Chapter 10 Characters, C-Strings, and More about the `string` Class

10.1 Character Testing

Concept:

- The C++ library has functions for character testing. You must include the `<cctype>` header file to use them.

- These functions test a single `char` argument and return either `true` or `false`. For example, the `isupper` function checks if a character is an uppercase letter.

```
char letter = 'a';
if (isupper(letter))
   cout << "Letter is uppercase.\n";
else
   cout << "Letter is lowercase.\n";
```

- In the example, `isupper` returns `false` for a lowercase character, causing the `else` block's message to be displayed.

Table 10-1 lists several character-testing functions. Each of these is prototyped in the `<cctype>` header file, so be sure to include that file when using the functions.

Table 10-1 Character-Testing Functions in `<cctype>`

CHARACTER FUNCTION	DESCRIPTION
`isalpha`	Returns `true` (a nonzero number) if the argument is a letter of the alphabet. Returns 0 if the argument is not a letter.
`isalnum`	Returns `true` (a nonzero number) if the argument is a letter of the alphabet or a digit. Otherwise, it returns 0.
`isdigit`	Returns `true` (a nonzero number) if the argument is a digit from 0 through 9. Otherwise, it returns 0.
`islower`	Returns `true` (a nonzero number) if the argument is a lowercase letter. Otherwise, it returns 0.
`isprint`	Returns `true` (a nonzero number) if the argument is a printable character (including a space). Returns 0 otherwise.
`ispunct`	Returns `true` (a nonzero number) if the argument is a printable character other than a digit, letter, or space. Returns 0 otherwise.
`isupper`	Returns `true` (a nonzero number) if the argument is an uppercase letter. Otherwise, it returns 0.
`isspace`	Returns `true` (a nonzero number) if the argument is a whitespace character. Whitespace characters are any of the following: `space` `vertical tab '\v'`

CHARACTER FUNCTION	DESCRIPTION
	`newline '\n'` `tab '\t'` Otherwise, it returns 0.

- Program 10-1 demonstrates several of these functions. It takes a character from the user and displays messages based on the results of each test.

Program 10-1

```
#include <iostream>
#include <cctype>
using namespace std;

int main()
{
    char input;

    cout << "Enter any character: ";
    cin.get(input);
    cout << "The character you entered is: " << input << endl;
    if (isalpha(input))
       cout << "That's an alphabetic character.\n";
    if (isdigit(input))
       cout << "That's a numeric digit.\n";
    if (islower(input))
       cout << "The letter you entered is lowercase.\n";
    if (isupper(input))
       cout << "The letter you entered is uppercase.\n";
    if (isspace(input))
       cout << "That's a whitespace character.\n";
    return 0;
}
```

Program Output

Program Output

- Program 10-2 provides a practical application by validating if a customer number follows the correct format.

Program 10-2

```
#include <iostream>
#include <cctype>
using namespace std;

bool testNum(char [], int);

```

```
 7  int main()
 8  {
 9      const int SIZE = 8;
10      char customer[SIZE];
11
12      cout << "Enter a customer number in the form ";
13      cout << "LLLNNNN\n";
14      cout << "(LLL = letters and NNNN = numbers): ";
15      cin.getline(customer, SIZE);
16
17      if (testNum(customer, SIZE))
18          cout << "That's a valid customer number.\n";
19      else
20      {
21          cout << "That is not the proper format of the ";
22          cout << "customer number.\nHere is an example:\n";
23          cout << "   ABC1234\n";
24      }
25      return 0;
26  }
27
28
29  bool testNum(char custNum[], int size)
30  {
31      int count;
32
33      for (count = 0; count < 3; count++)
34      {
35          if (!isalpha(custNum[count]))
36              return false;
37      }
38
39      for (count = 3; count < size - 1; count++)
40      {
41          if (!isdigit(custNum[count]))
42              return false;
43      }
44      return true;
45  }
```

Program Output

Program Output

- The program expects a customer number with three letters followed by four digits.
- The `testNum` function validates this format.
- It uses `isalpha` in a loop to check if the first three characters are letters. The `!` operator negates the result, so if a character is NOT a letter, the function returns `false`.

```
for (count = 0; count < 3; count++)
{
    if (!isalpha(custNum[count]))
       return false;
}
```

- A second loop uses `isdigit` to verify the remaining characters are digits. Again, if a character is NOT a digit, the function returns `false`.

```
for (count = 3; count < size - 1; count++)
{
    if (!isdigit(custNum[count]))
       return false;
}
```

- If the customer number passes both checks, the function returns `true`.

10.2 Character Case Conversion

Concept:

- The C++ library offers functions to convert a character's case.

- The `toupper` and `tolower` functions, part of the `<cctype>` header, are used for case conversion.

Table 10-2 Case-Conversion Functions in `<cctype>`

FUNCTION	DESCRIPTION
`toupper`	Returns the uppercase equivalent of its argument.
`tolower`	Returns the lowercase equivalent of its argument.

- These functions take a single character argument. `toupper` returns the uppercase version of a character.

```
cout << toupper('a');
```

- If the argument is already uppercase, `toupper` returns it unchanged.

```
cout << toupper('Z');
```

- Non-letter arguments are also returned unchanged.

```
cout << toupper('*');
cout << toupper ('&');
cout << toupper('%');
```

- These functions do not modify the original variable; they simply return the converted value.

```
char letter = 'A';
cout << tolower(letter) << endl;
cout << letter << endl;
```

These statements will cause the following to be displayed:

```
a
A
```

- Program 10-3 uses `toupper` inside an input validation loop.

Program 10-3

```
1   #include <iostream>
2   #include <cctype>
3   #include <iomanip>
4   using namespace std;
5
6   int main()
7   {
8       const double PI = 3.14159;
9       double radius;
10      char goAgain;
11
12      cout << "This program calculates the area of a circle.\n";
13      cout << fixed << setprecision(2);
14
15      do
16      {
17          cout << "Enter the circle's radius: ";
18          cin >> radius;
19          cout << "The area is " << (PI * radius * radius);
20          cout << endl;
21
22          cout << "Calculate another? (Y or N) ";
23          cin >> goAgain;
24
25          while (toupper(goAgain) != 'Y' && toupper(goAgain) != 'N')
26          {
27              cout << "Please enter Y or N: ";
28              cin >> goAgain;
29          }
30
31      } while (toupper(goAgain) == 'Y');
32      return 0;
33  }
```

Program Output

- The program needs to accept 'Y', 'y', 'N', or 'n' as valid input.
- A verbose way to check this would be:

```
while (goAgain != 'Y' && goAgain != 'y' &&
       goAgain != 'N' && goAgain != 'N')
```

- A simpler approach is to use `toupper` to convert the input to uppercase, reducing the number of comparisons.

```
while (toupper(goAgain) != 'Y' && toupper(goAgain) != 'N')
```

- Using `tolower` would also work.

```
while (tolower(goAgain) != 'y' && tolower(goAgain) != 'n')
```

Checkpoint

10.1 Write a short description of each of the following functions:

```
isalpha
isalnum
isdigit
islower
isprint
ispunct
isupper
isspace
toupper
tolower
```

10.2 Write a statement that will convert the contents of the `char` variable `big` to lowercase. The converted value should be assigned to the variable `little`.

10.3 Write an `if` statement that will display the word "digit" if the variable `ch` contains a numeric digit. Otherwise, it should display "Not a digit."

10.4 What is the output of the following statement?

```
cout << toupper(tolower('A'));
```

10.5 Write a loop that asks the user "Do you want to repeat the program or quit? (R/Q)". The loop should repeat until the user has entered an R or a Q (either uppercase or lowercase).

10.3 C-Strings

Concept:

- In C++, a C-string is a sequence of characters stored in consecutive memory locations, terminated by a null character.

- *String* refers to any sequence of characters. C++ can store strings either as `string` objects or as C-strings.
- A *C-string* is a sequence of characters in consecutive memory, ending with a null character (`\0`). This is the method used in the C language.
- All C++ string literals, like `"Bailey"`, are stored in memory as C-strings.

Note:

- The escape sequence `\0` represents the null terminator, which corresponds to ASCII code 0.

- The purpose of the null terminator is to mark the end of the C-string, allowing programs to know its length.

