

Structures

CS10003: PROGRAMMING AND DATA STRUCTURES



Basic Definitions

What is a Structure?

It is a convenient construct for representing a group of logically related data items.

- Examples:
 - Student name, roll number, and marks.
 - Real part and complex part of a complex number.

This is our first look at a non-trivial data structure.

- Helps in organizing complex data in a more meaningful way.

The individual structure elements are called *members*.

Defining a Structure

The composition of a structure may be defined as:

```
struct < name of structure > {  
    < data-type > < member-name1 >;  
    < data-type > < member-name2 >;  
    :  
    < data-type > < member-namek >;  
};
```

For example:

```
struct point {  
    float xcoord;  
    float ycoord;  
};
```

Structure Definition versus Structure Declaration

Structure Definition

```
struct point {  
    float xcoord;  
    float ycoord;  
};
```

- No memory is allocated
- Like defining a new data type

Structure Declaration

```
struct point a, b, c;
```

- Here **a, b, c** are variables of the type **struct point**
- Memory is allocated for **a, b, c**.
- Declaration is possible *after* definition

Structure Declaration can be clubbed with Definition

Separately:

```
struct point {  
    float xcoord;  
    float ycoord;  
};
```

```
struct point a, b, c;
```

Together:

```
struct point {  
    float xcoord;  
    float ycoord;  
} a, b, c;
```

- The struct definition can be reused elsewhere
- Like:

```
struct point p, q;
```

Another way:

```
struct {  
    float xcoord;  
    float ycoord;  
} a, b, c;
```

- In this case we do not have a name for the struct
- Hence we cannot reuse the struct definition

Accessing the members of a structure

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

`<variable-name>.<member-name>`

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

```
struct point {  
    float xcoord;  
    float ycoord;  
} a, b;  
a.xcoord = 2.5;    a.ycoord = 3.2;  
b.xcoord = b.ycoord = 0;
```

Example: Addition of two complex numbers

```
#include <stdio.h>
main( )
{
    struct complex
    {
        float real;
        float imag;
    } a, b, c;

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

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


Assignment of Structure Variables

```
struct class
```

```
{  
    int number;  
    char name[20];  
    float marks;  
};
```

```
main()
```

```
{  
    int x;  
    struct class student1 = {111, "Rao", 72.50};  
    struct class student2 = {222, "Reddy", 67.00};  
    struct class student3;  
  
    student3 = student2;  
}
```

A structure variable can be directly assigned to another

There is no way to directly compare two structure variables

-- except by comparing each member field individually

Arrays of Structures

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

```
struct class
{
    int number;
    char name[20];
    float marks;
};
struct class student[50];
```

- The individual members can be accessed as:

`student[k].marks`

marks of the k^{th} student

`student[k].name[j]`

j^{th} character in the name of the k^{th} student

An interesting observation

```
int a[5] = { 10, 20, 30, 40, 50 };
```

```
int b[5];
```

```
b = a;
```

X This is not allowed

```
struct list {
```

```
    int x[5];
```

```
};
```

```
struct list a, b;
```

```
a.x[0] = 10; a.x[1] = 20;
```

```
a.x[2] = 30; a.x[3] = 40;
```

```
a.x[4] = 50;
```

```
b = a;
```

This is allowed !!

Structures can be copied directly – even if they contain arrays !!

Type Definitions

The *typedef* construct

The *typedef* construct can be used to define new (derived) data types in C.

```
typedef float kilometers_per_hour;    // kilometers_per_hour is a new data type
                                       // Note that no variable is allocated space here
```

```
typedef char roll_number[ 10 ];
    // roll_number is a data type representing array of 10 characters
    // No array has been allocated yet – only the type has been defined
```

```
kilometers_per_hour speed;           // Here speed is a variable
roll_number p = "11AG10015";         // Here variable p is defined and initialized
speed = 40;
```

Structures and *typedef*

Without *typedef*

```
struct complex
{
    float real;
    float imag;
};
```

```
struct complex a, b, c;
```

Here struct complex is like a new data type.

With *typedef*

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

```
complex a, b, c;
```

Here complex is a new data type

Structures are passed by value to functions

```
#include <stdio.h>

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

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

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

```
void print ( _COMPLEX a)
{
    printf("(%.1f, %.1f) ", a.real, a.imag);
}

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

    print(x); print(y); printf("\n");
    swap(x, y);
    print(x); print(y); printf("\n");
}
```

Program output:

```
(4.000000, 5.000000) (10.000000, 15.000000)
(4.000000, 5.000000) (10.000000, 15.000000)
```

Structures can be returned from functions

```
#include <stdio.h>

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

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

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

    z = add(x, y);
    printf(" %f, %f \n", z.real, z.imag);
}
```

Program output:
14.000000, 20.000000

Union

- In a struct, space is allocated as the sum of the space required by its members.
- In a union, space is allocated as the **union** of the space required by its members.
 - We use union when we want only one of the members, but don't know which one.

Suppose we wish to store an ID for each employee.

- Some employees may provide passport ID (8 characters)
- Other employees may provide Aadhar Card Number (12 digit integer)
- If we use a structure with both these fields, we will waste space

Union example

```
typedef union {  
    char passport[9];  
    int aadhar;  
} id ;
```

```
struct employee {  
    char empname[20];  
    int empcode;  
    int idtype;  
    id idnumber;  
};
```

```
main ()  
{  
    struct employee x;  
    ... read employee name and employee code here ...  
    printf("What is your ID type: \n 1. Passport, 2. Aadhar\n");  
    scanf("%d", x.idtype);  
  
    if (idtype == 1) {  
        printf(" Enter passport number: ");  
        scanf( "%8s", x.id.passport ) ;  
    }  
    if (idtype == 2) {  
        printf("Enter Aadhar card number:");  
        scanf("%12d", &x.id.aadhar );  
    }  
}
```

Practice 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. Using this structure for a point
 - Write a function to return the distance between two given points
 - Write a function to return the middle point of the line segment joining two given points
 - Write a function to compute the area of a triangle formed by three given points
 - Write a main function and call the functions from there after reading in appropriate inputs (the points) from the keyboard
3. Define a structure STUDENT to store the following data for a student: name (null-terminated string of length at most 20 chars), roll no. (integer), CGPA (float). Then
 - In main, declare an array of 100 STUDENT structures. Read an integer n and then read in the details of n students in this array
 - Write a function to search the array for a student by name. Returns the structure for the student if found. If not found, return a special structure with the name field set to empty string (just a '\0')
 - Write a function to search the array for a student by roll no.
 - Write a function to print the details of all students with $CGPA > x$ for a given x
 - Call the functions from the main after reading in name/roll no/CGPA to search