## WORKED EXAMPLE 5.1 Matching and Replacing Parts of a String

Searching and replacing text is a common word processor function. Your task is to write a function that replaces the first occurrence of one string with another in a given string. For example, when asked to replace "ss" with "n" in "Mississippi", the result is "Minissippi".

**Step 1** Describe what the function should do.

This has been given to us already.

**Step 2** Determine the function's "inputs".

There are three arguments:

- The string that is being edited (such as "Mississippi")
- The string that is being replaced (such as "ss")
- The replacement string (such as "n")

At this point, we have enough information to document the function:

```
/**
   Replaces the first occurrence of match in str with repl.
   @param str the string that is being edited
   @param match the string that is being replaced
   @param repl the replacement string
*/
```

**Step 3** Determine the types of the parameter variables and the return value.

The parameter variables are all strings. However, because the first parameter is being modified, its type is string&. The function modifies the first parameter, and it does not return a value. Therefore, the return type is void.

The function will be defined as

```
void replace_first(string& str, string match, string repl)
```

**Step 4** Write pseudocode for obtaining the desired result.

We first need to find the position in which match occurs. (If it doesn't occur anywhere, we do nothing.) For example, the string "ss" is found in position 2 in "Mississippi". Let us assume that we know that position, and call it i. Then the answer is obtained by concatenating

- the substring of str from 0 to i - 1
- the replacement string
- the substring of str from i + match.length() to the end

How do we find that position? In the spirit of stepwise refinement, we will delegate that task to another function, find_index. That starts a new sequence of steps, which, for greater clarity, we will place after the steps for this function.

**Step 5** Implement the function body.

We simply translate the plan into C++:

```
{
   int i = find_index(str, match);
   if (i == -1) { return; } // No match
   str = str.substr(0, i) + repl + str.substr(i + match.length());
}
```

**Step 6** Test your function.

We first test whether "Mississippi" is edited as expected, and then we add a couple of additional tests.

```
int main()
{
   string river = "Mississippi";
   replace_first(river, "ss", "n");
   cout << river << endl;
   cout << "Expected: Minissippi" << endl;
   replace_first(river, "ss", "n");
   cout << river << endl;
   cout << "Expected: Mininippi" << endl;
   replace_first(river, "ss", "n"); // No more match—should do nothing
   cout << river << endl;
   cout << "Expected: Mininippi" << endl;
   return 0;
}
```

### Repeat for the Helper Function

Now it is time to turn to the helper function for finding the match position.

**Step 1** Describe what the function should do.

It should find and return the first index in which a substring occurs in a given string, or return –1 if the substring doesn't occur anywhere.

**Step 2** Determine the function's "inputs".

There are two arguments:

- The string that is being searched (such as "Mississippi")
- The substring that is being matched (such as "ss")

At this point, we have enough information to document the function:

```
/**
   Finds the index of the first occurrence of match in str.
   @param str the string that is being searched
   @param match the substring that is being matched
   @return the first index at which match occurs in str, or -1 if it doesn't
   occur anywhere
*/
```

**Step 3** Determine the types of the parameter variables and the return value.

The parameter variables are both strings. Neither of them is being modified. Therefore, they are both value parameters. The return type is int. The function will be defined as

```
int find_index(string str, string match)
```

**Step 4** Write pseudocode for obtaining the desired result.

We start at index 0. Let n be the length of match. If str.substr(0, n) equals match, we have found our match. If not, we increment the index. If i denotes the current index, we test whether str.substr(i, n) equals match. If so, we return i. We keep incrementing i while there is still a hope of a match. Eventually, the unexamined tail of str is shorter than match, and we stop, returning –1.

**Step 5** Implement the function body.

We translate the plan into C++:

```
{
   int n = match.length();
   for (int i = 0; i <= str.length() - match.length(); i++)
   {
      if (str.substr(i, n) == match) { return i; }
   }
   return -1;
}
```

**Step 6** Test your function.

We first test whether the `"ss"` in `"Mississippi"` is located at the expected position, and then we try a string that doesn't occur.

```
int main()
{
   string river = "Mississippi";
   int n = find_index(river, "ss");
   cout << n << endl;
   cout << "Expected: 2" << endl;
   n = find_index(river, "tt");
   cout << n << endl;
   cout << "Expected: -1" << endl;
   return 0;
}
```

See `ch05/match.cpp` for the complete program with both functions.