**Receiving Input**

In the program discussed above we assumed the values of **p**, **n** and **r** to be 1000, 3 and 8.5. Every time we run the program we would get the same value for simple interest. If we want to calculate simple interest for some other set of values then we are required to make the relevant change in the program, and again compile and execute it.

Thus the program is not general enough to calculate simple interest for any set of values without being required to make a change in the program.

Moreover, if you distribute the EXE file of this program to somebody he would not even be able to make changes in the program. Hence it is a good practice to create a program that is general enough to work for any set of values.

To make the program general the program itself should ask the user to supply the values of **p**, **n** and **r** through the keyboard during execution.

This can be achieved using a function called **scanf( )**. This function is a counter-part of the **printf( )** function. **printf( )** outputs the values to the screen whereas **scanf( )** receives them from the keyboard. This is illustrated in the program shown below.

main( )

{

int p, n ;

float r, si ;

printf ( "Enter values of p, n, r" ) ;

scanf ( "%d %d %f", &p, &n, &r ) ;

si = p \* n \* r / 100 ;

printf ( "%f" , si ) ;

}

The first **printf( )** outputs the message ‘Enter values of p, n, r’ on the screen. Here we have not used any expression in **printf( )** which means that using expressions in **printf( )** is optional.

Note that the ampersand (**&**) before the variables in the **scanf( )** function is a must. **&** is an ‘Address of’ operator. It gives the location number used by the variable in memory. When we say **&a**, we are telling **scanf( )** at which memory location should it store the value supplied by the user from the keyboard. The detailed working of the **&** operator would be taken up in Chapter 5.

Note that a blank, a tab or a new line must separate the values supplied to **scanf( )**. Note that a blank is creating using a spacebar, tab using the Tab key and new line using the Enter key. This is shown below:

Ex.: The three values separated by blank

1000 5 15.5

Ex.: The three values separated by tab.

1000 5 15.5

Ex.: The three values separated by newline.

1000

5

15.5

So much for the tips. How about another program to give you a feel of things...

/\* Just for fun. Author: Bozo \*/

main( )

{

int num ;

printf ( "Enter a number" ) ;

scanf ( "%d", &num ) ;

printf ( "Now I am letting you on a secret..." ) ;

printf ( "You have just entered the number %d", num ) ;

}