```
    printf("%d\n", delta);
    delta = alpha && beta && gamma++ - 2;
    printf("%d\n", delta);
    delta = alpha / beta >= beta / gamma;
    printf("%d\n", delta);
    delta == alpha / 2 == beta;
    printf("%d\n", delta);
    delta = ++delta || (delta = 5);
    printf("%d\n", delta);
    delta = 5 || ++delta;
    printf("%d\n", delta);
    delta == delta++;
    printf("%d\n", delta);
    return (0);
}
```

Listing 14: Boolean and arithmetic operators.

In the first assignment to delta:

```
delta = alpha > beta > gamma;
```

the assignment operator has the lowest precedence. Since > groups from left to right `alpha > beta` is evaluated first, and has the value **true**, which by rule (iii) is 1. The next expression to be computed is therefore:

```
1 > gamma
```

which evaluates to **false**, and has the value 0. `delta` gets the value 0.

Consider the second assignment of `delta`:

```
delta = alpha / beta == ++ gamma;
```

The pre-incremented value of `gamma` (2) is to be compared against the quotient `alpha / beta` (also 2). Note that the quotient is available before the comparison is done. The Boolean is **true**, so `delta` is 1.

In assignment:

```
delta = alpha || beta || gamma ++ - 2;
```

it is important to realise that a Boolean is evaluated only to the extent necessary to determine its truth value. Here `alpha` is non-zero and is therefore **true**, so the entire expression is **true**; `gamma` is not post-incremented; `delta` is 1. This deserves comparison with the next assignment to `delta`, where the **OR**s are replaced by **AND**s:

```
delta = alpha && beta && gamma ++ -2;
```

Post-incrementation of `gamma` implies that its last value (2) must be used in the evaluation of `delta`. `alpha` is 10 and `beta` is 5, so

```
alpha && beta
```

is **true**, but

```
gamma ++ - 2
```

is **false**, the entire expression is **false**, and `delta` is zero. Note that `gamma` is post-incremented when the expression for `delta` involves &&s; each sub-expression must be evaluated to determine the truth value of `delta`.

In the assignment:

```
delta = alpha / beta > = beta / gamma;
```

the quotients `alpha / beta` and `beta / gamma` are computed before the inequality is tested. With the current values of `alpha, beta` and `gamma`, the inequality is **true** and `delta` is 1.

The next statement:

```
delta == alpha / 2 == beta;
```

is not an assignment statement. No rvalue is modified by it, and this may cause a warning to be issued at compile time. `delta` is still 1.

Try the following exercises now.

E12) Calculate the value that `delta` gets in the remaining assignment statement, and verify you results by executing the program.

E13) Write a program that accepts three numbers and decides:

a) whether these can be the lengths of the sides of a triangle;

b) if they form an equilateral, isosceles or scalene triangle

c) if they form a right-angled triangle

E14) Write a program which accepts two numbers, and determines the following:

a) the first is positive, negative or zero

b) the second is positive, negative or zero

c) the first is even or odd

d) the second is even or odd

e) the first is larger than the second, in absolute value

f) if the first is exactly divisible by the second

Hint: Divisibility of the first by the second is determinable only if the second number is different from zero.

E15) Write a program to determine whether the value of a year input to it, such as 1900, or 1996, or 2000, represents a leap year or not. Your program should prompt the user to enter a value, and it should then output one of: “Leap year” or “Not a leap year”. Hint: A leap year value is evenly divisible by 4. However, a value that is divisible by 100 is not a leap year, unless it is also divisible by 400.

E16) Give the output of the following programs:

```
/* Program 4.15; file name:unit4-prog15.c */
#include <stdio.h>
int main()
{
    int x = 10, y = 5, z = 0;
    x %= y -= z == x < y;
    printf("x = %d, y = %d, z = %d\n", x, y, z);
    x *= y += z = x == (y /= 2);
    printf("x = %d, y = %d, z = %d\n", x, y, z);
    x = y-- && --z || (x + y - z);
    printf("x = %d, y = %d, z = %d\n", x, y, z);
    x = y++ < z++ > ++x / 12;
    x = --x <= -y > z || x && y;
    printf("x = %d, y = %d, z = %d\n", x, y, z);
    return (0);
}
```

E17) Write a program that accepts a digit from the keyboard, and displays it in English. So, if the user types 7, the program responds with “seven”, if she types 4, the program responds with “four”, etc.

