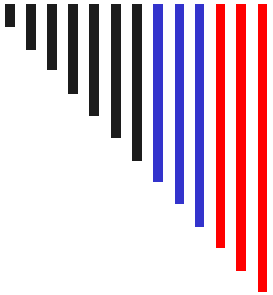


Chapter 3
Structure of a C Program

Objectives

- To be able to list and describe the six expression categories
- To understand the rules of precedence and associativity in evaluating expressions
- To understand the result of side effects in expression evaluation
- To be able to predict the results when an expression is evaluated
- To understand implicit and explicit type conversion
- To understand and use the first four statement types: null, expression, return, and compound

3-1 Expressions

An expression is a sequence of operands and operators that reduces to a single value. Expressions can be simple or complex. An operator is a syntactical token that requires an action be taken. An operand is an object on which an operation is performed; it receives an operator's action.

Topics discussed in this section:

Primary Expressions
Postfix Expressions
Prefix Expressions
Unary Expressions
Binary Expressions

Note

An expression always reduces to a single value.

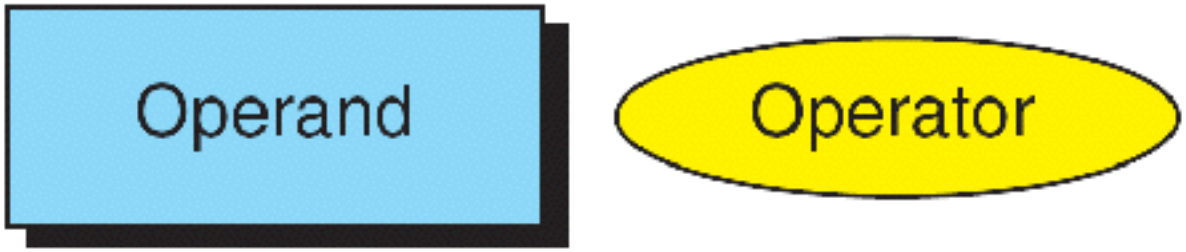


FIGURE 3-2 Postfix Expressions

Note

(a++) has the same effect as (a = a + 1)