More about String Literals

- A string literal is enclosed in double quotation marks (`" "`).

```
"Have a nice day."
"What is your name?"
"John Smith"
"Please enter your age:"
"Part Number 45Q1789"
```

- A program's string literals are stored in memory as C-strings with the null terminator automatically added, as seen in Program 10-4.

Program 10-4

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       char again;
7
8       do
9       {
10          cout << "C++ programming is great fun!" << endl;
11          cout << "Do you want to see the message again? ";
12          cin >> again;
13      } while (again == 'Y' || again == 'y');
14      return 0;
15  }
```

This program contains two string literals:

```
"C++ programming is great fun!"
"Do you want to see the message again? "
```

- The first string occupies 30 bytes (including `\0`) and the second occupies 39 bytes.

C	+	+		p	r	o	g	r	a	m	m	i	n	g		i	s		g	r	e	a	t		f	u	n	!	\0
D	o		y	o	u		w	a	n	t		t	o		s	e	e		t	h	e		m	e	s	s	a	g	
e		a	g	a	i	n	?		\0																				

- When a string literal appears in a statement, C++ uses its memory address.
- For example, in the following statement, the address of the string is passed to `cout`, which then displays characters until it encounters a null terminator.

```
cout << "Do you want to see the message again? ";
```

C-Strings Stored in Arrays

- Understanding C-strings is important for C++ programmers because:
    - You may encounter older *legacy code* that uses them.
    - Some C++ library functions work only with C-strings.
    - C libraries that you might use with C++ work with C-strings.

- To store a C-string, you define a `char` array large enough to hold the string plus one element for the null character.

```
const int SIZE = 21;
char name[SIZE];
```

- You can initialize a `char` array with a string literal, and the null terminator is added automatically.

```
const int SIZE = 21;
char name[SIZE] = "Jasmine";
```

- You can also let the compiler determine the array's size.

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

- You can use `cin` to read input into a `char` array, but this is unsafe as it does not check array boundaries and can cause a buffer overflow.

```
const int SIZE = 21;
char name[SIZE];
cin >> name;
```

- A safer way to get user input, including whitespace, is with `cin.getline`.
- This function requires a destination array and a size, preventing it from reading more characters than the array can hold. It also appends the null terminator.

```
const int SIZE = 80;
char line[SIZE];
```

```
cin.getline(line, SIZE);
```

- Program 10-5 shows how to process a C-string by looping through the `char` array until the null terminator is found.

Program 10-5

```
#include <iostream>
using namespace std;

int main()
{
    const int SIZE = 80;
    char line[SIZE];
    int count = 0;

    cout << "Enter a sentence of no more than "
         << (SIZE - 1) << " characters:\n";
    cin.getline(line, SIZE);

    cout << "The sentence you entered is:\n";
    while (line[count] != '\0')
    {
        cout << line[count];
        count++;
    }
    return 0;
}
```

💻 **Program Output**

10.4 Library Functions for Working with C-Strings

Concept:

- The C++ library provides numerous functions for C-string manipulation in the `<cstring>` header file.

The `strlen` Function

- Manipulating C-strings requires library functions from the `<cstring>` header.

```
#include <cstring>
```

- The `strlen` function returns the length of a C-string (the number of characters before the null terminator).

```
char name[] = "Thomas Edison";
int length;
length = strlen(name);
```

- The length of a string is different from the size of the array holding it.
- C-string functions can accept a `char` array name, a pointer to a `char`, or a string literal as an argument.

```
length = strlen("Thomas Edison");
```

The `strcat` Function

- The `strcat` function appends (concatenates) one C-string to the end of another.

```
const int SIZE = 13;
char string1[SIZE] = "Hello ";
char string2[] = "World!";

cout << string1 << endl;
cout << string2 << endl;
strcat(string1, string2);
cout << string1 << endl;
```

These statements will cause the following output:

```
Hello
World!
Hello World!
```

- `strcat` modifies the first string.
- The programmer must ensure the destination array is large enough for the combined string plus the null terminator.

- You can check the array's size before calling `strcat`.

```
if (sizeof(string1) >= (strlen(string1) + strlen(string2) + 1))
     strcat(string1, string2);
else
     cout << "String1 is not large enough for both strings.\n";
```

Warning!

- If the array holding the first string isn't large enough to hold both strings, `strcat` will overflow the boundaries of the array.

The `strcpy` Function

- The `strcpy` function copies one C-string to another, as the `=` operator cannot be used with `char` arrays for this purpose.

```
const int SIZE = 13;
char name[SIZE];
strcpy(name, "Albert Einstein");
```

- The contents of the second string are copied to the first, overwriting any existing data.

```
const int SIZE = 10;
char string1[SIZE] = "Hello", string2[SIZE] = "World!";
cout << string1 << endl;
cout << string2 << endl;
strcpy(string1, string2);
cout << string1 << endl;
cout << string2 << endl;
```

Here is the output:

```
Hello
World!
World!
World!
```

Warning!

- `strcpy` performs no bounds checking and can overflow the destination array if it isn't large enough.

The `strncat` and `strncpy` Functions

- `strncat` and `strncpy` are safer alternatives to `strcat` and `strcpy` because they prevent buffer overflows.
- `strncat` works like `strcat` but takes a third argument specifying the maximum number of characters to append.

```
strncat(string1, string2, 10);
```

- This example shows how to calculate the maximum number of characters that can be safely appended.

```
1   int maxChars;
2   const int SIZE_1 = 17;
3   const int SIZE_2 = 18;
4
```

```
  5   char string1[SIZE_1] = "Welcome ";
  6   char string2[SIZE_2] = "to North Carolina";
  7
  8   cout << string1 << endl;
  9   cout << string2 << endl;
 10   maxChars = sizeof(string1) - (strlen(string1) + 1);
 11   strncat(string1, string2, maxChars);
 12   cout << string1 << endl;
```

The output of this code is:

```
Welcome
to North Carolina
Welcome to North
```

- `strncpy` copies a specified number of characters from one string to another.

```
strncpy(string1, string2, 5);
```

- If the source string is longer than the number of characters to copy, `strncpy` does not automatically append a null terminator. You must add it manually.
- If the source string is shorter, the destination is padded with null terminators.

```
  1   int maxChars;
  2   const int SIZE = 11;
  3
  4   char string1[SIZE];
  5   char string2[] = "I love C++ programming!";
  6
  7   maxChars = sizeof(string1) - 1;
  8   strncpy(string1, string2, maxChars);
  9   string1[maxChars] = '\0';
 10   cout << string1 << endl;
```

The `strstr` Function

- The `strstr` function searches for the first occurrence of a substring within another string.
- It returns a pointer to the beginning of the found substring or `nullptr` if the substring is not found.

```
char arr[] = "Four score and seven years ago";
char *strPtr = nullptr;
cout << arr << endl;
strPtr = strstr(arr, "seven");
cout << strPtr << endl;
```

This code will display:

```
Four score and seven years ago
seven years ago
```

Note:

- The `nullptr` key word was introduced in C++ 11. In older versions, use the `NULL` constant instead.

- Program 10-6 demonstrates `strstr` by allowing a user to search for a product by its product number.

Program 10-6

