Chapter 11

DECISION CONSTRUCTS

11.1 OVERVIEW

While writing C programs, the programmer may need to choose a path to execute one or more statements through the program based on a certain criterion; Decision constructs provide a way to meet this requirement. This chapter introduces various selection structures available in C, which can be used in such situations.

11.1.1 Control Structures

Control structures are statements used to control the flow of execution in a program or function. The C control structures enable us to group individual instructions into a single logical unit with one entry point and one exit point.

Program instructions can be organized into three kinds of control structures to control execution flow i.e. *sequence, selection and repetition*. All programs, whether simple or complex, use these control structures to implement the program logic.

Until now we have been using only sequential flow, which is also called default flow. In case of *sequence* structure, instructions are executed in the same order in which they are specified in the program. A *compound statement* refers to a group of statements enclosed in opening and closing braces such as:

```
{
        statement1 ;
        statement2 ;
            .
            .
            .
        statementn ;
}
```

Control flows from $statement_1$ to $statement_n$ in a logical sequence. Here we shall discuss the *selection structure* in detail, and in the next chapter the *repetition structure* will be discussed. Solutions of some problems require steps with two or more alternative course of action. A *selection structure* chooses which statement or a block of statements is to execute. In C, there are two basic selection statements:

- *if* – else
- *switch*

There are some variations of *if – else* structure which will be discussed here. The third control structure is *repetition, which* is also called iteration or loop. It is a control structure, which repeats a statement or a group of statements in a program. C provides different types of loop structures to be used in various situations. These include *for* loop, *while* loop, and *do-while* loop.

11.2 IF STATEMENT

if is one of the keywords in C language. It is used to select a path flow in a program based on a condition. A *condition* is an expression that either evaluates to true (usually represented by 1) or false (represented by 0). The result of this evaluation can be assigned to a variable. For example, consider the following program:

```
#include <stdio.h>
void main ( )
{
      int age, status;

      printf("Enter the age :");
      scanf("%d", &age);

      status = (age > 60);
      printf("Status = %d", status);
}
```

The value of the variable status will be *1* if the age is greater 60, otherwise status will be *0*. Here's the *output* of the program:

Case 1 (When age is less than 60)	**Case 2** (When age is greater than 60)
Enter the age :45	Enter the age :78
Status = 0	Status = 1

11.2.1 Simple *if* Statement

if statement is the simplest form of decision constructs. It allows a statement or a set of statements to be executed conditionally. The general form of simple *if* statement is:

if (*condition*)
{
    $statement_1$;
    $statement_2$;
        .
        .
        .
    $statement_n$;
}

The statement(s) in the block of *if statement* are executed if the condition is *true*; otherwise these are skipped. If there are more statements in *if* block then these should be enclosed in braces as a compound statement. However, in case of a single statement the braces are optional.

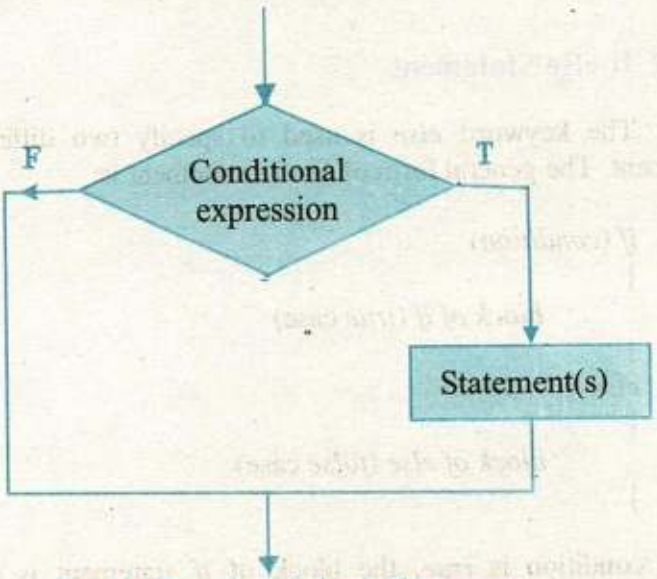

Fig. 11.1 Flow chart of simple *if* statement

This flow chart describes a situation where a simple *if* statement can be used.

Example 1 **A program that calculates the square root of a given number**

```
#include <stdio.h>
#include <math.h>

void main(void)
{
     double x = 0.0, square_root = 0.0;
     printf("Enter a number >");
     scanf("%lf", &x);

     if (x > 0)
          square_root = sqrt(x);
}
```

As the square root of negative numbers is imaginary, therefore the program finds the square root of positive numbers only. In this program we have used

math library. This includes the definition of sqrt and many other mathematical functions. We shall discuss it later in this book.

