2 The Decision Control Structure

- Decisions! Decisions!
- The *if* Statement
  - The Real Thing
  - Multiple Statements within *if*
- The *if-else* Statement
  - Nested *if-else*s
  - Forms of *if*
- Use of Logical Operators
  - The *else if* Clause
  - The ! Operator
  - Hierarchy of Operators Revisited
- A Word of Caution
- The Conditional Operators
- Summary
- Exercise

We all need to alter our actions in the face of changing circumstances. If the weather is fine, then I will go for a stroll. If the highway is busy I would take a diversion. If the pitch takes spin, we would win the match. If she says no, I would look elsewhere. If you like this book, I would write the next edition. You can notice that all these decisions depend on some condition being met.

C language too must be able to perform different sets of actions depending on the circumstances. In fact this is what makes it worth its salt. C has three major decision making instructions—the **if** statement, the **if-else** statement, and the **switch** statement. A fourth, somewhat less important structure is the one that uses conditional operators. In this chapter we will explore all these ways (except **switch**, which has a separate chapter devoted to it, later) in which a C program can react to changing circumstances.

Decisions! Decisions!

In the programs written in Chapter 1 we have used sequence control structure in which the various steps are executed sequentially, i.e. in the same order in which they appear in the program. In fact to execute the instructions sequentially, we don't have to do anything at all. By default the instructions in a program are executed sequentially. However, in serious programming situations, seldom do we want the instructions to be executed sequentially. Many a times, we want a set of instructions to be executed in one situation, and an entirely different set of instructions to be executed in another situation. This kind of situation is dealt in C programs using a decision control instruction. As mentioned earlier, a decision control instruction can be implemented in C using:

(a) The **if** statement
(b) The **if-else** statement
(c) The conditional operators

Now let us learn each of these and their variations in turn.

The *if* Statement

Like most languages, C uses the keyword **if** to implement the decision control instruction. The general form of **if** statement looks like this:

```
if ( this condition is true )
    execute this statement ;
```

The keyword **if** tells the compiler that what follows is a decision control instruction. The condition following the keyword **if** is always enclosed within a pair of parentheses. If the condition, whatever it is, is true, then the statement is executed. If the condition is not true then the statement is not executed; instead the program skips past it. But how do we express the condition itself in C? And how do we evaluate its truth or falsity? As a general rule, we express a condition using C's 'relational' operators. The relational operators allow us to compare two values to see whether they are equal to each other, unequal, or whether one is greater than the other. Here's how they look and how they are evaluated in C.

this expression	**is true if**
x == y	x is equal to y
x != y	x is not equal to y
x < y	x is less than y
x > y	x is greater than y
x <= y	x is less than or equal to y
x >= y	x is greater than or equal to y

Figure 2.1

The relational operators should be familiar to you except for the equality operator == and the inequality operator **!=**. Note that = is used for assignment, whereas, == is used for comparison of two quantities. Here is a simple program, which demonstrates the use of **if** and the relational operators.

```
/* Demonstration of if statement */
main( )
{
    int   num ;

    printf ( "Enter a number less than 10 " ) ;
    scanf ( "%d", &num ) ;

    if ( num <= 10 )
        printf ( "What an obedient servant you are !" ) ;
}
```

On execution of this program, if you type a number less than or equal to 10, you get a message on the screen through **printf( )**. If you type some other number the program doesn’t do anything. The following flowchart would help you understand the flow of control in the program.

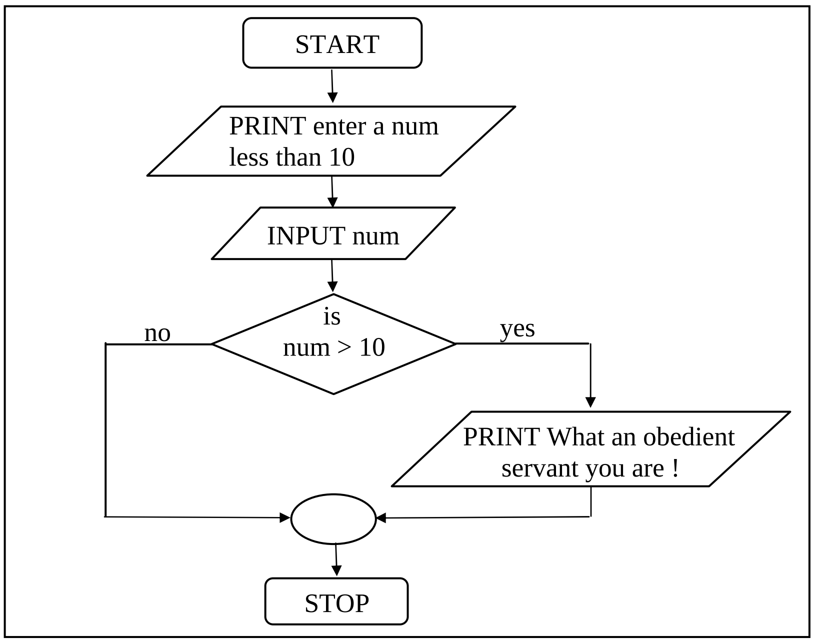
START
PRINT enter a num less than 10
INPUT num
is num > 10
no
yes
PRINT What an obedient servant you are !
STOP

Figure 2.2

To make you comfortable with the decision control instruction one more example has been given below. Study it carefully before reading further. To help you understand it easily, the program is accompanied by an appropriate flowchart.

Example 2.1: While purchasing certain items, a discount of 10% is offered if the quantity purchased is more than 1000. If quantity and price per item are input through the keyboard, write a program to calculate the total expenses.