```
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
    const int NUM_PRODS = 5;
    const int LENGTH = 27;

    char products[NUM_PRODS][LENGTH] =
                  { "TV327 31-inch Television",
                    "CD257 CD Player",
                    "TA677 Answering Machine",
                    "CS109 Car Stereo",
                    "PC955 Personal Computer" };

    char lookUp[LENGTH];
    char *strPtr = nullptr;
    int index;

    cout << "\tProduct Database\n\n";
    cout << "Enter a product number to search for: ";
    cin.getline(lookUp, LENGTH);

    for (index = 0; index < NUM_PRODS; index++)
    {
        strPtr = strstr(products[index], lookUp);
        if (strPtr != nullptr)
            break;
    }

    if (strPtr != nullptr)
        cout << products[index] << endl;
    else
        cout << "No matching product was found.\n";

    return 0;
}
```

Program Output

Program Output

Table 10-3 summarizes the string-handling functions discussed here, as well as the `strcmp` function that was discussed in Chapter 4. (All the functions listed require the `<cstring>` header file.)

Table 10-3 Some of the C-String Functions in `<cstring>`

FUNCTION	DESCRIPTION
`strlen`	Accepts a C-string or a pointer to a C-string as an argument. Returns the length of the C-string (not including the null terminator.) *Example Usage:* `len = strlen(name);`
`strcat`	Accepts two C-strings or pointers to two C-strings as arguments. The function appends the contents of the second string to the first C-string. (The first string is altered, the second string is left unchanged.) *Example Usage:* `strcat(string1, string2);`
`strcpy`	Accepts two C-strings or pointers to two C-strings as arguments. The function copies the second C-string to the first C-string. The second C-string is left unchanged. *Example Usage:* `strcpy(string1, string2);`
`strncat`	Accepts two C-strings or pointers to two C-strings, and an integer argument. The third argument, an integer, indicates the maximum number of characters to copy from the second C-string to the first C-string. *Example Usage:* `strncat(string1, string2, n);`
`strncpy`	Accepts two C-strings or pointers to two C-strings, and an integer argument. The third argument, an integer, indicates the maximum number of characters to copy from the second C-string to the first C-string. If `n` is less than the length of `string2`, the null terminator is not automatically appended to `string1`. If `n` is greater than the length of `string2`, `string1` is padded with '\0' characters. *Example Usage:* `strncpy(string1, string2, n);`
`strcmp`	Accepts two C-strings or pointers to two C-strings arguments. If `string1` and `string2` are the same, this function returns 0. If `string2` is alphabetically greater than `string1`, it returns a negative number. If `string2` is alphabetically less than `string1`, it returns a positive number. *Example Usage:* `if (strcmp(string1, string2))`
`strstr`	Accepts two C-strings or pointers to two C-strings as arguments. Searches for the first occurrence of `string2` in `string1`. If an occurrence of `string2` is found, the function returns a pointer to it. Otherwise, it returns `nullptr` (address 0).

FUNCTION	DESCRIPTION
	Example Usage: `cout << strstr(string1, string2);`

- In Program 10-6, a `for` loop iterates through the product array, calling `strstr` to search for the user's input.

```
strPtr = strstr(prods[index], lookUp);
```

- If a match is found, the returned pointer will not be `nullptr`, and a `break` statement exits the loop.

```
if (strPtr != nullptr)
    break;
```

- After the loop, an `if-else` statement checks the pointer's value to determine if a match was found and displays the result.

```
if (strPtr == nullptr)
    cout << "No matching product was found.\n";
else
    cout << prods[index] << endl;
```

The `strcmp` Function

- Relational operators (`==`, `>`, `<`) cannot be used to compare C-strings. Instead, use the `strcmp` function.
- `strcmp` takes two C-strings as arguments and returns an integer:
  - **0** if the strings are equal.
  - A **negative** value if the first string comes before the second alphabetically.
  - A **positive** value if the first string comes after the second alphabetically.

```
int strcmp(char *string1, char *string2);
```

- To check if two strings are equal, compare the return value of `strcmp` to 0.

```
if (strcmp(string1, string2) == 0)
    cout << "The strings are equal.\n";
else
    cout << "The strings are not equal.\n";
```

- Program 10-7 demonstrates this.

Program 10-7

```
1   #include <iostream>
2   #include <string>
3   using namespace std;
4
5   int main()
6   {
7       const int LENGTH = 40;
8       char firstString[LENGTH], secondString[LENGTH];
9
10      cout << "Enter a string: ";
11      cin.getline(firstString, LENGTH);
12      cout << "Enter another string: ";
13      cin.getline(secondString, LENGTH);
```

```
14
15      if (strcmp(firstString, secondString) == 0)
16          cout << "You entered the same string twice.\n";
17      else
18          cout << "The strings are not the same.\n";
19
20      return 0;
21  }
```

Program Output

- `strcmp` is case-sensitive. Some compilers provide nonstandard, case-insensitive versions like `stricmp`.
- Program 10-8 uses `strcmp` to validate a user-entered part number against a list of valid options.

Program 10-8

```
1   #include <iostream>
2   #include <cstring>
3   #include <iomanip>
4   using namespace std;
5
6   int main()
7   {
8       const double A_PRICE = 99.0,
9                    B_PRICE = 199.0;
10
11      const int PART_LENGTH = 9;
12      char partNum[PART_LENGTH];
13
14      cout << "The MP3 player part numbers are:\n"
15           << "\t16 Gigabyte, part number S147-29A\n"
16           << "\t32 Gigabyte, part number S147-29B\n"
17           << "Enter the part number of the MP3 player you\n"
18           << "wish to purchase: ";
19
20      cin >> partNum;
21
22      cout << showpoint << fixed << setprecision(2);
23      if (strcmp(partNum, "S147-29A") == 0)
24          cout << "The price is $" << A_PRICE << endl;
25      else if (strcmp(partNum, "S147-29B") == 0)
26          cout << "The price is $" << B_PRICE << endl;
27      else
28          cout << partNum << " is not a valid part number.\n";
29      return 0;
30  }
```

Program Output

Using `!` with `strcmp`

- The logical NOT (`!`) operator can simplify equality checks with `strcmp`.
- Since `strcmp` returns 0 (which is treated as `false`) when strings are equal, `!strcmp(...)` evaluates to `true` if they are the same.

```
if (strcmp(firstString, secondString) == 0)
if (!strcmp(firstString, secondString))
```

Sorting Strings

- `strcmp` is useful for sorting strings alphabetically.
- Program 10-9 asks the user for two names and uses `strcmp` to print them in alphabetical order.

Program 10-9

```
1   #include <iostream>
2   #include <string>
3   using namespace std;
4
5   int main()
6   {
7       const int NAME_LENGTH = 30;
8       char name1[NAME_LENGTH], name2[NAME_LENGTH];
9
10      cout << "Enter a name (last name first): ";
11      cin.getline(name1, NAME_LENGTH);
12      cout << "Enter another name: ";
13      cin.getline(name2, NAME_LENGTH);
14
15      cout << "Here are the names sorted alphabetically:\n";
16      if (strcmp(name1, name2) < 0)
17      cout << name1 << endl << name2 << endl;
18      else if (strcmp(name1, name2) > 0)
19      cout << name2 << endl << name1 << endl;
20      else
21      cout << "You entered the same name twice!\n";
22
23      return 0;
24  }
```

Program Output

Table 10-3 provides a summary of the C-string-handling functions we have discussed. All of the functions listed require the `<cstring>` header file.

Checkpoint

10.6 Write a short description of each of the following functions:

```
strlen
strcat
strcpy
strncat
strncpy
strcmp
strstr
```

10.7 What will the following program segment display?

```
char dog[] = "Fido";
cout << strlen(dog) << endl;
```

10.8 What will the following program segment display?

```
char string1[16] = "Have a ";
char string2[9] = "nice day";
strcat(string1, string2);
cout << string1 << endl;
cout << string2 << endl;
```

10.9 Write a statement that will copy the string "Beethoven" to the array `composer`.

10.10 When complete, the following program skeleton will search for the string "Windy" in the array `place`. If `place` contains "Windy" the program will display the message "Windy found." Otherwise, it will display "Windy not found."

```
#include <iostream>
using namespace std;
int main()
{
    char place[] = "The Windy City";
    return 0;
}
```

10.5 String/Numeric Conversion Functions

Concept:

- The C++ library provides functions to convert C-strings and `string` objects to numeric types, and vice versa.

- A number stored as a string, such as "`26792`", is a sequence of character codes and cannot be used in mathematical operations until it is converted to a numeric data type.
- The `<cstdlib>` header file provides several functions for converting C-strings into numeric values.

Table 10-4 C-String/Numeric Conversion Functions in `<cstdlib>`

FUNCTION	DESCRIPTION

FUNCTION	DESCRIPTION
`atoi`	Accepts a C-string as an argument. The function converts the C-string to an integer and returns that value. *Example Usage:* `int num = atoi("4569");`
`atol`	Accepts a C-string as an argument. The function converts the C-string to a `long` integer and returns that value. *Example Usage:* `long lnum = atol("500000");`
`atof`	Accepts a C-string as an argument. The function converts the C-string to a `double` and returns that value. *Example Usage:* `double fnum = atof("3.14159");`

Note:

- If a C-string that cannot be converted is passed to these functions, the behavior is undefined. Many compilers will convert characters up to the first invalid one. The functions might return 0 on failure.

The `string` to Number Functions

- C++ 11 introduced new functions in the `<string>` header to convert `string` objects to numeric values.

Table 10-5 `string` to Number Functions

FUNCTION	DESCRIPTION
`stoi(string str)`	Accepts a `string` argument and returns that argument's value converted to an `int`.
`stol(string str)`	Accepts a `string` argument and returns that argument's value converted to a `long`.
`stoul(string str)`	Accepts a `string` argument and returns that argument's value converted to an `unsigned long`.
`stoll(string str)`	Accepts a `string` argument and returns that argument's value converted to a `long long`.
`stoull(string str)`	Accepts a `string` argument and returns that argument's value converted to an `unsigned long long`.
`stof(string str)`	Accepts a `string` argument and returns that argument's value converted to a `float`.
`stod(string str)`	Accepts a `string` argument and returns that argument's value converted to a `double`.
`stold(string str)`	Accepts a `string` argument and returns that argument's value converted to a `long double`.

Note:

- If a `string` cannot be converted, these functions throw an `invalid_argument` exception. If the converted value is out of range for the data type, they throw an `out_of_range` exception.

- These functions can accept a `string` object, a string literal, or a C-string as an argument.