A **flow chart** is the pictorial representation of a program.

11.2.2 If-else Statement

The keyword *else* is used to specify two different choices with *if* statement. The general form of *if-else* statement is:

```
if (condition)
{
        block of if (true case)
}
else
{
        block of else (false case)
}
```

If the condition is *true*, the block of *if* statement is executed and if the condition is *false*, the block of *else* statement is executed. The following flow chart may help to explain the idea.

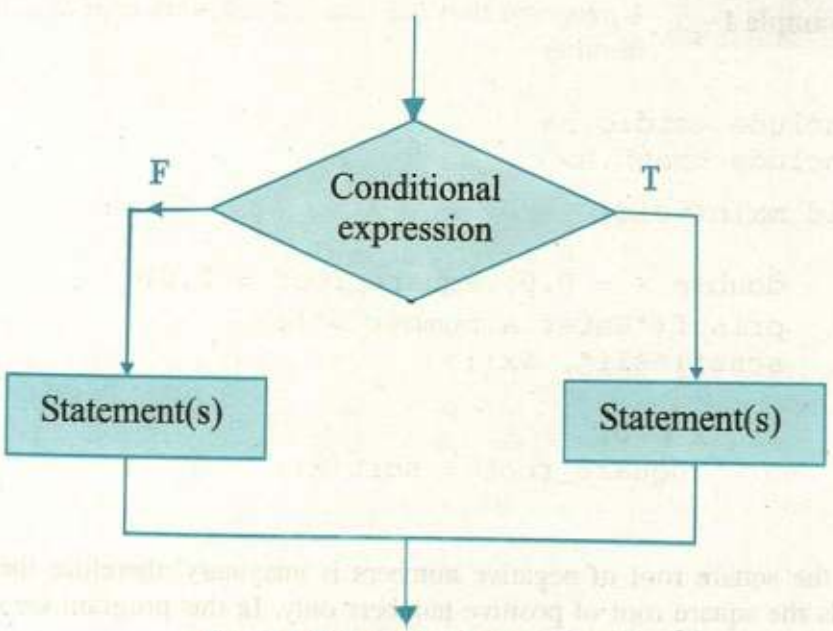

Fig. 11.2 Flow chart of *if-else* statement

The program in example 1 can be written using *if-else* statement as follows:

```
#include <stdio.h>
#include <math.h>
    void main(void)
    {
        double x = 0.0, square_root = 0.0;

        printf("Enter a number >");
        scanf("%lf", &x);

        if (x >= 0)
        {
            square_root = sqrt(x);
            printf("Square root of %lf = %lf", x,
            square_root);
        }
        else
            printf("Square root can not be
            calculated");
    }
```

There are two blocks of statements in this program, either of which is conditionally executed. If the condition evaluates to *true* the square root of **x** will be calculated and the output will be shown on the screen and in case the condition evaluates to *false*, the message "Square root can't be calculated" will be displayed.

It is very important to note that the block of *if* is enclosed in braces, and no bracket has been used in the body of *else* statement. The reason is that we want to execute multiple statements in case of *true* condition, so these must be represented as a compound statement. If we omit the braces from the body of *if* statement, the following error message will appear:

Illegal `else` without matching `if`

So, to avoid this message, one should always enclose a compound statement in braces. If there is a compound statement in the body of *else*, omitting braces will not cause the above-mentioned error, rather only one statement after the else will be treated as the body of *else* and the rest of the statements will always be executed sequentially.

11.2.3 Nested if Statement

Nested if statement means an *if* statement inside another *if* statement. Nesting can be done up to any level. The programmer may use as many *if* statements inside another *if* statement as (s)he wants. However, the increase in the level of nesting also increases the complexity of the *nested if* statement. The general form of *nested if* statement is as follows:

if (*condition*)

```
{
        if (condition)
        {
                block of inner if
        }
}
```

block of outer **if**

The *else* statement is optional, it may or may not be used with *outer* or *inner* **if** statement. However, if used, its block can also contain other if statements.