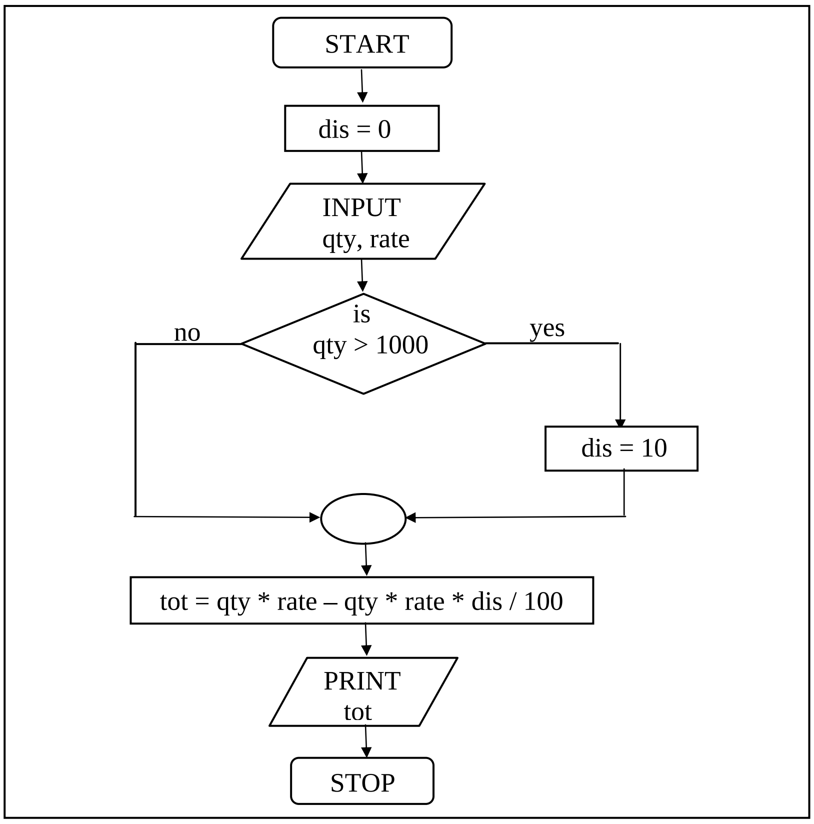

Figure 2.3

```
/* Calculation of total expenses */
main( )
{
    int   qty, dis = 0 ;
    float   rate, tot ;
    printf ( "Enter quantity and rate " ) ;
    scanf ( "%d %f", &qty, &rate) ;

    if ( qty > 1000 )
        dis = 10 ;
    tot = ( qty * rate ) - ( qty * rate * dis / 100 ) ;
    printf ( "Total expenses = Rs. %f", tot ) ;
}
```

Here is some sample interaction with the program.

```
Enter quantity and rate 1200 15.50
Total expenses = Rs. 16740.000000

Enter quantity and rate 200 15.50
Total expenses = Rs. 3100.000000
```

In the first run of the program, the condition evaluates to true, as 1200 (value of **qty**) is greater than 1000. Therefore, the variable **dis**, which was earlier set to 0, now gets a new value 10. Using this new value total expenses are calculated and printed.

In the second run the condition evaluates to false, as 200 (the value of **qty**) isn't greater than 1000. Thus, **dis**, which is earlier set to 0, remains 0, and hence the expression after the minus sign evaluates to zero, thereby offering no discount.

Is the statement **dis = 0** necessary? The answer is yes, since in C, a variable if not specifically initialized contains some unpredictable value (garbage value).

The Real Thing

We mentioned earlier that the general form of the if statement is as follows

```
if ( condition )
    statement ;
```

Truly speaking the general form is as follows:

```
if ( expression )
    statement ;
```

Here the expression can be any valid expression including a relational expression. We can even use arithmetic expressions in the **if** statement. For example all the following **if** statements are valid

```
if ( 3 + 2 % 5 )
    printf ( "This works" ) ;

if ( a = 10 )
    printf ( "Even this works" )  ;

if ( -5 )
    printf ( "Surprisingly even this works" ) ;
```

Note that in C a non-zero value is considered to be true, whereas a 0 is considered to be false. In the first **if**, the expression evaluates to **5** and since **5** is non-zero it is considered to be true. Hence the **printf( )** gets executed.

In the second **if**, 10 gets assigned to **a** so the **if** is now reduced to **if (a)** or **if (10)**. Since 10 is non-zero, it is true hence again **printf( )** goes to work.

In the third **if**, -5 is a non-zero number, hence true. So again **printf( )** goes to work. In place of -5 even if a float like 3.14 were used it would be considered to be true. So the issue is not whether the number is integer or float, or whether it is positive or negative. Issue is whether it is zero or non-zero.

Multiple Statements within *if*

It may so happen that in a program we want more than one statement to be executed if the expression following **if** is satisfied. If such multiple statements are to be executed then they must be placed within a pair of braces as illustrated in the following example.

Example 2.2: The current year and the year in which the employee joined the organization are entered through the keyboard. If the number of years for which the employee has served the organization is greater than 3 then a bonus of Rs. 2500/- is given to the employee. If the years of service are not greater than 3, then the program should do nothing.

```
/* Calculation of bonus */
main( )
{
    int   bonus, cy, yoj, yr_of_ser ;

    printf ( "Enter current year and year of joining " ) ;
    scanf ( "%d %d", &cy, &yoj ) ;

    yr_of_ser = cy - yoj ;

    if ( yr_of_ser > 3 )
    {
        bonus = 2500 ;
        printf ( "Bonus = Rs. %d", bonus ) ;
    }
}
```

Observe that here the two statements to be executed on satisfaction of the condition have been enclosed within a pair of braces. If a pair of braces is not used then the C compiler assumes that the programmer wants only the immediately next statement after the **if** to be executed on satisfaction of the condition. In other words we can say that the default scope of the **if** statement is the immediately next statement after it.

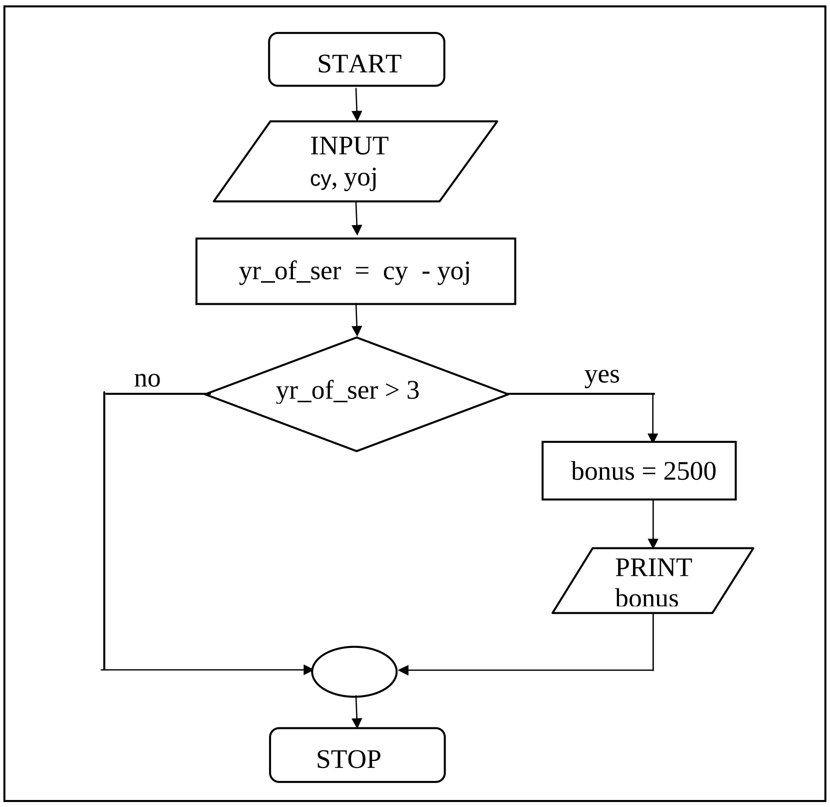

Figure 2.4

The *if-else* Statement

The **if** statement by itself will execute a single statement, or a group of statements, when the expression following **if** evaluates to true. It does nothing when the expression evaluates to false. Can we execute one group of statements if the expression evaluates to true and another group of statements if the expression evaluates to false? Of course! This is what is the purpose of the **else** statement that is demonstrated in the following example:

