Structured Types

- **array** - composite variable, all elements have same type.
- **structure** (or **struct**) - composite variable with components (called members or fields) which can be different types.

Parallel arrays

```c
char month[][4] = {"Feb", "Feb", "Mar"};
int day[] = {17, 25, 4};

printf("%s %d %d", month[1], day[1], year[1]);
```
structure

```c
struct date {
    char month[4];
    int day;
    int year;
};
```

- 3 fields or members:
  - an array of 4 characters named `month`,
  - an int named `day`,
  - an int named `year`
- `date` = structure tag or the name of this type of structure
- `month`, `day`, and `year` are not individual variables, but part of a larger object
- Structure field names can be the same as individual variables or the same as field names in other structures without conflict.

Declaring a variable:
- `struct date myDay;`
- `struct date d1, d2;`

typedef

```c
typedef struct {
    char month[4];
    int day;
    int year;
} Date;
```

- Defines a type named `Date` (and you don’t have to use the keyword “struct” with its tag).

The . (dot) operator

```c
structurevar.fieldname
```

- Structures of the same type can be assigned: `d2= d1;`
- Fields of a structure can be assigned: `d1.day = 21;`
- Structure fields can be used like other variables of that type:
  - `d1.day++;`
  - `diff= 2000 - d2.year;`
- Entire structures CAN NOT be read or printed
- Entire structures CAN NOT be compared
**Structure Initializers**

- values listed in braces
- fields initialized in the order they appear in the structure definition
- if too few values, remaining fields initialized to 0

```c
Date birthday = {"Dec", 25, 1964};
```

**Arrays of Structures**

```c
Date duedates[PROJCOUNT];
strncpy(duedates[0].month, "Feb");  
duedates[0].day= 17;
duedates[0].year= 1998;
strncpy(duedates[1].month, "Feb");  
duedates[1].day= 25;
duedates[1].year= 1998;
... etc.
int i;
for (i = 0; i < PROJCOUNT; i++){
    printf("%s %d %d", duedates[i].month,  
           duedates[i].day, duedates[i].year);
}
```

**Passing Structures, structure references and arrays of structures as parameters**

- Like Arrays
  - they are a composite type
  - you can pass the whole or pass just a part
  - they can be initialized using {} as they first come into existence
  - they can be compared using the == operator (or <, >, !=)
  - you can compare/process individual members one by one

- Unlike Arrays
  - they don't have to have all parts be of the same type
  - you must use the name of each part rather than the index
  - names of structures are not references
  - you must use & to get its address
  - a structure can be the return value of a function
  - they can be assigned using the = (assignment operator)