Strings

Class 32

Character Functions

- our programs so far have used numbers as the primary data
- since the purpose of programs is to model human activities, much computation involves characters and strings
- we begin with some useful character functions that reside in the cctype library, listed in Table 10-1 on page 558

Function	**Return Value**
isalpha	true if the argument is alphabetic
isalnum	true if the argument is alphabetic or a digit
isdigit	true if the argument is a digit
islower	true if the argument is lowercase alphabetic
isprint	true if the argument is a printable character
ispunct	true if the argument is a punctuation character (includes braces, dollar sign, less-than, etc.)
isupper	true if the argument is uppercase alphabetic
isspace	true if the argument is whitespace

Character Functions

- another pair of functions converts between upper and lower case
- toupper returns a uppercase equivalent to its argument if one exists, and returns the original character otherwise
- tolower returns an lowercase equivalent to its argument if one exists, and returns the original character otherwise

Character Functions

- another pair of functions converts between upper and lower case
- toupper returns a uppercase equivalent to its argument if one exists, and returns the original character otherwise
- tolower returns an lowercase equivalent to its argument if one exists, and returns the original character otherwise

- however, because of the historic screwiness of C++ types, these functions return an int, not a char, and so require a typecast:

```
char big_a = static_cast<char>(toupper('a'));
```

C-Strings

- we have discussed that C++ has two different kinds of strings
- old-fashioned C-strings inherited from the C language
- C++ string objects

C-Strings

- we have discussed that C++ has two different kinds of strings
- old-fashioned C-strings inherited from the C language
- C++ string objects

- similar to the situation with arrays and vectors, real programs written today use C++ strings
- but for historical reasons and because C++ still has many functions that assume C-strings, we must learn about them as well

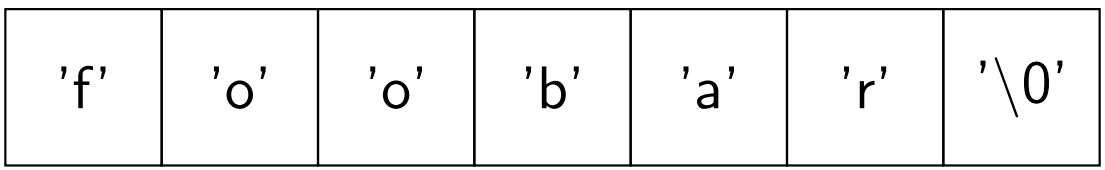
C-Strings

- a C-string is an array of characters (C-array)
- the final character in the array is the null character, the character with ASCII code 0
- the null character is written '\0'

C-Strings

- a C-string is an array of characters (C-array)
- the final character in the array is the null character, the character with ASCII code 0
- the null character is written '\0'

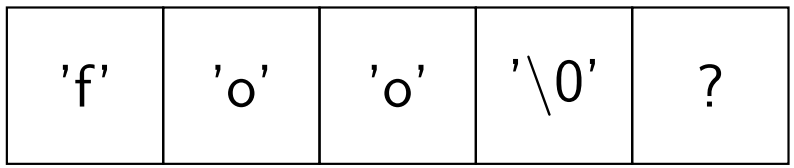
- a string literal enclosed in double quotes is stored in memory as a null-terminated C-string: `"foobar"` is



C-String Variables

- you can declare C-string variables

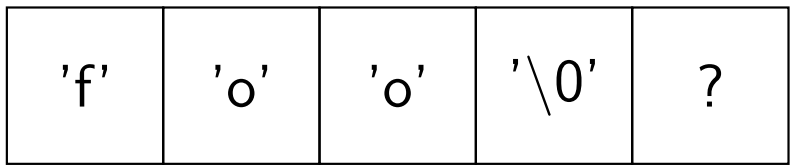
```
char name[5] = "foo";
```



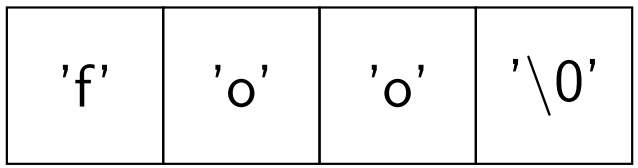
C-String Variables

- you can declare C-string variables

```
char name[5] = "foo";
```



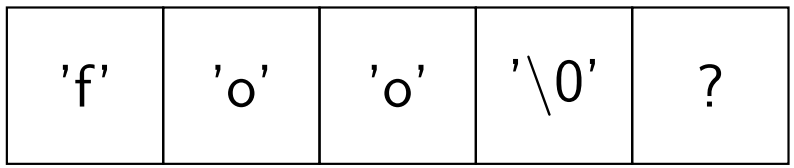

```
char name[] = "foo";
```



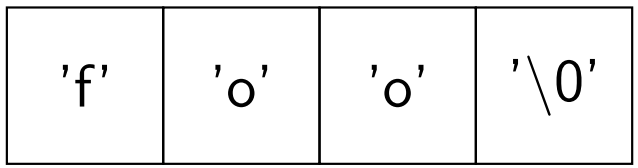
C-String Variables

- you can declare C-string variables

```
char name[5] = "foo";
```




```
char name[] = "foo";
```



- a three-character string occupies four array elements
- one element is required for the null character

C-string Output

- once a C-string is in an array of characters, it can be used for output

```
char name[10] = "Fred";
cout << name << endl;
```

C-string Output

- once a C-string is in an array of characters, it can be used for output

```
char name[10] = "Fred";
cout << name << endl;
```

- the stream insertion operator stops outputting characters when the null character is encountered
- regardless of the array size

'F'	'r'	'e'	'd'	\0	?	?	?	?	?

The Null Character

- everything about a C-string depends on the null character
- if something happens to that null character, everything goes south

```
char name[] = "Ann"; // name[3] is \0
name[3] = 'x'; // replace \0 with x
cout << name << endl;
Annxyy
```

The Null Character

- everything about a C-string depends on the null character
- if something happens to that null character, everything goes south

```
char name[] = "Ann"; // name[3] is \0
name[3] = 'x'; // replace \0 with x
cout << name << endl;
Annxyy
```

- once the null character is gone, cout doesn't know where to stop

C-string Input

- you can do stream extraction with C-strings, but it is very dangerous
- this is vulnerable to buffer overflow attacks

```
char name[5];
cout << "Enter a string (max 4 characters): ";
cin >> name;
cout << name << endl;

Enter a string (max 4 characters): abcd
abcd

Enter a string (max 4 characters): abcdxxxxxxxxxxxxxxxx
abcdxxxxxxxxxxxxxxxx
Segmentation fault (core dumped)
```

C-string Input

- a safer way to input into a C-string is with getline
- C++ has several different getline functions, all with the same name
- you have used one of them: `getline(cin, a_string);`
- this is a different one:

```
const size_t SIZE = 5;
char name[SIZE];
cout << "Enter a string (max 4 characters): ";
cin.getline(name, SIZE);
cout << name << endl;

Enter a string (max 4 characters): abcdxxxxxxxxxxxxxxx
abcd
```

- cin.getline does not exceed the limit of SIZE characters
- and thus is much safer than cin $\gg$