C-Style Strings

Strings are a fundamental data structure in programming. In this book, we use the standard library string class to handle string objects. Historically, the C programming language defined strings as an array of characters and provided a set of utility functions in the system file *string.h*. We refer to these string as *C-style strings.* The array definition of strings was inherited by C++ and woven into the declaration of many classes that are provided by C++ programming environments. You have experience with this fact in the stream open function that requires the file be given as a C-style string.

```
ifstream fin;                 // input stream
fin.open("inventory.dat");    // file name is C-style string
```

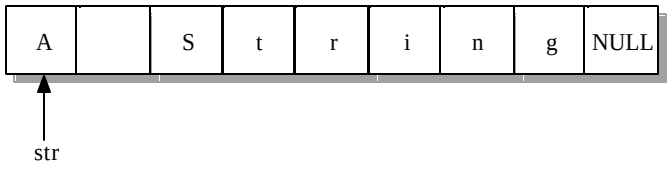
As a programmer, you should have some background with C-style strings.

Defining a C-style string.

A *C-style string* is a NULL-terminated sequence of characters stored in a C++ array. ASCII 0 designates the NULL character. A *string literal*, which is a sequence of characters enclosed in double quotes, is a C-style string. The following declaration creates a character array and assigns a string literal to the array.

```
char str[] = "A String";
```

The string is stored in memory as a 9 element character array.



Since the string is stored in an array, its name identifies the address of the first character and the location of NULL identifies the end of the string. For instance, str[0] is A and str[8] is the NULL character. The length of a C-style string is the number of characters preceding the NULL character. The length of string str is 8

A program has a variety of options for declaring a C-style string. In each case, we must allocate an array of characters with the NULL character terminating the string. The size of the array and the length of the string are often quite different.

```
// the character array str has 20 characters; the string uses 9 characters
char str[20] = "A String";

// str is the NULL string with length 0. str[0] is the NULL character
char str[20] = "";

// the compiler allocates four characters including NULL character
char str[] = "SOS";
```

EXAMPLE 1

A character array is not a C-style string until it contains a NULL character that indicates the end of the string.

```
char arr[10];
```

By placing a NULL character in the array block, we create a C-style string.

```
arr[0] = 'C';
arr[1] = '+';
arr[2] = '+';
arr[3] = 0;       // arr now contains the string "C++"
```

Converting String and C++ objects. The string class has a constructor that takes a C-style string as an argument. The constructor allows a program to convert a C-style string to a string object. The programmer can assign a C-style string to a string object.

```
// argument "Burrito" is a C-style string literal
string  strA("Burrito"), strB;

// assigns the C-style string "Chow Mein" to the string object strB
strB = "Chow Mein";
```

Conversion from a string object to a C-style string is provided by the member function c_str() that returns the address of the equivalent C-style string.

```
char *s;
string state = "Arizona";

s = state.c_str();   // s is the C-style string "Arizona"
```

This has important applications when opening a file stream. The open statement requires that the file name argument must be a C-style string.

```
string fileName("input.dat");

fin.open(fileName.c_str());      // return fileName as a C-style string
```

C-Style String I/O

C++ extends the stream operators "<<" and ">>" to C-style strings. With input, the >> operator extracts the next sequence of characters that are separated by whitespace. The output stream displays all of the characters in the string up to the NULL character. For instance,

```
char word1[32], word2[32];

cin >> word1 >> word2;     // Input: baseball umpire
cout << word2;             // Output: umpire
word1[4] = 0;              // insert NULL char after e in baseball
cout << word1;             // Output: base
```

The >> operator uses whitespace characters to separate C-style strings. When we want the input to include whitespace, we must use a special function, getline(), which is a member function in the istream class. The function has three arguments that include a character array, a positive integer n, and a character (delimeter) that terminates input. The delimiter is set to '\n' by default.

```
// member function prototype
istream& getline(char inputBuffer[], int n, char delimChar = '\n');
```

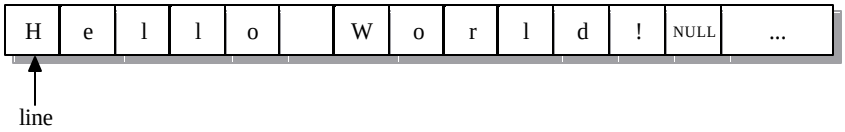
The array inputBuffer will contain the input string. The operation extracts characters from the input stream until either the character delimiter is found, n-1 characters are read, or end-of-file occurs. The characters are stored in the array inputBuffer followed by the NULL character. If delimChar is found, the character is extracted from the stream but not stored in the string. Note that you should not confuse this member function with the free function getline() which we use to input string objects.