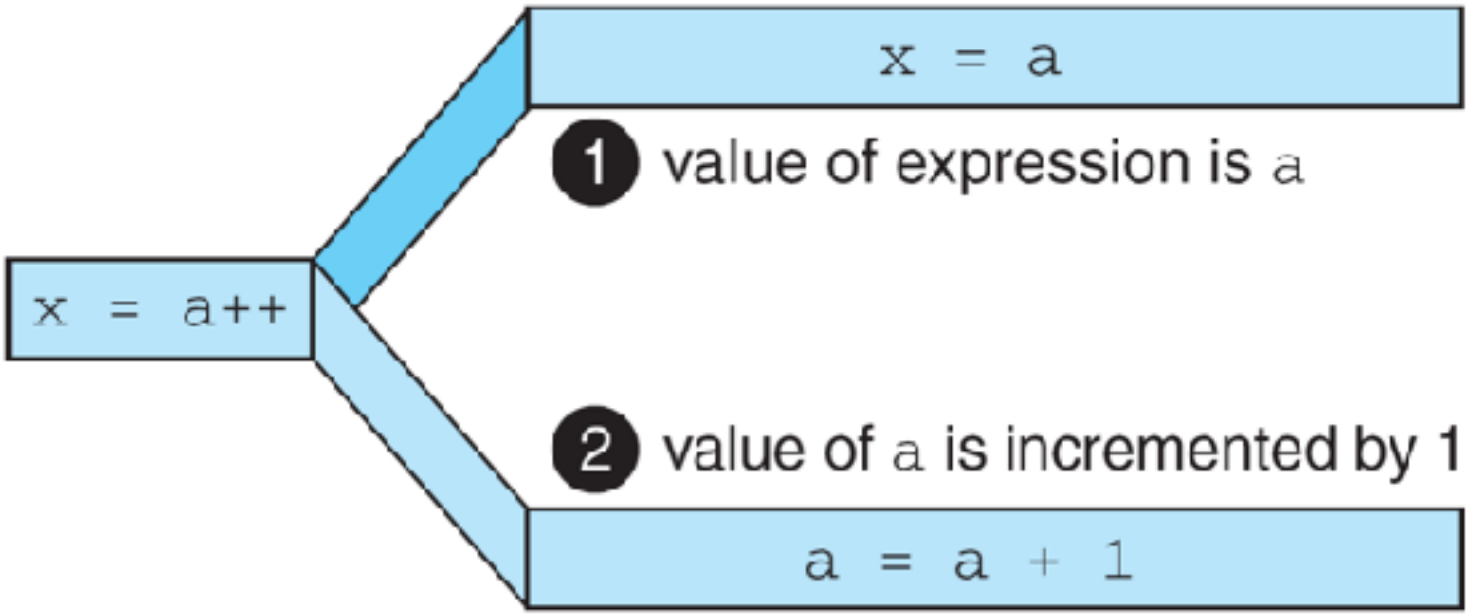


FIGURE 3-3 Result of Postfix a++

Note

The operand in a postfix expression must be a variable.

PROGRAM 3-1 Demonstrate Postfix Increment

```
 1  /* Example of postfix increment.
 2        Written by:
 3        Date:
 4  */
 5  #include <stdio.h>
 6  int main (void)
 7  {
 8  // Local Declarations
 9     int a;
10
11  // Statements
12     a = 4;
13     printf("value of a     : %2d\n", a);
14     printf("value of a++   : %2d\n",   a++);
15     printf("new value of a: %2d\n\n", a);
16     return 0;
17  }  // main
```

PROGRAM 3-1 Demonstrate Postfix Increment (continued)

```
Results:
value of a     :  4
value of a++   :  4
new value of a:  5
```

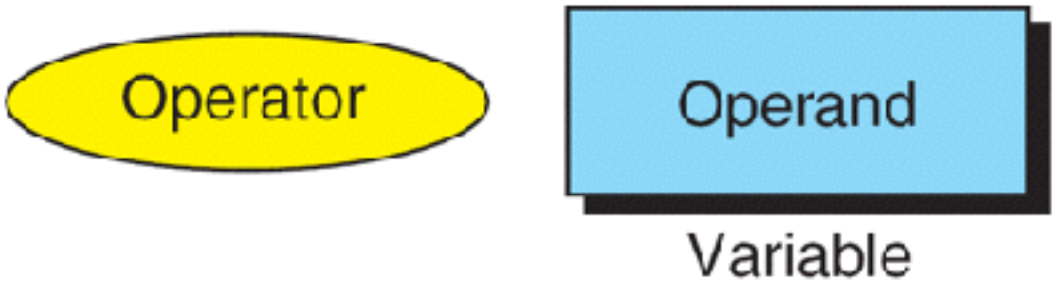


FIGURE 3-4 Prefix Expression

Note

The operand of a prefix expression must be a variable.

