C++

Structure-Union-Enum

1. Structure

- **A structure is same as that of records. It stores related information about an entity.**

- **Structure is basically a user defined data type that can store related information (even of different data types) together.**

- **A structure is declared using the keyword struct followed by a structure name. All the variables of the structures are declared within the structure. A structure type is defined by using the given syntax.**

- **The structure definition does not allocates any memory. It just gives a template that conveys to the C++ compiler how the structure is laid out in memory and gives details of the member names. Memory is allocated for the structure when we declare a variable of the structure. For ex, we can define a variable of student by writing**

```
struct student stud1;
```

1. Structure - Syntax

```
struct struct-name
{ data_type var-name;
  data_type var-name;
  ...
};

struct student
{ int r_no;
  char name[20];
  char course[20];
  float fees;
};
```

1. Structure - Typedef Declaration

```
typedef struct student
{
  int r_no;
  char name[20];
  char course[20];
  float fees;
};
```

- Now that you have preceded the structure's name with the keyword typedef, the student becomes a new data type. Therefore, now you can straight away declare variables of this new data type as you declare variables of type int, float, char, double, etc. to declare a variable of structure student you will just write,
- student stud1;

1. Structure - Initialization

```
struct struct_name
{ data_type member_name1;
  data_type member_name2;
  data_type member_name3;
  ....................................
}
struct_var = {constant1, constant2, constant
3,...};
```

- Now that you have preceded the structure's name with the keyword typedef, the student becomes a new data type. Therefore, now you can straight away declare variables of this new data type as you declare variables of type int, float, char, double, etc. to declare a variable of structure student you will just write,
- student stud1;

1. Structure - Member Accessing

```
struct struct_name
{ data_type member_name1;
  data_type member_name2;
  data_type member_name3;
  ....................................
}
struct_var = {constant1, constant2, constant
3,...};
```

- Initializing a structure means assigning some constants to the members of the structure.
- When the user does not explicitly initializes the structure then C automatically does that. For int and float members, the values are initialized to zero and char and string members are initialized to the '\0' by default.
- The initializers are enclosed in braces and are separated by commas. Note that initializers match their corresponding types in the structure definition.

1. Structure - Member Accessing

```
stud1.r_no = 01;
strcpy(stud1.name, "Rahul");
stud1.course = "BCA";
stud1.fees = 45000;
```

- Each member of a structure can be used just like a normal variable, but its name will be a bit longer. A structure member variable is generally accessed using a '.' (dot operator).
- The syntax of accessing a structure a member of a structure is:

  struct_var.member_name

1. Structure - Problem

- **Write a C++ program using structure to read and display the information about students.**
- **Write a C++ program to swap two numbers using structure.**
- **Write a C++ program using nested structure to read and display the information about students.**

1. Structure - Array of Structure

- **The general syntax for declaring an array of structure can be given as,**

```
struct struct_name struct_var[index];
struct student stud[30];
```

- **Now, to assign values to the i^{th} student of the class, we will write,**

```
stud[i].r_no = 09;
stud[i].name = "RASHI";
stud[i].course = "MCA";
stud[i].fees = 60000;
```

1. Structure - Problem

- **Write a C++ program using structure to read and display the information about students in a class.**

1. Structure - Passing to a Function

```
typedef struct
{
  int x;
  int y;
}POINT;

void display(int, int);
main()
{
  POINT p1 = {2, 3};
  display(p1.x, p1.y);
  return 0;
}
```

1. Structure - Problem

- **Write a C++ program using structure to read and display the information about students in a class using pointer.**

1. Structure - Self Referential Structure

```
struct node
{
  int val;
  struct node *next;
};
```

Self referential structures are those structures that contain a reference to data of its same type. That is, a self referential structure in addition to other data contains a pointer to a data that is of the same type as that of the structure.

