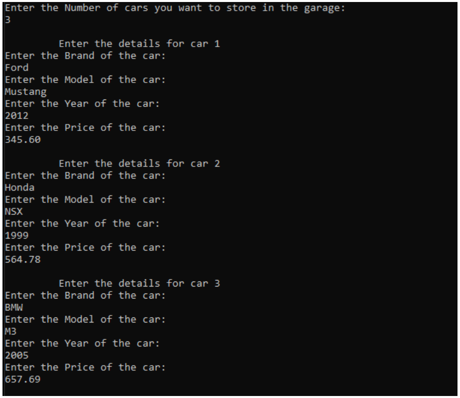


```c
main()
{
    // we created a Structure array variable to access the data members of Car
    struct Car car[100];
    int i, n;
    printf("Enter the Number of cars you want to store in the garage: \n");
    scanf("%d",&n);

    // Input the values:
    for(i=1;i<=n;i++)
    {
        printf("\n \t Enter the details for car %d \n",i);
        printf("Enter the Brand of the car: \n");
        scanf("%s",car[i].brand);
        printf("Enter the Model of the car: \n");
        scanf("%s",car[i].model);
        printf("Enter the Year of the car: \n");
        scanf("%d",&car[i].year);
        printf("Enter the Price of the car: \n");
        scanf("%f",&car[i].price);
    }


    // Displaying the values
    for(i=1;i<=n;i++)
    {
        printf("\n \t Car Details for car %d : \n",i);
        printf("Brand: %s \n",car[i].brand);
        printf("Model: %s \n",car[i].model);
        printf("Year: %d \n",car[i].year);
        printf("Price: %f \n",car[i].price);
    }

}
```

Figure: Main Function

Then we compile and run our program...

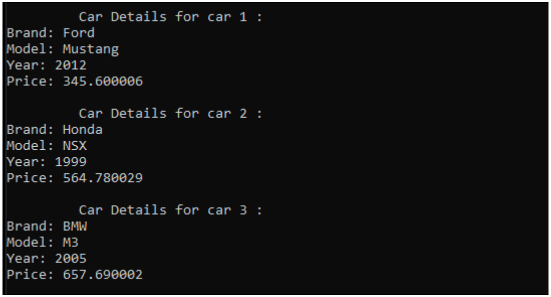
Output:
When we run the code we will get this output.
Then we enter our data.



```
Enter the Number of cars you want to store in the garage:
3

         Enter the details for car 1
Enter the Brand of the car:
Ford
Enter the Model of the car:
Mustang
Enter the Year of the car:
2012
Enter the Price of the car:
345.60

         Enter the details for car 2
Enter the Brand of the car:
Honda
Enter the Model of the car:
NSX
Enter the Year of the car:
1999
Enter the Price of the car:
564.78

         Enter the details for car 3
Enter the Brand of the car:
BMW
Enter the Model of the car:
M3
Enter the Year of the car:
2005
Enter the Price of the car:
657.69
```

Figure: Output

After we have entered our data, we click enter.
And we get this output.



```
         Car Details for car 1 :
Brand: Ford
Model: Mustang
Year: 2012
Price: 345.600006

         Car Details for car 2 :
Brand: Honda
Model: NSX
Year: 1999
Price: 564.780029

         Car Details for car 3 :
Brand: BMW
Model: M3
Year: 2005
Price: 657.690002
```

Figure: Output

- From the above example, we can say that Structures makes it way easier to manage big data of different types.
- The code for the management system was also readable and not messy.

Conclusion

- Structures help organize data faster, which can increase productivity.
- Structures make code easier to read.
- Structures can store data of different types, such as int, float, and char, under one name.

Thank You!