After bitwise operators, the next type of operators is 'Compound Assignment' operators.

These operators have the syntax,

variable operator= expression;

which translates into,

variable = variable operator expression;

The next type of operators is 'Member and Pointer operators.'

a) Subscript ⇝ Will be discussed in arrays.
b) Dereference and reference ⇝ Already discussed during 'Pointers'.
c) Structure dereference and structure reference ⇝ Will be discussed during "User defined types – Structures".

Finally we are left with the discussion of few of the miscellaneous operators:

The Comma Operator:

In C and C++, comma can be used as in two contexts:

a) Comma as an operator

The comma operator is a binary operator that evaluates its first operand and discards the result, it then evaluates the second operand and returns the value and type. The comma operator has the lowest precedence of any C operator.

```
int i = (5, 10);            // 10 is assigned to i

int j = ( f1(), f2() );

/* f1() is called and evaluated first followed by f2(). The returned value of f2 is collected in j. */
```

b) Comma as a separator

Comma often acts as separator when used with function calls, variable declarations, listing, etc.

```
int a = 1, b = 2;

void fun (x, y);
```

Here are few example programs: -

1. #include <stdio.h>

```
int main()
{
        int x = 10, y = 15;
        printf("%d\n", (x,y));
        return 0;
}
```

Output = 15

2. #include <stdio.h>

```
int main()
{
        int x = 10, y;
        y = (x++, printf("x = %d\n", x), ++x, printf("x = %d\n"),
                x++);
        printf ("y = %d\nx = %d\n", y, x);
        return 0;
}
```

Output:

Warning: Incompatible implicit declaration of built-in function 'printf'

x = 11

x = 12

y = 12

x = 13

Consider the following C programs:

a) #include <stdio.h>

```
int main()
{
```

```
        int a = 1, 2, 3;
        printf("%d", a);
        return 0;
    }
```

The above program fails in compilation with the following error –

[Error] expected identifier or '(' before numeric constant

This is because comma works just as a separator here.

b)
```
#include <stdio.h>
    int main()
    {
        int a;
        a = 1, 2, 3;
        printf("%d", a);
        return 0;
    }
```

But this program compiles fine and prints 1. Comma works as an operator here. Since precedence of comma operator is least in operator precedence table. So the assignment takes precedence over comma and expression is evaluated as (a = 1), 2, 3;

c)
```
#include <stdio.h>
    int main()
    {
        int a;
        (a = 1, 2, 3);
        printf("%d", a);
        return 0;
    }
```

Here because of parenthesis, comma operator is executed first and then assignment takes place. So the output is 3.

d) #include <stdio.h>

```
int main()
{
    int a = 10,b = 20;
    (b, a) = 30;
    printf("%d", a);
    return 0;
}
```

Using the result of comma operator as lvalue is not valid in C, but is possible in C++. So above program compiles in C++ and prints 30.