Example 2.3: In a company an employee is paid as under:

If his basic salary is less than Rs. 1500, then HRA = 10% of basic salary and DA = 90% of basic salary. If his salary is either equal to or above Rs. 1500, then HRA = Rs. 500 and DA = 98% of basic salary. If the employee's salary is input through the keyboard write a program to find his gross salary.

```
/* Calculation of gross salary */
main( )
{
	float   bs, gs, da, hra ;

	printf ( "Enter basic salary " ) ;
	scanf ( "%f", &bs ) ;

	if ( bs < 1500 )
	{
		hra = bs * 10 / 100 ;
		da = bs * 90 / 100 ;
	}
	else
	{
		hra = 500 ;
		da = bs * 98 / 100 ;
	}

	gs = bs + hra + da ;
	printf ( "gross salary = Rs. %f", gs ) ;
}
```

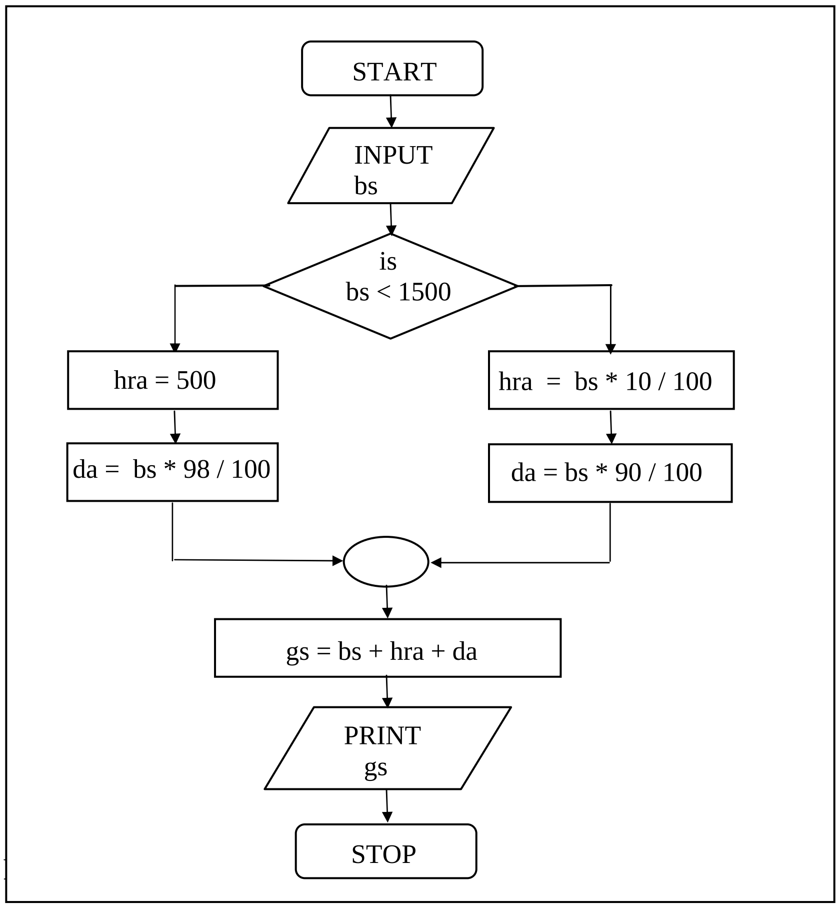

Figure 2.5

A few points worth noting...

(a) The group of statements after the **if** upto and not including the **else** is called an ‘if block’. Similarly, the statements after the **else** form the ‘else block’.

(b) Notice that the **else** is written exactly below the **if**. The statements in the if block and those in the else block have been indented to the right. This formatting convention is followed throughout the book to enable you to understand the working of the program better.

(c) Had there been only one statement to be executed in the if block and only one statement in the else block we could have dropped the pair of braces.

(d) As with the **if** statement, the default scope of **else** is also the statement immediately after the **else**. To override this default scope a pair of braces as shown in the above example must be used.

Nested *if-elses*

It is perfectly all right if we write an entire **if-else** construct within either the body of the **if** statement or the body of an **else** statement. This is called ‘nesting’of **if**s. This is shown in the following program.

```
/* A quick demo of nested if-else */
main( )
{
    int   i ;

    printf ( "Enter either 1 or 2 " ) ;
    scanf ( "%d", &i ) ;

    if ( i == 1 )
        printf ( "You would go to heaven !" ) ;
    else
    {
        if ( i == 2 )
            printf ( "Hell was created with you in mind" ) ;
        else
            printf ( "How about mother earth !" ) ;
    }
}
```

Note that the second **if-else** construct is nested in the first **else** statement. If the condition in the first **if** statement is false, then the condition in the second **if** statement is checked. If it is false as well, then the final **else** statement is executed.

You can see in the program how each time a **if-else** construct is nested within another **if-else** construct, it is also indented to add clarity to the program. Inculcate this habit of indentation, otherwise you would end up writing programs which nobody (you included) can understand easily at a later date.

In the above program an **if-else** occurs within the **else** block of the first **if** statement. Similarly, in some other program an **if-else** may occur in the **if** block as well. There is no limit on how deeply the **if**s and the **else**s can be nested.

Forms of *if*

The **if** statement can take any of the following forms:

```
(a)   if ( condition )
          do this ;

(b)   if ( condition )
      {
          do this ;
          and this ;
      }

(c)   if ( condition )
          do this ;
      else
          do this ;
(d)   if ( condition )
      {
          do this ;
      and this ;
}
else
{
      do this ;
      and this ;
}
```

(e)
```
if ( condition )
      do this ;
else
{
      if ( condition )
            do this ;
      else
      {
            do this ;
            and this ;
      }
}
```

(f)
```
if ( condition )
{
      if ( condition )
            do this ;
      else
      {
            do this ;
            and this ;
      }
}
else
      do this ;
```

Use of Logical Operators

C allows usage of three logical operators, namely, &&, || and !. These are to be read as 'AND' 'OR' and 'NOT' respectively.

