5.2 The `string` Class: An Introduction to Classes and Objects

5.2.1 Familiar Classes and Objects

We have been working with several classes since the first chapter. In Chapter 1, we introduced the standard input and output streams `cin` and `cout`. The C++ language comes with several predefined classes. Two of these are the input class `istream` and the output class `ostream`. The standard input stream `cin` is an `istream` class object. The definition of the `istream` and `ostream` classes and the definitions of `cin` and `cout` are contained in the header file `iostream`.

In a program, we can obtain data from the keyboard by applying the extraction operator >> to the `cin` object. We can view the statement

```
cin >> price;
```

as follows: the statement tells the `cin` object to extract a number from the input stream and place the number's value into the variable `price`. Thus, we send a message to the `cin` object and it responds in a certain way to that message. How that message is carried out (that is, the mechanics of extracting the characters and converting them into a number), is immaterial to us (information hiding).

Likewise, we can display data on the monitor by applying the insertion operator << to the `cout` object. We can view the statement

```
cout << "The value is " << price;
```

as sending a message to the `cout` object to display on the monitor screen the string

"The value is" followed by the value of the variable `price`.

We have used `get()` with `cin` as follows, where ch is a variable of type `char`.

```
ch = cin.get();
```

The function `get()` is a method of the `istream` class. When we code `cin.get()`, we are sending the `get()` message to the `cin` object. In response to this message, the object returns to the program the value of the next character in the input buffer.

Note that we use the same dot operator that we use with structures.

The same is true about `getline()`. When we code

```
cin.getline(name, 81);
```

we are sending the `getline()` message to the `cin` object. In response to the message, the `cin` object extracts an entire line of text from the input buffer and places a copy of it into the array `name`. Again, it is no concern to us how the `cin` object obtains the line of characters and places them into the array. That is the responsibility of the object.

In the remainder of this chapter, we shall illustrate these object-oriented concepts by studying the built-in C++ class `string`. In the following chapters, we shall discuss how you can code your own classes and objects and use them in your programs.

5.2.2 Introduction to string Objects

We have already studied C-strings, which are character arrays. As we shall see, `string` class objects are much easier, safer, and in many ways, more natural to use than C-strings. It is not important for us to know how `string` objects or the operations on them are implemented. What is important is that you know how to use string objects. This is in contrast to using C-strings where it is very important that we know that such strings are implemented as character arrays.

The `string` class is built into C++, so there is no need to declare it in any way except that to use the class in your programs you must use the following `#include` statement.

```
#include <string>
```

5.2.3. Instantiating and Outputting `string` Objects

A string object represents a set of characters. Before we learn how to use string objects, we must know how to **instantiate**, or create, them. **Value of the string object** is the set of characters a string object represents.

Note first that the word `string` is not a C++ keyword it is the name of a class. (You can easily see this in most C++ editors because the word `string` will not be highlighted or color-coded as a keyword.)

In C++, any class, including `string`, is a valid data type. Therefore, you can declare `string` objects in much the same way as you declare numeric data types. Consider the following declarations.

```
Example 1
        string str1;
```

The first declaration defines `str1` as an empty, or null, a string that contains no characters. Therefore, an uninitialized `string` object is set to the empty string by default. This is in contrast to declaring an uninitialized integer or character pointer, which has a garbage value. When an object is created, it executes a special method, called a **constructor**. The main role of a constructor is to initialize the object.

Because no initial value is specified in the first declaration, the object executes the so-called **no-argument constructor**, which initializes the string's value to the empty string. It is called the no-argument constructor because we do not provide any argument (string literal).

Note: A string that is uninitialized in its declaration is automatically set to the empty, or null, string, that is, a string with no characters.

```
Example 2 and 3
        string str2 = "Hello"; //uses a one-argument constructor method

        string str3("Hello"); //uses a one-argument constructor method
```

The second and third declarations define `str2` and `str3` as a `string` object having the value `"Hello"` (without the quotation marks). In both of these cases, a **one-argument constructor** initializes the value of the string to `"Hello"`. Note how enclosing the string `"Hello"` in parentheses in the third declaration is similar to a function call, although `str3` is not the name of a function.

You might wonder if the string literal that you use to initialize a `string` object has too many characters to fit in the string's value. Although there is a limit to the number of characters that can be stored in a `string` object, that number is very large. Therefore, there is no practical limit to the number of characters that a `string` object can store.

```
Example 4
        string str4(25, '*'); // uses a two-argument string constructor
```

The fourth declaration defines `str4` as a string of 25 asterisks (*). This declaration uses a **two-argument constructor** to initialize the value of the string. The first argument (an integer) is the number of times the second argument (a character) is to be repeated in the string's value.

We see from these declarations that it is possible to instantiate and initialize a `string` object in several ways by using different `string` constructors.

