Unit #: 2

Unit Name: Counting, Sorting and Searching

Topic: Enumerations

Course objectives:

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

COobj1: Acquire knowledge on how to solve relevant and logical problems using computing machine

COobj2: Map algorithmic solutions to relevant features of C programming language constructs

COobj3: Gain knowledge about C constructs and it's associated eco-system

COobj4: Appreciate and gain knowledge about the issues with C Standards and it's respective behaviors

COobj5: Get insights about testing and debugging C Programs

Course outcomes:

At the end of the course, the student will be able to

CO1: Understand and apply algorithmic solutions to counting problems using appropriate C Constructs

CO2: Understand, analyse and apply text processing and string manipulation methods using C Arrays, Pointers and functions

CO3: Understand prioritized scheduling and implement the same using C structures

CO4: Understand and apply sorting techniques using advanced C constructs

CO5: Understand and evaluate portable programming techniques using preprocessor directives and conditional compilation of C Programs

Team – PSWC,

Jan - May, 2022

Dept. of CSE,

PES University

PES University

Enumerations

Introduction

Enumeration (or enum) is a way of creating user-defined data type in C. It is mainly used to assign names to integral constants. The names make a program easy to read and maintain. It is easier to remember names than numbers. Example scenario: We all know Google.com. We don't remember the IP address of the website Google.com.

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 the program's memory. **Enums are used to replace #define chains**

An enumeration data type consists of named integer constants as a list. It starts with 0 (zero) by default and the 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 variables will be assigned to 0, 1 and 2 respectively by default

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

Note: These values are symbols for integer constants, which won't be stored anywhere in program's memory. It doesn't make sense to take its address.

Coding Example_1: Enum variables 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
```


PES University

```
        // symbol jan gets the value 0, all others previous+1
        //If we do not explicitly assign values to enum names, the compiler by default assigns
        //values starting from 0. jan gets value 0, feb gets 1, mar gets 3 and so on.
        // all constants are separated by comma(,).
        // no memory allocated for these constants. Available in this file anywhere directly.
};      // compulsory semi-colon
int main()
{       enum months m;      // memory is allocated for m.
        printf("sizeof m is %lu\n",sizeof(m));      // same as the size of integer
        m=apr;                                      // 3 is stored in m
        printf("%d",m);                             //3
        return 0;
}
```

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;
}
```

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, bef; // Value is not an integer constant
        abc="Hello", bef;     // Value is not an integer constant
};

int main()
{           enum examples e1;
            return 0;
}
```

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
```

PES University

```
        {       printf("%d\n", i);      }
        return 0;
}
```

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.

Below code results in Error: Redeclaration of enumerator 'def'.

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

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

Uncommenting the statement feb=10, results in error.

```
#include<stdio.h>
enum months
{
```


PES University

```
        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",n2);
        return 0;
}
```

Coding Example_9:

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>
```


PES University

```
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;
        }
   }
```

Coding Example_10: 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

Below code Results in Error: overflow in enumeration values.

```
 #include<limits.h>
#include<stdio.h>
enum examples
{
```

PES University

```
        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_11: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)
```

PES University

```
            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;
}
```

PES University