Advanced Object Oriented Programming

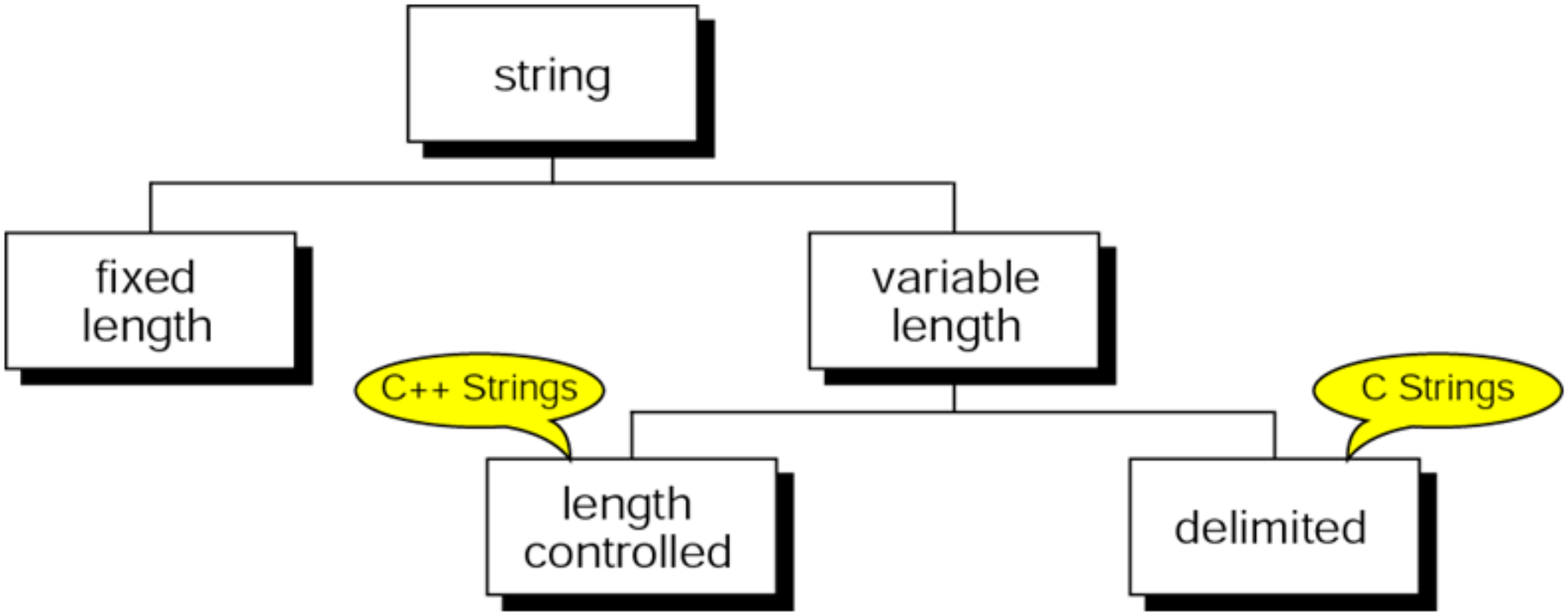
Strings

Seokhee Jeon
Department of Computer Engineering
Kyung Hee University
jeon@khu.ac.kr

String

- A String is a series of characters treated as a unit
- Examples:
  - “Dog”, “Steve Jobs”, “세종대왕”