5.2.4 What is a Constructor?

Constructors are important in instantiating an object. It is a function (method) where all the initializations are placed.

The following are the properties of a constructor:

- **Constructors have the same name as the class**
- **A constructor is just like an ordinary function(method), however only the following information can be placed in the header of the constructor,**
- **Constructors does not have any return value**

Note: You can output the value of a `string` object by sending the object to `cout` using the << operator.

The program dem051-1.cpp demonstrates the preceding declarations.

```
//dem051-0.cpp
//This program demonstrates
//different ways to declare
//string objects and how to output
//string objects.

#include <iostream>
#include <string>
using namespace std;
int main()
{

  string str1;
  string str2 = "Hello";
  string str3("Hello");
  string str4(25, '*');

  cout << "str1 = " << str1 << endl;
  cout << "str2 = " << str2 << endl;
  cout << "str3 = " << str3 << endl;
  cout << "str4 = " << str4 << endl;
  return 0;
}
```

```
str1 =
str2 = Hello
str3 = Hello
str4 = *************************
```

5.2.5. String Input

You can also input a `string` object's value using `cin` and the >> operator. For example, the following asks for a person's first name and stores it in the `string` object `first_name`.

```
string first_name;
cout << "Enter your first name: ";
cin >> first_name;
```

Program dem051-2.cpp shows how this works.

```
//dem051-2.cpp

//This program uses cin to input a
first and last name into a program.

#include <iostream>
#include <string>
```

```
Program Output

Enter your first name: John

Enter your last name: Molluzzo

Welcome, John Molluzzo
```

```

using namespace std;

int main()
{
  string first_name;
  string last_name;

  cout << "Enter your first name: ";
  cin >> first_name;

  cout << endl;
  cout << "Enter your last name: ";
  cin >> last_name;

  cout << endl << endl;
  cout << "Welcome, " << first_name
<< " " << last_name << endl;

  return 0;
}


```

In the last `cout` statement, we place a space as a string literal,"", into the output stream between the first and last names.

Now, what happens if I enter my first name and my middle initial when prompted to enter my first name? The following shows the output when I do that.

Program Output When Entering Two-Word First Name

```
Enter your first name: John C.

Enter your last name:

Welcome, John C.
```

When hitting the enter key after entering `John C`., the program does not wait for me to enter my last name, but just goes on to display the welcome message. The program behaves like this because when entering string data, `cin` uses white space (a blank, an enter, or a tab) to delimit strings. That is, `cin` can be used to input only one word at a time. It cannot be used to input a multi-word string.

To input a multi-word string, we must use the function `getline.` Suppose we want to ask the user in what state he or she lives. Because some province names, such as North Cotabato, are two words, we have to use `getline()` to extract the state name from the input stream. The following code will do that.

```
string state;

cout << "Enter your state of residence: ";

getline(cin,state);
```

Think of the `getline()` statement as saying the following: "Get a line of text from the `cin` input stream and place it into the value of the `string` object `state`." The `getline()` function asks `cin` to retrieve one line from the input stream that is, everything you type until you hit the enter key. The text you typed (not including the enter character) is then given to the `string` object `state`, which takes the text as its value. Note that `getline()` is not a method of the `string` class. Following is the program `dem10-3.cpp`, which demonstrates these concepts.

When running the program, multi-word responses are made to the two prompts. In each case, `getline()` retrieves the entire line entered and passes a copy of the line to the corresponding `string` object.

Note: Use `cin >> str1` to input one word into the string variable `str1`. Use `getline(cin, str1)` to input one line from `cin` into the string variable `str1`.

Program dem051-3.cpp shows how this works.

```
//dem051-3.cpp

//This program illustrates the use
of getline()
//to input multi-word strings.

#include <iostream>
#include <string>

using namespace std;

int main()
{
  string full_name;
  string state_name;

  cout << "Enter your full name: ";
  getline(cin,full_name);

  cout << endl;
```

```
Enter your full name: John C. Molluzzo

Enter your state of residence: New York

Welcome, John C. Molluzzo
New York is a nice place to live.
```

```
  cout << "Enter your state of
residence: ";
  getline(cin,state_name);

  cout << endl << endl;
  cout << "Welcome, " << full_name
<< endl;
  cout << state_name << " is a nice
place to live." << endl;

  return 0;
 }
```

5.2.5 Operations on `string` Objects

The `string` class has many methods that allow you to get information about a string and to manipulate a `string` object's value. Also, you can use many familiar operators on `string` objects in much the same way as you use them on numeric data types.

5.2.6 String Assignment

It is legal to assign one `string` object to another as follows.

String assignments	

`string str1 = "Hello";`
`string str2;`
 	
`str2 = str1;`
	The value of the `string` object `str2` is a copy of the value of the `string` object `str1`. Therefore, assigning one `string` object to another copies the value of the string on the right of the assignment to the value of the string on the left.