```
string str = "99";
int i = stoi(str);

int i = stoi("99");
char cstr[] = "99";
int i = stoi(cstr);
```

The `to_string` Function

- The C++ 11 `to_string` function, found in `<string>`, converts a numeric value to a `string` object.

Table 10-6 Overloaded Versions of the `to_string` Function

FUNCTION	DESCRIPTION
`to_string(int value);`	Accepts an `int` argument and returns that argument converted to a `string` object.
`to_string(long value);`	Accepts a `long` argument and returns that argument converted to a `string` object.
`to_string(long long value);`	Accepts a `long long` argument and returns that argument converted to a `string` object.
`to_string(unsigned value);`	Accepts an `unsigned` argument and returns that argument converted to a `string` object.
`to_string(unsigned long value);`	Accepts an `unsigned long` argument and returns that argument converted to a `string` object.
`to_string(unsigned long long value);`	Accepts an `unsigned long long` argument and returns that argument converted to a `string` object.
`to_string(float value);`	Accepts a `float` argument and returns that argument converted to a `string` object.
`to_string(double value);`	Accepts a `double` argument and returns that argument converted to a `string` object.
`to_string(long double value);`	Accepts a `long double` argument and returns that argument converted to a `string` object.

- Each version takes a numeric argument and returns its `string` representation.

```
int number = 99;
string output = to_string(number);
```

Here is another example:

```
double number = 3.14159;
cout << to_string(number) << endl;
```

- Program 10-10 demonstrates the `stoi` function. It allows a user to enter numbers, calculates their average, and stops when the user enters 'Q' or 'q'.

Program 10-10

```
#include <iostream>
#include <cctype>
#include <string>
using namespace std;

int main()
{
   string input;
   int total = 0;
   int count = 0;
   double average;

   cout << "This program will average a series of numbers.\n";
   cout << "Enter the first number or Q to quit: ";
   getline(cin, input);

   while (tolower(input[0]) != 'q')
   {
      total += stoi(input);
      count++;
      cout << "Enter the next number or Q to quit: ";
      getline(cin, input);
   }

   if (count != 0)
   {
      average = static_cast<double>(total) / count;
      cout << "Average: " << average << endl;
   }
   return 0;
   }
```

Program Output

- The program's `while` loop checks the user's input to see if it starts with 'q' or 'Q'.

```
while (tolower(input[0]) != 'q')
```

- If the input is not a quit command, `stoi` converts the string to an integer, which is added to a running total.

```
total += stoi(input);
```

- Mixing `cin >>` with `getline` can cause input problems because `cin >>` leaves the newline character in the input buffer, which `getline` then reads as an empty line.

```
1   cout << "What is your ID number? ";
2   cin >> idNumber;
3
4   cout << "What is your name? ";
5   getline(cin, name);
```

- A reliable solution is to use `getline` for all input, reading numeric values as strings and then converting them to their appropriate numeric types.
- Program 10-11 demonstrates this technique.

Program 10-11

```
1   #include <iostream>
2   #include <string>
3   #include <iomanip>
4   using namespace std;
5
6   int main()
7   {
8      string input;
9      string name;
10     int idNumber;
11     int age;
12     double income;
13
14     cout << "What is your ID number? ";
15     getline(cin, input);
16     idNumber = stoi(input);
17
18     cout << "What is your name? ";
19     getline(cin, name);
20
21     cout << "How old are you? ";
22     getline(cin, input);
23     age = stoi(input);
24
25     cout << "What is your annual income? ";
26     getline(cin, input);
27     income = stod(input);
28
29     cout << setprecision(2) << fixed << showpoint;
30     cout << "Your name is " << name
31          <<", you are " << age
32          << " years old,\nand you make $"
33          << income << " per year.\n";
34
35     return 0;
36  }
```

Program Output

Checkpoint

10.11 Write a short description of each of the following functions:

`atoi`	`stoi`
`atol`	`stol`
`atof`	`stof`
`itoa`	`stod`

10.12 Write a statement that will convert the string "`10`" to an integer and store the result in the variable `num`.

10.13 Write a statement that will convert the string "`100000`" to a `long` and store the result in the variable `num`.

10.14 Write a statement that will convert the string "`7.2389`" to a `double` and store the result in the variable `num`.

10.15 Write a statement that will convert the integer 127 to a string, and assign the result to a `string` object named `str`.

10.6 Focus on Software Engineering: Writing Your Own C-String-Handling Functions

Concept:

- You can design your own specialized functions for manipulating strings.

Writing a C-String-Handling Function

- Because arrays can be passed as arguments to functions, you can write your own functions to process C-strings.
- Program 10-12 shows a custom function that copies one C-string to another.

Program 10-12

```
1   #include <iostream>
2   using namespace std;
3
4   void stringCopy(char [], char []);
5
6   int main()
7   {
8       const int LENGTH = 30;
9       char first[LENGTH];
10      char second[LENGTH];
11
12      cout << "Enter a string with no more than "
13      << (LENGTH - 1) << " characters:\n";
14      cin.getline(first, LENGTH);
15
```

```
16      stringCopy(first, second);
17
18      cout << "The string you entered is:\n" << second << endl;
19      return 0;
20  }
21
22
23  void stringCopy(char string1[], char string2[])
24  {
25      int index = 0;
26
27      while (string1[index] != '\0')
28      {
29          string2[index] = string1[index];
30          index++;
31      }
32
33      string2[index] = '\0';
34  }
```

Program Output

- The `stringCopy` function copies characters from the source to the destination array until it encounters the null terminator. It then adds a null terminator to the end of the destination string to properly terminate it.

Warning!

- Because the `stringCopy` function doesn't know the size of the destination array, the programmer is responsible for ensuring it is large enough to hold the source string.

- Program 10-13 presents another custom function, `nameSlice`, which searches for the first space in a string and replaces it with a null terminator, effectively shortening the string.

Program 10-13

```
1   #include <iostream>
2   using namespace std;
3
4   void nameSlice(char []);
5
6   int main()
7   {
8       const int SIZE = 41;
9       char name[SIZE];
10
11      cout << "Enter your first and last names, separated ";
12      cout << "by a space:\n";
13      cin.getline(name, SIZE);
14      nameSlice(name);
15      cout << "Your first name is: " << name << endl;
16      return 0;
17  }
18
19
```

```
20  void nameSlice(char userName[])
21  {
22      int count = 0;
23
24      while (userName[count] != ' ' && userName[count] != '\0')
25          count++;
26
27      if (userName[count] == ' ')
28          userName[count] = '\0';
29  }
```

Program Output

- The function's `while` loop scans the array until it finds either a space or a null terminator.

```
while (userName[count] != ' ' && userName[count] != '\0')
   count++;
```

Note:

- The loop also stops at a null terminator to prevent going past the end of the array if no space is present.

- If a space is found, the `if` statement replaces it with a null terminator.

