Chapter 12: Enumerations

Section 12.1: Simple Enumeration

An enumeration is a user-defined data type consists of integral constants and each integral constant is given a name. Keyword `enum` is used to define enumerated data type.

If you use `enum` instead of `int` or `string/ char*`, you increase compile-time checking and avoid errors from passing in invalid constants, and you document which values are legal to use.

Example 1

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

void printColor(enum color chosenColor)
{
    const char *color_name = "Invalid color";
    switch (chosenColor)
    {
       case RED:
         color_name = "RED";
         break;

       case GREEN:
        color_name = "GREEN";
        break;

       case BLUE:
        color_name = "BLUE";
        break;
    }
    printf("%s\n", color_name);
}
```

With a main function defined as follows (for example):

```
int main(){
    enum color chosenColor;
    printf("Enter a number between 0 and 2");
    scanf("%d", (int*)&chosenColor);
    printColor(chosenColor);
    return 0;
}
```

Version ≥ C99

Example 2

(This example uses designated initializers which are standardized since C99.)

```
enum week{ MON, TUE, WED, THU, FRI, SAT, SUN };

static const char* const dow[] = {
  [MON] = "Mon", [TUE] = "Tue", [WED] = "Wed",
  [THU] = "Thu", [FRI] = "Fri", [SAT] = "Sat", [SUN] = "Sun" };

void printDayOfWeek(enum week day)
{
   printf("%s\n", dow[day]);
}
```

The same example using range checking:

```
enum week{ DOW_INVALID = -1,
  MON, TUE, WED, THU, FRI, SAT, SUN,
  DOW_MAX };

static const char* const dow[] = {
  [MON] = "Mon", [TUE] = "Tue", [WED] = "Wed",
  [THU] = "Thu", [FRI] = "Fri", [SAT] = "Sat", [SUN] = "Sun" };

void printDayOfWeek(enum week day)
{
   assert(day > DOW_INVALID && day < DOW_MAX);
   printf("%s\n", dow[day]);
}
```

Section 12.2: enumeration constant without typename

Enumeration types can also be declared without giving them a name:

```
  enum { buffersize = 256, };
  static unsigned char buffer [buffersize] = { 0 };
```

This enables us to define compile time constants of type `int` that can as in this example be used as array length.

Section 12.3: Enumeration with duplicate value

An enumerations value in no way needs to be unique:

```
#include <stdlib.h> /* for EXIT_SUCCESS */
#include <stdio.h> /* for printf() */


enum Dupes
{
   Base, /* Takes 0 */
   One, /* Takes Base + 1 */
   Two, /* Takes One + 1 */
   Negative = -1,
   AnotherZero /* Takes Negative + 1 == 0, sigh */
};

int main(void)
{
  printf("Base = %d\n", Base);
  printf("One = %d\n", One);
  printf("Two = %d\n", Two);
  printf("Negative = %d\n", Negative);
  printf("AnotherZero = %d\n", AnotherZero);

  return EXIT_SUCCESS;
}
```

The sample prints:

```
Base = 0
One = 1
Two = 2
```

```
Negative = -1
AnotherZero = 0
```

Section 12.4: Typedef enum

There are several possibilities and conventions to name an enumeration. The first is to use a *tag name* just after the `enum` keyword.

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

This enumeration must then always be used with the keyword *and* the tag like this:

```
enum color chosenColor = RED;
```

If we use `typedef` directly when declaring the `enum`, we can omit the tag name and then use the type without the `enum` keyword:

```
typedef enum
{
    RED,
    GREEN,
    BLUE
} color;

color chosenColor = RED;
```

But in this latter case we cannot use it as `enum color`, because we didn't use the tag name in the definition. One common convention is to use both, such that the same name can be used with or without `enum` keyword. This has the particular advantage of being compatible with C++

```
enum color                /* as in the first example */
{
    RED,
    GREEN,
    BLUE
};
typedef enum color color; /* also a typedef of same identifier */

color chosenColor  = RED;
enum color defaultColor = BLUE;
```

Function:

```
void printColor()
{
    if (chosenColor == RED)
    {
        printf("RED\n");
    }
    else if (chosenColor == GREEN)
    {
        printf("GREEN\n");
```

```
    }
    else if (chosenColor == BLUE)
    {
        printf("BLUE\n");
    }
}
```

For more on `typedef` see Typedef