



**Department of Computer Science and Engineering,
PES University, Bangalore, India**

**Lecture Notes
Problem Solving With C
UE24CS151B**

***Lecture #18
Enumeration in C***

**By,
Prof. Sindhu R Pai,
Theory Anchor, Feb-May, 2025
Assistant Professor
Dept. of CSE, PESU**

**Many Thanks to
Dr. Shylaja S S (Director, CCBD and CDSAML Research Center, PES University)
Prof. Nitin V Pujari (Dean, Internal Quality Assurance Cell, PES University)**

Unit #: 3**Unit Name: Text Processing and User-Defined Types****Topic: Enumeration in C**

Course objectives: The objective(s) of this course is to make students

- Acquire knowledge on how to solve relevant and logical problems using computing Machine.
- Map algorithmic solutions to relevant features of C programming language constructs.
- Gain knowledge about C constructs and its associated ecosystem.
- Appreciate and gain knowledge about the issues with C Standards and it's respective behaviours.

Course outcomes: At the end of the course, the student will be able to:

- Understand and Apply algorithmic solutions to counting problems using appropriate C Constructs.
- Understand, Analyze and Apply sorting and Searching techniques.
- Understand, Analyze and Apply text processing and string manipulation methods using Arrays, Pointers and functions.
- Understand user defined type creation and implement the same using C structures, unions and other ways by reading and storing the data in secondary systems which are portable.

Sindhu R Pai

Theory Anchor, Feb - May, 2025

Dept. of CSE,

PES University

Introduction

An **enumeration** (or **enum**) is a user-defined data type that **consists of a set of named constant values**. It helps make a program more readable by **replacing numeric constants with meaningful names**. These names enhance the program's maintainability. It is **easy to remember names** rather than the numbers. **We all know google.com. We don't remember the IP address of website google.com.** Typically, enums are used to **represent a collection of fixed sets of related constants**, such as **days of the week, directions, or states in a process**. Also, used in **state machines and flag management**.

The **keyword 'enum'** is used to declare new enumeration types in C. In Essence, Enumerated types provide a symbolic name to represent one state out of a list of states. The names are symbols for **integer constants, which won't be stored anywhere in program's memory**. Enums are used to **replace #define chains**.

Enumeration data type consists of named integer constants as a list. **It start with 0 (zero) by default and value is incremented by 1 for the sequential identifiers in the list.**

Syntax:

```
enum identifier { enumerator-list }; // semicolon compulsory & identifier optional
```

Examples:

```
enum month { Jan, Feb, Mar }; // Jan, Feb and Mar constants will be assigned to 0, 1 and 2 respectively by default
```

```
enum Direction { NORTH, SOUTH, EAST, WEST }; // 0, 1, 2, 3 by default
```

```
enum month { Jan = 1, Feb, Mar }; // Feb and Mar will be assigned with 2 and 3 respectively by default
```

Note: It doesn't make sense to take the addresses of the constants within the enumeration.

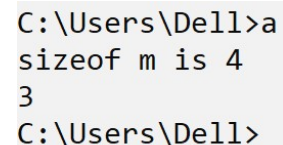
Characteristics of Enums in C

1. **Underlying type is int:** By default, enum constants are assigned integer values starting from 0, incremented by 1. Enum constants are automatically assigned values if no value specified.
2. **Values can be explicitly assigned:** You can manually assign specific integer values to enum constants.
3. **Same value can be assigned to multiple names.**
4. **Scoped within the same block or globally:** **Enum names must be unique** in their scope.
5. **No type safety in C:** Enum variables are essentially int, so you can assign any integer to an enum variable, even if it's not a valid constant defined within the enum.

Coding Example_1: Enum constants are automatically assigned values if no value specified.

```
#include<stdio.h>
enum months
{
    jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec
    // symbol jan gets the value 0, all others previous+1
    // all constants are separated by comma(.).
    // no memory allocated for these constants. Available in this file anywhere directly.
};

int main()
{
    enum months m;    // memory is allocated for m.
    printf("sizeof m is %lu\n",sizeof(m));
    m=apr;            // same as the size of integer
    printf("%d",m);
    return 0;
}
```



```
C:\Users\Dell>a
sizeof m is 4
3
C:\Users\Dell>
```