There are several things to note about these logical operators. Most obviously, two of them are composed of double symbols: **||** and **&&**. Don't use the single symbol **|** and **&**. These single symbols also have a meaning. They are bitwise operators, which we would examine in Chapter 14.

The first two operators, **&&** and **||**, allow two or more conditions to be combined in an **if** statement. Let us see how they are used in a program. Consider the following example.

Example 2.4: The marks obtained by a student in 5 different subjects are input through the keyboard. The student gets a division as per the following rules:

Percentage above or equal to 60 - First division
Percentage between 50 and 59 - Second division
Percentage between 40 and 49 - Third division
Percentage less than 40 - Fail

Write a program to calculate the division obtained by the student.

There are two ways in which we can write a program for this example. These methods are given below.

```
/* Method – I */
main( )
{
    int   m1, m2, m3, m4, m5, per ;

    printf ( "Enter marks in five subjects " ) ;
    scanf ( "%d %d %d %d %d", &m1, &m2, &m3, &m4, &m5 ) ;
    per = ( m1 + m2 + m3 + m4 + m5 ) / 5 ;
    if ( per >= 60 )
        printf ( "First division ") ;
    else
    {
        if ( per >= 50 )
            printf ( "Second division" ) ;
        else
        {
            if ( per >= 40 )
                printf ( "Third division" ) ;
            else
                printf ( "Fail" ) ;
        }
    }
}
```

This is a straight forward program. Observe that the program uses nested **if-else**s. This leads to three disadvantages:

(a) As the number of conditions go on increasing the level of indentation also goes on increasing. As a result the whole program creeps to the right.
(b) Care needs to be exercised to match the corresponding **if**s and **else**s.
(c) Care needs to be exercised to match the corresponding pair of braces.

All these three problems can be eliminated by usage of ‘Logical operators’. The following program illustrates this.

```
/* Method – II  */
main( )
{
    int  m1, m2, m3, m4, m5, per ;

    printf ( "Enter marks in five subjects " ) ;
    scanf ( "%d %d %d %d %d", &m1, &m2, &m3, &m4, &m5 ) ;
    per = ( m1 + m2 + m3 + m4 + m5 ) / 5 ;
    if ( per >= 60 )
        printf ( "First division" ) ;

    if ( ( per >= 50 ) && ( per < 60 ) )
        printf ( "Second division" ) ;

    if ( ( per >= 40 ) && ( per < 50 ) )
        printf ( "Third division" ) ;

    if ( per < 40 )
        printf ( "Fail" ) ;
}
```

As can be seen from the second **if** statement, the **&&** operator is used to combine two conditions. ‘Second division’ gets printed if both the conditions evaluate to true. If one of the conditions evaluate to false then the whole thing is treated as false.

Two distinct advantages can be cited in favour of this program:

(a) The matching (or do I say mismatching) of the **if**s with their corresponding **else**s gets avoided, since there are no **else**s in this program.

(b) In spite of using several conditions, the program doesn't creep to the right. In the previous program the statements went on creeping to the right. This effect becomes more pronounced as the number of conditions go on increasing. This would make the task of matching the **if**s with their corresponding **else**s and matching of opening and closing braces that much more difficult.

The *else if* Clause

There is one more way in which we can write program for Example 2.4. This involves usage of **else if** blocks as shown below:

```
/* else if ladder demo */
main( )
{
	int   m1, m2, m3, m4, m5, per ;

	per = ( m1+ m2 + m3 +  m4+ m5 ) / per ;

	if ( per >= 60 )
		printf ( "First division" ) ;
	else if ( per >= 50 )
		printf ( "Second division" ) ;
	else if ( per >= 40 )
		printf ( "Third division" ) ;
	else
		printf ( "fail" ) ;
}
```

You can note that this program reduces the indentation of the statements. In this case every **else** is associated with its previous **if**. The last **else** goes to work only if all the conditions fail. Even in **else if** ladder the last **else** is optional.

Note that the **else if** clause is nothing different. It is just a way of rearranging the **else** with the **if** that follows it. This would be evident if you look at the following code:

```
if ( i == 2 )
	printf (  "With you…" ) ;
else
{
	if ( j == 2 )
	printf ( "…All the time" ) ;
}
```

```
if ( i == 2 )
	printf (  "With you…" ) ;
else if ( j == 2 )
	printf ( "…All the time " ) ;
```

Another place where logical operators are useful is when we want to write programs for complicated logics that ultimately boil down to only two answers. For example, consider the following example:

Example 2.5: A company insures its drivers in the following cases:

– If the driver is married.
– If the driver is unmarried, male & above 30 years of age.
– If the driver is unmarried, female & above 25 years of age.

In all other cases the driver is not insured. If the marital status, sex and age of the driver are the inputs, write a program to determine whether the driver is to be insured or not.

Here after checking a complicated set of instructions the final output of the program would be one of the two—Either the driver should be ensured or the driver should not be ensured. As mentioned above, since these are the only two outcomes this problem can be solved using logical operators. But before we do that let us write a program that does not make use of logical operators.

```
/* Insurance of driver - without using logical operators */
main( )
{
    char   sex, ms ;
    int   age ;

    printf ( "Enter age, sex, marital status " ) ;
    scanf ( "%d %c %c", &age, &sex, &ms ) ;

    if ( ms == 'M' )
        printf ( "Driver is insured" ) ;
    else
    {
        if ( sex == 'M' )
        {
            if ( age > 30 )
                printf ( "Driver is insured" ) ;
            else
                printf ( "Driver is not insured" ) ;
        }
        else
        {
            if ( age > 25 )
                printf ( "Driver is insured" ) ;
            else
                printf ( "Driver is not insured" ) ;
        }
    }
}
```

From the program it is evident that we are required to match several **if**s and **else**s and several pairs of braces. In a more real-life situation there would be more conditions to check leading to the program creeping to the right. Let us now see how to avoid these problems by using logical operators.

As mentioned above, in this example we expect the answer to be either ‘Driver is insured’ or ‘Driver is not insured’. If we list down all those cases in which the driver is insured, then they would be:

(a) Driver is married.
(b) Driver is an unmarried male above 30 years of age.
(c) Driver is an unmarried female above 25 years of age.