E18) Write a program that gets a character such as +, -, * or /, then scans two numbers, and then yields the result of the operation denoted by the character. In the division of one value by another, your program should behave intelligently if the denominator == 0,

---

We close this unit here. We summarise the Unit in the next section.

4.6 SUMMARY

In this unit:

1. We introduced Boolean expressions which perform certain tests for taking decisions on the flow of control in the program. They are built up from Boolean operators like ||, && and comparison operators like ==.Boolean expressions are assigned the value 1 when they are true and 0 when they are false.
2. We discussed the **goto** statement. While this has to be avoided for the sake of clarity, it could be useful some times.
3. We discussed the **if ()** statement whose usage is as follows:

   if()

   if(*Expression*)
   statement or compound statement;

   If the value of the *Expression* is non-zero, the instructions in the statement or compound statement following it is executed; otherwise it is skipped. We can use any valid C expression here, not necessarily a Boolean expression.
4. We discussed an extension of **if ()**, the **if ()–else** whose usage is as follows:

   if()-else

   if(*Expression*)
   statement or compound statement;
   else
   statement or compound statement;

If the value of the expression is non-zero, the instructions in the statement or compound statement following it are executed. Otherwise, the instructions in the statement or compound statement after the **else** are executed. As in the case of **if ()** any valid C expression can be used, not necessarily a Boolean expression.

4.7 SOLUTIONS/ANSWERS

E1)
```
1
1
1 1
0 0
```

E2)
```
x > y
x !  = y
```

E3) No, because the unary operator ! has a higher precedence than the binary operator ==. Hence !5 will be calculated and the value is 0. So, the value of !x == y will be true and x != y will be printed. However, if y is different from 0, you will get entirely different results! Try it now! Change the value of y to 1, remove the brackets around x != y and see what you get!

E4) No, since && has lower priority than the < or >, the expression is evaluated we want.

E5) No, if the statement is not there, the program will divide input by 2 and increase the cycle length 1 also. So, the program will correctly perform the required operations and do the book keeping also correctly without the iterate.

E7)
```
/* Program4.4 :File unit4-ans-ex7.c*/
#include <stdio.h>
int main()
{
    int input, cycle_length = 0;
    begin_again:
    printf("Enter a positive trail number:");
    scanf("%d", &input);
    if (input <= 0)/* only a positive input permitted */
        goto begin_again;
    iterate:
    if (input == 1)
        goto finished;
    if (input % 2 == 1) /* input was odd */
        goto odd_input;
    input /= 2;
    cycle_length++;
    goto iterate;
    odd_input:input = input * 3 + 1;
    cycle_length++;
    goto iterate;
    finished:
    printf("1 appeared as the terminating digit \
after %d iterations", cycle_length);
    return (0);
}
```

E8)
```
/*File name: unit4-ans-ex8.c*/
#include<stdio.h>
int main()
{
    int first, second, third;
    printf("Enter the first number..\n");
    scanf("%d", &first);
    printf("Enter the second number..\n");
    scanf("%d", &second);
    printf("Enter the third number..\n");
    scanf("%d", &third);
    /*If all the numbers are equal
    print them*/
    if(first == second && second == third)
        goto caseb;
    if(first <= second)
        goto casea;
    /*Now, second < first;
    If third <= second,
    the order is first >= second >= third*/
    if(third<=second)
        printf("%d, %d, %d", first, second, third);
```

```
    /* If first <= third, the order is
    third>=first>=second*/
    if(first <= third)
        printf("%d, %d, %d", third, first,second);
    /*The only possibility now is
    second<=third<=first*/
    if(second <= third && third <= first)
    printf("%d, %d, %d", first, third, second);
    goto finished;
    caseb:
    /*All the numbers are equal; print the numbers
    and end the program*/
    printf("%d, %d, %d", first, second, third);
    goto finished;
    casea:
    /* Now, first <= second;If third <= first,
    the order is third, first, second*/
    if(third <= first)
        printf("%d, %d, %d", second, first, third);
    /*If second is less than third, the order is
    third, second, first*/
    if(second <= third)
        printf("%d, %d, %d", third, second,first);
    /*The only possiblity is
    first, third, second*/
    if(first <= third && third<=second)
        printf("%d, %d, %d",second,third,first);
    finished:
    return 0;
}
```

E9)
```
z = 4
x = 98, y = 99, z = 99
x = 2, y = 3, z = 4
```

E10) No, they are not required. Because == has higher priority than ||, the expression will be evaluated correctly without the parentheses. The parentheses improve the readability of the programme.

