C++ Expression

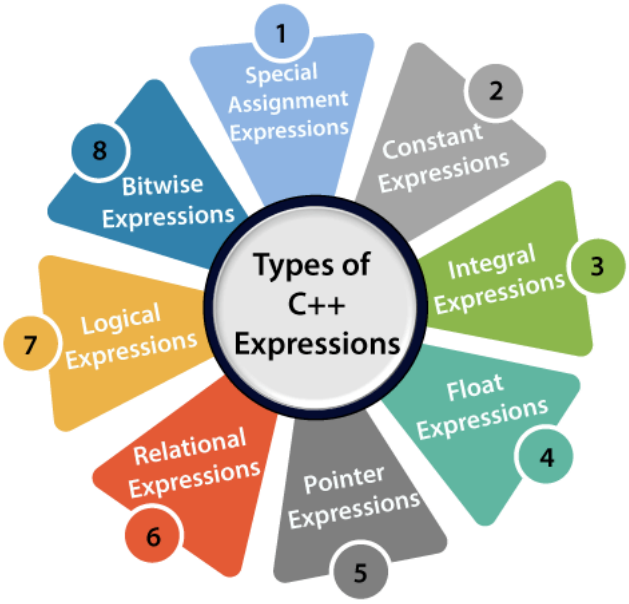
C++ expression consists of operators, constants, and variables which are arranged according to the rules of the language. It can also contain function calls which return values. An expression can consist of one or more operands, zero or more operators to compute a value. Every expression produces some value which is assigned to the variable with the help of an assignment operator.

Examples of C++ expression:

```
(a+b) - c
(x/y) -z
4a2 - 5b +c
(a+b) * (x+y)
```

An expression can be of following types:

- Constant expressions
- Integral expressions
- Float expressions
- Pointer expressions
- Relational expressions
- Logical expressions
- Bitwise expressions
- Special assignment expressions



If the expression is a combination of the above expressions, such expressions are known as compound expressions.

Constant expressions

A constant expression is an expression that consists of only constant values. It is an expression whose value is determined at the compile-time but evaluated at the run-time. It can be composed of integer, character, floating-point, and enumeration constants.

⇧ SCROLL TO TOP

Constants are used in the following situations:

- It is used in the subscript declarator to describe the array bound.
- It is used after the case keyword in the switch statement.
- It is used as a numeric value in an **enum**
- It specifies a bit-field width.
- It is used in the pre-processor **#if**

In the above scenarios, the constant expression can have integer, character, and enumeration constants. We can use the static and extern keyword with the constants to define the function-scope.

The following table shows the expression containing constant value:

Expression containing constant	Constant value
x = (2/3) * 4	(2/3) * 4
extern int y = 67	67
int z = 43	43
static int a = 56	56

Let's see a simple program containing constant expression:

```
#include <iostream>
using namespace std;
int main()
{
   int x;       // variable declaration.
   x=(3/2) + 2;  // constant expression
   cout<<"Value of x is : "<<x;  // displaying the value of x.
   return 0;
}
```

In the above code, we have first declared the 'x' variable of integer type. After declaration, we assign the simple constant expression to the 'x' variable.

Output

```
Value of x is : 3
```

Integral Expressions

An integer expression is an expression that produces the integer value as output after performing all the explicit and implicit conversions.

Examples of integral expression:

```
(x * y) -5
x + int(9.0)
```

where x and y are the integers.

Let's see a simple example of integral expression:

```
#include <iostream>
using namespace std;
int main()
{
   int x;  // variable declaration.
   int y;  // variable declaration
   int z;  // variable declaration
   cout<<"Enter the values of x and y";
   cin>>x>>y;
   z=x+y;
   cout<<"\n"<<"Value of z is :"<<z; //  displaying the value of z.
   return 0;
}
```

In the above code, we have declared three variables, i.e., x, y, and z. After declaration, we take the user input for the values of 'x' and 'y'. Then, we add the values of 'x' and 'y' and stores their result in 'z' variable.

