Case Control Structure

Decisions Using switch

- allows us to make a decision from the number of choices

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

Decisions Using switch

- 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

Executing 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

Example

```
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

Using break statement with switch

```
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

Tips and Traps

- You can put the cases in any order you please
- Output of program
- I am in case 22

```
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" ) ;
    }
}
```

- You are also allowed to use char values in case and switch
- The output of this program would be:
- I am in case x
- when we use ‘v’, ‘a’, ‘x’ they are actually replaced by the ASCII values (118, 97, 120) of these character constants.

```
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" ) ;
    }
}
```

- At times we may want to execute a common set of statements for multiple cases
- 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.

```
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" ) ;
    }
}
```

- Even if there are multiple statements to be executed in each case there is no need to enclose them within a pair of braces
- 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.
- 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.

```
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 ;
    }
}
```

Switch v/s if

- 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.

Value of expression in switch

- 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

- 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
- In principle, a switch may occur within another, but in practice it is rarely done. Such statements would be called nested switch statements
- 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 float expression cannot be tested using a switch
- Cases can never have variable expressions (for example it is wrong to say case a +3 :)
- Multiple cases cannot use same expressions. Thus the following switch is illegal:

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

switch Versus if-else Ladder

- For switch the compiler looks up a jump table
- If-else is evaluated at run time
- For longer if-else nesting, equivalent switch case will take less time
- For shorter if-else, if-else will take less time as compared to switch

The goto Keyword

- Avoid goto keyword!
- 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.
- The big problem with gotos 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.

```
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

Remarks

- 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 gotos 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!" ) ;
}
```