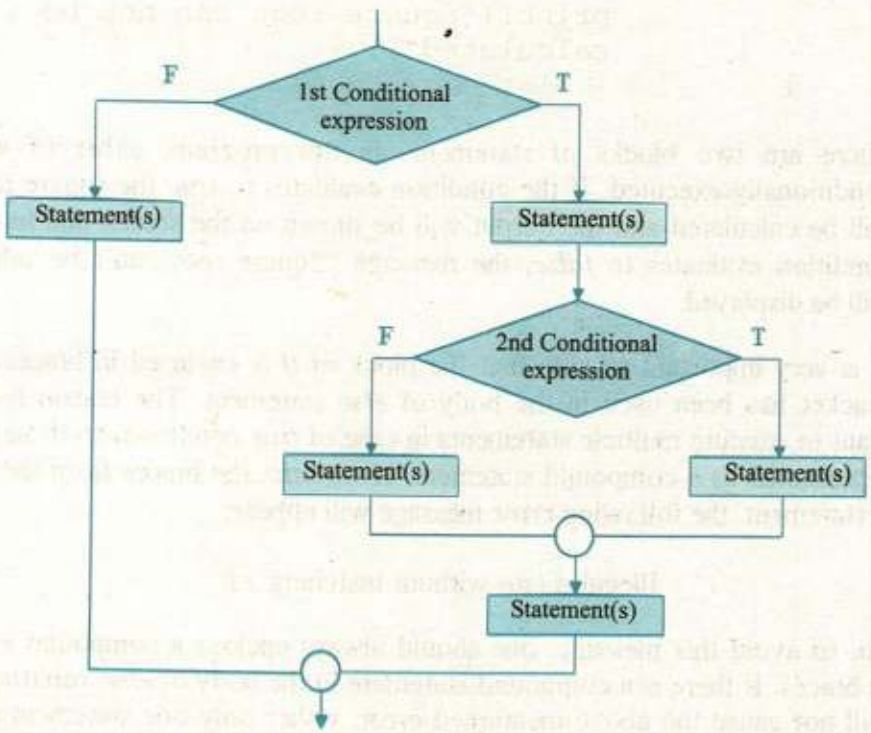

Fig. 11.3 Flow chart of *nested if* statement

Example 2 **A program that accepts three numbers from the user and displays the largest number.**

```
#include <stdio.h>

void main(void)
{
   int a, b, c;

   printf("Enter three numbers >");
   scanf("%d %d %d", &a, &b, &c);

   if (a > b)
   {
        if(a > c)
             printf("%d is largest", a);
        else
             printf("%d is largest ", c);

   }
   else
   {
    if (b > c)
         printf("%d is largest", b);
    else
         printf("%d is largest", c);

   }
}
```

This is a simple program that compares three numbers to find the largest one. The body of *if* and the body of *else*, both contains other if statements (nested *if*s are boxed). This sort of arrangement of multiple *if* statements is called *nested if*.

The execution of the nested *if* proceeds as follows: The first condition (a > b) is tested, if it is *true* then the second condition (a > c) is tested, if it is also *true*, the rest of the conditions will be skipped and we conclude that *a* is the

largest number. If the second condition is *false*, this means *a* is greater than *b* but smaller than **c**. Therefore **c** is the largest number.

If the first condition (a > b) is *false*, the body of *if* is skipped and the flow of control is transferred to the body of *else* where the condition (b > c) is tested. If it is *true* then the second condition (b > c) is tested, if it is also *true*, this means *b* is greater than *a* (from the first condition) and *b* is greater than *c* (from the second condition), therefore we conclude that *b* is the largest number. If the second condition in the body of *else* is *false*, it means *b* is greater than *a* (from the first condition), but *b* is smaller than *c*, therefore we conclude that **c** is the largest number.

In this way, we can implement decision making in the program using nested *if* statement.

11.2.4 Comparison of Nested if and Sequence of ifs

It is due to the complexity of *nested if* statement that beginners usually prefer to use a sequence of *if* statements rather than a single nested *if* statement. However, sometimes it is useful to use a *nested if* instead of sequence of ifs such as in example 2. In case of *nested if* statement, when the control flow reaches a logical decision, the rest of the conditions are skipped. Whereas in a sequence of *if* statements, all conditions are tested in any case. This can be understood by the following example.

Example 3 **A program that inputs a number from the user and determines whether it is positive, negative or zero.**

```
#include <stdio.h>

void main(void)
{
        int num;

        printf("Enter a number >");
        scanf("%d", &num);

        if (num > 0)
                printf("The number is positive");
        if (num < 0)
                printf("The number is negative");
        if (num == 0)
                printf("The number is zero");
```

This program implements the solution using a sequence of *if* statements. Suppose, the user enters a positive number, the answer is decided in the first comparison i.e. the number is positive. So there is no need to check the number for its being negative or zero as it is impossible. But, this solution suggests doing the last two comparisons unnecessarily, wasting the CPU time. This situation may be avoided by using some other form of *if* statement such as *if-else* statement.

