4 The Case Control Structure

In real life we are often faced with situations where we are required to make a choice between a number of alternatives rather than only one or two. For example, which school to join or which hotel to visit or still harder which girl to marry (you almost always end up making a wrong decision is a different matter altogether!). Serious C programming is same; the choice we are asked to make is more complicated than merely selecting between two alternatives. C provides a special control statement that allows us to handle such cases effectively; rather than using a series of **if** statements. This control instruction is in fact the topic of this chapter. Towards the end of the chapter we would also study a keyword called **goto**, and understand why we should avoid its usage in C programming.

Decisions Using *switch*

The control statement that allows us to make a decision from the number of choices is called a **switch**, or more correctly a **switch-case-default**, since these three keywords go together to make up the control statement. They most often appear as follows:

```
switch ( integer expression )
{
    case constant 1 :
        do this ;
    case constant 2 :
        do this ;
    case constant 3 :
        do this ;
    default :
        do this ;
}
```

The integer expression following the keyword **switch** is any C expression that will yield an integer value. It could be an integer constant like 1, 2 or 3, or an expression that evaluates to an integer. The keyword **case** is followed by an integer or a character constant. Each constant in each **case** must be different from all the others. The “do this” lines in the above form of **switch** represent any valid C statement.

What happens when we run a program containing a **switch**? First, the integer expression following the keyword **switch** is evaluated. The value it gives is then matched, one by one, against the constant values that follow the **case** statements. When a match is found, the program executes the statements following that **case**, and all subsequent **case** and **default** statements as well. If no match is found with any of the **case** statements, only the statements following the **default** are executed. A few examples will show how this control structure works.

Consider the following program:

```
main( )
{
    int   i = 2 ;

    switch ( i )
    {
        case 1 :
            printf ( "I am in case 1 \n" ) ;
        case 2 :
            printf ( "I am in case 2 \n" ) ;
        case 3 :
            printf ( "I am in case 3 \n" ) ;
        default :
            printf ( "I am in default \n" ) ;
    }
}
```

The output of this program would be:

```
I am in case 2
```

```
I am in case 3
I am in default
```

The output is definitely not what we expected! We didn't expect the second and third line in the above output. The program prints case 2 and 3 and the default case. Well, yes. We said the **switch** executes the case where a match is found and all the subsequent **cases** and the **default** as well.

If you want that only case 2 should get executed, it is upto you to get out of the **switch** then and there by using a **break** statement. The following example shows how this is done. Note that there is no need for a **break** statement after the **default**, since the control comes out of the **switch** anyway.

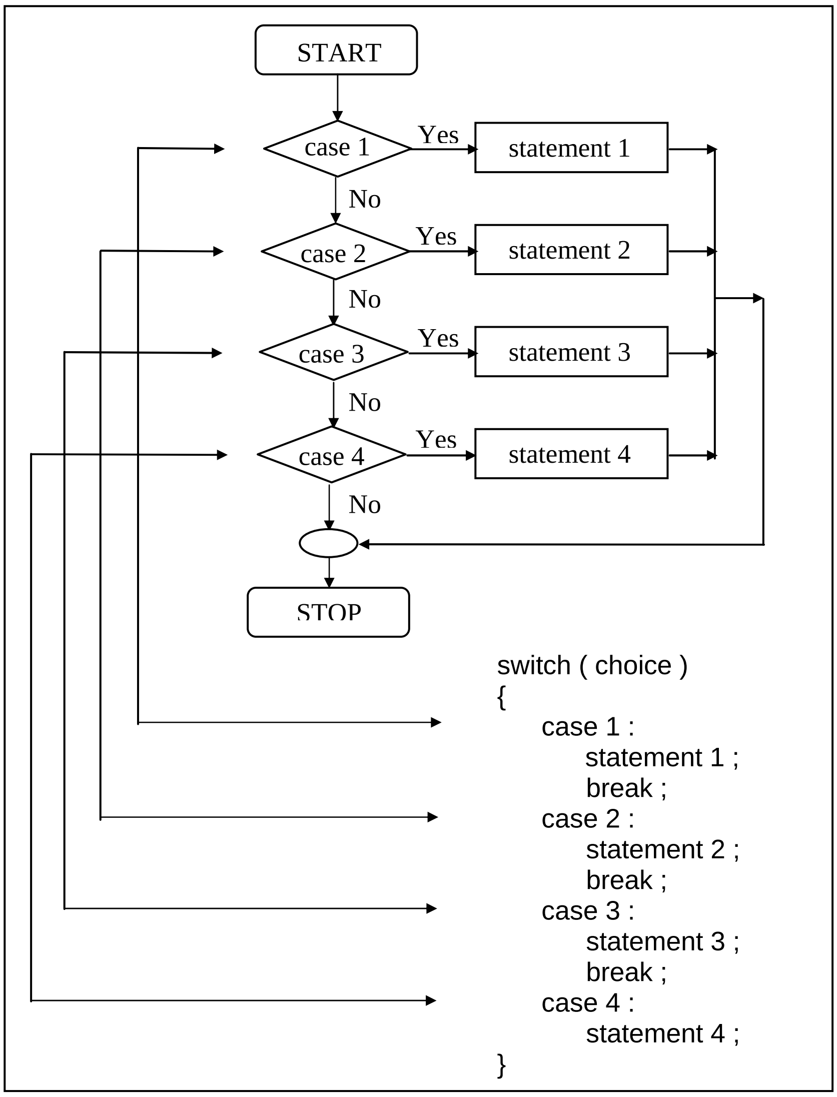
```
main( )
{
    int   i = 2 ;

    switch ( i )
    {
        case 1 :
            printf ( "I am in case 1 \n" ) ;
            break ;
        case 2 :
            printf ( "I am in case 2 \n" ) ;
            break ;
        case 3 :
            printf ( "I am in case 3 \n" ) ;
            break ;
        default :
            printf ( "I am in default \n" ) ;
    }
}
```