Since all these cases lead to the driver being insured, they can be combined together using **&&** and **||** as shown in the program below:

```
/* Insurance of driver - using logical operators */
main( )
{
    char   sex, ms ;
    int  age ;

    printf ( "Enter age, sex, marital status " ) ;
    scanf ( "%d %c %c" &age, &sex, &ms ) ;

    if ( ( ms == 'M') || ( ms == 'U' && sex == 'M' && age > 30 ) ||
                 ( ms == 'U' && sex == 'F' && age > 25 ) )
        printf ( "Driver is insured" ) ;
    else
        printf ( "Driver is not insured" ) ;
}
```

In this program it is important to note that:

– The driver will be insured only if one of the conditions enclosed in parentheses evaluates to true.

– For the second pair of parentheses to evaluate to true, each condition in the parentheses separated by **&&** must evaluate to true.

– Even if one of the conditions in the second parentheses evaluates to false, then the whole of the second parentheses evaluates to false.

– The last two of the above arguments apply to third pair of parentheses as well.

Thus we can conclude that the **&&** and **||** are useful in the following programming situations:

(a) When it is to be tested whether a value falls within a particular range or not.
(b) When after testing several conditions the outcome is only one of the two answers (This problem is often called yes/no problem).

There can be one more situation other than checking ranges or yes/no problem where you might find logical operators useful. The following program demonstrates it.

Example 2.6: Write a program to calculate the salary as per the following table:

Gender	Years of Service	Qualifications	Salary
Male	>= 10	Post-Graduate	15000
	>= 10	Graduate	10000
	< 10	Post-Graduate	10000
	< 10	Graduate	7000
Female	>= 10	Post-Graduate	12000
	>= 10	Graduate	9000
	< 10	Post-Graduate	10000
	< 10	Graduate	6000

Figure 2.6

```
main( )
{
    char   g ;
    int   yos, qual, sal ;

    printf ( "Enter Gender, Years of Service and
                Qualifications ( 0 = G, 1 = PG ):" ) ;
    scanf ( "%c%d%d", &g, &yos, &qual ) ;

    if ( g == 'm' && yos >= 10 && qual == 1 )
        sal = 15000 ;
    else if ( ( g == 'm' && yos >= 10 && qual == 0 ) ||
        ( g == 'm' && yos < 10 && qual == 1 ) )
        sal = 10000 ;
    else if ( g == 'm' && yos < 10 && qual == 0 )
        sal = 7000 ;
    else if ( g == 'f' && yos >= 10 && qual == 1 )
        sal = 12000 ;
    else if ( g == 'f' && yos >= 10 && qual == 0 )
        sal = 9000 ;
    else if ( g == 'f' && yos < 10 && qual == 1 )
        sal = 10000 ;
    else if ( g == 'f' && yos < 10 && qual == 0 )
        sal = 6000 ;

    printf ( "\nSalary of Employee = %d", sal ) ;
}
```

The ! Operator

So far we have used only the logical operators **&&** and ||. The third logical operator is the NOT operator, written as **!**. This operator reverses the result of the expression it operates on. For example, if the expression evaluates to a non-zero value, then applying **!** operator to it results into a 0. Vice versa, if the expression evaluates to zero then on applying **!** operator to it makes it 1, a non-zero value. The final result (after applying **!**) 0 or 1 is considered to be false or true respectively. Here is an example of the NOT operator applied to a relational expression.

```
! ( y < 10 )
```

This means “not **y** less than 10”. In other words, if **y** is less than 10, the expression will be false, since **(y < 10)** is true. We can express the same condition as **(y >= 10)**.

The NOT operator is often used to reverse the logical value of a single variable, as in the expression

```
if ( ! flag )
```

This is another way of saying

```
if ( flag == 0 )
```

Does the NOT operator sound confusing? Avoid it if you want, as the same thing can be achieved without using the NOT operator.

Hierarchy of Operators Revisited

Since we have now added the logical operators to the list of operators we know, it is time to review these operators and their priorities. Figure 2.7 summarizes the operators we have seen so far. The higher the position of an operator is in the table, higher is its priority. (A full-fledged precedence table of operators is given in Appendix A.)

Operators	**Type**
!	Logical NOT
* / %	Arithmetic and modulus
+ -	Arithmetic
< > <= >=	Relational
== !=	Relational
&&	Logical AND
||	Logical OR
=	Assignment

Figure 2.7

A Word of Caution

What will be the output of the following program:

```
main( )
{
    int  i ;

    printf ( "Enter value of i " ) ;
    scanf ( "%d", &i ) ;
    if ( i = 5 )
        printf ( "You entered 5" ) ;
    else
        printf ( "You entered something other than 5" ) ;
}
```

And here is the output of two runs of this program...

```
Enter value of i 200
You entered 5
Enter value of i 9999
You entered 5
```

Surprising? You have entered 200 and 9999, and still you find in either case the output is ‘You entered 5’. This is because we have written the condition wrongly. We have used the assignment operator = instead of the relational operator ==. As a result, the condition gets reduced to **if (5)**, irrespective of what you supply as the value of **i**. And remember that in C ‘truth’ is always non-zero, whereas ‘falsity’ is always zero. Therefore, **if (5)** always evaluates to true and hence the result.

Another common mistake while using the **if** statement is to write a semicolon (**;**) after the condition, as shown below:

```
main( )
{
    int  i ;

    printf ( "Enter value of i " ) ;
    scanf ( "%d", &i ) ;
    if ( i == 5 ) ;
        printf ( "You entered 5" ) ;
}
```

The ; makes the compiler to interpret the statement as if you have written it in following manner:

```
if ( i == 5 )
    ;
printf ( "You entered 5" ) ;
```

Here, if the condition evaluates to true the ; (null statement, which does nothing on execution) gets executed, following which the **printf( )** gets executed. If the condition fails then straightaway the **printf( )** gets executed. Thus, irrespective of whether the condition evaluates to true or false the **printf( )** is bound to get executed. Remember that the compiler would not point out this as an error, since as far as the syntax is concerned nothing has gone wrong, but the logic has certainly gone awry. Moral is, beware of such pitfalls.

The following figure summarizes the working of all the three logical operators.

Operands		**Results**			
x	**y**	**!x**	**!y**	**x && y**	**x || y**
0	0	1	1	0	0
0	non-zero	1	0	0	0
non-zero	0	0	1	0	1
non-zero	non-zero	0	0	1	1

Figure 2.8

The Conditional Operators

The conditional operators **?** and **:** are sometimes called ternary operators since they take three arguments. In fact, they form a kind of foreshortened if-then-else. Their general form is,

```
expression 1 ? expression 2 : expression 3
```

What this expression says is: "if **expression 1** is true (that is, if its value is non-zero), then the value returned will be **expression 2**, otherwise the value returned will be **expression 3**". Let us understand this with the help of a few examples:

(a)
```
int  x, y ;
scanf ( "%d", &x ) ;
y = ( x > 5 ? 3 : 4 ) ;
```

This statement will store 3 in **y** if **x** is greater than 5, otherwise it will store 4 in y.

