05: The C++ string class

Programming Technique II

(SCSJ1023)

Adapted from Tony Gaddis and Barret Krupnow (2016), Starting out with C++: From Control Structures through Objects

(Last update: Oct. 7, 2018)

The C++ string Class

- The `string` class offers several advantages over C-style strings
  - large body of member functions
  - overloaded operators to simplify expressions

- Special data type supports working strings

```
#include <string>
```

String output - Example

Program 10-15

```
1  // This program demonstrates the string class.
2  #include <iostream>
3  #include <string>    // Required for the string class.
4  using namespace std;
5
6  int main()
7  {
8     string movieTitle;
9
10    movieTitle = "Wheels of Fury";
11    cout << "My favorite movie is " << movieTitle << endl;
12    return 0;
13 }
```

Program Output

```
My favorite movie is Wheels of Fury
```

The C++ string Class

- Can define string variables in programs:

```
string firstName, lastName;
```

- Can receive values with assignment operator:

```
firstName = "George";
lastName = "Washington";
```

- Can be displayed via cout

```
cout << firstName << " " << lastName;
```

The C++ string Class

- Objects of type string can be declared and initialized in several ways:

```
string s1;
string s2 = "New York";
string s3(60, '*');
string s4 = s3;
string s5 = 'N';  //Error. s5 must hold a string, not a character
string s6 =78;    //Error.
```

- If the string is not initialized, it represents the empty string.

Other ways to define C++ strings

Definition	Meaning
`string name;`	defines an empty string object
`string myName("Abu Bakar");`	defines a string and initializes it with a literal. `myName: "Abu Bakar"`
`string yourName(myName);`	defines a string and initializes it with values from other string. `yourName: "Abu Bakar"`
`string aName(myName, 5);`	defines a string and initializes it with string from `myName` starting at position 5 (i.e., at 6th char.) `aName: "akar"`
`string newName(myname,0,3);`	defines a string and initializes it with 3 characters from `myname` starting at position 0 `aName: "Abu"`
`string noName(5, 'A');`	defines string and initializes it to 5 `'A'`s `noName: "AAAAA"`

Input into a `string` Object

- Use **`cin >>`** to read an item into a string:

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

String Input - example

Program 10-16

```
1  // This program demonstrates how cin can read a string into
2  // a string class object.
3  #include <iostream>
4  #include <string>
5  using namespace std;
6
7  int main()
8  {
9     string name;
10
11    cout << "What is your name? ";
12    cin >> name;
13    cout << "Good morning " << name << endl;
14    return 0;
15 }
```

Program Output with Example Input Shown in Bold

```
What is your name? Peggy [Enter]
Good morning Peggy
```

String input : cin

- When the input stream **`cin`** is used, white space characters (space, tab, newline etc.) are used
  - Separators – read next data
  - Terminators – terminate reading a value

String input getline()

- Use **`getline()`** to read a line of input (with spaces) into a string object
- The **`getline()`** reads from the current position in the input stream until a newline character is found
- The **`getline()`** takes 2 parameters:
  - specifies an input stream eg. **`cin`** or file name
  - specifies a string variable
- Example :

```
getline(cin, name)
```

String input : getline()

- The **`getline()`** with 3 parameters
  - 1st and 2nd as the **`getline()`** with 2 parameters
  - 3rd specifies the terminator character i.e. the character at which **`getline()`** will stop reading from the input stream

- Example :

```
iFile.open("String1.dat");
getline(iFIle, name, '\t');
```

string operator

```
string s3, s4;
string s1("Hi");
string s2(" Dad");

s3 = s1 + s2;          // string concatenation
s4 = "Hello Mom!"      // string assignment
```

Overloaded string Operators

OPERATOR	MEANING
>>	reads whitespace-delimited strings into string object
<<	outputs string object to a stream
=	assigns string on right to string object on left
+=	appends string on right to end of contents of string on left

continued …

OPERATOR	MEANING
`+`	concatenates two strings
`[]`	references character in string using array notation
`>, >=, <, <=, ==, !=`	relational operators for string comparison. Return **`true`** or **`false`**

continued …

```
string word1, phrase;
string word2 = " Dog";
cin >> word1;  // user enters "Hot“, word1 has "Hot"
phrase = word1 + word2;  // phrase has "Hot Dog"

phrase += " on a bun";

cout << phrase <<endl;// displays "Hot Dog on a bun"
```

string Methods

- Are behind many overloaded operators

- Categories
  - assignment : **`assign, copy, data`**
  - modification : **`append, clear, erase, insert, replace, swap`**
  - space management : **`capacity, empty, length, resize, size`**
  - substrings : **`find, substr`**
  - comparison : **`compare`**

- See Table 10-7 for A list of functions

Table 10-7

Member Function Example	Description
`theString.append(str);`	Appends `str` to `theString`. `str` can be a `string` object or character array.
`theString.append(str, x, n);`	n number of characters from `str`, starting at position `x`, are appended to `theString`. If `theString` is too small, the function will copy as many characters as possible.
`theString.append(str, n);`	The first n characters of the character array `str` are appended to `theString`.
`theString.append(n, 'z');`	Appends n copies of 'z' to `theString`.
`theString.assign(str);`	Assigns `str` to `theString`. `str` can be a `string` object or character array.
`theString.assign(str, x, n);`	n number of characters from `str`, starting at position `x`, are assigned to `theString`. If `theString` is too small, the function will copy as many characters as possible.
`theString.assign(str, n);`	The first n characters of the character array `str` are assigned to `theString`.
`theString.assign(n, 'z');`	Assigns n copies of `'z'` to `theString`.
`theString.at(x);`	Returns the character at position `x` in the `string`.
`theString.begin();`	Returns an iterator pointing to the first character in the `string`. (For more information on iterators, see Chapter 15.)
`theString.capacity();`	Returns the size of the storage allocated for the `string`.
`theString.clear();`	Clears the `string` by deleting all the characters stored in it.