String taxonomy



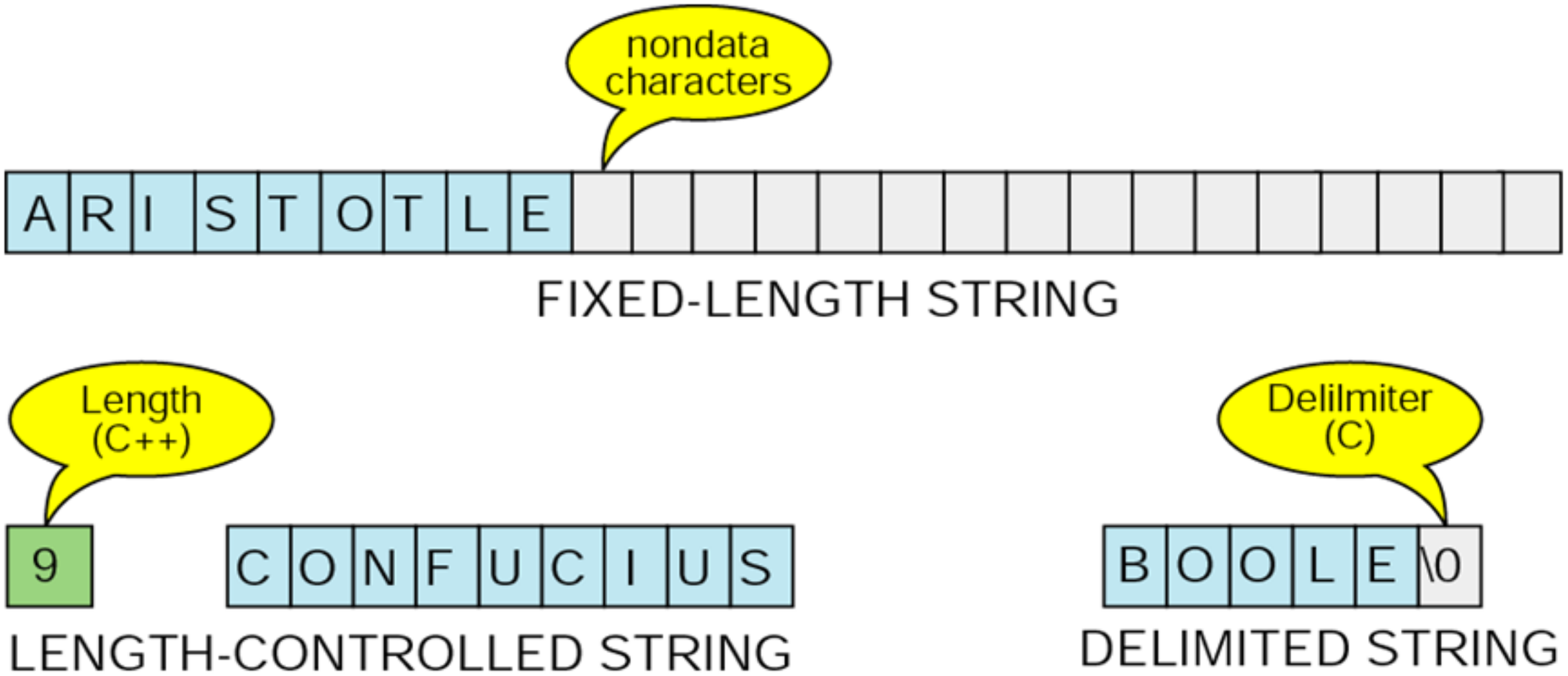
Fixed-length strings

- A fixed-length string is implemented as an array of characters
- We must first decide what size to make the variable
- Problem: how to tell the data from the nondata

Variable-length strings

- Create a structure that can expand and contract to accommodate the data
- Length-Controlled Strings
  - Add a count that specifies the number of characters in the string
  - The amount of bytes used for the count determines the max length of possible strings
- Delimited Strings
  - Add a delimiter to identify the end of the string
  - It eliminates one character from being used for data