Coding Example_2: We can assign values to some of the symbol names in any order. All unassigned names get value as, value of previous name plus one.

```
#include <stdio.h>
enum day {sunday = 1, monday, tuesday = 5, wednesday, thursday = 10, friday, saturday};
int main()
{
    //printf("enter the value for monday\n");
    //scanf("%d",&monday); // error: not legal
}
```

```
// Memory not allocated for constants in enum. So ampersand(&) can't be used with those.
//Also value already assigned to monday that can't be changed
printf("%d %d %d %d %d %d %d", sunday, monday, tuesday, wednesday, thursday,
friday, saturday);
return 0;
}
```

```
C:\Users\De11>a
1 2 5 6 10 11 12
C:\Users\De11>
```

Coding Example_3: Only integer constants are allowed. Below code results in Error if abc=3.5 is uncommented.

```
#include<stdio.h>
enum examples
{
    abc = 3.5, def, // Value is not an integer constant
    pqr = "Hello", lmn; // Value is not an integer constant
};
int main()
{
    enum examples e1;
    return 0;
}
```

```
C:\Users\De11>gcc -c check.c
check.c:3:13: error: enumerator value for 'abc' is not an integer constant
{
  abc = 3.5, def, // Value is not an integer constant
  ^~~
check.c:4:10: error: enumerator value for 'pqr' is not an integer constant
pqr = "Hello", lmn; // Value is not an integer constant
  ^~~~~~
check.c:4:22: error: expected ',' or '}' before ';' token
pqr = "Hello", lmn; // Value is not an integer constant
  ^
```

Coding Example_4: Valid arithmetic operators are +, - * and / and %

```
#include<stdio.h>
enum months{          jan,feb,mar,apr,may,jun,july,aug,sep,oct,nov,dec          };
int main()
{
    printf("%d\n",mar-jan); // valid
    printf("%d\n",mar*jan); // valid
    printf("%d\n",mar&&feb); // valid
    //mar++; //error : mar is a constant
    //printf("after incrementing %d\n",mar); // error
    enum months m=feb;
    m=(enum months)(m + jan); // m can be changed. m is a variable of type enum months.
    // But all constants in enum cannot be changed its value
    printf("m, after incrementing %d\n",m);
    for(enum months i=jan;i<=dec;i++) // loop to iterate through all constants in enum
    {
        printf("%d\n", i);
    }
}
```

Coding Example_5: Enumerated Types are Not Strings .Two enum symbols/names can have same value.

```
#include <stdio.h>
enum State {Working = 1, Failed = 0, Freezed = 0};
int main()
{
    printf("%d, %d, %d", Working, Failed, Freezed); // associated values are printed
    return 0;
}
```

Coding Example_6: All enum constants must be unique in their scope.

```
#include<stdio.h>
enum example1 {def, cdt};
enum example2 {abc, def};
int main()
{
    enum example1 e; return 0;
}
```

```
C:\Users\Dell>gcc -c check.c
check.c:2:47: error: redeclaration of enumerator 'def'
enum example1 {def, cdt}; enum example2 {abc, def};
               ^~~~
check.c:2:16: note: previous definition of 'def' was here
enum example1 {def, cdt}; enum example2 {abc, def};
               ^~~~
```

Coding Example_7: It is not possible to change the constants.

```
#include<stdio.h> // Uncommenting the statement feb=10, results in error.
enum months { jan,feb=11,mar,apr,may=23,jun,july };
int main()
{
    enum months m;
    //feb=10; // error
    printf("%d",feb); // no error // we can access it directly
    return 0;
}
```

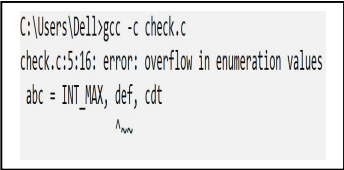
Coding Example_8: Storing the symbol of one enum in another enum variable. It is allowed.

```
#include<stdio.h>
enum nation{ India=1, Nepal };
enum States{ Delhi, Katmandu };
int main()
{
    enum States n=Delhi; enum States n1=India;
    // Value 1 is stored in n1 which is of type enum States
    enum nation n2=Katmandu;
    printf("%d\n",n); printf("%d\n",n1); printf("%d\n",n2);
    return 0;
}
```

```
C:\Users\Dell>a
0
1
1
```

Coding Example_9: The value assigned to enum names must be some integral constant, i.e., the value must be in range from minimum possible integer value to maximum possible integer value

```
#include<limits.h>
#include<stdio.h>
enum examples
{
    abc = INT_MAX, def, cdt
    // INT_MAX is defined in limits.h
    // def must get INT_MAX+1 value which is an error
    // delete def and cdt and compile and run again
};
int main()
{
    enum examples e; e = abc;    printf("%d",e);
    return 0;
}
```

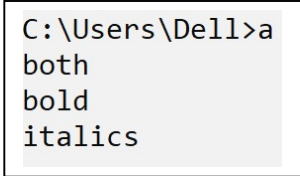


Coding Example_10: Enumerations are actually used to restrict the values to a set of integers within the range of values. But outside the range, if any value is used it is not an error in C standards like C11, C99 and C89. Other languages(ex: Java) doesn't support any value outside the range.