The equivalent **if** statement will be,

```
if ( x > 5 )
    y = 3 ;
else
    y = 4 ;
```

(b)
```
char  a ;
int  y ;
scanf ( "%c", &a ) ;
y = ( a >= 65 && a <= 90 ? 1 : 0 ) ;
```

Here 1 would be assigned to **y** if **a >=65 && a <=90** evaluates to true, otherwise 0 would be assigned.

The following points may be noted about the conditional operators:

(a) It's not necessary that the conditional operators should be used only in arithmetic statements. This is illustrated in the following examples:

```
Ex.:    int   i ;
        scanf ( "%d", &i ) ;
        ( i == 1 ? printf ( "Amit" ) : printf ( "All and sundry" ) ) ;

Ex.:    char   a = 'z' ;
        printf ( "%c" , ( a >= 'a' ? a : '!' ) ) ;
```

(b) The conditional operators can be nested as shown below.

```
int   big, a, b, c ;
big = ( a > b ? ( a > c ? 3: 4 ) : ( b > c ? 6: 8 ) ) ;
```

(c) Check out the following conditional expression:

```
a > b ? g = a : g = b ;
```

This will give you an error 'Lvalue Required'. The error can be overcome by enclosing the statement in the **:** part within a pair of parenthesis. This is shown below:

```
a > b ? g = a : ( g = b ) ;
```

In absence of parentheses the compiler believes that **b** is being assigned to the result of the expression to the left of second =. Hence it reports an error.

The limitation of the conditional operators is that after the **?** or after the **:** only one C statement can occur. In practice rarely is this the requirement. Therefore, in serious C programming conditional operators aren't as frequently used as the **if-else**.

Summary

(a) There are three ways for taking decisions in a program. First way is to use the **if-else** statement, second way is to use the

conditional operators and third way is to use the **switch** statement.
- (b) The default scope of the **if** statement is only the next statement. So, to execute more than one statement they must be written in a pair of braces.
- (c) An **if** block need not always be associated with an **else** block. However, an **else** block is always associated with an **if** statement.
- (d) If the outcome of an **if-else** ladder is only one of two answers then the ladder should be replaced either with an **else-if** clause or by logical operators.
- (e) **&&** and **||** are binary operators, whereas, **!** is a unary operator.
- (f) In C every test expression is evaluated in terms of zero and non-zero values. A zero value is considered to be false and a non-zero value is considered to be true.
- (g) Assignment statements used with conditional operators must be enclosed within a pair of parenthesis.

Exercise

if, if-else, Nested *if-elses*

[A] What would be the output of the following programs:

(a)
```
main( )
{
    int   a = 300, b, c ;
    if ( a >= 400 )
         b = 300 ;
         c = 200 ;
         printf ( "\n%d %d", b, c ) ;
}
```

(b)
```
main( )
{
    int   a = 500, b, c ;
    if ( a >= 400 )
```

```
            b = 300 ;
            c = 200 ;
            printf ( "\n%d %d", b, c ) ;
    }
```

(c)
```
    main( )
    {
        int   x = 10, y = 20 ;
        if ( x == y ) ;
            printf ( "\n%d %d", x, y ) ;
    }
```

(d)
```
    main( )
    {
        int   x = 3, y = 5 ;
        if ( x == 3 )
            printf ( "\n%d", x ) ;
        else ;
            printf ( "\n%d", y ) ;
    }
```

(e)
```
    main( )
    {
        int   x = 3 ;
        float   y = 3.0 ;

        if ( x == y )
            printf ( "\nx and y are equal" ) ;
        else
            printf ( "\nx and y are not equal" ) ;
    }
```

(f)
```
    main( )
    {
        int   x = 3, y, z ;
        y = x = 10 ;
        z = x < 10 ;
        printf ( "\nx = %d y = %d z = %d", x, y, z ) ;
    }
```

(g)
```
main( )
{
    int  k = 35 ;
    printf ( "\n%d %d %d", k == 35, k = 50, k > 40 ) ;
}
```

(h)
```
main( )
{
    int i = 65 ;
    char j = 'A' ;
    if ( i == j )
        printf ( "C is WOW" ) ;
    else
        printf( "C is a headache" ) ;
}
```

(i)
```
main( )
{
    int   a = 5, b, c ;
    b = a = 15 ;
    c = a < 15 ;
    printf ( "\na = %d b = %d c = %d", a, b, c ) ;
}
```

(j)
```
main( )
{
    int  x = 15 ;
    printf ( "\n%d %d %d", x != 15, x = 20, x < 30 ) ;
}
```

[B] Point out the errors, if any, in the following programs:

(a)
```
main( )
{
    float   a = 12.25, b = 12.52 ;
    if ( a = b )
        printf ( "\na and b are equal" ) ;
}
```

(b)
```
main( )
{
    int   j = 10, k = 12 ;
    if ( k >= j )
    {
        {
            k = j ;
            j = k ;
        }
    }
}
```

(c)
```
main( )
{
    if ( 'X' < 'x' )
        printf ( "\nascii value of X is smaller than that of x" ) ;
}
```

(d)
```
main( )
{
    int  x = 10 ;
    if ( x >= 2 ) then
        printf ( "\n%d", x ) ;
}
```

(e)
```
main( )
{
    int  x = 10 ;
    if x >= 2
        printf ( "\n%d", x ) ;
}
```

(f)
```
main( )
{
    int  x = 10, y = 15 ;
    if ( x % 2 = y % 3 )
```

```
            printf ( "\nCarpathians" ) ;
    }
```

(g)
```
    main( )
    {
        int x = 30 , y = 40 ;
        if ( x == y )
            printf( "x is equal to y" ) ;
        elseif ( x > y )
            printf( "x is greater than y" ) ;
        elseif ( x < y )
            printf( "x is less than y" ) ;
    }
```

(h)
```
    main( )
    {
        int  x = 10 ;
        if ( x >= 2 ) then
            printf ( "\n%d", x ) ;
    }
```

(i)
```
    main( )
    {
        int a, b ;
        scanf ( "%d %d",a, b ) ;
        if ( a > b ) ;
            printf ( "This is a game" ) ;
        else
            printf ( "You have to play it" ) ;
    }
```

[C] Attempt the following:

(a) If cost price and selling price of an item is input through the keyboard, write a program to determine whether the seller has made profit or incurred loss. Also determine how much profit he made or loss he incurred.

