Representation:

A C-string is an array of characters terminated by a null character. For example, the string **"toy"** is represented as shown in figure 1.

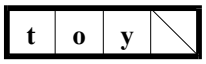

Figure 1. C-string **"toy"**.

Declaration:

A C-string is declared as an array of characters. Examples are given below.

```
char s[10];                 //s is an array of 10 characters having elements s[0] .. s[9].
                            //s can hold up to 9 characters
char t[]="one";             //t is a string initialized to the characters 'o', 'n', 'e', '\0'
char u[3]="one";            // t is initialized to the characters 'o', 'n', 'e'
                            //t is not a string.
char v[]={'o','n','e','\0'};//v is a string having four (4) characters. Each character is initialized.
char e[]="";                //e is a string having a single character, the null terminator.
                            //e is the empty string
```

Strings and pointers to strings:

1. Strings are referenced by pointers to the actual string. For example, variable *t*, is used to reference string *t* declared as **char *t*="toy";**
2. When the name of an array appears without a subscripting operator **[]**, the type of the array name is changed to a pointer to the element type. For example, *t* has type **char*** because elements of *t* have type char and *t* is an array.
3. String pointers can be declared directly. For example, **char*** ***s*;**. Variable ***s*** can be assigned to point to a string but no such assignment has been made yet. Variable *s* is said to be undefined. References to s will likely cause an execution-time error.
4. A string pointer can be initialized. For example **char*** ***s*="toy";** Storage for string **"toy"** is allocated in the constant area of the program. The string **"toy"** cannot be changed. String *s*, however, can be reassigned. Refer to figure 2.

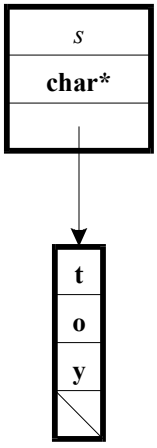

Figure 2. char* ***s*="toy";**

Operations:

#include <*string***>**

Declaration	Description	Example
int *strlen*(**char*** *s*);	Function *strlen* returns the number of characters in the string referenced by parameter *s*. The terminating character is excluded from the count returned by function *strlen*.	**char** *s*[]="one"; **int** *c*=**strlen**(*s*); *cout* << *c*; Output `3`
char* *strcpy*(**char*** *d*,**char*** *s*);	Function *strcpy* copies the contents of the string *s* to the string d, overwriting the contents of *d*. The entire contests of *s* are copied, plus the terminating null character even if *s* is longer than *d*. The argument d is returned.	**char** *d*[]="destinataion"; **char** *s*[]="source"; **char*** *t*=**strcpy**(*d*,*s*); *cout* << *d*; Output `source`
char* *strcat*(**char*** *d*,**char*** *s*);	Function *strcat* appends the contents of string *s* to string *d*. A pointer to string *d* is returned. The null character that terminates *d* (and perhaps other characters following it in memory) is overwritten with characters from *s* and a new terminating null character. Characters are copied from *s* until a null character is encountered in *s*. The memory beginning with *d* is assumed to be large enough to hold both strings.	**char** *d*[10]="One"; **char** *s*[]=", two"; **char*** *t*=*strcat*(*d*,*s*); Ouput `One, two`
int strcmp(char* u,char* v);	Function *strcmp* lexicographically compares the contents of the null-terminated string *u* with the contents of the null-terminated string *v*. It returns a value of type **int** that is less than zero if $u < v$; equal zero if $u = v$; and greater than zero if $u > v$.	**char u[]="ted";** **char v[]="tom";** **int c=strcmp(u,v);** *cout* << *c*; Ouput `-1`

Table 1. Selected functions in library **string.h.** (**#include <***string***>**) continued