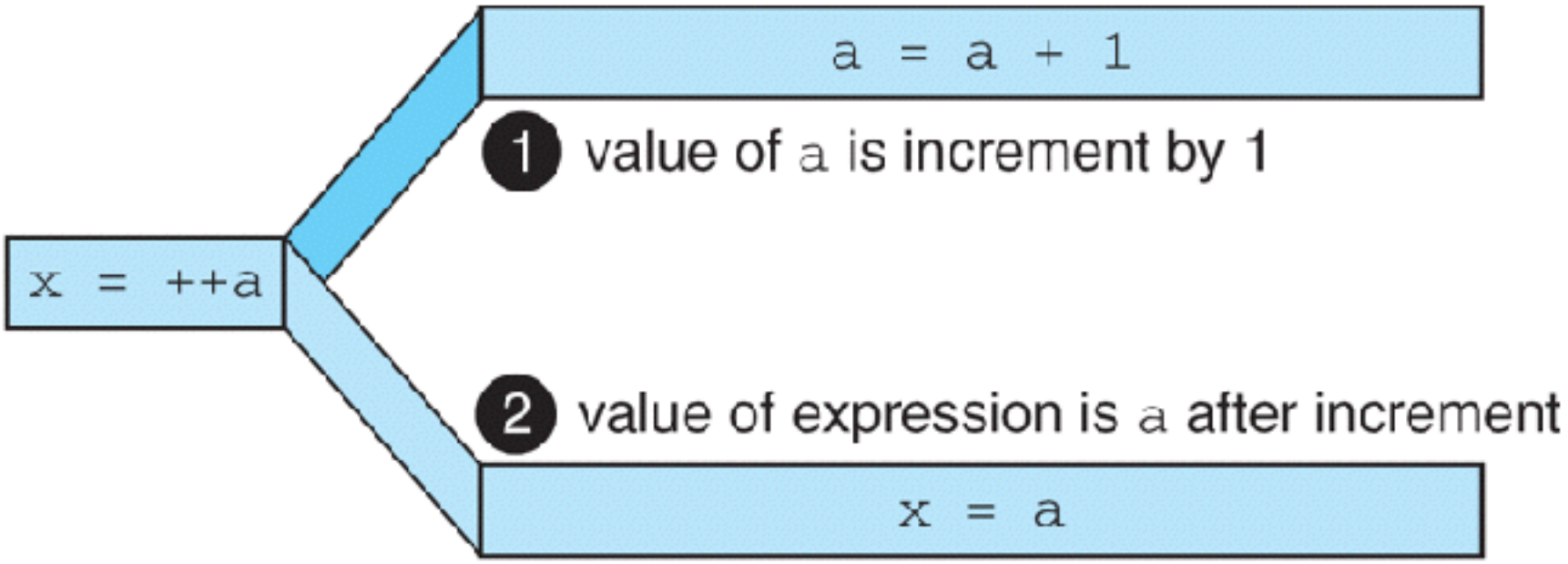


FIGURE 3-5 Result of Prefix ++a

Note

(++a) has the same effect as (a = a + 1)

PROGRAM 3-2 Demonstrate Prefix Increment

```
1  /* Example of prefix increment.
2        Written by:
3        Date:
4  */
5  #include <stdio.h>
6  int main (void)
7  {
8  // Local Declarations
9     int a;
10
11 // Statements
12    a = 4;
13    printf("value of a     : %2d\n", a);
14    printf("value of ++a   : %2d\n", ++a);
15    printf("new value of a: %2d\n", a);
16    return 0;
17 }  // main
```

PROGRAM 3-2 Demonstrate Prefix Increment (continued)

```
Results:
value of a     :  4
value of ++a   :  5
new value of a:  5
```

Note

If ++ is after the operand, as in a++, the increment takes place after the expression is evaluated.

If ++ is before the operand, as in ++a, the increment takes place before the expression is evaluated.

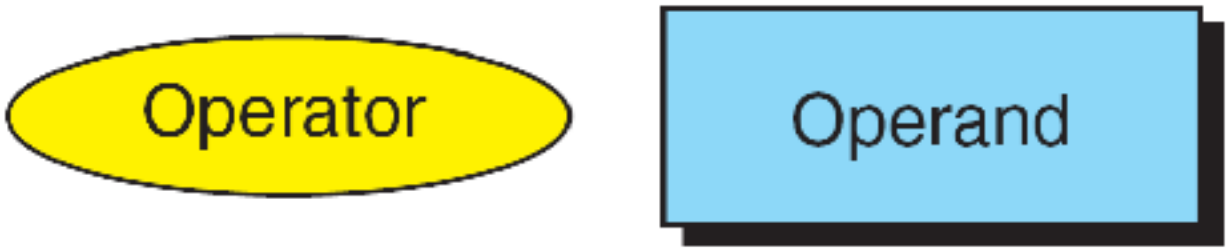


FIGURE 3-6 Unary Expressions

Expression	Contents of a Before *and* After Expression	Expression Value
+a	3	+3
-a	3	-3
+a	-5	-5
-a	-5	+5

Table 3-1 Examples of Unary Plus And Minus Expressions

FIGURE 3-7 Binary Expressions

Note

Both operands of the modulo operator (%) must be integral types.

