"C-String" (_s Secure Version)

strcpy_s, strcat_s, strncpy_s --- proposed for standard, OK in Visual Studio

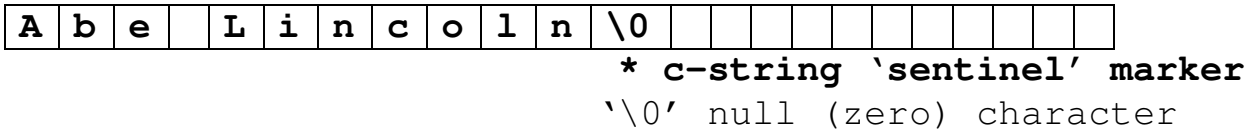
#include <cstring>

Use a "buffer" (an array of characters) to store a string-value.
Use "str…" library routines to store, copy, or compare values.

```
char  name [ 50 ];     // must always allocate maximum length
char  first [ 30 ];
char  last[30];

name = "Abe Lincoln";      // illegal - cannot "assign" arrays

strcpy( name, "Abe Lincoln");           // deprecated
strcpy_s( name, 50, "Abe Lincoln");     // ← Use secure version
```



Null sentinel is automatically appended with c-string "literals":

"HELLO" 6 bytes long: 'H' 'E' 'L' 'L' 'O' '\0'

Unnecessary to explicitly end c-string literal with null char:
"HELLO\0" 7 bytes long: 'H' 'E' 'L' 'L' 'O' '\0' '\0'

Determine c-string length (doesn't count the null char at the end) :

```
int  length = strlen( "Hello" );       // 5
int  lenName = strlen( name );         // 11
```

Accessing individual chars in c-string value – use array subscripting:

```
int  len = strlen(name);  // Count of actual chars in array
for(int i=0; i < len; ++i)
{
     cout << name[ i ] << endl;
}
```

Note: you can also output the whole c-string at once with output streams, e.g.,

```
cout << name << endl;
```

Copy c-string values:

```
strcpy_s(name, 50, "Fred"); // must use function to copy
```

Caution: strcpy(d, s) implements NO automatic overflow checking in C or C++.

```
int  a;
char buf[5];
int  c;
strcpy(buf, "Hi there");        // illegal, causes overflow of buf array, no error generated
strcpy_s(buf, 5, "Hi there"); // illegal, throws debugging error condition
```

Copy c-string values, with overflow limits:

```
char small[5];      // room for 4 chars + null

strcpy_s(  small, 5, "csc");             // c s c \0
strncpy_s( small, 5, "csc", 3);          // same result
strncpy_s( small, 5, "csc", 4);          // same result

strncpy_s( small, 5, "computer", 3);          // c o m \0
strncpy_s( small, 5, "computer", 4);          // c o m p \0
strncpy_s( small, 5, "computer", 5);          // Error- too big
strncpy_s( small, 5, "computer", _TRUNCATE); // c o m p \0
```

Concatenate a c-string to the end of another cstring: **strcat_s**

```
const int MAXLEN = 50;
char name[MAXLEN];

strcpy_s(name, MAXLEN, "Fred");
strcat_s(name, MAXLEN, " "); // append a blank char
strcat_s(name, MAXLEN, "Flintstone");
strncat_s(name, MAXLEN, " Rocks a lot", 6);
```

Extracting a c-string substring from the middle of another c-string:

```
char longstring[50] = "Hello There";
char histring[50], thestring[50];

for(int i=0; i<5; i++)    // Hello
     histring[i] = longstring[i];  // 5 chars from longstring
histring[5] = 0;   // append 0 for end of string sentinel

// Equivalent: strncpy_s(histring, 50, longstring, 5);
```

```
for(int i=0; i<3; i++)    // The
     thestring[i] = longstring[6 + i]; // 3 chars, start pos = 6
thestring[3] = 0;   // append 0 for end of string sentinel

// Equivalent: strncpy_s(thestring, 50,  longstring + 6, 3);
// Notice- Compute array base-address → ^^^^^^^^^^^^^^^
```

Input c-string value from stream: → read a line at a time into an array buffer

```
char inline[260];

// get: stores input sentinel; getline: does not store sentinel
cin.getline(inline, 260); // must specify maxlen
cin.getline(inline, 260, '\n');    // sentinel

cin.get(inline, 260);
cin.get(inline, 260, '\n');
```

Compare 2 c-string values (arrays cannot use ==, <, <=, >, >=, or !=):

```
char s1[20] = "...some string value ...";
char s2[50] = "...another string value";

if(s1 == "csc") ……      // illegal
if(s1 > s2) ……          // illegal
```

Compare 2 c-strings (**strcmp** compares contents char-by-char):

```
if(strcmp(s1, s2) == 0)        // equal
if(strcmp(s1, s2) < 0)         // less than
if(strcmp(s1, s2) > 0)         // gtr than
```

Uses underlying character set (e.g., http://www.asciitable.com/)

```
strcmp(s1, s2) :        compare
stricmp(s1, s2):        case-insensitive
    if(stricmp("Csc", "csc") == 0)     // true
    if(stricmp(answer, "yes") == 0 )   // yes,YES,Yes
strncmp(s1, s2, n):     compare first n chars only
strnicmp(s1, s2, n):    case-insensitive, n chars only
```

Examples:

```
strcmp("abc", "abc"):           == 0
strcmp(" abc", "abc"):          <  0
strcmp("abc", "cba"):           <  0
strcmp("aaa", "aab"):           <  0
strcmp("abc", "abcd"):          <  0
strcmp("aa" , "b"):             <  0
strcmp("a"  , "bbbb"):          <  0
strcmp("Abc", "abc"):           <  0
strcmp("HELLO",  "HELLO"):      == 0
strcmp("HELLO!", "HELLO"):      >  0
```