# C++ Enumeration

An enumeration is a user-defined data type that consists of integral constants. To define an enumeration, keyword **enum** is used.

enum season { spring, summer, autumn, winter };

Here, the name of the enumeration is season.

And, spring, summer and winter are values of type season.

By default, spring is 0, summer is 1 and so on. You can change the default value of an enum element during declaration (if necessary).

enum season

{ spring = 0,

summer = 4,

autumn = 8,

winter = 12

};

## Enumerated Type Declaration

When you create an enumerated type, only blueprint for the variable is created. Here's how you can create variables of enum type.

enum boolean { false, true };

// inside function

enum boolean check;

Here, a variable check of type enum boolean is created.

Here is another way to declare same check variable using different syntax.

enum boolean

{

false, true

} check;

### Example 1: Enumeration Type

#include <iostream>

using namespace std;

enum week { Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday };

int main()

{

week today;

today = Wednesday;

cout << "Day " << today+1;

return 0;

}

**Output**

Day 4

### Example2: Changing Default Value of Enums

#include <iostream>

using namespace std;

enum seasons { spring = 34, summer = 4, autumn = 9, winter = 32};

int main() {

seasons s;

s = summer;

cout << "Summer = " << s << endl;

return 0;

}

**Output**

Summer = 4

## Why enums are used in C++ programming?

An enum variable takes only one value out of many possible values. Example to demonstrate it,

#include <iostream>

using namespace std;

enum suit {

club = 0,

diamonds = 10,

hearts = 20,

spades = 3

} card;

int main()

{

card = club;

cout << "Size of enum variable " << sizeof(card) << " bytes.";

return 0;

}

**Output**

Size of enum variable 4 bytes.

It's because the size of an integer is 4 bytes.;

This makes enum a good choice to work with flags.

You can accomplish the same task using [C++ structures](https://www.programiz.com/cpp-programming/structure). However, working with enums gives you efficiency along with flexibility.

### How to use enums for flags?

Let us take an example,

enum designFlags {

ITALICS = 1,

BOLD = 2,

UNDERLINE = 4

} button;

Suppose you are designing a button for Windows application. You can set flags ITALICS, BOLD and UNDERLINE to work with text.

There is a reason why all the integral constants are power of 2 in above pseudocode.

// In binary

ITALICS = 00000001

BOLD = 00000010

UNDERLINE = 00000100

Since, the integral constants are power of 2, you can combine two or more flags at once without overlapping using bitwise OR **|** operator. This allows you to choose two or more flags at once. For example,

#include <iostream>

using namespace std;

enum designFlags {

BOLD = 1,

ITALICS = 2,

UNDERLINE = 4

};

int main()

{

int myDesign = BOLD | UNDERLINE;

// 00000001

// | 00000100

// \_\_\_\_\_\_\_\_\_\_\_

// 00000101

cout << myDesign;

return 0;

}

Output

5

When the output is 5, you always know that bold and underline is used.

Also, you can add flag to your requirements.

if (myDesign & ITALICS) {

// code for italics

}