Strings in C++

C++ offer ways of working with strings: Cstring and the standard string class. Cstrings are the mechanism for strings that was originally provided in the C language, and hence has been inherited by C++. The C++ string class also supports string manipulation, and provides some additional error checking which was not present with the Cstring.

Cstrings

As you know, a char array is an array which can store characters in it's elements. For example:

Char list[5];

list[0]	list[1]	list[2]	list[3]	list[4]
'd'	'r'	'1'	'p'	'x'

A *Cstring* is a character array which contains not only characters, but, also a null ('\0'). The null is placed in the element following the final character in the string of characters that is stored. The purpose of the null is to indicate the ending of the string.

Char name[7];

name[0]	name[1]	name[2]	name[3]	name[4]	name[5]	name[6]
's'	't'	'a'	'c'	'e'	'y'	'\0'

How does a Cstring get initialized?

* A Cstring can be initialized when it is declared.

For example: char text[] = "hello"; //creates a cstring with 6 elements

char word[10] = "hello"; //larger array – last 4 element unused

```
char   strng[4];
 strng = "bye" ;  // ILLEGAL
```

* A Cstring can be initialized using an input mechanism.

```
char in_string[20];
cin >> in_string;        // will read and store characters up
                         //until a blank or newline occurs in the input
                         // !! TROUBLE if there are more than 19 chars input
```

A Cstring can be initialized element by element, as a regular array, but the '\0' must be specifically assigned to the correct element.

```
char strng[6];
strng[0] = 'h';
strng[1] = 'i';
strng[2] = '\0';
```

How is a Cstring used?

The '\0' which terminates a Cstring is recognized by a variety of applications. A Cstring can be used for output of strings and with functions provided by C++ for handling Cstrings, a couple of which are listed on page 613 of your text.

The following example illustrates the use of Cstrings for *output*:

```
char done[] = "hello";
char undone[4];

undone[0] = 'b';
undone[1] = 'y';
undone[2] = 'e';

cout << done << "\n\n";   //outputs the string 'hello'
cout << undone << "\n";   //outputs 'bye' plus RAM contents until '\0' encountered
```

A number of *functions* are provided for neat manipulation of Cstrings. You may find a few of them helpful in your function, in particular strlen and strcat, which are exemplified below:

```
char done[20] = "hello"; //declare big enough for future needs
int lastindex;
char temp;

lastindex = strlen(done);  //swap 1st and last char
temp = done[lastindex -1];
done[lastindex -1] = done[0];
done[0] = temp;

strcat(done, "-goodbye");  //concatenate two strings
cout << done << "\n\n";
```

Standard string Class

C++ provides a string class, with overloaded operators and many methods to make string manipulation more natural.

The *string* class provides the following:

- overloading of the '=' operator (you can assign one string to another, rather than using strcpy as with Cstrings)
- overloading of the '+' operator (you can concatenate more naturally, no need for strcat as with Cstrings)
- overloading of the << and >> operators for I/O
- overloading of the relational operators ==, != , etc
- over 100 methods for manipulation of string objects, including at(int) method for safe indexing of the string (boundary checking IS done)
-

A nice overview of contstructors, assignment techniques and a few methods is given in your text on page 650. Look at the definitions for length() and substr(pos,len) prior to reading the following code. It performs the same task as the Cstring code above (perhaps not in the most efficient manner, I was trying to exhibit many string features).

```
string done;      // declare string object
int last;
char temp1,temp2;

  done = "hello";                  //assignment is overloaded for type string
  last = done.length();            // length method will return 5 here

  temp1 = done.at(0);              //get 1st char – it's at index 0
  temp2 = done.at(last -1);        //get last char – remember length is an absolute
                                   // and indexing begins at 0

 done = done.substr(0,last-1);     //reset string to itself minus the last char
 last = done.length();             //length is 4 now

 done = done.substr(1,last-1);     //change string again – minus 1st character

 done = temp2 + done + temp1;      //concatenate original w/ overloaded '+'
                                   //first and last characters

  done = done + "-goodbye";        //concatenate two strings
  cout << done << "\n\n";
```