EXAMPLE 2

1. Read an entire line from the keyboard by using the delimiter `'\n'`.

```
char line[81];    // space for 80 chars and NULL

cin.getline(line,81);   // delimeter defaults to '\n'
```

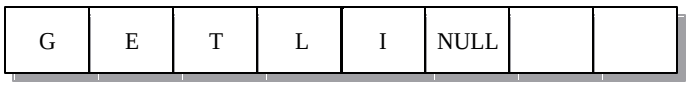
Input: Hello World!



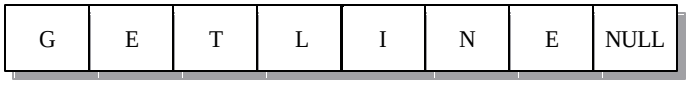
2. The getline() function uses separate count and delimiter conditions to terminate the input of characters. For parts (a) and (b), assume the input is "GETLINE-DEMO IT".

```
char buffer[25];
```

(a) `fin.getline(buffer,6,'-');    // read 5 chars. append NULL`



(b) `fin.getline(buffer,20,'-');      // read to - and discard char`



3. Read a social security number and a name from a stream fin. The social security number is in the format xxx-xx-xxxx and is followed by a colon (:). The name occupies the rest of the line.

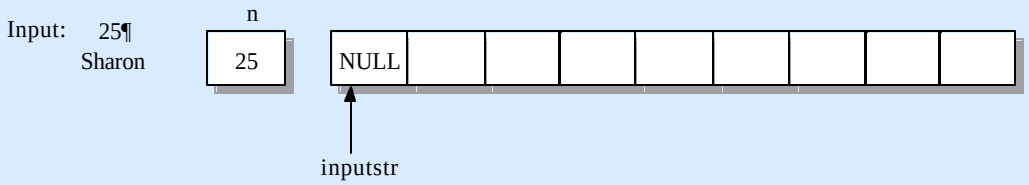
```
char ssn[12], name[51];

fin.getline(ssn,12,':');      // read the ssn and discard ':'
fin.getline(name,51,'\n');    // rest of the line is the name
```

```
For input:  459-23-4718:Wilson, John
   ssn is  "459-23-4718"
   name is "Wilson, John"
```

☞ Note

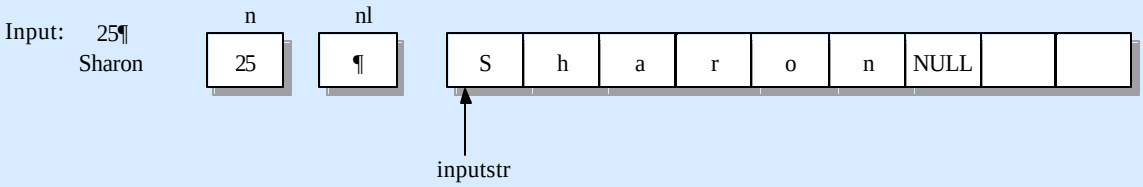
The getline() function must be used carefully with other forms of input. Assume input includes an integer followed by a string on the next line. An attempt to read the string with getline() extracts only the newline character (¶). This causes the input string to be NULL and is a frequent problem when mixing numeric and string input.

```
char inputstr[9];
int n;

// incorrect approach
cin >> n;                          // read integer 25
cin.getline(inputstr,9,'\n');      // read and discard ¶
```



Avoid the problem by extracting the newline character with a simple get(). The input for getline() is then taken from the next line.

```
// correct approach!
char nl;                           // declare char object for get()

cin >> n;                          // read integer 25
cin.get(nl);                       // extract the newline from the stream
cin.getline(inputstr,9,'\n');  // read string "Sharon"
```



C-style string Handling Functions

In the book, we introduce a series of operations for string objects. C-style strings have similar operations provided by free functions in the C++ system file *string.h*. This section introduces selected functions from the file including strlen() that returns the length of a string and strcmp() that uses ASCII ordering to compare two strings. To modify a string, we discuss functions strcpy() that copies one string to another, strcat() that concatenates two strings, and strstr() that identifies the location of a substring in larger string. For each function, we include the prototype, a description of its action and an example.

