SCALAR DATA TYPE
ENUMERATION

An Enumerated data type is a data type whose domain values are given in a list or ordered list and whose only operations are equality and assignment.

OR

An Enumeration is an ordered list of distinct values.

OR

An Enumeration is a complete ordered listing of all items in a collection.

Pascal was first language which introduced Enumeration.

To make Enumeration facility useful, a programming language must provide a mechanism for declaring and defining the new data type and for declaring variables whose values will come from the elements of type.

It is assumed that these literals are distinct and thus equality can be directly defined.

Before an Era of Enumeration, what we had

for Example :- A variable **StudentClass** might have only four possible values representing freshers, Sophomore, Junior and senior.

Similarly, a variable **StudentSex** might have only two values

representing Male and female.

Before the Concept of Enumeration the languages like FORTRAN or COBOL Such variables is declared as integer type and distinct values are assigned.

Like

Freshers = 1 , Sophomore = 2, and so on

and

Male = 0, Female = 1

Then translator manipulates values as integers.

That Creates big problem like

Sophomore = 1

&

Female = 1

as both has same values so can we apply integer based operation on it

As a point of view of programmer it should not be but according to Translator it can apply as they are of integer types.

Then

Languages Such as C, Pascal and Ada includes an enumeration data type that allows the programmer to define and manipulate Such variables more directly.

Specification of Enumeration :-

The programmers defines both the literal names to be used for the values and their ordering using a declaration such as in pascal

```
type months=(Jan, Feb, Mar, Apr, May, June,
             Jul, Aug, Sep, Oct, Nov, Dec);
```

in C

```
enum StudentClass { Fresh , Soph , Junior , Senior };
enum StudentSex { Male, Female };
```

In pascal, C example can be written as

```
type class = (fresh, Soph, Junior, Senior);
```

followed by declarations for variables such as

```
Studentclass : Class;
Student Sex class : Class;
```

Here type definition introduces the type name **Class**, which may be used wherever the **primitive type** name such as integer might be used.

It also introduces the **literals** of **Fresh, Soph, Junior and Senior**, which may be used where ever a language-defined literal such as "27" might be used. Thus we can write.

if StudentClass = Junior then ....

instead of the less understandable

if StudentClass = 3 then ....

which would be required if integer variables were used.

Static Compiler can find error such as

if StudentClass = Male then

As Male is part of StudentClass.

Operations which we can perform

- Relational operations (equal, less-than, greater-than, etc)
- Assignment
- Successor and predecessor

Implementation :-

→ Each value in the enumeration sequence is represented at runtime by one of integess 0, 1, 2, ... as only a small set of values is involved and the values are never negative.

→ In this integer representation is often shortened to omit the sign bit and use only enough bits for the range of values required, as with the subrange values.

→ only and maximom 2 bits are required to represent the Senior=3 in memory because 3 = 11 (binary)

2 bits only

In C, the programmer may override this default and set any values desired for enumeration values. for example

```
enum Class { Fresh=74, Soph=89, Junior=7
             Senior=28 };
```

With this storage representation for enumeration types, Relational operations such as =, >, and < may be implemented.