Structure

A structure is a collection of variable which can be same or different types. You can refer to a structure as a single variable and to its parts as **members** of that variable by using the dot (.) operator. The power of structures lies in the fact that once defined, the structure name becomes a **user-defined data type** and may be used the same way as other built-in data types, such as int, double, char.

```
struct Student
{
    int rollno, age;
    char name[80];
    float marks;
};

int main()
{

    // declare two variables of the new type
    Student  s1, s3;

    //accessing of data members
    cin >> s1.rollno >> s1.age >> s1.name >> s1.marks;
    cout << s1.rollno << s1.age << s1.name << s1.marks;

    //initialization of structure variable
    Student s2 = {100, 17, "Aniket", 92};
    cout << s2.rollno << s2.age << s2.name << s2.marks;

    //structure variable in assignment statement
    s3 = s2;
    cout << s3.rollno << s3.age << s3.name << s3.marks;

    return 0;
}
```

Defining a structure

When dealing with the students in a school, many variables of different types are needed. It may be necessary to keep track of name, age, Rollno, and marks point for example.

```
struct Student
{
    int rollno, age;
    char name[80];
    float marks;
};
```

Student is called the **structure tag,** and is your brand new data type, like int, double or char.

rollno, name, age, and **marks** are **structure members.**

Declaring Variables of Type struct

The most efficient method of dealing with structure variables is to define the structure **globally**. This tells "the whole world", namely main and any functions in the program, that a new data type exists. To declare a structure globally, place it **BEFORE** void main(). The structure variables can then be defined locally in main, for example…

```
struct Student
{
    int rollno, age;
    char name[80];
    float marks;
};


int main()
{
    // declare two variables of the new type
    Student  s1, s3;
     ……
     ……
     return 0;
}
```

Alternate method of declaring variables of type struct:

```
struct Student
{
    int rollno, age;
    char name[80];
    float marks;
} s1, s3;
```

Accessing of data members

The accessing of data members is done by using the following format:
structure variable.member name
for example

```
cin >> s1.rollno >> s1.age >> s1.name >> s1.marks;
```

Initialization of structure variable

Initialization is done at the time of declaration of a variable. For example

```
Student s2 = {100, 17, "Aniket", 92};
```

Structure variable in assignment statement

```
  s3 = s2;
```

The statement assigns the value of each member of s2 to the corresponding member of s3. Note that one structure variable can be assigned to another only when they are of the same structure type, otherwise complier will give an error.

Nested structure (Structure within structure)

It is possible to use a structure to define another structure. This is called nesting of structure. Consider the following program

```
struct Day
{
    int month, date, year;
};

struct Student
{
    int rollno, age;
    char name[80];
    Day date_of_birth;
    float marks;
};
```

Accessing Member variables of Student

To access members of date_of_birth we can write the statements as below :

```
Student s;  // Structure variable of Student

s.date_of_birth.month = 11;
s.date_of_birth.date = 5;
s.date_of_birth.year = 1999;
```

typedef

It is used to define new data type for an existing data type. It provides and alternative name for standard data type. It is used for self documenting the code by allowing descriptive name for the standard data type.

The general format is:
typedef existing datatype new datatype
for example:
`typedef float real;`
Now, in a program one can use datatype real instead of float.
Therefore, the following statement is valid:
`real amount;`

Enumerated data type

The enum specifier defines the set of names which are stored internally as integer constant. The first name was given the integer value 0, the second value 1 and so on.

for example:

```
enum months{jan, feb, mar, apr, may} ;
```

It has the following features:

- It is user defined.
- It works if you know in advance a finite list of values that a data type can take.
- The list cannot be input by the user or output on the screen.

#define preprocessor directive

The #define preprocessor allows to define symbolic names and constants e.g.
`#define pi 3.14159`
This statement will translate every occurrence of PI in the program to 3.14159

Macros

Macros are built on the #define preprocessor. Normally a macro would look like:
`#define square(x) x*x`
Its arguments substituted for replacement text, when the macro is expanded.