String Length

Prototype
```
int strlen(char* s);
```

Action
```
Returns the length of the C-style string s.
```

Example
```
char s[10] = "Compile";
cout << strlen(s);         // output is  7

str[strlen(s)-1] = 0;      // eliminate 'e' from s \
```

String Compare

Prototype
```
int strcmp(char* s, char* t);
```

Action
Compares the relative alphabetic ordering of the C-style strings s and t.
Returns a negative value if s is less than t.
Returns 0 if s is equal to t.
Returns a positive value if s is greater than t.

Example
```
char *s = "township";
int cmpValue;

// cmpValue is negative (< 0) since "township" < "village"
cmpValue = strcmp(s,"village");

// cmpValue is positive (> 0) since "township" > "city"
cmpValue = strcmp(s,"city");

// cmpValue is 0 since &s[4] is starting address for "ship"
cmpValue = strcmp(&s[4],"ship");
```

Pattern Matching (Single Character)

Prototype
```
char *strchr(char* s, char ch);
```

Action
Returns a pointer to the first occurrence of ch in the C-style string s. Returns NULL if ch is not in s.

Example
```
char *s = "mississippi", *p;

p = strchr(s,'s');      // p is the address of s[2]
p = strchr(s,'t');      // p is NULL

// &s[6] is the starting address for string "sippi"
p = strchr(&s[6],'p');  // p is the address of s[8]
cout << p;              // Output: ppi
```

Pattern Matching (Substring)

Prototype
```
char *strstr(char* s, char* t);
```

Action
Returns a pointer to the first occurrence of string t in the C-style string s. Returns NULL if t is not in s.

Example

```
char *s = "division", *t = "vision";

p = strstr(s,t);        // p is the address of s[2]
p = strstr(s,"visor");  // p is NULL
```

String Copy

Prototype

```
char *strcpy(char* s, char* t);
```

Action

The function copies string t (including the NULL character) into string s and then returns the address of s.

Example

```
char s[255], *t = "Good Morning", *u = "Evening";

// s = t is an invalid statement, since it attempts to
// assign pointer t to constant pointer s. C-style strings
// use strcpy() to copy one string to another

strcpy(s,t);            // assigns s = "Good Morning"
cout << strcpy(s,u);    // copy u to s; Output "Evening"

strcpy(s,&t[5]);        // copies "Morning" to s
```

String Concatenate

Prototype

```
char *strcpy(char* s, char* t);
```

Action

Copies string t onto the end of string s and returns the address of s.

Example

```
char s[255] = "Good", *t = "Morning";

strcat(s," ");          // add a blank onto the end of s
cout << strcat(s,t);    // append t. s is "Good Morning"
```

☞ Note

The copy and concatenate functions assume that the program provides sufficient memory for the target string. Since the C++ compiler does not perform array bounds checking, errors that are hard to detect may occur. For instance, consider this code.

```
char s[5], t[] = "too big!";

strcpy(s,t);
```

The function copies nine characters to s, but only 5 positions are available. The additional four characters stream into memory, possibly overwriting other program data values. When string objects are used, problems like these do not occur, since the class provides sufficient dynamic memory to hold the result.

```
// string class solution
string s, t = "Not too big!";

s = t;
```

PROGRAM ILLUSTRATING C-STYLE STRINGS

The program inputs a name into a C-style string in the format "First Last" and copies the name to another string in the format "Last, First". The steps are coded in the function reverseName() that takes a pointer to the original string as its first argument and pointer to the new string as its second argument. In a loop, the program input three strings and outputs the reversed string in each case.

```
#include <iostream>
#include <string.h>
using namespace std;

// reverse first and last name and separate them
// with a comma. copy result to newName.
void reverseName(char *name, char *newName);

int main()
{  char name[32], newName[32];
   int  i;

   // read and process three names
   for (i = 0; i < 3; i++)
   {  cin.getline (name,32,'\n');
      reverseName(name,newName);
      cout << "Reversed name: " << newName << endl << endl;
   }
   return 0;
}

void reverseName(char *name, char *newName)
{  char *p;

   // search for first blank in name and replace with NULL char
   p = strchr(name,' ');
   *p = 0;

   // copy last name to newName, append ", " and
   // concatenate first name
   strcpy(newName,p+1);
   strcat(newName,", ");
   strcat(newName,name);

   // replace the NULL char with the original blank
   *p = ' ';
}
```

```
Run:

Abraham Lincoln
Reversed name: Lincoln, Abraham

Debbie Rogers
Reversed name: Rogers, Debbie

Jim Brady
Reversed name: Brady, Jim
```