E11)
```
z = 7
z = 102, y = 99, Z = 101
x = 5, y = 5, z = 5
```

E12) The value of `delta` in all the statements except the last one is 1 because `delta` is assigned the result of **OR** operator applied to 2 non-zero quantities. In the statement `delta == delta++` no assignment is made to `delta` but it is post-incremented and gets the value 2.

E13)
```
#include <stdio.h>
int main()
{
    int a, b, c;
    printf("Enter Side 1 ..");
    scanf("%d",&a);
    printf("Enter Side 2 ..");
    scanf("%d",&b);
    printf("Enter Side 3 ..");
    scanf("%d",&c);
    if( (a+b <= c) || (a+c <= b) || (b+c <= a))
        {
```

```
            printf("%d, %d and %d do not form a triangle",a,b,c);
            goto finished;
        }
    else
        printf("%d, %d and %d do form a triangle.\n",a,b,c);
    if((a == b) || (b == c))
        {
         printf("%d, %d and %d form an \
isoceles triangle.\n",a,b,c);
         if( (a ==b )&& (b == c))
             printf("%d, %d and %d actually form an \
equilateral triangle.\n", a,b,c);
        }
    else
        printf("%d, %d and %d form a scalene \
triangle.\n",a,b,c);
    /*Apply Pythagoras theorem.*/
    if( (a*a + b*b == c*c) || (a*a + c*c == b*b)
        || (b*b + c*c == a*a))
        printf("%d, %d and %d form a right angled \
triangle.",a,b,c);
    else
        printf("%d, %d and %d do not form a right \
angled triangle.",a,b,c);
    finished:
    return (0);
}
```

E14) The following program checks which of the numbers are positive, which are negative and which are zero. Complete the program to carry out the remaining checks.

```
#include <stdio.h>
int main ()
{
    int a, b;
    printf("Enter the first number ...\n");
    scanf("%d",&a);
    printf("Enter the second number ...\n");
    scanf("%d",&b);
    if ( a*b == 0)
        {
         if (a == 0)
             {
              printf("First number is zero.\n");
              if (b > 0)
                  printf("Second number is positive");
              else
                  if(b < 0)
                      printf("Second number is negative\n");
              }
         else
             {
              printf("The second number is zero.\n");
              if (a > 0)
                  printf("The first number is positive.\n");
              else
                  printf("The first number is negative.\n");
             }
        }
    else
        {
         if( a*b > 0)
```

```
                {
                 if ( a > 0)
                     printf("Both the numbers are positive");
                 else
                     printf("Both the numbers are negative.\n");
                }
            else
            /* a*b < 0 is the only possibility now.*/
                {
                 if ( a < 0)
                     printf("First number is negative, \
    second number is positive.\n");
                 else
                     printf("First number is positive, \
    second number is negative.\n");
                }
          }
        return (0);
    }
```

E15)
```
    #include <stdio.h>
    int main()
    {
        int year;
        printf("Enter the year...\n");
        scanf("%d",&year);
        if ( year % 4 == 0)
            if (year % 100 == 0)
                if( year % 400 !=0)
                    printf("The year %d is not a leap year.",
    year);
                else
                    printf("The year %d is a leap year.",year);
            else
                printf("The year %d is a leap year.",year);
        else
            printf("The year %d is not a leap year.",year);
        return (0);
    }
```

E16) Output of program:

```
x = 2, y = 4, z = 0
x = 6, y = 3, z = 1
x = 1, y = 2, z = 0
x = 1, y = 3, z = 1
```

E17) We have given a program skeleton that reads out 0 and 1. Complete the rest of the program on your own.

```
#include <stdio.h>
int main()
{
    char number;
    printf("Enter a number from 0 to 9 ...\n");
    scanf("%c",&number);
    if (number == '0')
        printf("You entered zero.");
    if (number == '1')
        printf("You entered one.");
    /* Fill in the skeleton.*/
    return (0);
}
```

E18) We have given a program skeleton that performs division. Complete the rest of the program on your own.

```
#include <stdio.h>
int main()
{
    char operator;
    float a, b;
        printf("Type a simple expression like 3*4 to \
evaluate:\n");
        printf("Do not include spaces.\n");
        scanf("%f%c%f",&a,&operator,&b);
    if(operator == '/')
            {
                 if( b == 0)
                    printf("Division by 0 not possible.");
             else
                    printf("The answer is %f",a/b);
            }
    return (0);
}
```