The output of this program would be:

```
I am in case 2
```

The operation of **switch** is shown below in the form of a flowchart for a better understanding.



```
switch ( choice )
{
    case 1 :
        statement 1 ;
        break ;
    case 2 :
        statement 2 ;
        break ;
    case 3 :
        statement 3 ;
        break ;
    case 4 :
        statement 4 ;
}
```

Figure 4.1

The Tips and Traps

A few useful tips about the usage of **switch** and a few pitfalls to be avoided:

(a) The earlier program that used **switch** may give you the wrong impression that you can use only cases arranged in ascending order, 1, 2, 3 and default. You can in fact put the cases in any order you please. Here is an example of scrambled case order:

```
main( )
{
    int  i = 22 ;

    switch ( i )
    {
        case 121 :
            printf ( "I am in case 121 \n" ) ;
            break ;
        case 7 :
            printf ( "I am in case 7 \n" ) ;
            break ;
        case 22 :
            printf ( "I am in case 22 \n" ) ;
            break ;
        default :
            printf ( "I am in default \n" ) ;
    }
}
```

The output of this program would be:

```
I am in case 22
```

(b) You are also allowed to use **char** values in **case** and **switch** as shown in the following program:

```
main( )
```

```
{
    char  c = 'x' ;

    switch ( c )
    {
        case 'v' :
            printf ( "I am in case v \n" ) ;
            break ;
        case 'a' :
            printf ( "I am in case a \n" ) ;
            break ;
        case 'x' :
            printf ( "I am in case x \n" ) ;
            break ;
        default :
            printf ( "I am in default \n" ) ;
    }
}
```

The output of this program would be:

```
I am in case x
```

In fact here when we use ‘v’, ‘a’, ‘x’ they are actually replaced by the ASCII values (118, 97, 120) of these character constants.

(c) At times we may want to execute a common set of statements for multiple **case**s. How this can be done is shown in the following example.

```
main( )
{
    char  ch ;

    printf ( "Enter any of the alphabet a, b, or c " ) ;
    scanf ( "%c", &ch ) ;
    switch ( ch )
    {
        case 'a' :
        case 'A' :
            printf ( "a as in ashar" ) ;
            break ;
        case 'b' :
        case 'B' :
            printf ( "b as in brain" ) ;
            break ;
        case 'c' :
        case 'C' :
            printf ( "c as in cookie" ) ;
            break ;
        default :
            printf ( "wish you knew what are alphabets" ) ;
    }
}
```

Here, we are making use of the fact that once a **case** is satisfied the control simply falls through the **case** till it doesn't encounter a **break** statement. That is why if an alphabet **a** is entered the **case 'a'** is satisfied and since there are no statements to be executed in this **case** the control automatically reaches the next **case** i.e. **case 'A'** and executes all the statements in this **case**.