`string str1 = "Hello";`
`string str2 = str1;`
	You can also initialize one `string` object to another in a declaration.

`string str1 = "Hello";`
`string str2(str1);`
	You can also initialize one `string` object to another using the this technique

5.2.7. Concatenation

You can also paste (or **concatenate**) two strings together using the + operator, the same operator that we use for the addition of numbers.

In C++, it is possible to make an operator, such as the addition operator (+), do different things depending on the types of data the operator is working on. This is called **operator overloading**. Thus, the addition operator in the expression `i + j`, where `i` and `j` are integers, will work differently than the addition operator in the expression `str1` + `str2` of the previous example. In the first case, the + is operating on integers. Therefore, C++ does integer addition. In the second case, the + is operating on `string` objects. Therefore, C++ does concatenation. Overloading addition is not done automatically for any combination of objects. The overloading must be coded somewhere. In the case of the `string` class, the overload is coded in the `<string>` header file.

The += operator also has meaning (that is, is overloaded) for `string` objects and has the effect of appending the value of the string on the right of += to the value of the string on the left of +=.

You can read the += when operating on strings as "is appended with." The following code has the same result as the previous example.

Concatenator	

`string str1 = "Hello, "`
`string str2 = "World!"`
`string str3 = str1 + str2;`
	Declares the string str1 with an initial value of "Hello", the same with str2 which assigns a value "World". After which it declare the string str3 that assigns the concatenated value of str2 and str1.

`string str3 = "Hello, ";`
 	
`str3 += "World!";`
	The declaration initializes `str3` to the string `"Hello, "`. The second statement is equivalent to the assignment `str3 = str3 + "World!";`. Thus, the string `"World!"` is appended to the end of the existing string, which results in `str3` containing the string `"Hello, World!"`.

Overloaded operators for string Objects	
= (assignment operator)	The assignment operator assigns the value of one string to the value of another.
+ operator	The + operator concatenates together the string values of its operands.
+= (compound assignment operator)	The compound assignment operator += appends the value of the string on the right of the operator to the end of the value of the string on the left of the operator.

5.2.8. Applying a Class Method

To apply a class method to an object, use the **dot (.) operator** as follows:

```
class_object.method_name()
```

This sends a message, namely `method_name`, to the object `class_object`, which in turn executes the method.

For example, to find the number of characters in the value of a `string` object (that is, the length of the string), use the `length()` method, which is a method of the `string` class. Recall that a method represents a message, or request, that is sent to an object to do something. To apply a method to an object, you must use the **dot (.) operator** as follows.

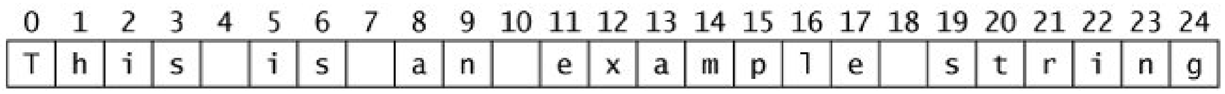
```
str1.length()
```

This statement sends the `length()` message to the `string` object `str1`: "Compute your length and return its value."

In this section, we discuss some useful `string` class methods that allow you to manipulate `string` objects.

Recall that the character positions in a `string` object's value are numbered from 0 to one less than the number of characters in the string's value. Figure 1 shows a `string` object's value with the character positions numbered.

Figure 1



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
T	h	i	s		i	s		a	n		e	x	a	m	p	l	e		s	t	r	i	n	g

We will now discuss several `string` class methods that we will use to manipulate the string in Figure 1.

string Class method	Meaning	Syntax	Example Declaration	Output
length()	Return the numeric length of string.	length()	`string str=”This is an example string”`	cout << str.length(); will return 15
`replace()`	`Allows to replace one substring by another.`	`replace(int,int,string)` `where: it replace no. of characters in the second argument beginning`	`string str=”This is an example string”`	`str.replace(0,4,”Here”)` `cout<<str;` `will return “Here is an example of string”` `str.replace(18,0,” of a “);` `cout<<str;` `will return “This is an example of a`

		at location in the first argument by the string in the third argument		string”
insert()	Places a string at a certain location in another string with the characters that follow being moved to the right.	insert(int ,string)	string str=”This is an example string”	str.insert(19,”of a “); will return “This is an example of a string” as it moved the word “string” that is at position 19 and insert “of a “ before it. (Start counting at left and 0 as you first index position)
append()	Allows you to add a string to the end of the string to which you apply the method.	append(str ing)	string str=”This is an example string”	str.append(“.”) will return “This is an example string.” Note that “.” is already appended at the end of the string.
erase()	Erase a substring from a string.	erase(int, int) where: first int argument is the starting position and second int argument is the number of characters to erase.	string str=”This is an example string”	str.erase(5,3); will return “This an example of string” as it remove the “is “(including the space after is) because it start erasing at position 5 and deletes 3 characters.
empty()	Returns true if string is empty otherwise returns false	empty()	string str1; string str2=”Hello”;	str1.empty() will return true str2.empty() will return false
find_first_of()	Find and return the index location of	find_first _of(string ,[int])	string str=”Hello, how are you today?”;	str.find_first_of(punc); will return 5 since “,” is the first punctuation

