Each character in the array occupies one byte of memory and the last character is always '\0'. '\0' is called null character. **A string not terminated by a '\0' is not really a string, but merely a collection of characters.**

```
/* Program to demonstrate printing of a string */
main( )
{
char name[ ] = "Klinsman" ;
int i = 0 ;
while ( i <= 7 )
{
printf ( "%c", name[i] ) ;
i++ ;
}
```

And here is the output...

Klinsman

Can we write the **while** loop without using the final value 7? We can; because we know that each character array always ends with a '\0'. Following program illustrates this.

```
main( )
{
char name[ ] = "Klinsman" ;
int i = 0 ;
while ( name[i] != `\0' )
{
printf ( "%c", name[i] ) ;
i++ ;
}
}
```


And here is the output...

Klinsman

Here is another version of the same program; this one uses a pointer to access the array elements.

```
main( )
{
char name[ ] = "Klinsman" ;
char *ptr ;
ptr = name ; /* store base address of string */
while ( *ptr != `\0' )
{
printf ( "%c", *ptr ) ;
ptr++ ;
}
}
```

The **%s** used in **printf( )** is a format specification for printing out a string. The same specification can be used to receive a string from the keyboard, as shown below.

```
main( )
```

target string = Sayonara
Note that having copied the entire source string into the target string, it is necessary to place a '\0' into the target string, to mark its end.

- **strcat( )**

This function concatenates the source string at the end of the target string. For example, "Bombay" and "Nagpur" on concatenation would result into a string "BombayNagpur". Here is an example of **strcat( )** at work.

```
main( )
{
char source[ ] = "Folks!" ;
char target[30] = "Hello" ;
strcat ( target, source ) ;
printf ( "\nsource string = %s", source ) ;
printf ( "\ntarget string = %s", target ) ;
}
```

And here is the output...

```
source string = Folks!
target string = HelloFolks!
```

- **strcmp( )**

This is a function which compares two strings to find out whether they are same or different. The two strings are compared character by character until there is a mismatch or end of one of the strings is reached, whichever occurs first. If the two strings are identical, **strcmp( )** returns a value zero. If they're not, it returns the numeric difference between the ASCII values of the first non-matching pair of characters. Here is a program which puts **strcmp( )** in action.

```
main( )
{
char string1[ ] = "Jerry" ;
char string2[ ] = "Ferry" ;
int i, j, k ;
i = strcmp ( string1, "Jerry" ) ;
j = strcmp ( string1, string2 ) ;
k = strcmp ( string1, "Jerry boy" ) ;
printf ( "\n%d %d %d", i, j, k ) ;
}
```

And here is the output...

```
0 4 -32
```

In the first call to **strcmp( )**, the two strings are identical—"Jerry" and "Jerry"—and the value returned by **strcmp( )** is zero. In the second call, the first character of "Jerry" doesn't match with the first character of "Ferry" and the result is 4, which is the numeric difference between ASCII value of 'J' and ASCII value of 'F'. In the third call to **strcmp( )** "Jerry" doesn't match with "Jerry boy", because the null character at the end of "Jerry" doesn't match the blank in "Jerry boy". The value returned is -32, which is the value of null character minus the ASCII value of space, i.e., '\0' minus ' ', which is equal to -32.