5.5 Case Study: Decoding Phone Numbers

PROBLEM

To dial a telephone number, we use the telephone's keypad to enter a sequence of digits. For a long-distance call, the telephone system must divide this number into its area code, exchange, and local number; but if the number is a local call, it splits the number into its exchange and local number. For example, if a person enters 12345556789 in the United States, the system extracts the area code (234), the exchange (555), and the local number (6789); but if the person enters 5556789, the system extracts the exchange (555) and the local number (6789).

We want to develop a program that, given a phone number (a sequence of seven digits or a sequence of 11 digits beginning with 1), extracts and displays the area code (if a long distance number), the exchange, and the local number. For example, for the input 12345556789, the program should display (234)-555-6789; and for the input 5556789, it should display 555-6789.

OBJECT-CENTERED DESIGN

BEHAVIOR. The program should display a prompt for a phone number on the screen, and then read a phone number from the keyboard. It should check that the number is a valid telephone number. If the number begins with 1, the program should extract and display the area code, exchange, and local number. Otherwise, the program should extract and display the exchange and local number.

OBJECTS. From our behavioral description, we can identify the following objects:

	Software Objects		
Problem Objects	**Type**	**Kind**	**Name**
prompt	`string`	constant	*PROMPT*
phone number	`string`	varying	*phoneNumber*
screen	`ostream`	varying	`cout`
keyboard	`istream`	varying	`cin`
area code	`string`	varying	none
exchange	`string`	varying	none
local number	`string`	varying	none

Note that the phone number is not represented by an integer, because the maximum (`unsigned long`) integer value on 32-bit systems is 429496725, which is considerably less than a long distance number such as 12345556789. Thus, we will treat phone numbers as sequences of characters and use the type `string`to store them.

OPERATIONS. From the statement of the problem, we can identify these operations:

i. Display a string on the screen (the input prompt)
ii. Read a string from the keyboard (*phoneNumber*)
iii. Check that a string constitutes a valid phone number
iv. If the number begins with ‘1’:
   - **a.** Extract and display the second, third, and fourth characters from a string (the area code)
   - **b.** Extract and display the fifth, sixth, and seventh characters from a string (the exchange)
   - **c.** Extract and display the eighth through the eleventh characters from a string (the local number)

v. Otherwise
   - **a.** Extract and display the first, second, and third characters from a string (the exchange)
   - **b.** Extract and display the fourth through seventh characters from a string (the local number)

ALGORITHM. Assuming the availability of a function for each of the operations, we can arrange them into an algorithm for the problem.

Algorithm for Phone Number Decoder

```
1. Display PROMPT.
2. From cin, read phoneNumber.
3. Check that phoneNumber is a valid phone number.
4. If phoneNumber begins with 1:
     a. Display the area code = characters 1–3 of phoneNumber.
     b. Display the exchange = characters 4–6 of phoneNumber.
     c. Display the local number = characters 7–10 of phoneNumber.
5. Otherwise
     a. Display the exchange = characters 0–2 of phoneNumber.
     b. Display the local number = characters 3–6 of phoneNumber.
```

REFINEMENT. All of the operations except the third are provided in C++ or by the `string` class. We need to define a function to perform this operation.

FUNCTION’S PROBLEM. Write a function to check that the value stored in a string is a valid phone number.

FUNCTION’S BEHAVIOR. The function should receive a string from its caller and check that it contains either 7 or 11 characters. For a string with 11 characters, it should check that the first digit is 1. For a string with 7 characters, it should check that the first digit is not 1 or 0. It should also check that each of the characters in the string is a digit. For a string that fails any of these tests, the function should display an appropriate diagnostic message and terminate execution.

FUNCTION'S OBJECTS. From our behavioral description, we can identify the following objects for this function:

	Software Objects			
Problem Objects	**Type**	**Kind**	**Movement**	**Name**
a string	`string`	varying	received	*aString*
first digit of a string	`char`	constant	none	none

We will not name the first digit of a string, since it can be accessed directly using the subscript operator.

FUNCTION'S OPERATIONS. From our behavioral description, we can identify the following operations:

i. Check that the string contains either 7 characters or 11 characters
ii. If the string contains 11 characters, check that the first digit is a 1
iii. If the string contains 7 characters, check that the first digit is not 1 or 0
iv. Check that each of the characters in the string is a digit
v. If any of tests i–iv fail, display a diagnostic message and terminate the program

As we shall see, each of these operations is readily available, so we can go on to organize them into an algorithm.