	the first occurrence of any of a set of characters. It return -1 if it does not find any in the set of characters.	Note: int argument is optional that refers where to start the searching.	string punc=".,!?:;";	found in the set that is in the string str.find_first_of(punc,7); will return 24 since "?" is next punctuation in the set that is in the string
substr()	Extracts strings starting from the 1st argument, the number of characters to extracts depends on the 2nd argument specified	substr(int,int)	string str="C++ Programming"	str.substr(0,3); will return "C++" str.substr(7,4); will return "gram" str.substr(12,2); will return "in"
find()	Find first occurrence in string and returns it index position. If the string is not found it returns -1.	find(string)	string str="C++ Programming"	str.find("+"); will return 1 str.find("Programs"); will return -1 since Programs is not found in str
at()	Retrieve or change an individual character in a string object	at(int) where: int represents index in string starting at 0 up to size - 1	string str="C++ Programming"	cout << str.at(4); will return P str.at(7) = "Z" cout << str; will return C++ ProZramming; replace the letter "g" with letter "Z"

5.2.9 Program dem051-4.cpp

Program dem051-4.cpp illustrates all of the methods discussed in this section.

```cpp
//dem051-4.cpp

//This program illustrates some of methods that are available
//to manipulate string objects.

#include <iostream>
#include <string>

using namespace std;

int main()
{
    string str1 = "This is an example string";
    string str2 = "sample";
    string str3;
    string punctuation = ".,:;!?";

    int position;

    cout << "The original string, str1, is : "
         << str1 << endl << endl;

    //Finds the first occurrence of "is" in str1
    position = str1.find("is");

    cout << "The first occurrence of \"is\" is at position "
         << position << endl << endl;

    //Finds the first occurrence of "is" in str1 after position 3
    position = str1.find("is", 3);

    cout << "The first occurrence of \"is\" after position 3 is "
         << "at position " << position << endl << endl;

    //Finds the first occurrence of "is" from the end of str1
    position = str1.rfind("is");

    cout << "The first occurrence of \"is\" from the end of str1 is "
         << "at position " << position << endl << endl;

    //Try to find a substring that is not in str1.
    position = str1.find("Hello");

    if (position == -1)
        cerr << "The substring \"Hello\" was not found in str1."
             << endl << endl;

    //Extracts a 8-character substring from str1 beginning at position 11
    str3 = str1.substr(11, 8);
    cout << "The 8-charater substring of str1 beginning at position 11 is: "
         << str3 << endl << endl;

    //Replace 7 characters beginning at position 11 with the string str2
    str1.replace(11, 7, str2);

    cout << "After replacing the word \"example\" " << endl
```

```
         << "by the word \"sample\", in str1 the string is: "
         << str1 << endl << endl;

    //Erases a 7-character substring from str1 beginning at position 11
    str1.erase(11, 7);

    cout << "After erasing the word \"sample \", in str1 the string is: "
         << str1 << endl << endl;

    //Insert the string "example " into the string beginning at position 11
    str1.insert(11, "example ");

    cout << "After putting back the word \"example \", in str1 the string is: "
         << endl << str1 << endl << endl;

    //Append a period at the end of the string
    str1.append(".");

    cout << "After appending a period, the string str1 is: "
         << str1 << endl << endl;

    //Finds the first occurrence of a punctuation mark in str1
    position =     str1.find_first_of(punctuation);

    cout << "The first punctuation mark in str1 is "
         << "at position " << position << endl << endl;
    return 0;
}

```

Program Output

```
  The original string, str1, is : This is an example
   string

The first occurrence of "is" is at position 2

The first occurrence of "is" after position 3 is
   at position 5

The first occurrence of "is" from the end of str1
   is at position 5

The substring "Hello" was not found in str1.

  The 8-charater substring of str1 beginning at
   position 11 is: example

After replacing the word "example"
```

Program Output

```
by the word "sample", in str1 the string is: This
   is an sample string

  After erasing the word "sample ", in str1 the
   string is: This is an string

After putting back the word "example ", in str1
   the string is:
This is an example string

  After appending a period, the string str1 is: This
   is an example string.

The first punctuation mark in str1 is at position 25
```

The program comments and output explain the code. Note that the program uses the escape sequence `\"` several times to allow `cout` to display the quotes character,`"`, inside a quoted string.