(d) Even if there are multiple statements to be executed in each **case** there is no need to enclose them within a pair of braces (unlike **if**, and **else**).

(e) Every statement in a **switch** must belong to some **case** or the other. If a statement doesn't belong to any **case** the compiler won't report an error. However, the statement would never get executed. For example, in the following program the **printf( )** never goes to work.

```
main( )
{
    int  i, j ;

    printf ( "Enter value of i" ) ;
    scanf ( "%d", &i ) ;

    switch ( i )
    {
        printf ( "Hello" ) ;
        case 1 :
            j = 10 ;
            break ;
        case 2 :
            j = 20 ;
            break ;
    }
}
```

(f) If we have no **default** case, then the program simply falls through the entire **switch** and continues with the next instruction (if any,) that follows the closing brace of **switch**.

(g) Is **switch** a replacement for **if**? Yes and no. Yes, because it offers a better way of writing programs as compared to **if**, and no because in certain situations we are left with no choice but to use **if**. The disadvantage of **switch** is that one cannot have a case in a **switch** which looks like:

```
case i <= 20 :
```

All that we can have after the case is an **int** constant or a **char** constant or an expression that evaluates to one of these constants. Even a **float** is not allowed.

The advantage of **switch** over **if** is that it leads to a more structured program and the level of indentation is manageable, more so if there are multiple statements within each **case** of a **switch**.

(h) We can check the value of any expression in a **switch**. Thus the following **switch** statements are legal.

```
switch ( i + j * k )
switch ( 23 + 45 % 4 * k )
switch ( a < 4 && b > 7 )
```

Expressions can also be used in cases provided they are constant expressions. Thus **case 3 + 7** is correct, however, **case a + b** is incorrect.

(i) The **break** statement when used in a **switch** takes the control outside the **switch**. However, use of **continue** will not take the control to the beginning of **switch** as one is likely to believe.

(j) In principle, a **switch** may occur within another, but in practice it is rarely done. Such statements would be called nested **switch** statements.

(k) The **switch** statement is very useful while writing menu driven programs. This aspect of **switch** is discussed in the exercise at the end of this chapter.

switch Versus *if-else* Ladder

There are some things that you simply cannot do with a **switch**. These are:

(a) A float expression cannot be tested using a **switch**
(b) Cases can never have variable expressions (for example it is wrong to say **case a +3 :**)
(c) Multiple cases cannot use same expressions. Thus the following **switch** is illegal:

```
switch ( a )
{
    case 3 :
         ...
    case 1 + 2 :
         ...
}
```

(a), (b) and (c) above may lead you to believe that these are obvious disadvantages with a **switch**, especially since there weren't any such limitations with **if-else**. Then why use a **switch** at all? For speed—**switch** works faster than an equivalent **if-else** ladder. How come? This is because the compiler generates a jump table for a **switch** during compilation. As a result, during execution it simply refers the jump table to decide which case should be executed, rather than actually checking which case is satisfied. As against this, **if-else**s are slower because they are evaluated at execution time. A **switch** with 10 cases would work faster than an equivalent **if-else** ladder. Also, a **switch** with 2 **case**s would work slower than **if-else** ladder. Why? If the 10th **case** is satisfied then jump table would be referred and statements for the 10th **case** would be executed. As against this, in an **if-else** ladder 10 conditions would be evaluated at execution time, which makes it slow. Note that a lookup in the jump table is faster than evaluation of a condition, especially if the condition is complex.

If on the other hand the conditions in the **if-else** were simple and less in number then **if-else** would work out faster than the lookup mechanism of a **switch**. Hence a **switch** with two **case**s would work slower than an equivalent **if-else**. Thus, you as a programmer should take a decision which of the two should be used when.

The *goto* Keyword

Avoid **goto** keyword! They make a C programmer's life miserable. There is seldom a legitimate reason for using **goto**, and its use is one of the reasons that programs become unreliable, unreadable, and hard to debug. And yet many programmers find **goto** seductive.

In a difficult programming situation it seems so easy to use a **goto** to take the control where you want. However, almost always, there is a more elegant way of writing the same program using **if**, **for**, **while** and **switch**. These constructs are far more logical and easy to understand.