FUNCTION'S ALGORITHM. With a bit of thought, we can arrange the preceding operations into the following algorithm:

Algorithm for Function that Checks a Phone Number

```
1. Receive aString.
2. If the size of aString is neither 7 nor 11
       Display a diagnostic message and terminate.
   Otherwise
       If aString has size 7 and begins with 0 or 1
           Display a diagnostic message and terminate.
       Otherwise if aString has size 11 and does not begin with 1
           Display a diagnostic message and terminate.
       Otherwise if there are any non-digit characters in aString
           Display a diagnostic message and terminate.
```

FUNCTION'S CODING. The following function `checkValidity()` implements this algorithm:

Case Study 5.5-1 Verifying Phone Numbers.

```
/* checkValidity ensures that a string is a valid phone number.
 *
 * Receive: aString, a string
 * Return:  nothing if aString is valid
 *          (terminate the program otherwise)
 ***************************************************************/

void phoneError(string message);

void checkValidity(string aString)
{
   if (aString.size() != 7 && aString.size() != 11)
      phoneError("A phone number has 7 or 11 characters");
   else
   {
      if (aString.size() == 7 &&
          (aString[0] == '0' || aString[0] == '1'))
          phoneError("A local call cannot begin with 0 or 1");
      else if (aString.size() == 11 && aString[0] != '1')
          phoneError("A long distance call must begin with 1");
      if (aString.find_first_not_of("0123456789",0) != string::npos)
         phoneError("A phone number must consist of all digits");
   }
}

/* phoneError displays an error msg and terminates the program
 *
 * Receive: message, a string.
 * Output:  message, with phone-specific text
 * Postcondition: Program has been terminated.
 **************************************************************/

#include <cstdlib>                    // exit()
using namespace std;

void phoneError(string message)
{
   cerr << "Your call cannot be completed because:\n"
        << message <<  ".\nPlease try again." << endl;
   exit(1);
}
```

Note the use of the various string operations. We use the `size()` function to check that the number of characters is correct,

```
if (aString.!= 7 && aString.!= 11)
```

the subscript operator to check that the first character of `aString` is correct,

```
if (aString.size() == 7 && (aString == '0' || aString == '1'))
```

and the `find_first_not_of()` function to search `aString` for non-digits:

```
if (aString.find_first_not_of("0123456789", 0)!= NPOS)
```

Also note that to "clean up" the function, we wrote a simple `PhoneError()` function which, given a diagnostic message, displays that message, and terminates the program using the `exit()` function provided in `cstdlib` This provides a convenient way to customize the diagnostic messages without cluttering the function with redundant code.

CODING AND TESTING. The following C++ program solves the original problem.

Case Study 5.5-2 Processing Phone Numbers.

```
/* simPhone.cpp simulates the processing of a phone number
 * by the telephone company.
 *
 * Input:  phoneNumber.
 * Output: areaCode, exchange and localNumber.
 ***************************************************************/

#include <iostream>              // cin, cout, >>, <<
#include <string>                  // string
using namespace std;

void checkValidity(string aString);

int main()
{
  const string PROMPT = "\nEnter a phone number: ";
  cout << PROMPT;

  string phoneNumber;
  cin >> phoneNumber;

  checkValidity(phoneNumber);

  if (phoneNumber[0] == '1')
     cout << '(' << phoneNumber.substr(1, 3) << ")-"
          << phoneNumber.substr(4, 3) << '-'
          << phoneNumber.substr(7, 4) << endl;
  else
     cout << phoneNumber.substr(0, 3) << '-'
          << phoneNumber.substr(3, 4) << endl;
}

/*** Insert the contents of Figure 5.2 here:
       the prototype for phoneError
       the definition of checkValidity
       #include <cstdlib>
       using namespace std;
       the definition of phoneError   ***/
```

Sample runs:

```
Enter a phone number 12345556789
(234)-555-6789

Enter a phone number 5556789
555-6789

Enter a phone number 1234

Your call cannot be completed because:
A phone number has 7 or 11 characters.
Please try again.
```