Output

```
Enter the values of x and y
8
9
Value of z is :17
```

Let's see another example of integral expression.

```
#include <iostream>
using namespace std;
int main()
{

  int x;   // variable declaration
  int y=9;    // variable initialization
  x=y+int(10.0);    // integral expression
 cout<<"Value of x : "<<x;   // displaying the value of x.
 return 0;
}
```

⇧ SCROLL TO TOP : declare two variables, i.e., x and y. We store the value of expression (y+int(10.0)) in a 'x' variable.

Output

```
Value of x : 19
```

Float Expressions

A float expression is an expression that produces floating-point value as output after performing all the explicit and implicit conversions.

The following are the examples of float expressions:

```
x+y
(x/10) + y
34.5
x+float(10)
```

Let's understand through an example.

```
#include <iostream>
using namespace std;
int main()
{

  float x=8.9;     // variable initialization
  float y=5.6;     // variable initialization
  float z;         // variable declaration
  z=x+y;
  std::cout <<"value of z is :"  << z<<std::endl;  // displaying the value of z.


   return 0;
}
```

Output

```
value of z is :14.5
```

Let's see another example of float expression.

```
#include <iostream>
using namespace std;
int main()
{
  float x=6.7;   // variable initialization
  float y;     // variable declaration
  y=x+float(10);   // float expression
  std::cout <<"value of y is :"  << y<<std::endl;  // displaying the value of y
```

⇧ SCROLL TO TOP

In the above code, we have declared two variables, i.e., x and y. After declaration, we store the value of expression (x+float(10)) in variable 'y'.

Output

```
value of y is :16.7
```

Pointer Expressions

A pointer expression is an expression that produces address value as an output.

The following are the examples of pointer expression:

```
&x
ptr
ptr++
ptr-
```

Let's understand through an example.

```
#include <iostream>
using namespace std;
int main()
{

  int a[]={1,2,3,4,5};  // array initialization
  int *ptr;      // pointer declaration
  ptr=a;    // assigning base address of array to the pointer ptr
  ptr=ptr+1;   // incrementing the value of pointer
  std::cout <<"value of second element of an array : "  << *ptr<<std::endl;
  return 0;
}
```

In the above code, we declare the array and a pointer ptr. We assign the base address to the variable 'ptr'. After assigning the address, we increment the value of pointer 'ptr'. When pointer is incremented then 'ptr' will be pointing to the second element of the array.

Output

```
value of second element of an array : 2
```

Relational Expressions

A relational expression is an expression that produces a value of type bool, which can be either true or false. It is also known as a boolean expression. When arithmetic expressions are used on both sides of the relational operator, arithmetic expressions are evaluated first, and then their results are compared.

The following are the examples of the relational expression:

```
a>b
a-b >= x-y
a+b>80
```

Let's understand through an example

⇧ SCROLL TO TOP >>

```
using namespace std;
int main()
{
   int a=45;    // variable declaration
   int b=78;    // variable declaration
   bool y= a>b;   // relational expression
   cout<<"Value of y is :"<<y;  // displaying the value of y.
   return 0;
}
```

In the above code, we have declared two variables, i.e., 'a' and 'b'. After declaration, we have applied the relational operator between the variables to check whether 'a' is greater than 'b' or not.

Output

```
Value of y is :0
```

Let's see another example.

```
#include <iostream>
using namespace std;
int main()
{
 int a=4;     // variable declaration
 int b=5;     // variable declaration
 int x=3;     // variable declaration
 int y=6;    // variable declaration
 cout<<((a+b)>=(x+y));   // relational expression
 return 0;
}
```

In the above code, we have declared four variables, i.e., 'a', 'b', 'x' and 'y'. Then, we apply the relational operator (>=) between these variables.

Output

```
1
```

Logical Expressions

A logical expression is an expression that combines two or more relational expressions and produces a bool type value. The logical operators are '&&' and '||' that combines two or more relational expressions.