The big problem with **goto**s is that when we do use them we can never be sure how we got to a certain point in our code. They obscure the flow of control. So as far as possible skip them. You can always get the job done without them. Trust me, with good programming skills **goto** can always be avoided. This is the first and last time that we are going to use **goto** in this book. However, for sake of completeness of the book, the following program shows how to use **goto**.

```
main( )
{
    int  goals ;

    printf ( "Enter the number of goals scored against India" ) ;
    scanf ( "%d", &goals ) ;

    if ( goals <= 5 )
        goto sos ;
    else
    {
        printf ( "About time soccer players learnt C\n" ) ;
        printf ( "and said goodbye! adieu! to soccer" ) ;
        exit(  ) ;  /* terminates program execution */
    }

    sos :
        printf ( "To err is human!" ) ;
}
```

And here are two sample runs of the program...

```
Enter the number of goals scored against India 3
To err is human!
Enter the number of goals scored against India 7
About time soccer players learnt C
and said goodbye! adieu! to soccer
```

A few remarks about the program would make the things clearer.

– If the condition is satisfied the **goto** statement transfers control to the label ‘sos’, causing **printf( )** following **sos** to be executed.

– The label can be on a separate line or on the same line as the statement following it, as in,

```
sos : printf ( "To err is human!" ) ;
```

– Any number of **goto**s can take the control to the same label.

– The **exit( )** function is a standard library function which terminates the execution of the program. It is necessary to use this function since we don't want the statement

```
printf ( "To err is human!" )
```

to get executed after execution of the **else** block.

– The only programming situation in favour of using **goto** is when we want to take the control out of the loop that is contained in several other loops. The following program illustrates this.

```
main( )
{
    int  i, j, k ;

    for ( i = 1 ; i <= 3 ; i++ )
    {
        for ( j = 1 ; j <= 3 ; j++ )
        {
            for ( k = 1 ; k <= 3 ; k++ )
            {
                if ( i == 3 && j == 3 && k == 3 )
                    goto out ;
                else
                    printf ( "%d %d %d\n", i, j, k ) ;
            }
        }
    }
    out :
        printf ( "Out of the loop at last!" ) ;
}
```

Go through the program carefully and find out how it works. Also write down the same program without using **goto**.

Summary

(a) When we need to choose one among number of alternatives, a **switch** statement is used.
(b) The **switch** keyword is followed by an integer or an expression that evaluates to an integer.
(c) The **case** keyword is followed by an integer or a character constant.
(d) The control falls through all the cases unless the **break** statement is given.
(e) The usage of the **goto** keyword should be avoided as it usually violets the normal flow of execution.

Exercise

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

(a)
```
main( )
{
    char  suite = 3 ;
    switch ( suite )
    {
        case 1 :
            printf ( "\nDiamond" ) ;
        case 2 :
            printf ( "\nSpade" ) ;
        default :
            printf ( "\nHeart") ;
    }
    printf ( "\nI thought one wears a suite" ) ;
}
```

(b)
```
main( )
{
    int  c = 3 ;

    switch ( c )
    {
        case 'v' :
            printf ( "I am in case v \n" ) ;
            break ;
        case 3 :
            printf ( "I am in case 3 \n" ) ;
            break ;
        case 12 :
            printf ( "I am in case 12 \n" ) ;
            break ;
        default :
            printf ( "I am in default \n" ) ;
    }
```

```
     }

(c)  main( )
     {
         int  k, j = 2 ;
         switch ( k = j + 1 )
         {
              case 0 :
                   printf ( "\nTailor") ;
              case 1 :
                   printf ( "\nTutor") ;
              case 2 :
                   printf ( "\nTramp") ;
              default :
                   printf ( "\nPure Simple Egghead!" ) ;
         }
     }

(d)  main( )
     {
         int  i = 0 ;
         switch ( i )
         {
              case 0 :
                   printf ( "\nCustomers are dicey" ) ;
              case 1 :
                   printf ( "\nMarkets are pricey" ) ;
              case 2 :
                   printf ( "\nInvestors are moody" ) ;
              case 3 :
                   printf ( "\nAt least employees are good" ) ;
         }
     }

(e)  main( )
     {
         int  k ;
         float j = 2.0 ;
```

```
          switch ( k = j + 1 )
          {
               case 3 :
                    printf ( "\nTrapped" ) ;
                    break ;
               default :
                    printf ( "\nCaught!" ) ;
          }
     }
```

