Structures

.... and other user-defined data types
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:

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

For example:

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

A structure definition

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

Defining structure variables:

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

A new data-type
Important

• **struct** is the required C keyword

• Do not forget the ; at the end!

• 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
Structure Definition versus Structure Declaration

**Structure Definition**

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

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

**Structure Declaration**

```c
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:

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

struct point a, b, c;
```

Together:

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

• The struct definition can be reused elsewhere
• Like:
  ```c
  struct point p, q;
  ```

Another way:

```c
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

  \[
  \text{variable-name}.\text{member-name}
  \]

  where \textit{variable} refers to the name of a structure-type variable, and \textit{member} refers to the name of a member within the structure.

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

a.xcoord = 2.5;  a.ycoord = 3.2;
b.xcoord = b.ycoord = 0;
```
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:

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

```c
a.real=1.0;   a.img=2.0;
b.real=-3.0;  b.img=4.0;
```
Example: Addition of two complex numbers

```c
#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

```c
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

Two structure variables can not be compared for equality or inequality

if (student1 == student2)…… this cannot be done
Arrays of Structures

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

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

- The individual members can be accessed as:
  ```c
  student[ k ].marks  \hspace{1cm} \textit{marks of the }k^{\text{th}}\text{ student}
  student[ k ].name[ j ]  \hspace{1cm} \textit{j}^{\text{th}}\text{ character in the name of the }k^{\text{th}}\text{ student}
  ```
An interesting observation

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

b = a;
```

```
struct list {
    int x[5];
};

struct list a, b;
a.x[0] = 10;  a.x[1] = 20;
a.x[4] = 50;

b = a;
```

X This is not allowed

This is allowed !!

Structures can be copied directly – even if they contain arrays !!
Type Definitions
The \textit{typedef} construct

The \textit{typedef} construct can be used to define new (derived) data types in C.

```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 \textit{speed} is a variable
roll_number p = "11AG10015"; // Here variable \textit{p} is defined and initialized
speed = 40;
```
Structures and *typedef*

**Without typedef**

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

struct complex a, b, c;
```

*Here struct complex is like a new data type.*

**With typedef**

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

complex a, b, c;
```

*Here complex is a new data type*
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;
Structures are passed by value to functions

```c
#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("(%f, %f) ", 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
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 coordinate system. Using this structure for a point
   i. Write a function to return the distance between two given points
   ii. Write a function to return the middle point of the line segment joining two given points
   iii. Write a function to compute the area of a triangle formed by three given points
   iv. 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

1. 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
2. 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’)
3. Write a function to search the array for a student by roll no.
4. Write a function to print the details of all students with CGPA > x for a given x
5. Call the functions from the main after reading in name/roll no/CGPA to search