PROGRAM 3-3 Binary Expressions

```
 1  /* This program demonstrates binary expressions.
 2        Written by:
 3        Date:
 4  */
 5  #include <stdio.h>
 6  int main (void)
 7  {
 8  // Local Declarations
 9     int   a = 17;
10     int   b = 5;
11     float x = 17.67;
12     float y = 5.1;
13
14  // Statements
15     printf("Integral calculations\n");
16     printf("%d + %d  = %d\n", a, b, a + b);
```

PROGRAM 3-3 Binary Expressions (continued)

```
17      printf("%d - %d  = %d\n", a, b, a - b);
18      printf("%d * %d  = %d\n", a, b, a * b);
19      printf("%d / %d  = %d\n", a, b, a / b);
20      printf("%d %% %d  = %d\n", a, b, a % b);
21      printf("\n");
25      printf("%f - %f  = %f\n", x, y, x - y);
26      printf("%f * %f  = %f\n", x, y, x * y);
27      printf("%f / %f  = %f\n", x, y, x / y);
28      return 0;
29  }   // main
```

PROGRAM 3-3 Binary Expressions (continued)

```
Results:
Integral calculations
17 + 5 = 22
17 - 5 = 12
17 * 5 = 85
17 / 5 = 3
17 % 5 = 2

Floating-point calculations
17.670000 + 5.100000  = 22.770000
17.670000 - 5.100000  = 12.570000
17.670000 * 5.100000  = 90.116997
17.670000 / 5.100000  = 3.464706
```

Note

The left operand in an assignment expression must be a single variable.

Compound Expression	Equivalent Simple Expression
`x *= expression`	`x = x * expression`
`x /= expression`	`x = x / expression`
`x %= expression`	`x = x % expression`
`x += expression`	`x = x + expression`
`x -= expression`	`x = x - expression`

Table 3-2 Expansion of Compound Expressions

PROGRAM 3-4 Demonstration of Compound Assignments

```
 1  /* Demonstrate examples of compound assignments.
 2        Written by:
 3        Date:
 4  */
 5  #include <stdio.h>
 6
 7  int main (void)
 8  {
 9  // Local Declarations
10     int x;
11     int y;
12
13  // Statements
14     x = 10;
15     y = 5;
16
```

PROGRAM 3-4 Demonstration of Compound Assignments

```
17      printf("x: %2d  |  y: %2d ",     x, y);
18      printf("  |  x *= y + 2: %2d ", x *= y + 2);
19      printf("  |  x is now: %2d\n",  x);
20
21      x = 10;
22      printf("x: %2d  |  y: %2d ",     x, y);
23      printf("  |  x /= y + 1: %2d ", x /= y + 1);
24      printf("  |  x is now: %2d\n",  x);
25
26      x = 10;
27      printf("x: %2d  |  y: %2d ",      x, y);
28      printf("  |  x %%= y - 3: %2d ", x %= y - 3);
29      printf("  |  x is now: %2d\n", x);
30
31      return 0;
32  }   // main
```

PROGRAM 3-4 Demonstration of Compound Assignments

```
Results:
x: 10  |  y:  5  |  x *= y + 2: 70  |  x is now: 70
x: 10  |  y:  5  |  x /= y + 1:  1  |  x is now:  1
x: 10  |  y:  5  |  x %= y - 3:  0  |  x is now:  0
```

3-2 Precedence and Associativity

Precedence is used to determine the order in which different operators in a complex expression are evaluated. Associativity is used to determine the order in which operators with the same precedence are evaluated in a complex expression.

Topics discussed in this section:

Precedence
Associativity

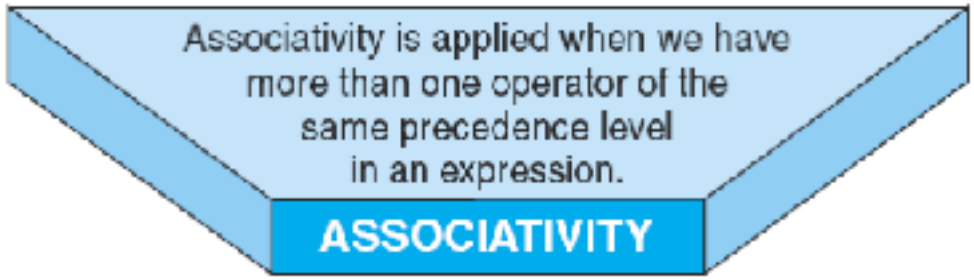
PROGRAM 3-5 Precedence

```
1  /* Examine the effect of precedence on an expression.
2        Written by:
3        Date:
4  */
5  #include <stdio.h>
6
7  int main (void)
8  {
9  // Local Declarations
10    int a = 10;
11    int b = 20;
12    int c = 30;
13
14 // Statements
15    printf ("a *  b + c  is: %d\n", a *  b + c);
16    printf ("a * (b + c) is: %d\n", a * (b + c));
17    return 0;
18 }  // main
```

