Example 3.1

The program in Fig.3.1 shows the use of integer arithmetic to convert a given number of days into months and days.

PROGRAM TO CONVERT DAYS TO MONTHS AND DAYS

Program

```
main ()
{
    int   months, days ;

    printf("Enter days\n") ;
    scanf("%d", &days) ;

    months = days / 30 ;
    days   = days % 30 ;

    printf("Months = %d   Days = %d", months, days) ;
}
```

Output

```
Enter days
265
Months = 8   Days = 25

Enter days
364
Months = 12   Days = 4

Enter days
45
Months = 1   Days = 15
```

Fig. 3.1 *Illustration of integer arithmetic*

Example 3.2

Program of Fig.3.2 prints a sequence of squares of numbers. Note the use of the shorthand operator *= .

The program attempts to print a sequence of squares of numbers starting from 2. The statement

a *= a;

which is identical to

a = a*a;

replaces the current value of **a** by its square. When the value of **a** becomes equal or greater than **N** (=100) the **while** is terminated. Note that the output contains only three values 2, 4 and 16.

USE OF SHORTHAND OPERATORS

Program

```
#define    N   100
#define    A   2

main()
{
    int  a;

    a  =  A;
    while( a < N )
    {
        printf("%d\n", a);
        a *= a;
    }
}
```

Output

```
2
4
16
```

*Fig. 3.2 Use of shorthand operator *=*

Example 3.3

In Fig.3.3, the program employs different kinds of operators. The results of their evaluation are also shown for comparison.

Notice the way the increment operator **++** works when used in an expression. In the statement

c = ++a - b;

new value of **a** (= 16) is used thus giving the value 6 to c. That is, a is incremented by 1 before it is used in the expression. However, in the statement

d = b++ + a;

the old value of **b** (=10) is used in the expression. Here, b is incremented by 1 after it is used in the expression.

We can print the character % by placing it immediately after another % character in the control string. This is illustrated by the statement

printf("a%%b = %d\n", a%b);

The program also illustrates that the expression

c > d ? 1 : 0

assumes the value 0 when c is less than d and 1 when c is greater than d.

ILLUSTRATION OF OPERATORS

Program

```
main()
{
    int a, b, c, d;

    a = 15;
    b = 10;
    c = ++a - b;

    printf("a = %d  b = %d  c = %d\n",a, b, c);

    d = b++ +a;

     printf("a = %d  b = %d  d = %d\n",a, b, d);

     printf("a/b = %d\n", a/b);
     printf("a%%b = %d\n", a%b);
     printf("a *= b = %d\n", a*=b);
     printf("%d\n", (c>d) ? 1 : 0);
     printf("%d\n", (c<d) ? 1 : 0);
}
```

Output

```
a = 16  b = 10  c = 6
a = 16  b = 11  d = 26
a/b = 1
a%b = 5
a *= b = 176
0
1
```

Fig. 3.3 Further illustration of arithmetic operators

Example 3.4

The program in Fig.3.4 illustrates the use of variables in expressions and their evaluation.

Output of the program also illustrates the effect of presence of parentheses in expressions. This is discussed in the next section.

EVALUATION OF EXPRESSIONS

Program

```
main()
{
    float  a, b, c, x, y, z;

    a = 9;
    b = 12;
    c = 3;

    x = a - b / 3 + c * 2 - 1;
    y = a - b / (3 + c) * (2 - 1);
    z = a -(b / (3 + c) * 2) - 1;

    printf("x = %f\n", x);
    printf("y = %f\n", y);
    printf("z = %f\n", z);
}
```

Output

```
x = 10.000000
y = 7.000000
z = 4.000000
```

Fig.3.4 Illustrations of evaluation of expressions

Example 3.5

Output of the program in Fig.3.6 shows round-off errors that can occur in computation of floating point numbers.

PROGRAM SHOWING ROUND-OFF ERRORS

Program

```
/*------------------- Sum of n terms of 1/n -------------------*/
main()
{
    float  sum, n, term ;
    int    count = 1 ;
```

```
    sum = 0 ;
    printf("Enter value of n\n") ;
     scanf("%f", &n) ;
    term = 1.0/n ;
    while( count <= n )
    {
         sum = sum + term ;
         count++ ;
    }
    printf("Sum = %f\n", sum) ;
}
```

Output

```
Enter value of n
99
Sum = 1.000001
Enter value of n
143
Sum = 0.999999
```

Fig.3.6 Round-off errors in floating point computations

We know that the sum of n terms of 1/n is 1. However, due to errors in floating point representation, the result is not always 1.

Example 3.6

Figure 3.8 shows a program using a cast to evaluate the equation

$$sum = \sum_{i=1}^{n} (1/i)$$

PROGRAM SHOWING THE USE OF A CAST

Program

```
main()
{
    float  sum ;
    int    n ;

    sum = 0 ;

    for( n = 1 ; n <= 10 ; ++n )
    {
       sum = sum + 1/(float)n ;
       printf("%2d  %6.4f\n", n, sum) ;
    }
```

```
    }
```

Output

```
    1  1.0000
    2  1.5000
    3  1.8333
    4  2.0833
    5  2.2833
    6  2.4500
    7  2.5929
    8  2.7179
    9  2.8290
   10  2.9290
```

Fig. 3.8 Use of a cast