```
if (userName[count] == ' ')
   userName[count] = '\0';
```

Using Pointers to Pass C-String Arguments

- Pointers are very useful for writing C-string functions, as they only require the starting address of the string; the null terminator marks its end.
- Program 10-14 demonstrates a function that uses a pointer to count the occurrences of a specific character in a C-string.

Program 10-14

```
1   #include <iostream>
2   using namespace std;
3
4   int countChars(char *, char);
5
6   int main()
7   {
8       const int SIZE = 51;
9       char userString[SIZE];
10      char letter;
11
12      cout << "Enter a string (up to 50 characters): ";
13      cin.getline(userString, SIZE);
14
15      cout << "Enter a character and I will tell you how many\n";
16      cout << "times it appears in the string: ";
17      cin >> letter;
18
19      cout << letter << " appears ";
```

```
20        cout << countChars(userString, letter) << " times.\n";
21        return 0;
22    }
23
24
25    int countChars(char *strPtr, char ch)
26    {
27        int times = 0;
28
29        while (*strPtr != '\0')
30        {
31            if (*strPtr == ch)
32                times++;
33            strPtr++;
34        }
35
36        return times;
37    }
```

Program Output

- In the `countChars` function, the `while` loop continues as long as the character being pointed to (`*strPtr`) is not the null terminator.

```
while (*strPtr != '\0')
```

- Inside the loop, an `if` statement compares the current character to the target character.

```
if (*strPtr == ch)
```

- The pointer is then incremented to point to the next character in the string.

```
strPtr++;
```

Checkpoint

10.16 What is the output of the following program?

```
#include <iostream>
using namespace std;
void mess(char []);
int main()
{
    char stuff[] = "Tom Talbert Tried Trains";
    cout << stuff << endl;
    mess(stuff);
    cout << stuff << endl;
    return 0;
}
void mess(char str[])
{
    int step = 0;
    while (str[step] != '\0')
```

```
    {
      if (str[step] == 'T')
         str[step] = 'D';
      step++;
    }
}
```

10.7 More about the C++ `string` Class

Concept:

- Standard C++ provides the `string` class, a special data type for storing and working with strings.

- The `string` class is an abstract data type, not a primitive type, that provides powerful features for string manipulation.

More about the `string` Class

Using the `string` Class

- To use the `string` class, you must include the `<string>` header file.

```
#include <string>
```

- Defining a `string` object is similar to defining a primitive variable.

```
string movieTitle;
```

- You can use the assignment operator to store a value in it.

```
movieTitle = "Wheels of Fury";
```

- The `cout` object can be used to display its contents.

```
cout << "My favorite movie is " << movieTitle << endl;
```

- Program 10-15 provides a complete example.

Program 10-15

```
1   #include <iostream>
2   #include <string>
3   using namespace std;
4
5   int main()
6   {
7       string movieTitle;
8
9       movieTitle = "Wheels of Fury";
10      cout << "My favorite movie is " << movieTitle << endl;
11      return 0;
12  }
```

Program Output

- As shown in Program 10-16, you can use `cin` to read input from the keyboard into a `string` object.

Program 10-16

```
1   #include <iostream>
2   #include <string>
3   using namespace std;
4
5   int main()
6   {
7       string name;
8
9       cout << "What is your name? ";
10      cin >> name;
11      cout << "Good morning " << name << endl;
12      return 0;
13  }
```

Program Output

Reading a Line of Input into a `string` Object

- To read an entire line of input, including spaces, into a `string` object, use the `getline` function.

```
string name;
cout << "What is your name? ";
getline(cin, name);
```

Comparing and Sorting `string` Objects

- `string` objects can be compared directly with relational operators (`<`, `>`, `==`, etc.), so functions like `strcmp` are not needed.

```
string set1 = "ABC";
string set2 = "XYZ";
```

- The comparison is performed lexicographically (alphabetically).

```
if (set1 < set2)
   cout << "set1 is less than set2.\n";
```

- You can also compare `string` objects to C-strings.

```
str > "Joseph"
"Kimberly" < str
str == "William"
```

- Program 10-17 demonstrates comparing `string` objects.
- Program 10-18 shows how relational operators can be used to sort `string` objects.

Program 10-17

```cpp
#include <iostream>
#include <iomanip>
#include <string>
using namespace std;

int main()
{
  const double APRICE = 249.0;
  const double BPRICE = 199.0;
  string partNum;

  cout << "The headphone part numbers are:\n";
  cout << "\tNoise canceling, part number S147-29A\n";
  cout << "\tWireless, part number S147-29B\n";
  cout << "Enter the part number of the desired headphones: ";
  cin >> partNum;
  cout << fixed << showpoint << setprecision(2);

  if (partNum == "S147-29A")
  cout << "The price is $" << APRICE << endl;
  else if (partNum == "S147-29B")
  cout << "The price is $" << BPRICE << endl;
  else
  cout << partNum << " is not a valid part number.\n";
   return 0;
}
```

Program Output

Program 10-18

```cpp
#include <iostream>
#include <string>
using namespace std;

int main ()
{
    string name1, name2;

    cout << "Enter a name (last name first): ";
    getline(cin, name1);

    cout << "Enter another name: ";
    getline(cin, name2);

    cout << "Here are the names sorted alphabetically:\n";
    if (name1 < name2)
        cout << name1 << endl << name2 << endl;
    else if (name1 > name2)
```

```
        cout << name2 << endl << name1 << endl;
    else
        cout << "You entered the same name twice!\n";
    return 0;
}
```

Program Output

Other Ways to Define `string` Objects

- `string` objects can be initialized in various ways at the time of definition.

Table 10-7 Examples of `string` Object Definitions

DEFINITION	DESCRIPTION
`string address;`	Defines an empty `string` object named `address`.
`string name("William Smith");`	Defines a `string` object named `name`, initialized with "William Smith."
`string person1(person2);`	Defines a `string` object named `person1`, which is a copy of `person2`. `person2` may be either a `string` object or character array.
`string str1(str2, 5);`	Defines a `string` object named `str1`, which is initialized to the first five characters in the character array `str2`.
`string lineFull('z', 10);`	Defines a `string` object named `lineFull` initialized with 10 `'z'` characters.
`string firstName(fullName, 0, 7);`	Defines a `string` object named `firstName`, initialized with a substring of the `string fullName`. The substring is seven characters long, beginning at position 0.

Program 10-19

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

int main()
{
    string greeting;
    string name("William Smith");

    greeting = "Hello ";
    cout << greeting << name << endl;
    return 0;
}
```

Program Output

- The `string` class also supports several operators for common operations.

Table 10-8 `string` Class Operators

SUPPORTED OPERATOR	DESCRIPTION
`>>`	Extracts characters from a stream and inserts them into the `string`. Characters are copied until a whitespace or the end of the `string` is encountered.
`<<`	Inserts the `string` into a stream.
`=`	Assigns the `string` on the right to the `string` object on the left.
`+=`	Appends a copy of the `string` on the right to the `string` object on the left.
`+`	Returns a `string` that is the concatenation of the two `string` operands.
`[]`	Implements array-subscript notation, as in `name[x]`. A reference to the character in the `x` position is returned.
Relational Operators	Each of the relational operators is implemented:
	`< > <= >= == ! =`

- Program 10-20 demonstrates the use of several `string` operators.

Program 10-20

```
1   #include <iostream>
2   #include <string>
3   using namespace std;
4
5   int main ()
6   {
7       string str1, str2, str3;
8
9       str1 = "ABC";
10      str2 = "DEF";
11      str3 = str1 + str2;
12
13      cout << str1 << endl;
14      cout << str2 << endl;
15      cout << str3 << endl;
16
17      str3 += "GHI";
18      cout << str3 << endl;
19      return 0;
20  }
```

Program Output

Using `string` Class Member Functions

- The `string` class provides member functions for various operations. For instance, the `length` function returns the number of characters in the string.

```
string town = "Charleston";
```

```
x = town.length();
```

- Program 10-21 demonstrates the `length` member function.

Program 10-21

```
1   #include <iostream>
2   #include <string>
3   using namespace std;
4
5   int main ()
6   {
7       string town;
8
9       cout << "Where do you live? ";
10      cin >> town;
11      cout << "Your town's name has " << town.length() ;
12      cout << " characters\n";
13      return 0;
14  }
```

Program Output

- The `size` member function also returns the string's length and is often used in loops, as shown in Program 10-22.