The following are some examples of logical expressions:

```
a>b && x>y
a>10 || b==5
```

Let's see a simple example of logical expression.

```
#include <iostream>
using namespace std;
int main()
{
 int a=2;
 int b=7:
```

```
cout<<((a>b)||(a>c));
return 0;
}
```

Output

```
0
```

Bitwise Expressions

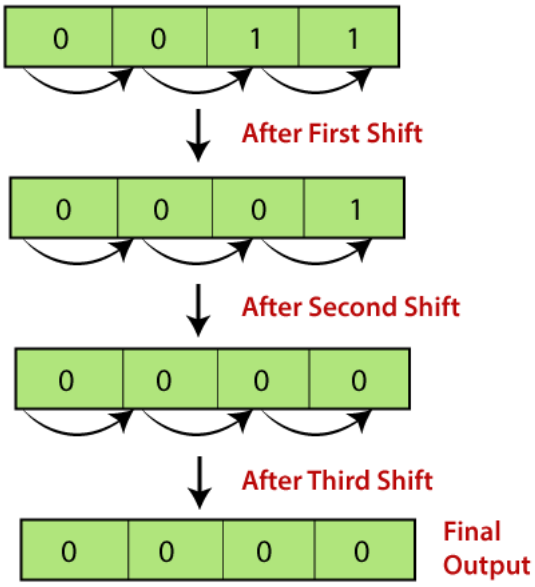
A bitwise expression is an expression which is used to manipulate the data at a bit level. They are basically used to shift the bits.

For example:

x=3

x>>3 // This statement means that we are shifting the three-bit position to the right.

In the above example, the value of 'x' is 3 and its binary value is 0011. We are shifting the value of 'x' by three-bit position to the right. Let's understand through the diagrammatic representation.



Let's see a simple example.

```
#include <iostream>
using namespace std;
int main()
{
 int x=5;   // variable declaration
std::cout << (x>>1) << std::endl;
return 0;
}
```

In the above code, we have declared a variable 'x'. After declaration, we applied the bitwise operator, i.e., right shift operator to shift one-bit position to right.

Output

```
2
```

⇧ SCROLL TO TOP

Let's look at another example.

```
#include <iostream>
using namespace std;
int main()
{
 int x=7;   // variable declaration
std::cout << (x<<3) << std::endl;
return 0;
}
```

In the above code, we have declared a variable 'x'. After declaration, we applied the left shift operator to variable 'x' to shift the three-bit position to the left.

Output

```
56
```

Special Assignment Expressions

Special assignment expressions are the expressions which can be further classified depending upon the value assigned to the variable.

- **Chained Assignment**

Chained assignment expression is an expression in which the same value is assigned to more than one variable by using single statement.

For example:

```
a=b=20
 or
(a=b) = 20
```

Let's understand through an example.

```
#include <iostream>
using namespace std;
int main()

 int a;   // variable declaration
 int b;   // variable declaration
 a=b=80;  // chained assignment
 std::cout <<"Values of 'a' and 'b' are : " <<a<<","<<b<< std::endl;
 return 0;
}
```

In the above code, we have declared two variables, i.e., 'a' and 'b'. Then, we have assigned the same value to both the variables using chained assignment expression.

Output

```
Values of 'a' and 'b' are : 80,80
```

Note: Using chained assignment expression, the value cannot be assigned to the variable at the time of declaration. For example, int a=b=c=90 is not a valid statement.

- **Embedded Assignment Expression**

An embedded assignment expression is an assignment expression in which assignment expression is enclosed within another assignment expression.

Let's understand through an example.

```
#include <iostream>
using namespace std;
int main()
{
 int a;  // variable declaration
 int b;  // variable declaration
 a=10+(b=90);  // embedded assignment expression
 std::cout <<"Values of 'a' is " <<a<< std::endl;
 return 0;
}
```

In the above code, we have declared two variables, i.e., 'a' and 'b'. Then, we applied embedded assignment expression (a=10+(b=90)).

Output

```
Values of 'a' is 100
```

- **Compound Assignment**

A compound assignment expression is an expression which is a combination of an assignment operator and binary operator.

For example,

```
a+=10;
```

In the above statement, 'a' is a variable and '+=' is a compound statement.

Let's understand through an example.

```
#include <iostream>
using namespace std;
int main()
{
  int a=10;   // variable declaration
  a+=10;    // compound assignment
  std::cout << "Value of a is :" <<a<< std::endl; // displaying the value of a.
  return 0;
}
```

In the above code, we have declared a variable 'a' and assigns 10 value to this variable. Then, we applied compound assignment operator (+=) to 'a' variable, i.e., a+=10 which is equal to (a=a+10). This statement increments the value of 'a' by 10.

Output

```
Value of a is :20
```

← Prev Next →

⇧ SCROLL TO TOP