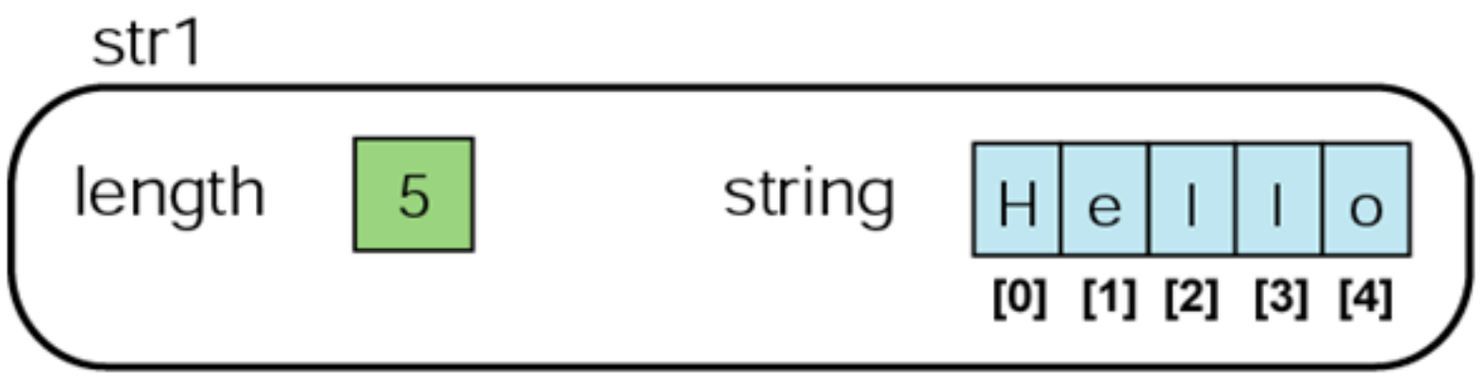
String formats



C++ String (It's a Class)

- A C++ string is a sequence of characters implemented as a length-controlled string object (an instantiation of the string class)
- The C++ name for the string class is basic_string
- Within the basic string class is a type definition for the type string, which equates the two

C++ string (It's a Class)



String constructors

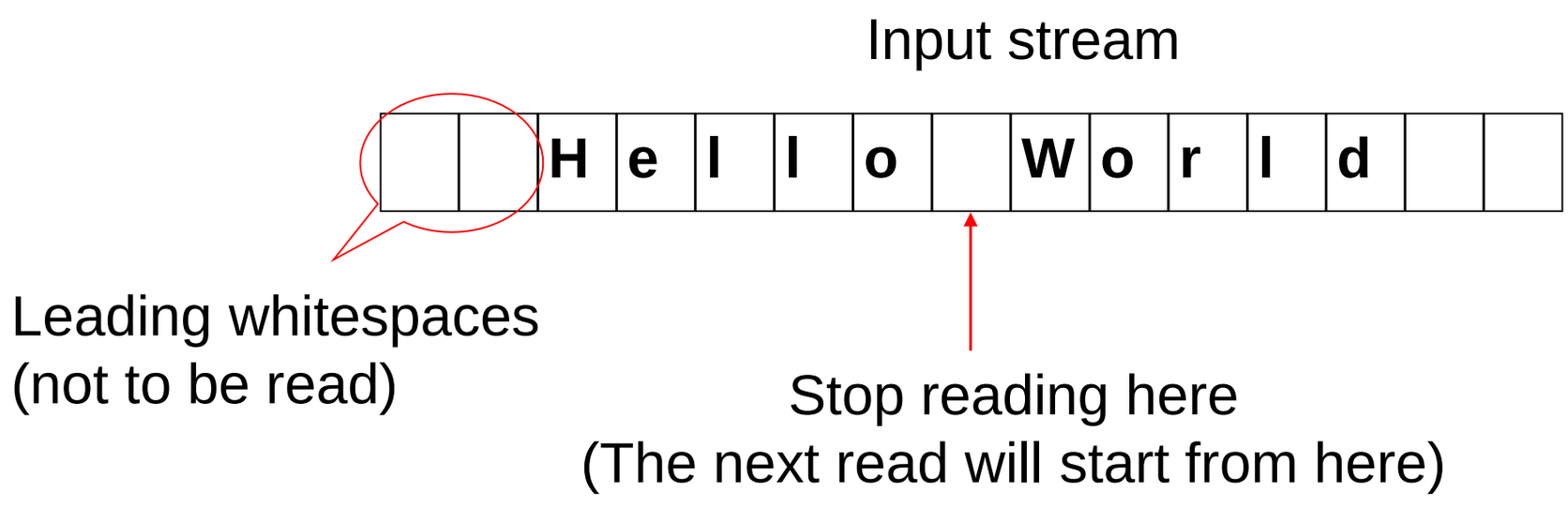
Constructor format	Operation
string s1;	Default constructor (empty string)
string s2("Hello World");	Initialization constructor using C string
string s3(num,'c');	Initialization constructor using *num* identical characters
string s4(s2);	Copy constructor
string s5(s2, num);	Copy constructor that copies *num* characters from beginning of string
string s6(s2, start, num);	Copy constructor that copies *num* characters from index location *start* in s2
string s7("Hello", num);	Initialization constructor using the first *num* characters of the C string
string s8("Hello", start, num);	Same as s6, but with C string