11.2.5 if-else if Statement

Nested if statements can become quite complex, if there are more than three alternatives and indentation is not consistent, it may be difficult to determine the logical structure of the *if* statement. In such situations, *if* statement with multiple alternatives (if-else if) can be a good option. The general form of *if-else if* statement is as follows:

```
if (condition1)
        statement1;
else if (condition2)
        statement2;
                .
                .
                .
else if (conditionn)
        statementn;
else
        statementk;
```

The test conditions in *if statement with multiple alternatives* are executed in sequence until a *true* condition is reached. If a condition is true, the statement(s) following it is executed, and the rest of the alternatives are skipped. If a condition is false, the statement following it is skipped and the next condition is tested. If all conditions are falls, then $statement_k$ following the last *else* is executed. The following flowchart shows the execution flow of the program through *if-else if* statement.

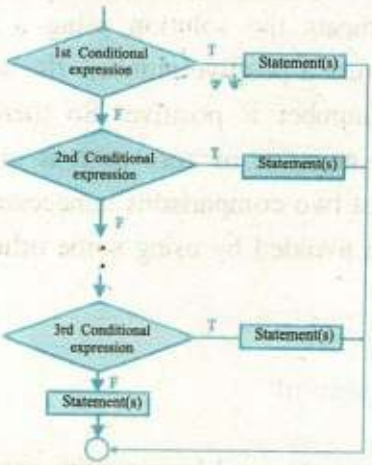

Fig. 11.4 Flowchart of *if-else if* statement

The program in *example 3* can be re-written using *if-else if* structure as follows:

```
#include <stdio.h>
void main(void)
{
     int num;
     printf("Enter a number >");
     scanf("%d", &num);
     if (num > 0)
          printf("The number is positive");
     else if (num < 0)
          printf("The number is negative");
     else
          printf("The number is zero");
}
```

It can be seen that the *else-if* construct has greatly simplified the program, while preserving the efficiency as well. If a positive number is entered, we will reach the answer in the first comparison and rest of the conditions will be skipped.

11.3 USE OF LOGICAL OPERATORS

We have studied the three logical operators (AND, OR, and NOT represented by &&, ||, and ! respectively) in chapter 2. These operators play an important role in constructing certain compound conditions to be used with *if* statement. Until now, we have been using simple conditions with *if* statement and its variations. In this section, we shall observe how complex program logic can be simplified using logical operators.

To grasp the concept, we re-write the program in *example* 2 using logical AND operator. This will demonstrate how much the task is simplified.

```
#include <stdio.h>
void main (void)
{
        int a, b, c;
        printf("Enter three numbers >");
        scanf("%d %d %d", &a, &b, &c);
        if ( a > b && a > c)
             printf("%d is largest", a);
        else if( b > a && b > c)
             printf("%d is largest", b);
        else
             printf("%d is largest", c);
}
```

There may be situations where the use of logical operators may simplify the program logic. We just need to concentrate the underlying problem. We shall see more examples of using logical operators in next chapters.

11.4 SWITCH STATEMENT

The *switch* statement can also be used in C to select one of many alternatives. Like *if* statement, it is also a conditional statement that compares the value of an expression against a list of cases. The case label can be an integral or character constant. Similarly, the value of the expression must be an integer or a character; it must not be a double or float value. Here's the syntax of switch statement.

```
switch(expression)
{
        case val1:
              statements1;
              break;
        case val2:
              statements2;
              break;
                .
                .
                .
        case valn:
              statementsn;
              break;
```

```
        default:
                statements_k;
}
```

The value of expression is compared to each *case* label. The statements following the first label with a matching evaluated value are executed until a *break* statement is encountered. The break causes the rest of the switch statement to be skipped. If all *break* statements are omitted from the switch statement, the code from the first true *case* down to the end of the switch statement will execute sequentially. A *default* label may be used to provide code to be executed if none of the *case* label is matched. However the position of *default* label is not fixed. It may be placed before the first *case* statement or after the last *case*. The *switch* statement is especially useful when the selection is based on the value.