Code to demonstrate why enum support values outside range values in C.

```
#include<stdio.h>
enum design_flags
{
    bold=1, italics=4, underline=8    }df;
int main()
{
    df=bold;
    df=italics|bold; // to set bold and italics. Now df contains 5 which is not defined in enum
    // if you write again df=italics, 5 is replaced with 4. So text is no more bold. It is only italics
    if(df==(bold|italics))
        printf("both\n");
    else if(df==bold)
        printf("bold it is\n");
    else if(df==italics)
        printf("italics it is\n");

    if(df&bold)    // whether the text is bold
        printf("bold\n");
    if(df & italics) // whether the text is italics
        printf("italics\n");
}
```



```
    if(df & underline)    // whether the text is underlined
        printf("underline\n");
    return 0;
}
```

Array of Enums

C allows you to store **multiple values of an enumeration type in a single collection**, just like an array of integers. Since enums are internally treated as integers, you can easily declare an array using the enum type and assign it values using the defined enum constants. It combines the **readability of named constants (from enums) with the power of indexed access (from arrays)**, allowing you to **handle a fixed set of meaningful values in a structured manner**.

Coding Example_11: Demo of array of enums

```
#include <stdio.h>
enum Day { MON, TUE, WED, THU, FRI, SAT, SUN };
int main() {
    enum Day workWeek[5] = { MON, TUE, WED, THU, FRI };
    for (int i = 0; i < 5; i++) {
        printf("Day %d in enum is %d\n", i + 1, workWeek[i]);
    }
    return 0;
}
```

```
C:\Users\Dell>a
Day 1 in enum is 0
Day 2 in enum is 1
Day 3 in enum is 2
Day 4 in enum is 3
Day 5 in enum is 4
```

Passing Enum to a Function

Since enums are internally treated as int in C, passing them to functions is straightforward — just like passing an integer.

Coding Example_12: Demo of passing enum to a function

```
#include <stdio.h>
enum Status { SUCCESS = 1, FAILURE = 0 };
void printStatus(enum Status s) {
    if (s == SUCCESS)    printf("Operation was successful.\n");
    else                printf("Operation failed.\n");
}
int main() {
    enum Status currentStatus = SUCCESS; //enum Status currentStatus = FAILURE;
    printStatus(currentStatus); // Pass enum to function
    return 0; }
}
```

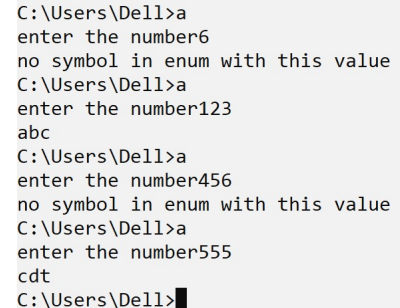
```
C:\Users\Dell>a
Operation was successful.

C:\Users\Dell>gcc -c check.c
C:\Users\Dell>gcc check.o
C:\Users\Dell>a
Operation failed.
```


Coding Example_13: One of the short comings of Enumerated Types is that they don't print nicely.

To print the "String"(symbol) associated with the defined enumerated value, you must use the following cumbersome code.

```
#include<stdio.h>
enum example1 {    abc=123, bef=345, cdt=555    };
void printing(enum example1 e1);
int main()
{
    enum example1 e1;
    // how to print abc, bef and cdt based on user choice?? printf("enter the number");
    scanf("%d",&e1); //enter any number
    printing(e1);    // user defined function to print the symbol name
    return 0;
}
void printing(enum example1 e1)
{
    switch(e1)
    {
        case abc: printf("abc");break;
        case bef:printf("bef");break;
        case cdt:printf("cdt");break;
        default:printf("no symbol in enum with this value"); break;
    }
}
```



```
C:\Users\Dell>a
enter the number6
no symbol in enum with this value
C:\Users\Dell>a
enter the number123
abc
C:\Users\Dell>a
enter the number456
no symbol in enum with this value
C:\Users\Dell>a
enter the number555
cdt
C:\Users\Dell>
```

Think about it!!

- Is pointer to enum same as pointer to integer?
- Can a pointer to integer hold the address of enum variable?
- Can a pointer to integer hold the address of enum constant?
- Can a pointer to enum hold the address of any integer?

Happy Mastering Enumerations in C!!