C++ String Input/Output

- The string class is overloaded for the insertion and extraction operators
  - We can read a string just like any other variables
- String output (<<)
  - E.g., `cout << month;` or `fsOut << month;`
- String input (>>)
  - E.g., `cout >> month;` or `fsIn >> month;`

String extraction operator as a ‘cin >>’

- Skips any leading whitespace
- Extracts all contiguous non-whitespace characters
- Stops at any whitespace character
- The terminating whitespace character is left in the input stream
- Example:



String extraction operator as a ‘cin >>’

The extraction operator stops at whitespace.

To read a string with spaces, we must use getline.

‘getline’ function

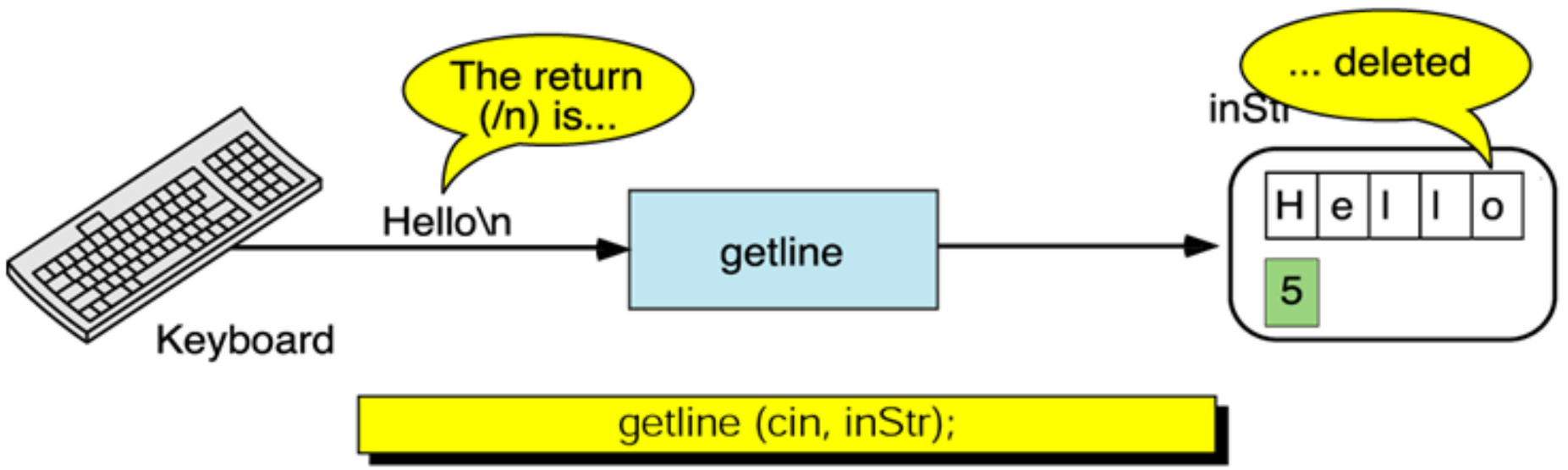
- All characters, including whitespace, are read into the string until the terminating character is found
- The terminating character, usually a new line, is deleted (extracted and discarded)
- The getline function is a stand-alone function (not a class member)