(b) Any integer is input through the keyboard. Write a program to find out whether it is an odd number or even number.

(c) Any year is input through the keyboard. Write a program to determine whether the year is a leap year or not.

(Hint: Use the % (modulus) operator)

(d) According to the Gregorian calendar, it was Monday on the date 01/01/1900. If any year is input through the keyboard write a program to find out what is the day on 1st January of this year.

(e) A five-digit number is entered through the keyboard. Write a program to obtain the reversed number and to determine whether the original and reversed numbers are equal or not.

(f) If the ages of Ram, Shyam and Ajay are input through the keyboard, write a program to determine the youngest of the three.

(g) Write a program to check whether a triangle is valid or not, when the three angles of the triangle are entered through the keyboard. A triangle is valid if the sum of all the three angles is equal to 180 degrees.

(h) Find the absolute value of a number entered through the keyboard.

(i) Given the length and breadth of a rectangle, write a program to find whether the area of the rectangle is greater than its perimeter. For example, the area of the rectangle with length = 5 and breadth = 4 is greater than its perimeter.

(j) Given three points **(x1, y1)**, **(x2, y2)** and **(x3, y3)**, write a program to check if all the three points fall on one straight line.

(k) Given the coordinates **(x, y)** of a center of a circle and it's radius, write a program which will determine whether a point lies inside the circle, on the circle or outside the circle.

(Hint: Use **sqrt( )** and **pow( )** functions)

(l) Given a point **(x, y)**, write a program to find out if it lies on the x-axis, y-axis or at the origin, viz. (0, 0).

Logical Operators

If a = 10, b = 12, c = 0, find the values of the expressions in the following table:

Expression	**Value**
a != 6 && b > 5	1
a == 9 || b < 3	
! (a < 10)	
! (a > 5 && c)	
5 && c != 8 || !c	

[D] What would be the output of the following programs:

(a)
```
main( )
{
    int  i = 4, z = 12 ;
    if ( i = 5 || z > 50 )
         printf ( "\nDean of students affairs" ) ;
    else
         printf ( "\nDosa" ) ;
}
```

(b)
```
main( )
{
    int  i = 4, z = 12 ;
    if ( i = 5 && z > 5 )
        printf ( "\nLet us C" ) ;
    else
        printf ( "\nWish C was free !" ) ;
}
```

(c)
```
main( )
{
    int   i = 4, j = -1, k = 0, w, x, y, z ;
    w = i || j || k ;
    x = i && j && k ;
    y = i || j && k ;
    z = i && j || k ;
    printf ( "\nw = %d x = %d y = %d z = %d", w, x, y, z ) ;
}
```

(d)
```
main( )
{
    int   i = 4, j = -1, k = 0, y, z ;
    y = i + 5 && j + 1 || k + 2 ;
    z = i + 5 || j + 1 && k + 2 ;
    printf ( "\ny = %d z = %d", y, z ) ;
}
```

(e)
```
main( )
{
    int   i = -3, j = 3 ;
    if ( !i + !j * 1 )
        printf ( "\nMassaro" ) ;
    else
        printf ( "\nBennarivo" ) ;
}
```

(f)
```
main( )
{
    int  a = 40 ;
    if ( a > 40 && a < 45 )
        printf ( "a is greater than 40 and less than 45" ) ;
        else
              printf ( "%d", a ) ;
   }
```

(g)
```
   main( )
   {
        int   p = 8, q = 20 ;
        if ( p == 5 && q > 5 )
              printf ( "\nWhy not C" ) ;
        else
              printf ( "\nDefinitely C !" ) ;
   }
```

(h)
```
   main( )
   {
        int i = -1, j = 1, k ,l ;
        k = i && j ;
        l = i || j ;
        printf ( "%d %d", I, j ) ;
   }
```

(i)
```
   main( )
   {
        int x = 20 , y = 40 , z = 45 ;
        if ( x > y && x > z )
              printf( "x is big" ) ;
        else if ( y > x && y > z )
              printf( "y is big" ) ;
        else if ( z > x && z > y )
              printf( "z is big" ) ;
   }
```

(j)
```
   main( )
   {
        int i = -1, j = 1, k ,l ;
        k = !i && j ;
        l = !i  ||  j ;
        printf ( "%d %d", i, j ) ;
```

```
}
```

(k)
```
main( )
{
    int   j = 4, k ;
    k = !5 && j ;
    printf ( "\nk = %d", k ) ;
}
```

[E] Point out the errors, if any, in the following programs:

(a)
```
/* This program
/* is an example of
/* using Logical operators */
main( )
{
    int   i = 2, j = 5 ;
    if ( i == 2 && j == 5 )
        printf ( "\nSatisfied at last" ) ;
}
```

(b)
```
main( )
{
    int   code, flag ;
    if ( code == 1 & flag == 0 )
        printf ( "\nThe eagle has landed" ) ;
}
```

(c)
```
main( )
{
    char   spy = 'a', password = 'z' ;
    if ( spy == 'a' or password == 'z' )
        printf ( "\nAll the birds are safe in the nest" ) ;
}
```

(d)
```
main( )
{
    int   i = 10, j = 20 ;
    if ( i = 5 ) && if ( j = 10 )
        printf ( "\nHave a nice day" ) ;
}
```

(a)
```
main( )
{
    int  x = 10 , y = 20;
    if ( x >= 2 and y <=50 )
        printf ( "\n%d", x ) ;
}
```

(b)
```
main( )
{
    int   a, b ;
    if ( a == 1 & b == 0 )
        printf ( "\nGod is Great" ) ;
}
```

(c)
```
main( )
{
    int x = 2;
    if ( x == 2 && x != 0 ) ;
    {
        printf ( "\nHi" ) ;
        printf( "\nHello" ) ;
    }
    else
        printf( "Bye" ) ;
}
```

(d)
```
main( )
{
    int   i = 10, j = 10 ;
    if ( i && j == 10)
        printf ( "\nHave a nice day" ) ;
 }
```

[F] Attempt the following:

(a) Any year is entered through the keyboard, write a program to determine whether the year is leap or not. Use the logical operators **&&** and **||**.

(b) Any character is entered through the keyboard, write a program to determine whether the character entered is a capital letter, a small case letter, a digit or a special symbol.

The following table shows the range of ASCII values for various characters.

Characters	**ASCII Values**
A – Z	65 – 90
a – z	97 – 122
0 – 9	48 – 57
special symbols	0 - 47, 58 - 64, 91 - 96, 123 - 127

(c) An Insurance company follows following rules to calculate premium.