(f)
```
     main( )
     {
          int  ch = 'a' + 'b' ;
          switch ( ch )
          {
               case 'a' :
               case 'b' :
                    printf ( "\nYou entered b" ) ;
               case 'A' :
                    printf ( "\na as in ashar" ) ;
               case 'b' + 'a' :
                    printf ( "\nYou entered a and b" ) ;
          }
     }
```

(g)
```
     main( )
     {
          int  i = 1 ;
          switch ( i - 2 )
          {
               case -1 :
                    printf ( "\nFeeding fish" ) ;
               case 0 :
                    printf ( "\nWeeding grass" ) ;
               case 1 :
                    printf ( "\nmending roof" ) ;
               default :
                    printf ( "\nJust to survive" ) ;
```

```
    }
}
```

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

(a)
```
main( )
{
    int  suite = 1 ;
    switch ( suite ) ;
    {
        case 0 ;
            printf ( "\nClub" ) ;
        case 1 ;
            printf ( "\nDiamond" ) ;
    }
}
```

(b)
```
main( )
{
    int  temp ;
    scanf ( "%d", &temp ) ;
    switch ( temp )
    {
        case ( temp <= 20 ) :
            printf ( "\nOoooooohhhh! Damn cool!" ) ;
        case ( temp > 20 && temp <= 30 ) :
            printf ( "\nRain rain here again!" ) ;
        case ( temp > 30 && temp <= 40 ) :
            printf ( "\nWish I am on Everest" ) ;
        default :
            printf ( "\nGood old nagpur weather" ) ;
    }
}
```

(c)
```
main( )
{
    float  a = 3.5 ;
    switch ( a )
```

```
        {
            case 0.5 :
                printf ( "\nThe art of C" ) ;
                break ;
            case 1.5 :
                printf ( "\nThe spirit of C" ) ;
                break ;
            case 2.5 :
                printf ( "\nSee through C" ) ;
                break ;
            case 3.5 :
                printf ( "\nSimply c" ) ;
        }
    }
```

```
(d) main( )
    {
        int a = 3, b = 4, c ;
        c = b – a ;
        switch ( c )
        {
            case 1 || 2 :
                printf ( "God give me an opportunity to change things" ) ;
                break ;

            case a || b :
                printf ( "God give me an opportunity to run my show" ) ;
                break ;
        }
    }
```

[C] Write a menu driven program which has following options:

1. Factorial of a number.
2. Prime or not
3. Odd or even
4. Exit

Make use of *switch* statement.

The outline of this program is given below:

```
/* A menu driven program */
main( )
{
    int  choice ;
    while ( 1 )
    {
        printf ( "\n1.  Factorial" ) ;
        printf ( "\n2.  Prime" ) ;
        printf ( "\n3.  Odd/Even" ) ;
        printf ( "\n4.  Exit" ) ;
        printf ( "\nYour choice? " ) ;
        scanf ( "%d", &choice ) ;

        switch ( choice )
        {
            case 1 :
                /* logic for factorial of a number  */
                break ;
            case 2 :
                /* logic for deciding prime number */
                break ;
            case 3 :
                /* logic for odd/even */
                break ;
            case 4 :
                exit( ) ;
        }
    }
}
```

Note:

The statement **while (1)** puts the entire logic in an infinite loop. This is necessary since the menu must keep reappearing on the screen once an item is selected and an appropriate action taken.

[D] Write a program which to find the grace marks for a student using **switch**. The user should enter the class obtained by the student and the number of subjects he has failed in.

- If the student gets first class and the number of subjects he failed in is greater than 3, then he does not get any grace. If the number of subjects he failed in is less than or equal to 3 then the grace is of 5 marks per subject.

- If the student gets second class and the number of subjects he failed in is greater than 2, then he does not get any grace. If the number of subjects he failed in is less than or equal to 2 then the grace is of 4 marks per subject.

- If the student gets third class and the number of subjects he failed in is greater than 1, then he does not get any grace. If the number of subjects he failed in is equal to 1 then the grace is of 5 marks per subject