(table continues)

Display 9.7 **Member Functions of the Standard Class string**

EXAMPLE	REMARKS
Constructors	
`string str;`	Default constructor; creates empty `string` object `str`.
`string str("string");`	Creates a `string` object with data `"string"`.
`string str(aString);`	Creates a `string` object `str` that is a copy of `aString`. `aString` is an object of the class `string`.
Element access	
`str[i]`	Returns read/write reference to character in `str` at index `i`.
`str.at(i)`	Returns read/write reference to character in `str` at index `i`.
`str.substr(position, length)`	Returns the substring of the calling object starting at `position` and having `length` characters.
Assignment/Modifiers	
`str1 = str2;`	Allocates space and initializes it to `str2`'s data, releases memory allocated for `str1`, and sets `str1`'s size to that of `str2`.
`str1 += str2;`	Character data of `str2` is concatenated to the end of `str1`; the size is set appropriately.
`str.empty( )`	Returns `true` if `str` is an empty `string`; returns `false` otherwise.
`str1 + str2`	Returns a `string` that has `str2`'s data concatenated to the end of `str1`'s data. The size is set appropriately.
`str.insert(pos, str2)`	Inserts `str2` into `str` beginning at position `pos`.
`str.remove(pos, length)`	Removes substring of size `length`, starting at position `pos`.
Comparisons	
`str1 == str2` `str1 != str2`	Compare for equality or inequality; returns a Boolean value.
`str1 < str2` `str1 > str2` `str1 <= str2` `str1 >= str2`	Four comparisons. All are lexicographical comparisons.
`str.find(str1)`	Returns index of the first occurrence of `str1` in `str`.
`str.find(str1, pos)`	Returns index of the first occurrence of string `str1` in `str`; the search starts at position `pos`.
`str.find_first_of(str1, pos)`	Returns the index of the first instance in `str` of any character in `str1`, starting the search at position `pos`.
`str.find_first_not_of (str1, pos)`	Returns the index of the first instance in `str` of any character *not* in `str1`, starting search at position `pos`.

`theString.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.
`theString.compare(x, n, str);`	Compares `theString` 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.
`theString.copy(str, x, n);`	Copies the character array `str` to `theString`, beginning at position `x`, for `n` characters. If `theString` is too small, the function will copy as many characters as possible.
`theString.data();`	Returns a character array containing a null terminated string, as stored in `theString`.
`theString.empty();`	Returns true if `theString` is empty.
`theString.end();`	Returns an iterator pointing to the last character of the string in `theString`. (For more information on iterators, see Chapter 15.)
`theString.erase(x, n);`	Erases `n` characters from `theString`, beginning at position `x`.
`theString.find(str, x);`	Returns the first position at or beyond position `x` where the string `str` is found in `theString`. `str` may be either a `string` object or a character array.
`theString.find('z', x);`	Returns the first position at or beyond position `x` where '`z`' is found in `theString`.
`theString.insert(x, str);`	Inserts a copy of `str` into `theString`, beginning at position `x`, `str` may be either a `string` object or a character array.
`theString.insert(x, n, 'z');`	Inserts '`z`' `n` times into `theString` at position `x`.
`theString.length();`	Returns the length of the string in `theString`.
`theString.replace(x, n, str);`	Replaces the `n` characters in `theString` beginning at position `x` with the characters in string object `str`.
`theString.resize(n, 'z');`	Changes the size of the allocation in `theString` 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.
`theString.size();`	Returns the length of the string in `theString`.
`theString.substr(x, n);`	Returns a copy of a substring. The substring is `n` characters long and begins at position `x` of `theString`.
`theString.swap(str);`	Swaps the contents of `theString` with `str`.

Length of a String

- The number of characters it contains including whitespace characters, if any.

- Example:

```
string s1 = "UTM Skudai";
int len = s1.length(); // len is 10
```

Modification of string objects

- Appending strings (to the end of a string)

```
string str1 = "UTM ";
string str2 = "Skudai ";

str1.append(str2);    // str1 becomes "UTM Skudai "
str2.append(str1);    // str2 becomes "Skudai UTM Skudai"
```

- Inserting strings (within a string)

```
string str3 = "Skudai ";
string str4("UTM Johor");  // str4 has "UTM Johor"
// insert str3 into str4 starting at position 4
str4.insert(4, str3);  // str4 becomes "UTM Skudai Johor"
```

Substrings

- `substr` method returns a substring

- syntax is :

```
s.substr(position, length);
```

- Example:

```
string name = "objects";
// copy a sub string from name starting at position 2 and taking 3 characters
string sub = name.substr(2, 3); // sub has "jec"

// copy a sub string from name starting at position 3
sub = name.substr(3, 15);       // sub has "ects"
```