Program 10-22

```
1   #include <iostream>
2   #include <string>
3   using namespace std;
4
5   int main()
6   {
7       string str1, str2, str3;
8
9       str1 = "ABC";
10      str2 = "DEF";
11      str3 = str1 + str2;
```

```
12
13      for (int x = 0; x < str3.size(); x++)
14          cout << str3[x];
15      cout << endl;
16
17      if (str1 < str2)
18          cout << "str1 is less than str2\n";
19      else
20          cout << "str1 is not less than str2\n";
21      return 0;
22  }
```

Program Output

- Table 10-9 lists many of the available `string` class member functions.

Table 10-9 `string` Class Member Functions

MEMBER FUNCTION EXAMPLE	DESCRIPTION
`mystring.append(n, 'z')`	Appends `n` copies of `'z'` to `mystring`.
`mystring.append(str)`	Appends `str` to `mystring`. `str` can be a `string` object or character array.
`mystring.append(str, n)`	The first `n` characters of the character array `str` are appended to `mystring`.
`mystring.append(str, x, n)`	`n` number of characters from `str`, starting at position `x`, are appended to `mystring`. If `mystring` is too small, the function will copy as many characters as possible.
`mystring.assign(n, 'z')`	Assigns `n` copies of `'z'` to `mystring`.
`mystring.assign(str)`	Assigns `str` to `mystring`. `str` can be a `string` object or character array.
`mystring.assign(str, n)`	The first `n` characters of the character array `str` are assigned to `mystring`.
`mystring.assign(str, x, n)`	`n` number of characters from `str`, starting at position `x`, are assigned to `mystring`. If `mystring` is too small, the function will copy as many characters as possible.
`mystring.at(x)`	Returns the character at position `x` in the `string`.
`mystring.back()`	Returns the last character in the `string`. (This member function was introduced in C++ 11.)
`mystring.begin()`	Returns an iterator pointing to the first character in the `string`. (For more information on iterators, see Chapter 16.)
`mystring.c_str()`	Converts the contents of `mystring` to a C-string, and returns a pointer to the C-string.
`mystring.capacity()`	Returns the size of the storage allocated for the `string`.
`mystring.clear()`	Clears the `string` by deleting all the characters stored in it.
`mystring.compare(str)`	Performs a comparison like the `strcmp` function (see Chapter 4), with the same return values. `str` can be a `string` object or a character array.

MEMBER FUNCTION EXAMPLE	DESCRIPTION
`mystring.compare(x, n, str)`	Compares `mystring` and `str`, starting at position `x`, and continuing for `n` characters. The return value is like `strcmp`. `str` can be a `string` object or character array.
`mystring.copy(str, x, n)`	Copies the character array `str` to `mystring`, beginning at position `x`, for `n` characters. If `mystring` is too small, the function will copy as many characters as possible.
`mystring.empty()`	Returns `true` if `mystring` is empty.
`mystring.end()`	Returns an iterator pointing to the last character of the string in `mystring`. (For more information on iterators, see Chapter 17.)
`mystring.erase(x, n)`	Erases `n` characters from `mystring`, beginning at position `x`.
`mystring.find(str, x)`	Returns the first position at or beyond position `x` where the string `str` is found in `mystring`. `str` may be either a `string` object or a character array.
`mystring.find('z', x)`	Returns the first position at or beyond position `x` where `'z'` is found in `mystring`. If `'z'` is not found, the function returns the special value `string::npos`.
`mystring.front()`	Returns the first character in the `string`. (This member function was introduced in C++ 11.)
`mystring.insert(x, n, 'z')`	Inserts `'z'` `n` times into `mystring` at position `x`.
`mystring.insert(x, str)`	Inserts a copy of `str` into `mystring`, beginning at position `x`. `str` may be either a `string` object or a character array.
`mystring.length()`	Returns the length of the string in `mystring`.
`mystring.replace(x, n, str)`	Replaces the `n` characters in `mystring` beginning at position `x` with the characters in string object `str`.
`mystring.resize(n, 'z')`	Changes the size of the allocation in `mystring` to `n`. If `n` is less than the current size of the string, the string is truncated to `n` characters. If `n` is greater, the string is expanded and `'z'` is appended at the end enough times to fill the new spaces.
`mystring.size()`	Returns the length of the string in `mystring`.
`mystring.substr(x, n)`	Returns a copy of a substring. The substring is `n` characters long and begins at position `x` of `mystring`.
`mystring.swap(str)`	Swaps the contents of `mystring` with `str`.

In the Spotlight:

String Tokenizing

Sometimes a string will contain a series of words or other items of data separated by spaces or other characters. For example, look at the following string:

```
"peach raspberry strawberry vanilla"
```

This string contains the following four items of data: `peach`, `raspberry`, `strawberry`, and `vanilla`. In programming terms, items such as these are known as *tokens*. Notice a space appears between the items. The character that separates tokens is

known as a *delimiter*. Here is another example:

```
"17;92;81;12;46;5"
```

This string contains the following tokens: `17`, `92`, `81`, `12`, `46`, and `5`. Notice a semicolon appears between each item. In this example, the semicolon is used as a delimiter. Some programming problems require you to read a string that contains a list of items then extract all of the tokens from the string for processing. For example, look at the following string that contains a date:

```
"3-22-2018"
```

The tokens in this string are `3`, `22`, and `2018`, and the delimiter is the hyphen character. Perhaps a program needs to extract the month, day, and year from such a string. Another example is an operating system pathname, such as the following:

```
/home/rsullivan/data
```

The tokens in this string are `home`, `rsullivan`, and `data`, and the delimiter is the '`/`' character. Perhaps a program needs to extract all of the directory names from such a pathname. The process of breaking a string into tokens is known as *tokenizing*, or *splitting* a string.

The following function, `split`, shows an example of how we can split a string into tokens. The function has three parameters:

- `s`—the string that we want to split into tokens
- `delim`—the character that is used as a delimiter
- `tokens`—a `vector` that will hold the tokens once they are extracted

```
1   void split(const string& s, char delim, vector<string>& tokens)
2   {
3      int tokenStart = 0;
4
5      int delimPosition = s.find(delim);
6
7      while (delimPosition != string::npos)
8      {
9         string tok = s.substr(tokenStart, delimPosition - tokenStart);
10
11        tokens.push_back(tok);
12
13        delimPosition++;
14
15        tokenStart = delimPosition;
16
17        delimPosition = s.find(delim, delimPosition);
18
19        if (delimPosition == string::npos)
20        {
21            string tok = s.substr(tokenStart, delimPosition - tokenStart);
22
23           tokens.push_back(tok);
24        }
25     }
26  }
```

Let's take a closer look at the code:

- The `tokenStart` variable defined in line 3 will be used to hold the starting position of the next token. This variable is initialized with 0, assuming the first token starts at position 0.
- In line 6, we call the `s.find()` function, passing `delim` as an argument. The function will return the position of the first occurrence of `delim` in `s`. That value is assigned to the `delimPosition` variable. Note if the `find()` member function does not find the specified delimiter, it will return the special constant `string::npos` (which is defined as −1).
- The `while` loop that begins in line 9 iterates as long as `delimPosition` is not equal to `string::npos`.
- Inside the `while` loop, line 12 extracts the substring beginning at `tokenStart`, and continuing up to the delimiter. The substring, which is a token, is assigned to the `tok` object.
- In line 15, we push the `tok` object to the back of the `tokens` vector.
- Now we are ready to find the next token. Line 18 increments `delimPosition`, and line 21 sets `tokenStart` to the same value as `delimPosition`.
- Line 24 calls the `s.find()` function, passing `delim` and `delimPosition` as arguments. The function will return the position of the next occurrence of `delim`, appearing at or after `delimPosition`. That value is assigned to the `delimPosition` variable. If the specified delimiter is not found, the `find()` member function will return the constant `string::npos`. When that happens in this statement, it means we are processing the last token in the string.
- The `if` statement that starts in line 27 determines whether `delimPosition` is equal to `string::npos`. As previously stated, if this is true, it means we are processing the last token. If that is the case, line 30 extracts the token and assigns it to `tok`, and line 33 pushes `tok` to the back of the `tokens` vector.

Program 10-23 demonstrates the `split` function.