(1) If a person's health is excellent and the person is between 25 and 35 years of age and lives in a city and is a male then the premium is Rs. 4 per thousand and his policy amount cannot exceed Rs. 2 lakhs.

(2) If a person satisfies all the above conditions except that the sex is female then the premium is Rs. 3 per thousand and her policy amount cannot exceed Rs. 1 lakh.

(3) If a person's health is poor and the person is between 25 and 35 years of age and lives in a village and is a male

then the premium is Rs. 6 per thousand and his policy cannot exceed Rs. 10,000.

(4) In all other cases the person is not insured.

Write a program to output whether the person should be insured or not, his/her premium rate and maximum amount for which he/she can be insured.

(d) A certain grade of steel is graded according to the following conditions:

(i) Hardness must be greater than 50
(ii) Carbon content must be less than 0.7
(iii) Tensile strength must be greater than 5600

The grades are as follows:

Grade is 10 if all three conditions are met
Grade is 9 if conditions (i) and (ii) are met
Grade is 8 if conditions (ii) and (iii) are met
Grade is 7 if conditions (i) and (iii) are met
Grade is 6 if only one condition is met
Grade is 5 if none of the conditions are met

Write a program, which will require the user to give values of hardness, carbon content and tensile strength of the steel under consideration and output the grade of the steel.

(e) A library charges a fine for every book returned late. For first 5 days the fine is 50 paise, for 6-10 days fine is one rupee and above 10 days fine is 5 rupees. If you return the book after 30 days your membership will be cancelled. Write a program to accept the number of days the member is late to return the book and display the fine or the appropriate message.

(f) If the three sides of a triangle are entered through the keyboard, write a program to check whether the triangle is valid or not. The triangle is valid if the sum of two sides is greater than the largest of the three sides.

(g) If the three sides of a triangle are entered through the keyboard, write a program to check whether the triangle is isosceles, equilateral, scalene or right angled triangle.

(h) In a company, worker efficiency is determined on the basis of the time required for a worker to complete a particular job. If the time taken by the worker is between 2 – 3 hours, then the worker is said to be highly efficient. If the time required by the worker is between 3 – 4 hours, then the worker is ordered to improve speed. If the time taken is between 4 – 5 hours, the worker is given training to improve his speed, and if the time taken by the worker is more than 5 hours, then the worker has to leave the company. If the time taken by the worker is input through the keyboard, find the efficiency of the worker.

(i) A university has the following rules for a student to qualify for a degree with A as the main subject and B as the subsidiary subject:
   (a) He should get 55 percent or more in A and 45 percent or more in B.
   (b) If he gets than 55 percent in A he should get 55 percent or more in B. However, he should get at least 45 percent in A.
   (c) If he gets less than 45 percent in B and 65 percent or more in A he is allowed to reappear in an examination in B to qualify.
   (d) In all other cases he is declared to have failed.

   Write a program to receive marks in A and B and Output whether the student has passed, failed or is allowed to reappear in B.

(j) The policy followed by a company to process customer orders is given by the following rules:

(a) If a customer order is less than or equal to that in stock and has credit is OK, supply has requirement.

(b) If has credit is not OK do not supply. Send him intimation.

(c) If has credit is Ok but the item in stock is less than has order, supply what is in stock. Intimate to him data the balance will be shipped.

Write a C program to implement the company policy.

Conditional operators

[G] What would be the output of the following programs:

(a)
```
main( )
{
    int   i = -4, j, num ;
    j = ( num < 0 ? 0 : num * num ) ;
    printf ( "\n%d", j ) ;
}
```

(b)
```
main( )
{
    int   k, num = 30 ;
    k = ( num > 5 ? ( num <= 10 ? 100 : 200 ) : 500 ) ;
    printf ( "\n%d", num ) ;
}
```

(c)
```
main( )
{
    int   j = 4 ;
    ( !j != 1 ? printf ( "\nWelcome") : printf ( "\nGood Bye") ) ;
}
```

[H] Point out the errors, if any, in the following programs:

(a)
```
main( )
{
    int   tag = 0, code = 1 ;
    if ( tag == 0 )
        ( code > 1 ? printf ( "\nHello" ) ? printf ( "\nHi" ) ) ;
    else
        printf ( "\nHello Hi !!" ) ;
}
```

(b)
```
main( )
{
    int   ji = 65 ;
    printf ( "\nji >= 65 ? %d : %c", ji ) ;
}
```

(c)
```
main( )
{
    int   i = 10, j ;
    i >= 5 ? ( j = 10 ) : ( j = 15 ) ;
    printf ( "\n%d %d", i, j ) ;
}
```

(d)
```
main( )
{
    int a = 5 , b = 6 ;
    ( a == b ? printf( "%d",a) ) ;
}
```

(e)
```
main( )
{
    int n = 9 ;
    ( n == 9 ? printf( "You are correct" ) ; : printf( "You are wrong" ) ;) ;
}
```

(f)
```
main( )
{
    int  kk = 65 ,ll ;
    ll = ( kk == 65 : printf ( "\n kk is equal to 65" ) : printf ( "\n kk is not
equal to 65" ) ) ;
    printf( "%d", ll ) ;
}
```

(g)
```
main( )
{
    int  x = 10, y = 20 ;
    x == 20 && y != 10 ? printf( "True" ) : printf( "False" ) ;
}
```

[I] Rewrite the following programs using conditional operators.

(a)
```
main( )
{
    int  x, min, max ;
    scanf ( "\n%d %d", &max, &x ) ;
    if ( x > max )
        max = x ;
    else
        min = x ;
}
```

(b)
```
main( )
{
    int  code ;
    scanf ( "%d", &code ) ;
    if ( code > 1 )
        printf ( "\nJerusalem" ) ;
    else
        if ( code < 1 )
            printf ( "\nEddie" ) ;
        else
            printf ( "\nC Brain" ) ;
}
```

(c)
```
main( )
{
    float sal ;
    printf ("Enter the salary" ) ;
    scanf ( "%f", &sal ) ;
    if ( sal < 40000 && sal > 25000 )
        printf ( "Manager" ) ;
    else
        if ( sal < 25000 && sal > 15000 )
            printf ( "Accountant" ) ;
        else
            printf ( "Clerk" ) ;
}
```

[J] Attempt the following:

(a) Using conditional operators determine:

(1) Whether the character entered through the keyboard is a lower case alphabet or not.

(2) Whether a character entered through the keyboard is a special symbol or not.

(b) Write a program using conditional operators to determine whether a year entered through the keyboard is a leap year or not.

(c) Write a program to find the greatest of the three numbers entered through the keyboard using conditional operators.