12. *endl*, Constants and Operators

- Shivam Malhotra

endl keyword

When used with cout, *endl* directs the output to the next line just like '\n'

```
cout << "I am bond" << endl;
cout << "James bond";
```

Output :

```
I am bond
James bond
```

```
cout << "Bleep" << endl << "Bloop";
```

Output :

```
Bleep
Bloop
```

Why use endl?

Compare the following two statements :

```
cout << "Bleep" << endl << "Bloop";
```

```
cout << "Bleep\nBloop";
```

Note that both the statements give the same output :

```
Bleep
Bloop
```

But the left one is cleaner and easier to read.

Hence using *endl* is preferred over '\n' in such cases.

Constants

Constants are named storage locations whose value can not change during the program execution

Compare this definition with that of a variable:

A variable is a named storage location whose value can change during program execution

Constants

Declaration of a constant is similar to the declaration of a variable with *const* keyword in front of it. For example,

```
const int sum = 25;
```

A constant must be given an initial value at the time of its declaration.

Example

```
#include <iostream.h>
#include <conio.h>
int main()
{
        clrscr();
        const double number = 15.23;
        // number = 17;
        cout << "number = " << number;
        getch();
        return 0;
}
```

Cannot modify the value

Initial value must be provided

Why use Constants?

Constants are a reminder to the programmer that some particular values must not be modified. This prevent us from accidentally changing them, and also enforces consistency.

For example, if we want to enforce that the value of pi should be used as 3.14, we can make it a constant.

```
const float pi = 3.14;
cin >> radius;
float circum = 2 * pi * radius;
float area = pi * radius * radius;
```

Operators

An operator is a symbol or character or word which trigger some operation (computation) on its operands, ex. +, *, **<<, >>** etc.

In the coming topics, we will study many different operators.

We will start by studying arithmetic operators:

Binary Arithmetic Operators

Operator	Example
Addition Operator (+)	If a = 13 and b = 5, (a+b) will be 18
Subtraction Operator (-)	If a = 13 and b = 5, (a-b) will be 8
Multiplication Operator (*)	If a = 13 and b = 5, (a*b) will be 65
Division Operator (/)	If a = 13 and b = 5, (a/b) will be 2
Modulus Operator (%)	If a = 13 and b = 5, (a%b) will be 3

Unary Arithmetic Operators

Unary operator is an operator which takes only one operand. Thus the expression looks like :

(operator) operand

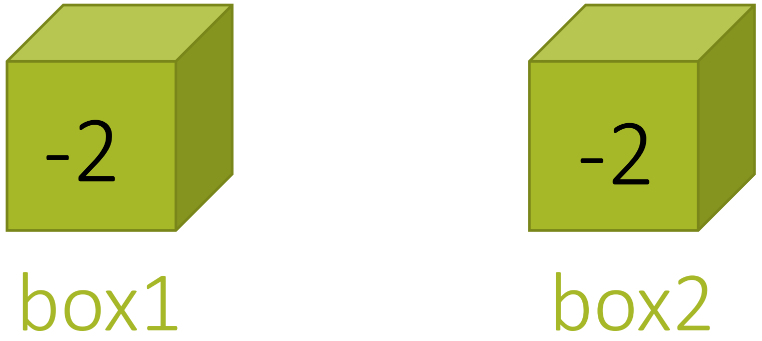
There are two unary arithmetic operators :

1. Unary + : For example, +someVariable
2. Unary - : For example, -someVariable

Unary +

Unary + operator is applied in front of its operand and the value of expression is same as the value of operand.

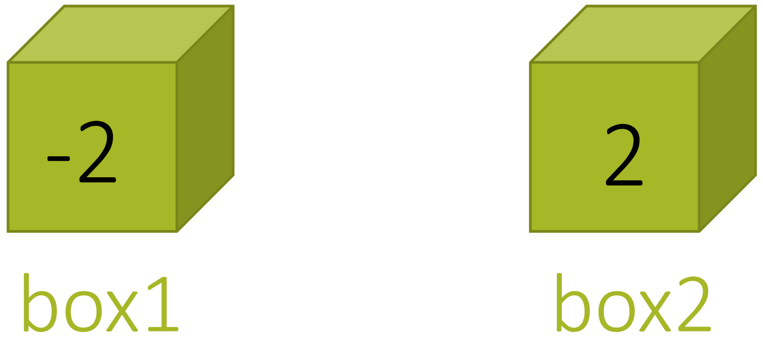
```
int box1, box2;
box1 = -2;
box2 = +box1;
```



Unary -

Unary - operator is also applied in front of its operand and the value of expression is negative of the value of operand.

```
int box1, box2;
box1 = -2;
box2 = -box1;
```



What’s ahead?

In the next video, we will study about increment (++) and decrement (--) operators.