



Structures

What is a Structure?

- Used for handling a group of logically related data items
 - Examples:
 - Student name, roll number, and marks
 - Real part and complex part of a complex number
- Helps in organizing complex data in a more meaningful way
- The individual structure elements are called **members**

Defining a Structure

```
struct tag {  
    member 1;  
    member 2;  
    :  
    member m;  
};
```

- **struct** is the required C keyword
- **tag** is the name of the structure
- **member 1, member 2, ...** are individual member declarations

Contd.

- The individual members can be ordinary variables, pointers, arrays, or other structures (any data type)
 - The member names within a particular structure must be distinct from one another
 - A member name can be the same as the name of a variable defined outside of the structure
- Once a structure has been defined, the individual structure-type variables can be declared as:

```
struct tag var_1, var_2, ..., var_n;
```


Example

- A structure definition

```
struct student {  
    char name[30];  
    int  roll_number;  
    int  total_marks;  
    char dob[10];  
};
```

- Defining structure variables:

```
struct student a1, a2, a3;
```



A new data-type

A Compact Form

- It is possible to combine the declaration of the structure with that of the structure variables:

```
struct tag {  
    member 1;  
    member 2;  
    :  
    member m;  
} var_1, var_2, ..., var_n;
```

- Declares three variables of type **struct tag**
- In this form, **tag** is optional

Accessing a Structure

- The members of a structure are processed individually, as separate entities
 - Each member is a separate variable
- A structure member can be accessed by writing `variable.member`

where `variable` refers to the name of a structure-type variable, and `member` refers to the name of a member within the structure

- Examples:
`a1.name, a2.name, a1.roll_number, a3.dob`

Example: Complex number addition

```
int main()
{
    struct complex
    {
        float real;
        float cmplex;
    } a, b, c;

    scanf ("%f %f", &a.real, &a.cmplex);
    scanf ("%f %f", &b.real, &b.cmplex);

    c.real = a.real + b.real;
    c.cmplex = a.cmplex + b.cmplex;
    printf ("\n %f + %f j", c.real, c.cmplex);
    return 0;
}
```


Operations on Structure Variables

- Unlike arrays, a structure variable can be directly assigned to another structure variable of the same type

`a1 = a2;`

- All the individual members get assigned
- Two structure variables can not be compared for equality or inequality

`if (a1 == a2).....` ← **this cannot be done**

Arrays of Structures

- Once a structure has been defined, we can declare an array of structures

```
struct student class[50];
```



type name

- The individual members can be accessed as:

```
class[i].name
```

```
class[5].roll_number
```

Arrays within Structures

- A structure member can be an array

```
struct student
{
    char name[30];
    int roll_number;
    int marks[5];
    char dob[10];
} a1, a2, a3;
```

- The array element within the structure can be accessed as:

a1.marks[2], a1.dob[3],...

Structure Initialization

- Structure variables may be initialized following similar rules of an array. The values are provided within the second braces separated by commas
- An example:

```
struct complex a={1.0,2.0}, b={-3.0,4.0};
```



```
a.real=1.0; a.imag=2.0;  
b.real=-3.0; b.imag=4.0;
```

Parameter Passing in a Function

- Structure variables can be passed as parameters like any other variables. Only the values will be copied during function invocation

```
void swap (struct complex a, struct complex b)
{
    struct complex tmp;

    tmp=a;
    a=b;
    b=tmp;
}
```

Returning structures

- It is also possible to return structure values from a function. The return data type of the function should be as same as the data type of the structure itself

```
struct complex add(struct complex a, struct complex b)  
{  
    struct complex tmp;  
  
    tmp.real = a.real + b.real;  
    tmp.imag = a.imag + b.imag;  
    return(tmp);  
}
```

Direct arithmetic operations are not possible with structure variables

Defining data type: using `typedef`

- One may define a structure data-type with a single name

```
typedef struct newtype {  
    member-variable1;  
    member-variable2;  
    .  
    member-variableN;  
} mytype;
```

- `mytype` is the name of the new data-type
 - Also called an **alias** for `struct newtype`
 - Writing the tag name `newtype` is optional, can be skipped
 - Naming follows rules of variable naming

typedef : An example

```
typedef struct {  
    float real;  
    float imag;  
} _COMPLEX;
```

- Defined a new data type named **_COMPLEX**. Now can declare and use variables of this type

```
_COMPLEX a, b, c;
```


- Note: typedef is not restricted to just structures, can define new types from any existing type
- Example:
 - typedef int INTEGER
 - Defines a new type named **INTEGER** from the known type **int**
 - Can now define variables of type **INTEGER** which will have all properties of the **int** type

```
INTEGER a, b, c;
```

The earlier program using typedef

```
typedef struct{
    float real;
    float imag;
} _COMPLEX;
```

```
void swap (_COMPLEX a, _COMPLEX b)
{
    _COMPLEX tmp;

    tmp = a;
    a = b;
    b = tmp;
}
```

Contd.

```
void print (_COMPLEX a)
{
    printf("(%f, %f) \n",a.real,a.imag);
}

int main()
{
    _COMPLEX x={4.0,5.0}, y={10.0,15.0};

    print(x); print(y);
    swap(x,y);
    print(x); print(y); return 0;
}
```



■ Output:

(4.000000, 5.000000)

(10.000000, 15.000000)

(4.000000, 5.000000)

(10.000000, 15.000000)

- x and y are not swapped! But that has got nothing to do with structures specially. We will see its reason shortly



Structures and Functions

- A structure can be passed as argument to a function
- A function can also return a structure

Example: complex number addition

```
int main()
{
    _COMPLEX a, b, c;
    scanf("%f %f", &a.real, &a.imag);
    scanf("%f %f", &b.real, &b.imag);
    c = add (a, b) ;
    printf("\n %f %f", c,real, c.imag);
    return 0;
}
```

```
_COMPLEX add(_COMPLEX x, _COMPLEX
y)
{
    _COMPLEX t;

    t.real = x.real + y.real;
    t.imag = x.imag + y.imag ;
    return (t) ;
}
```

Exercise Problems

1. Extend the complex number program to include functions for addition, subtraction, multiplication, and division
2. Define a structure for representing a point in two-dimensional Cartesian co-ordinate system
 - Write a function to compute the distance between two given points
 - Write a function to compute the middle point of the line segment joining two given points
 - Write a function to compute the area of a triangle, given the co-ordinates of its three vertices