1. Structure - with Pointers

```
struct struct_name
{
  data_type member_name1;
  data_type member_name2;

  ....................................
}*ptr;

struct struct_name *ptr;
```

```
struct student *ptr_stud, stud;
ptr_stud = &stud;
```

- 1st Way to Access The Members

```
(*ptr_stud).roll_no;
```

- 2nd Way to Access The Members

```
ptr_stud->roll_no = 01;
```

3. Enum - Definition

- The enumerated data type is a user defined type based on the standard integer type.

- An enumeration consists of a set of named integer constants. That is, in an enumerated type, each integer value is assigned an identifier. This identifier (also known as an enumeration constant) can be used as symbolic names to make the program more readable.

- Enumerations create new data types to contain values that are not limited to the values fundamental data types may take. The syntax of creating an enumerated data type can be given as below.

```
enum enumeration_name{ identifier1, identifier2, ....., identifiern };
```

3. Enum - Declaration

```
enum COLORS {RED, BLUE, BLACK, GREEN, YELLOW, PURPLE, WHITE};
```

In case you do not assign any value to a constant, the default value for the first one in the list - RED (in our case), has the value of 0. The rest of the undefined constants have a value 1 more than its previous one. So in our example,

RED = 0, BLUE = 1, BLACK = 2, GREEN = 3, YELLOW = 4, PURPLE = 5, WHITE =6

If you want to explicitly assign values to these integer constants then you should specifically mention those values as shown below.

```
enum COLORS {RED = 2, BLUE, BLACK = 5, GREEN = 7, YELLOW, PURPLE , WHITE = 15};
```

As a result of the above statement, now

3. Enum - Declaration with Example

```
enum COLORS {RED, BLUE, BLACK, GREEN, YELLOW, PURPLE, WHITE};
```

In case you do not assign any value to a constant, the default value for the first one in the list - RED (in our case), has the value of 0. The rest of the undefined constants have a value 1 more than its previous one. So in our example,

RED = 0, BLUE = 1, BLACK = 2, GREEN = 3, YELLOW = 4, PURPLE = 5, WHITE =6

If you want to explicitly assign values to these integer constants then you should specifically mention those values as shown below.

```
enum COLORS {RED = 2, BLUE, BLACK = 5, GREEN = 7, YELLOW, PURPLE , WHITE = 15};
```

As a result of the above statement, now

RED = 2, BLUE = 3, BLACK = 5, GREEN = 7, YELLOW = 8, PURPLE = 9, WHITE = 15

3. Enum - Variable Declaration

```
enumeration_name  variable_name;
```

```
typedef enum COLORS color;
color forecolor = RED;
```

3. Enum - Variable Declaration

Once the enumerated variable has been declared, values can be stored in it. However, an enumerated variable can hold only declared values for the type. For example, to assign the color black to the background color, we will write

```
bg_color = BLACK;
```

Once an enumerated variable has been assigned a value, we can store its value in another variable of the same type as shown below.

```
enum COLORS bg_color, border_color;
bg_color = BLACK;
border_color = bg_color;
```

3. Enum - Variable Declaration

Enumerated types can be implicitly or explicitly cast. For ex, the compiler can implicitly cast an enumerated type to an integer when required. However, when we implicitly cast an integer to an enumerated type, the compiler will either generate an error or warning message.

```
enum COLORS{RED, BLUE, BLACK, GREEN, YELLOW, PURPLE, WHITE};
enum COLORS c;c = BLACK + WHITE;
```

Here, c is an enumerate data type variable. If we write, c = BLACK + WHITE, then logically, it should be 2 + 6 = 8; which is basically a value of type int. However, the left hand side of the assignment operator is of the type enum COLORS. SO the statement would complain an error. To remove the error, you can do either of two things. First, declare c to be an int and secondly, cast the right hand side in the following manner

```
c = enum COLORS(BLACK + WHITE);
```

3. Enum - Problem

- **Write a C++ program to display the name of the colors using enumerated type data.**