Program 10-23

```
1   #include <iostream>
2   #include <string>
3   #include <vector>
4   using namespace std;
5
6   void split(const string&, char, vector<string>&);
7
8   int main()
9   {
10     string str1 = "one two three four";
11     string str2 = "10:20:30:40:50";
12     string str3 = "a/b/c/d/e/f";
13
14     vector<string> tokens;
15
16     split(str1, ' ', tokens);
17     for (auto e : tokens)
18        cout << e << " ";
19     cout << endl;
20     tokens.clear();
21
22     split(str2, ':', tokens);
23     for (auto e : tokens)
24        cout << e << " ";
25     cout << endl;
26     tokens.clear();
27
28     split(str3, '/', tokens);
```

```
   for (auto e : tokens)
      cout << e << " ";
   cout << endl;
   return 0;
}

 The split function is not shown here.
```

Program Output

10.8 Focus on Problem Solving and Program Design: A Case Study

- This case study involves creating a `dollarFormat` function.
- The function accepts a `string` reference containing an unformatted number (e.g., "1084567.89") and modifies it to include a dollar sign and commas (e.g., "$1,084,567.89").

```
void dollarFormat(string &currency)
{
   int dp;
   dp = currency.find('.');
   if (dp > 3)
   {
      for (int x = dp - 3; x > 0; x -= 3)
         currency.insert(x, ",");
   }
   currency.insert(0, "$");
}
```

- The function first finds the position of the decimal point.

```
dp = currency.find('.');
```

- It then checks if the number of digits before the decimal is greater than three.

```
if (dp > 3)
```

- If so, a `for` loop inserts commas at the appropriate three-digit intervals.

```
for (int x = dp - 3; x > 0; x -= 3)
     currency.insert(x, ",");
```

- Finally, a dollar sign is inserted at the beginning of the string.
- Program 10-24 provides a full demonstration of this function.

Program 10-24

```
1   #include <iostream>
2   #include <string>
3   using namespace std;
4
```

```
 5   void dollarFormat(string &);
 6
 7   int main ()
 8   {
 9        string input;
10
11        cout << "Enter a dollar amount in the form nnnnn.nn : ";
12        cin >> input;
13        dollarFormat(input);
14        cout << "Here is the amount formatted:\n";
15        cout << input << endl;
16        return 0;
17   }
18
19
20   void dollarFormat(string &currency)
21   {
22        int dp;
23
24        dp = currency.find('.');
25        if (dp > 3)
26        {
27        for (int x = dp - 3; x > 0; x -= 3)
28        currency.insert(x, ",");
29        }
30        currency.insert(0, "$");
31   }
```

Program Output

Review Questions and Exercises

Short Answer

1. What header file must you include in a program using character-testing functions such as `isalpha` and `isdigit`?
2. What header file must you include in a program using the character conversion functions `toupper` and `tolower`?
3. Assume `c` is a `char` variable. What value does `c` hold after each of the following statements executes?

Statement	*Contents of* `c`
`c = toupper('a');`	_____
`c = toupper('B');`	_____
`c = tolower('D');`	_____
`c = toupper('e');`	_____

4. Look at the following code. What value will be stored in `s` after the code executes?

```
char name[10];
int s;
strcpy(name, "Jimmy");
s = strlen(name);
```

5. What header file must you include in a program using string functions such as `strlen` and `strcpy`?
6. What header file must you include in a program using string/numeric conversion functions such as `atoi` and `atof`?
7. What header file must you include in a program using `string` class objects?
8. How do you compare `string` class objects?

Fill-in-the-Blank

1. The _____ function returns `true` if the character argument is uppercase.
2. The _____ function returns `true` if the character argument is a letter of the alphabet.
3. The _____ function returns `true` if the character argument is a digit.
4. The _____ function returns `true` if the character argument is a whitespace character.
5. The _____ function returns the uppercase equivalent of its character argument.
6. The _____ function returns the lowercase equivalent of its character argument.
7. The _____ file must be included in a program that uses character-testing functions.
8. The _____ function returns the length of a string.
9. To _____ two strings means to append one string to the other.
10. The _____ function concatenates two strings.
11. The _____ function copies one string to another.
12. The _____ function searches for a string inside of another one.
13. The _____ function compares two strings.
14. The _____ function copies, at most, *n* number of characters from one string to another.
15. The _____ function returns the value of a string converted to an integer.
16. The _____ function returns the value of a string converted to a long integer.
17. The _____ function returns the value of a string converted to a float.
18. The _____ function converts an integer to a string.

Algorithm Workbench

1. The following `if` statement determines whether choice is equal to `'Y'` or `'y'`:

   ```
   if (choice == 'Y' || choice == 'y')
   ```

   Simplify this statement by using either the `toupper` or `tolower` function.

2. Assume `input` is a `char` array holding a C-string. Write code that counts the number of elements in the array that contain an alphabetic character.

3. Look at the following array definition:

   `char str[10];`

   Assume `name` is also a `char` array, and it holds a C-string. Write code that copies the contents of `name` to `str` if the C-string in `name` is not too big to fit in `str`.

4. Look at the following statements:

   `char str[] = "237.89";`

   `double value;`

   Write a statement that converts the string in `str` to a `double` and stores the result in `value`.

5. Write a function that accepts a pointer to a C-string as its argument. The function should count the number of times the character 'w' occurs in the argument and return that number.

6. Assume `str1` and `str2` are string class objects. Write code that displays "They are the same!" if the two objects contain the same string.

True or False

1. T F Character-testing functions, such as `isupper`, accept strings as arguments and test each character in the string.
2. T F If `toupper`'s argument is already uppercase, it is returned as is, with no changes.
3. T F If `tolower`'s argument is already lowercase, it will be inadvertently converted to uppercase.
4. T F The `strlen` function returns the size of the array containing a string.
5. T F If the starting address of a C-string is passed into a pointer parameter, it can be assumed that all the characters, from that address up to the byte that holds the null terminator, are part of the string.
6. T F C-string-handling functions accept as arguments pointers to strings (array names or pointer variables), or literal strings.
7. T F The `strcat` function checks to make sure the first string is large enough to hold both strings before performing the concatenation.
8. T F The `strcpy` function will overwrite the contents of its first string argument.
9. T F The `strcpy` function performs no bounds checking on the first argument.
10. T F There is no difference between "847" and 847.

Find the Errors

Each of the following programs or program segments has errors. Find as many as you can.

1. ```
   char str[] = "Stop";
   if (isupper(str) == "STOP")
      exit(0);
   ```

2. ```
   char numeric[5];
   int x = 123;
   numeric = atoi(x);
   ```

3.
```
char string1[] = "Billy";
char string2[] = " Bob Jones";
strcat(string1, string2);
```

4.
```
char x = 'a', y = 'a';
if (strcmp(x, y) == 0)
   exit(0);
```

Programming Challenges

1. String Length

   Write a function that returns an integer and accepts a pointer to a C-string as an argument. The function should count the number of characters in the string and return that number. Demonstrate the function in a simple program that asks the user to input a string, passes it to the function, then displays the function's return value.

   

   Solving the Backward String Problem

2. Backward String

   Write a function that accepts a pointer to a C-string as an argument and displays its contents backward. For instance, if the string argument is `"Gravity"` the function should display `"ytivarG"`. Demonstrate the function in a program that asks the user to input a string then passes it to the function.

3. Word Counter

   Write a function that accepts a pointer to a C-string as an argument and returns the number of words contained in the string. For instance, if the string argument is "Four score and seven years ago" the function should return the number 6. Demonstrate the function in a program that asks the user to input a string then passes it to the function. The number of words in the string should be displayed on the screen. *Optional Exercise:* Write an overloaded version of this function that accepts a `string` class object as its argument.

4. Average Number of Letters

   Modify the program you wrote for Programming Challenge 3 (Word Counter), so it also displays the average number of letters in each word.

5. Sentence Capitalizer

   Write a function that accepts a pointer to a C-string as an argument and capitalizes the first character of each sentence in the string. For instance, if the string argument is "`hello. my name is Joe. what is your name?`" the function should manipulate the string so that it contains "`Hello. My name is Joe. What is your name?`" Demonstrate the function in a program that asks the user to input a string then passes it to the function. The modified string should be displayed on the screen. *Optional Exercise:* Write an overloaded version of this function that accepts a `string` class object as its argument.

6. Vowels and Consonants

   Write a function that accepts a pointer to a C-string as its argument. The function should count the number of vowels appearing in the string and return that number.

   Write another function that accepts a pointer to a C-string as its argument. This function should count the number of consonants appearing in the string and return that number.

