

# CS 240 Programming in C

## Structures

March 21, 2023

# Structures in C

A structure is a collection of one or more variables, possibly of different types, grouped together under a single name for convenient handling. (Structures are called "records" in some languages, notably Pascal.) Structures help to organize complicated data, particularly in large programs, because they permit a group of related variables to be treated as a unit instead of as separate entities. <sup>1</sup>

---

<sup>1</sup>Page no. 127 of textbook

# Structures syntax

Syntax: To define a structure in C, we use the `struct` keyword followed by the structure name and a list of member variables enclosed in braces.

Example:

```
struct car
{
    int year;
    char brand[50];
    char model[50];
};
```

Note: Unlike arrays, structures allow us to store data of different types in a single variable.

# Purpose and benefits

- Structures allow us to group related data items together, making our code more organized and easier to read.
- Simplifies complex data structures: Structures allow us to represent complex data structures in a more natural and intuitive way.
- Enables passing of multiple values: Structures enable us to pass multiple values to functions without having to use global variables or multiple parameters.
- Facilitates modularity: Structures can be used to create modular code that is easier to maintain and debug.
- Enhances code readability: Structures provide a way to logically group related data items, making the code easier to read and understand.

# Structures syntax

We can define structures with any combination of data types and member names.

```
struct Car {  
    char make[50];  
    char model[50];  
    int year;  
};
```

Assign value:

```
struct Car c1 = {"Toyota", "Corolla", 2016};
```

To initialize a structure, we can use a curly brace-enclosed list of values in the order they are defined in the structure.

# Accessing Structure Members

To access the members of a structure, we use the dot (.) operator to refer to the member by name.

```
struct Car c1 = {"Toyota", "Corolla", 2016};
```

```
printf("Brand: %s", c1.make);  
printf("Model: %s", c1.model);  
printf("Year: %d", c1.year);
```

```
c1.year = 2020;
```

We can also modify the values of structure members using the dot operator.

# Defining Structures Within Structures

- Structures can contain members that are themselves structures, creating a nested structure.
- To define a nested structure, we simply include the definition of one structure inside the definition of another.

```
struct address {  
    int building_number;  
    char street[50];  
    char city[50];  
    char pincode[5];  
};
```

```
struct user {  
    char name[50];  
    struct address lives_at;  
};
```

# Array of Structures

- We can define arrays of structures to store multiple values of a structure type.
- To define an array of structures, we include the array size after the structure name in the declaration.

```
struct student {  
    char name[50];  
    int student_id;  
    float marks;  
};
```

```
// array of 100 student structures  
struct student cs240[100];
```



# Array of Structures


To access elements of an array of structures, we use the array index operator `[]` to refer to the element by index, and then use the dot `(.)` operator to access the member by name.

```
struct student cs240[10];  
  
cs240[0].student_id = 101;  
strcpy(cs240[0].name, "John");  
cs240[0].marks = 85.5;
```

Note: We can also use loops to iterate over an array of structures and perform operations on each element.

# Array of Structures

```
struct Car
{
    int year;
    char *make;
    char *model;
};

void printCar(struct Car car) {}

int main()
{
    struct Car cars[] = {
        {2019, "Toyota", "Camry"},
        {2018, "Honda", "Civic"},
        {2017, "Ford", "Fusion"},
        {2016, "Chevy", "Cruze"},
        {2015, "Nissan", "Altima"}};

    for (int i = 0; i < 5; i++)
        printCar(cars[i]);
}
```

# Pointer of Structures

Like primitive types, we can have a pointer to a structure. If we have a pointer to the structure, members are accessed using the arrow ( `->` ) operator.

# Typedef

We use the `typedef` keyword to create an alias name for data types. It is commonly used with structures to simplify the syntax of declaring variables.

A union is a user-defined type similar to structs in C except for one key difference.

Structures allocate enough space to store all their members, whereas unions can only hold one member value at a time.

A bit field is a data structure that consists of one or more adjacent bits which have been allocated for specific purposes, so that any single bit or group of bits within the structure can be set or inspected. A bit field is most commonly used to represent integral types of known, fixed bit-width, such as single-bit Booleans.<sup>2</sup>

---

<sup>2</sup>[https://en.wikipedia.org/wiki/Bit\\_field](https://en.wikipedia.org/wiki/Bit_field)

- Phonebook Program
- Matrix Calculator