data types such as structures and classes. References work wonderfully well with these user-defined data types.

3.14 OPERATORS IN C++

C++ has a rich set of operators. All C operators are valid in C++ also. In addition, C++ introduces some new operators. We have already seen two such operators, namely, the insertion operator <<, and the extraction operator >>. Other new operators are:

```
::        Scope resolution operator
::*       Pointer-to-member declarator
->*       Pointer-to-member operator
.*        Pointer-to-member operator
delete    Memory release operator
endl      Line feed operator
new       Memory allocation operator
setw      Field width operator
```

In addition, C++ also allows us to provide new definitions to some of the built-in operators. That is, we can give several meanings to an operator, depending upon the types of arguments used. This process is known as *operator overloading*.

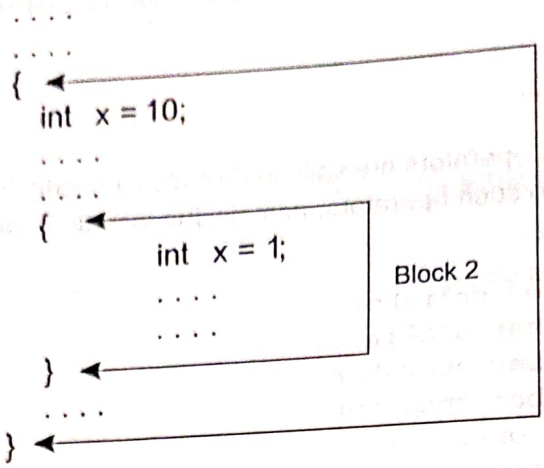
3.15 SCOPE RESOLUTION OPERATOR

Like C, C++ is also a block-structured language. Blocks and scopes can be used in constructing programs. We know that the same variable name can be used to have different meanings in different blocks. The scope of the variable extends from the point of its declaration till the end of the block containing the declaration. A variable declared inside a block is said to be local to that block. Consider the following segment of a program:

```
.....
.....
{
      int x = 10;
      .....
      .....
}
.....
.....
{
      int x = 1;
      .....
      .....
}
```

The two declarations of x refer to two different memory locations containing different values. Statements in the second block cannot refer to the variable x declared in the first block, and vice versa. Blocks in C++ are often nested. For example, the following style is common:

Scanned with CamScanner



Block2 is contained in block1. Note that a declaration in an inner block *hides* a declaration of the same variable in an outer block and, therefore, each declaration of **x** causes it to refer to a different data object. Within the inner block, the variable **x** will refer to the data object declared therein.

In C, the global version of a variable cannot be accessed from within the inner block. C++ resolves this problem by introducing a new operator :: called the *scope resolution operator*. This can be used to uncover a hidden variable. It takes the following form:

```
:: variable-name
```

This operator allows access to the global version of a variable. For example, ::count means the global version of the variable count (and not the local variable count declared in that block). Program 3.1 illustrates this feature.

Program 3.1 Scope Resolution Operator

```
#include <iostream>

using namespace std;

int m = 10;             // global m

int main()
{
    int m = 20;         // m redeclared, local to main

  {
    int k = m;
    int m = 30;         // m declared again
                        // local to inner block
    cout << "we are in inner block \n";
    cout << "k = " << k << "\n";
    cout << "m = " << m << "\n";
```

Scanned with CamScanner

```
        cout << "::m =" << ::m << "\n";
    }
    cout << "\nWe are in outer block \n";
    cout << "m = " << m << "\n";
    cout << "::m = " << ::m << "\n";

    return 0;
}
```

The output of Program 3.1 would be:

```
We are in inner block
k = 20
m = 30
::m = 10

We are in outer block
m = 20
::m = 10
```

In the above program, the variable **m** is declared at three places, namely, outside the **main()** function inside the **main()**, and inside the inner block.

Scanned with CamScanner