Example 4 **Write a program that inputs a character from the user and checks whether it is a vowel or a consonant.**

```
#include <stdio.h>
void main(void)
{
        char ch;
        printf("Enter an alphabet");
        ch = getche();
        switch(ch)
        {
                case 'a':
                case 'A':
                        printf("It's a vowel");
                        break;
                case 'e':
                case 'E':
                        printf("It's a vowel");
                        break;
                case 'i':
                case 'I':
                        printf("It's a vowel");
                        break;
                case 'o':
                case 'O':
```

```
                printf("It's a vowel");
                break;
            case 'u':
            case 'U':
                printf("It's a vowel");
                break;
            default:
                printf("It is not a vowel");
                break;
        }
    }
```

In this program, we have used two *case* labels, one after the other, without having any other statement between them. This sort of arrangement of *case* labels works equivalent to the logical OR operator i.e., whether the value of the variable *ch* is 'a' or 'A', the code following the labels *case* '*a* and *case* '*A*' will execute.

- The value of *expression* in *switch* statement must be of type *int* or *char*, but not of type *float* or *double*.
- If the value of *expression* in *switch* statement is of type *float* or *double*, the compiler will generate the following error message:

Switch selection expression must be of integral type

11.5 CONDITIONAL OPERATOR

Conditional operator is used as an alternative to the if-else statement. It is a ternary operator (requires three operand). Its general form is:

conditional expression? *true-case statement* : *false-case statement*;

The *expression* may be a relational or logical expression. If the expression is true then the true-case statement will be executed otherwise the false-case statement is executed. e.g.,

```
#include <stdio.h>
#include <conio.h>
void main()
{
    int a, b;
    printf("Please enter two numbers: ");
    scanf("%d %d", &a, &b);
    a > b ? printf("%d is larger", a) : printf("%d is
    larger", b);
}
```

11.6 CASE STUDY: Locating a point in the coordinate plane

PROBLEM

Write a program that input x- and y-coordinates of a point in the coordinate plane. Based on these values, it then determines where it lies, on x- or y-axis, or in any of the four quadrants.

SOLUTION

The first step towards the solution of any problem (simple or complex) is to understand it. Think on the problem from all aspects; identify input and output requirements and different types of restrictions on the data. After analyzing the problem, try to develop a simple solution. Don't indulge yourself in unnecessary details. You can trace the solution on the paper in the form of an *algorithm. An algorithm is a step-by-step procedure to solve any problem in finite number of steps.* Let's work out the problem described above:

It is clear from the figure 11.1 that each point will lay on either of the two axes or in any of the four quadrants.

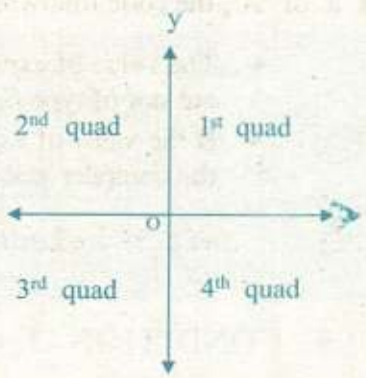

Fig. 11.1 -Co-ordinate plane

Input Values: The program will input the values of x-coordinate and y-coordinate. These values will be of type *int*.

Output: The program will output a message describing the position of the point in the coordinate plane.

What do we know?

- If both x- and y-coordinates are zero, the point will be at the origin.
- If x-coordinate is zero and y-coordinate is non-zero, then the point will be on y-axis.
- If y-coordinate is zero and x-coordinate is non-zero, then the point will be on x-axis.
- If both x- and y-coordinates are positive (> 0), the point will lie in the 1st quadrant.
- If x-coordinate is negative (< 0) and y-coordinate is positive (> 0), the point will lie in the 2nd quadrant.
- If both x- and y-coordinates are negative (< 0), the point will lie in the 3rd quadrant.
- If x-coordinate is positive (> 0) and y-coordinate is negative (< 0), the point will lie in the 4th quadrant.

Now, keeping in view all this information, let's try to write a C program to solve this problem:

Program:

```
#include <stdio.h>
void main(void)
{
     int x, y;
     printf("Enter value for x- and y-coordinates
>");
     scanf("%d %d", &x, &y);
     if(x  == 0)
     {
          if(y  == 0)
               printf("The   point   lie   on   the
origin");
          else
               printf("The point lie on y-axis");
     }
     else if(x > 0)
     {
          if(y  == 0)
               printf("The point lie on x-axis");
          else if(y > 0)
               printf("The   point   lies   in   1st
               quadrant");
          else
               printf("The   point   lies   in   4th
               quadrant");
     }
     else
     {
          if(y  == 0)
               printf("The point lies on x-axis");
          else if(y > 0)
               printf("The   point   lies   in   2nd
               quadrant");
          else
               printf("The   point   lies   in   3rd
               quadrant");
     }
}
```