Demonstrate these two functions in a program that performs the following steps:

1. The user is asked to enter a string.
2. The program displays the following menu:
    1. Count the number of vowels in the string
    2. Count the number of consonants in the string
    3. Count both the vowels and consonants in the string
    4. Enter another string
    5. Exit the program
3. The program performs the operation selected by the user and repeats until the user selects E to exit the program.

7. Name Arranger

    Write a program that asks for the user's first, middle, and last names. The names should be stored in three different character arrays. The program should then store, in a fourth array, the name arranged in the following manner: the last name followed by a comma and a space, followed by the first name and a space, followed by the middle name. For example, if the user entered "`Carol Lynn Smith`", it should store "`Smith, Carol Lynn`" in the fourth array. Display the contents of the fourth array on the screen.

8. Sum of Digits in a String

    Write a program that asks the user to enter a series of single-digit numbers with nothing separating them. Read the input as a C-string or a `string` object. The program should display the sum of all the single-digit numbers in the string. For example, if the user enters 2514, the program should display 12, which is the sum of 2, 5, 1, and 4. The program should also display the highest and lowest digits in the string.

9. Most Frequent Character

    Write a function that accepts either a pointer to a C-string, or a `string` object, as its argument. The function should return the character that appears most frequently in the string. Demonstrate the function in a complete program.

10. `replaceSubstring` Function

    Write a function named `replaceSubstring`. The function should accept three C-string or `string` object arguments. Let's call them *string1*, *string2*, and *string3*. It should search *string1* for all occurrences of *string2*. When it finds an occurrence of *string2*, it should replace it with *string3*. For example, suppose the three arguments have the following values:

`string1`:	"the dog jumped over the fence"
`string2`:	"the"
`string3`:	"that"

    With these three arguments, the function would return a `string` object with the value "that dog jumped over that fence." Demonstrate the function in a complete program.

11. Case Manipulator

    Write a program with three functions: `upper`, `lower`, and `reverse`. The `upper` function should accept a pointer to a C-string as an argument. It should step through each character in the string, converting it to uppercase. The `lower` function, too, should accept a pointer to a C-string as an argument. It should step through each character in the string, converting it to lowercase. Like `upper` and `lower`, `reverse` should also accept a pointer to a string. As it steps through the string, it

should test each character to determine whether it is uppercase or lowercase. If a character is uppercase, it should be converted to lowercase. Likewise, if a character is lowercase, it should be converted to uppercase.

Test the functions by asking for a string in function `main`, then passing it to them in the following order: `reverse`, `lower`, and `upper`.

12. Password Verifier

    Imagine you are developing a software package that requires users to enter their own passwords. Your software requires that users' passwords meet the following criteria:

    - The password should be at least six characters long.
    - The password should contain at least one uppercase and at least one lowercase letter.
    - The password should have at least one digit.

    Write a program that asks for a password then verifies that it meets the stated criteria. If it doesn't, the program should display a message telling the user why.

13. Date Printer

    Write a program that reads a string from the user containing a date in the form mm/dd/yyyy. It should print the date in the form March 12, 2018.

14. Word Separator

    Write a program that accepts as input a sentence in which all of the words are run together, but the first character of each word is uppercase. Convert the sentence to a string in which the words are separated by spaces and only the first word starts with an uppercase letter. For example, the string "StopAndSmellTheRoses." would be converted to "Stop and smell the roses."

15. Character Analysis

    If you have downloaded this book's source code, you will find a file named `text.txt` in the Chapter 10 folder. Write a program that reads the file's contents and determines the following:

    - The number of uppercase letters in the file
    - The number of lowercase letters in the file
    - The number of digits in the file

16. Pig Latin

    Write a program that reads a sentence as input and converts each word to "Pig Latin." In one version, to convert a word to Pig Latin, you remove the first letter and place that letter at the end of the word. Then you append the string "ay" to the word. Here is an example:

English:	I SLEPT MOST OF THE NIGHT
Pig Latin:	IAY LEPTSAY OSTMAY FOAY HETAY IGHTNAY

17. Morse Code Converter

    Morse code is a code where each letter of the English alphabet, each digit, and various punctuation characters are represented by a series of dots and dashes. Table 10-10 shows part of the code.

    Write a program that asks the user to enter a string then converts that string to Morse code.

18. Phone Number List

    Write a program that has an array of at least 10 `string` objects that hold people's names and phone numbers. You may make up your own strings, or use the following:

    ```
    "Alejandra Cruz, 555-1223"
    "Joe Looney, 555-0097"
    "Geri Palmer, 555-8787"
    "Li Chen, 555-1212"
    "Holly Gaddis, 555-8878"
    "Sam Wiggins, 555-0998"
    "Bob Kain, 555-8712"
    "Tim Haynes, 555-7676"
    "Warren Gaddis, 555-9037"
    "Jean James, 555-4939"
    "Ron Palmer, 555-2783"
    ```

    The program should ask the user to enter a name or partial name to search for in the array. Any entries in the array that match the string entered should be displayed. For example, if the user enters "`Palmer`" the program should display the following names from the list:

    ```
    Geri Palmer, 555-8787
    Ron Palmer, 555-2783
    ```

19. Check Writer

    Write a program that displays a simulated paycheck. The program should ask the user to enter the date, the payee's name, and the amount of the check (up to $10,000). It should then display a simulated check with the dollar amount spelled out, as shown here:

	Date: 11/24/2018
Pay to the Order of: John Phillips	$1920.85

    One thousand nine hundred twenty and 85 cents

    Be sure to format the numeric value of the check in fixed-point notation with two decimal places of precision. Be sure the decimal place always displays, even when the number is zero or has no fractional part. Use either C-strings or `string` class objects in this program.

    Input Validation: Do not accept negative dollar amounts, or amounts over $10,000.

20. Lottery Statistics

    To play the PowerBall lottery, you buy a ticket that has five numbers in the range of 1–69, and a "PowerBall" number in the range of 1-26. (You can pick the numbers yourself, or you can let the ticket machine randomly pick them for you.) Then, on a specified date, a winning set of numbers are randomly selected by a machine. If your first five numbers match the first five winning numbers in any order, and your PowerBall number matches the winning PowerBall number, then you win the jackpot, which is a very large amount of money. If your numbers match only some of the winning numbers, you win a lesser amount, depending on how many of the winning numbers you have matched.

    In the student sample programs for this book, you will find a file named `pbnumbers.txt`, containing the winning lottery numbers that were selected between February 3, 2010 and May 11, 2016 (the file contains 654 sets of winning numbers). Here is an example of the first few lines of the file's contents:

```
17 22 36 37 52 24
14 22 52 54 59 04
05 08 29 37 38 34
10 14 30 40 51 01
07 08 19 26 36 15
```

and so on ...

Each line in the file contains the set of six numbers that were selected on a given date. The numbers are separated by a space, and the last number in each line is the PowerBall number for that day. For example, the first line in the file shows the numbers for February 3, 2010, which are 17, 22, 36, 37, 52, and the PowerBall number 24.

Write one or more programs that work with this file to perform the following:

- Display the 10 most common numbers, ordered by frequency.
- Display the 10 least common numbers, ordered by frequency.
- Display the 10 most overdue numbers (numbers that haven't been drawn in a long time), ordered from most overdue to least overdue.
- Display the frequency of each number 1-69, and the frequency of each Powerball number 1-26.

Table 10-10 Morse Code

CHARACTER	CODE	CHARACTER	CODE	CHARACTER	CODE	CHARACTER	CODE
space	*space*	`6`	`-....`	`G`	`--.`	`Q`	`--.-`
comma	`--..--`	`7`	`--...`	`H`	`....`	`R`	`.-.`
period	`.-.-.-`	`8`	`---..`	`I`	`..`	`S`	`...`
question mark	`..--..`	`9`	`----.`	`J`	`.---`	`T`	`-`
`0`	`-----`	`A`	`.-`	`K`	`-.-`	`U`	`..-`
`1`	`.----`	`B`	`-...`	`L`	`.-..`	`V`	`...-`
`2`	`..---`	`C`	`-.-.`	`M`	`--`	`W`	`.--`
`3`	`...--`	`D`	`-..`	`N`	`-.`	`X`	`-..-`
`4`	`....-`	`E`	`.`	`O`	`---`	`Y`	`-.--`
`5`	`.....`	`F`	`..-.`	`P`	`.--.`	`Z`	`--..`