1. Introduction to Enumeration

An **Enumeration (enum)** in C language is a **user-defined data type** that consists of a set of **named integer constants**.
It is mainly used to assign **meaningful names to integer values**, which improves program readability and maintainability.

Definition

Enumeration is a user-defined data type that allows a variable to take only one value from a predefined set of named constants.

---

2. Need for Enumeration

Enumeration is needed to:

- Improve code readability
- Avoid using meaningless numbers (magic numbers)
- Make programs easier to understand and maintain
- Restrict variable values to a fixed set
- Reduce logical errors

Example Without enum

```
int day = 3;
```

Example With enum

```
enum Day {MON, TUE, WED, THU, FRI, SAT, SUN};
enum Day day = WED;
```

---

3. Declaration of Enumeration

Syntax

```
enum enum_name
{
   constant1,
   constant2,
   constant3,
   …
};
```

Example

```
enum Color
{
   RED,
   GREEN,
   BLUE
};
```

Here:

- `Color` is the enum name
- `RED`, `GREEN`, `BLUE` are enum constants

---

4. Enumeration Variable Declaration

Syntax

```
enum enum_name variable_name;
```

Example

```
enum Color c;
c = RED;
```

---

5. Default Values in Enumeration

By default:

- First enum constant = **0**
- Next constants increase by **1**

Example

```
enum Week
{
   MON,   // 0
   TUE,   // 1
   WED,   // 2
   THU,   // 3
   FRI    // 4
};
```

---

6. Assigning Custom Values to Enum Constants

Enum constants can be assigned **explicit values**.

Example

```
enum Status
{
   FAIL = 0,
   PASS = 1
};
```

Partial Assignment

```
enum Number
{
   ONE = 1,
   TWO,
   THREE
};
```

Values will be:

- ONE = 1
- TWO = 2
- THREE = 3

---

7. Enumeration and Integer Relationship

- Enum constants are internally stored as **integers**
- Enum variables can be compared using relational operators

Example

```
enum Level {LOW, MEDIUM, HIGH};
enum Level l = MEDIUM;

if(l == MEDIUM)
   printf("Medium Level");
```

---

8. Enumeration with switch Statement

Enums are commonly used with **switch-case**.

Example

```
enum Day {MON, TUE, WED, THU, FRI};

enum Day d = WED;

switch(d)
{
   case MON: printf("Monday"); break;
   case WED: printf("Wednesday"); break;
   default: printf("Other Day");
}
```

---

9. Enumeration Without Enum Name (Anonymous Enum)

Enum can be declared without a name.

Example

```
enum
{
   FALSE,
   TRUE
};
```

---

10. typedef with Enumeration

`typedef` is used to create an alias for enum.

Example

```
typedef enum
{
   OFF,
   ON
} Switch;
```

Usage:

```
Switch s = ON;
```

---

11. Enumeration vs #define

Feature	enum	#define
Type safety	Yes	No
Debugging	Easy	Difficult
Scope	Controlled	Global
Readability	High	Moderate

---

12. Size of Enumeration

- Size of enum is generally same as **int**
- Can vary depending on compiler

Example

```
printf("%d", sizeof(enum Color));
```

---

13. Enumeration in Programs

Example: Menu Driven Program

```
enum Menu {ADD = 1, SUB, MUL, DIV};

int choice = ADD;

if(choice == ADD)
   printf("Addition Selected");
```

14. Advantages of Enumeration

- Improves code readability
- Makes program self-documenting
- Reduces errors
- Easy to maintain
- Useful in decision making

15. Limitations of Enumeration

- Cannot store floating-point values
- Limited to integer constants
- No string values directly
- Less flexible than variables

16. Common Errors with Enumeration

1. Assigning invalid values
2. Confusing enum with variables
3. Forgetting enum keyword
4. Assuming enum is string type

17. Enumeration and Functions

Enum values can be:

- Passed to functions

- Returned from functions

Example

```
enum Result {FAIL, PASS};

enum Result check(int marks)
{
   if(marks >= 40)
      return PASS;
   else
      return FAIL;
}
```

---

18. Applications of Enumeration

- Days of week
- Months of year
- Menu options
- Error codes
- State machines
- Embedded systems

---

19. Difference Between Enumeration and Structure

Feature	Enumeration	Structure
Data types	Single (int)	Multiple
Purpose	Named constants	Group data
Memory	Single value	Multiple values

---

20. Best Practices

- Use meaningful enum names
- Prefer enum over #define
- Use typedef for simplicity
- Avoid assigning invalid values

---

21. Interview / Exam Important Points

- Enum constants are integers
- Default value starts from 0
- Enum improves readability
- Used with switch statements
- Enum is a user-defined data type

---

22. Conclusion

Enumeration in C language is a powerful feature that allows programmers to define a set of named integer constants. It enhances program clarity, reduces errors, and improves maintainability. Enums are widely used in real-world applications, especially in menu-driven programs, system programming, and embedded systems.