PROGRAM 3-5 Precedence

```
Results:
a *  b + c  is: 230
a * (b + c) is: 500
```

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right


Associativity is applied when we have
more than one operator of the
same precedence level
in an expression.
ASSOCIATIVITY

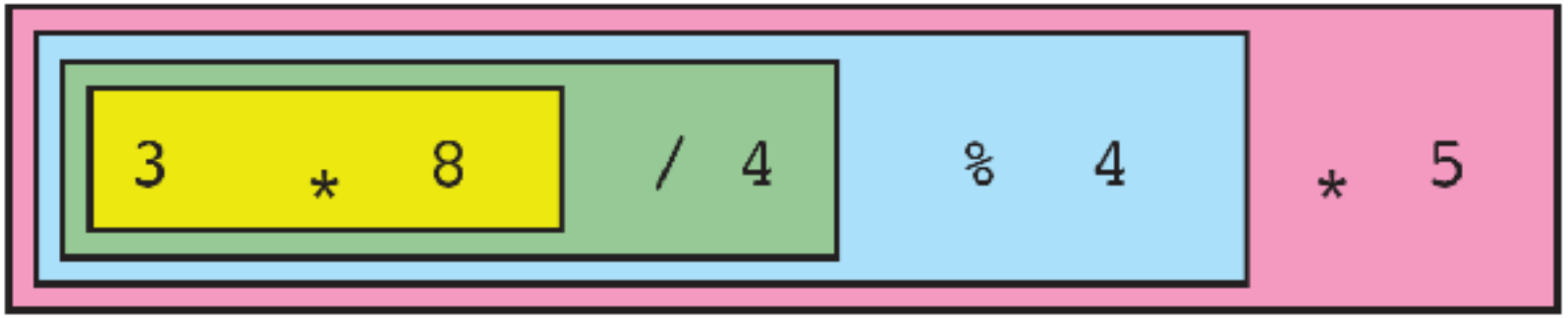


FIGURE 3-8 Left-to-Right Associativity

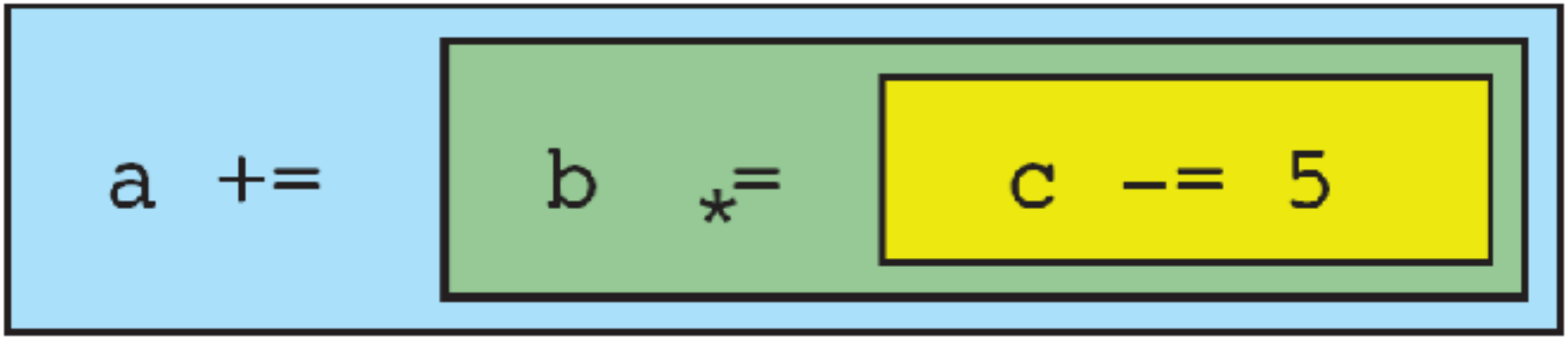


FIGURE 3-9 Right-to-Left Associativity

So you can have things like:

```
int i =1;
int j =2;

i = j*= 2; // i = j = 4 after

i += i + j+=2;  // j=6, i = 10
                // after
```

You can actually say

```
int i, sum;

i = sum = 3 +1; // Right to left
                // associativity
```

And i will equal 4, but it is not good programming practice.