Exercise 11c

1. Fill in the blanks:

   (i) _____ control the flow of execution in a program.
   (ii) _____ is a group statements enclosed in braces.
   (iii) *if* statement is used to select a pathflow in a program based on a _____.
   (iv) A _____ is a pictorial representation of a program.
   (v) _____ refers to a structure with one if statement inside another if statement.
   (vi) _____ statement can be used as an alternative to if-else if statement.
   (vii) In switch statement the value of expression must be _____ or _____.
   (viii) The _____statement switches the control outside the block in which it is used.
   (ix) The purpose of _____ label in switch-case statement is the same as that of *else* in *if-else* statement.
   (x) A condition is an expression that is either _____ or _____.

2. Write T for true and F for false statement.

   (i) An arithmetic expression can not be used as condition in **if** statement.
   (ii) The conditional operator is a ternary operator.
   (iii) Logical operators are used to compare values.
   (iv) A switch statement can not be used within the block of an *if* statement.
   (v) The break statement stops the execution of a program for a moment and then resumes.
   (vi) In sequence structure, statements are executed in the same order in which they appear in the program.
   (vii) A false condition always evaluates to zero.
   (viii) Use of the statement terminator at the end of an *if* statement causes a syntax error.
   (ix) The switch expression can not be of float or double type.
   (x) C is an unstructured programming language.

3. What is a *control structure*? Briefly describe the basic control structures for writing programs.

4. How many selection statements are available in C? Discuss the difference between them.

5. Write the general form of the following statements:
   (i) *if* statement with one alternative
   (ii) *if* statement with two alternatives
   (iii) *if* statement with multiple alternatives
   (iv) *switch* statement

6. Rewrite the program given in the example 4 using *if* statement.

7. Attempt the following parts:
   (i) Assuming x is 10.0 and y is 15.0, what are the values of the following conditions:
      a) x != y
      b) x < x
      c) x >= y – x
      d) x == y + x – y

   [*Hint*: The value of a true condition is **1** and a false condition is *0*]

   (ii) Write an expression to test each of the following relationships.
      a) *age* is from 18 to 25.
      b) *temperature* is less than 40.0 and greater than 25.0.
      c) *year* is divisible by 4 (Hint: use %).
      d) *speed* is not greater than 80.
      e) *y* is greater than x and less than z.
      f) *w* is either equal to 6 or not greater than 3.

   Note: The bold face words represent variables' names.

   (iii) Write assignment statement for the following:
      a) Assigns a value of 1(one) to the variable *test* if *k* is in the range –*m* through +*m*, inclusive. Otherwise, assigns a value of zero.
      b) Assigns a value of one (1) to the variable *lowercase* if *ch* is a lowercase letter; otherwise, assigns a value of zero (0).
      c) Assigns a value of one (1) to the variable *divisor* if *m* is a divisor of *n*; otherwise, assigns a value of zero(0).

   Note: The bold face words represent variables' names.

8. Write an interactive program that contains an *if* statement that may be used to computer the area of a square ($area = side^2$) or a triangle ($area = ½ \times base \times height$) after prompting the user to type the first character of the figure name (S or T).

9. A year is a leap year if it is divisible four, except that any year divisible by 100 is a leap year only if it is divisible by 400. Write a program that inputs a year such as 1996, 1800, and 2010, and displays "Leap year" if it is a leap year, otherwise displays "Not a leap year".

10. Write a program that inputs temperature and displays a message according to the following data:

Temperature	Message
Greater than 35	Hot day
Between 25 and 35 (inclusive)	Pleasant day
Less than 25	Cool day

11. Write a program that inputs obtained marks of a student, calculates percentage (assume total marks are 1100), and displays his/her grade. The grade should be calculated according to the following rules:

Percentage	Grade
More than or equal to 80	A+
Between 70 (inclusive) and 80	A
Between 60 (inclusive) and 70	B
Between 50 (inclusive) and 60	C
Between 40 (inclusive) and 50	D
Between 33 (inclusive) and 40	E
Less than 33	F

12. Write a program that inputs two numbers and asks for the choice of the user, if user enters 1 then displays the sum of numbers, if user enters 2 then displays result of subtraction of the numbers, if user enters 3 then displays the result of the multiplication of the numbers and if user enters 4 then displays result of the division of the numbers (divide the larger number by the smaller number), otherwise displays the message "Wrong Choice". [Use a switch statement to implement the solution]