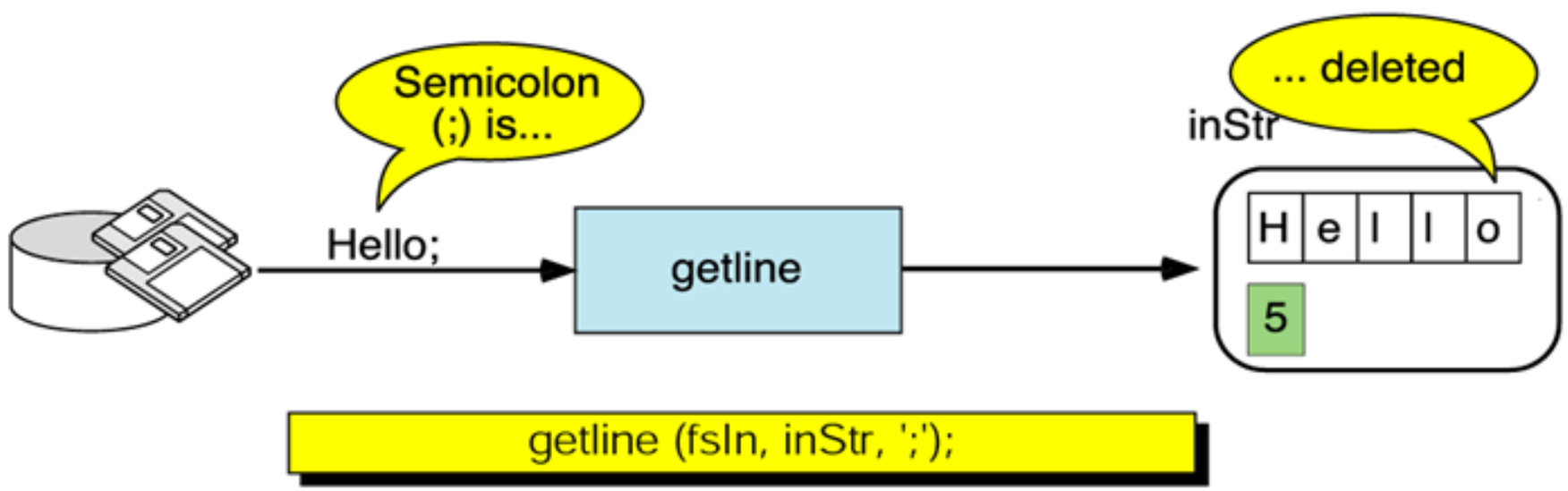
Overloaded functions

getline(stream, StringName); // Delimiter : ‘\n’

getline(stream, StringName, Delimiter);

‘getline’ function





Program 14-2 Demonstrate *getline* operation

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

int main ()
{
   cout << "Enter a name in the form <last,first>: \n";
   string  lastName;
   getline (cin, lastName, ',');
   string  firstName;
   getline (cin, firstName);

   cout << "Here is your name:\n\t|"
        << firstName << ' '
         << lastName << "|\n";
   return 0;
}  // main
```

The string input /output operators and functions are defined in the string header file, not the I/O stream header file.

```
/*       Results:
Enter a name in the form <last,first>:
Washington,George
Here is your name:
   |George Washington|

User Error: spaces after comma
Enter a name in the form <last,first>:
Washington,   George
Here is your name:
   |   George Washington|
*/
```

Assignment operator

- Overloaded for three source types:
  - The value of a C++ string
  - The value of a C string
  - A single character

Program 14-3 Demonstrate string assignment

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

int main ()
{
    string str1 ("String 1");
    string str2;
    string str3;
    string str4;
    string str5 = "String 5";

    cout << "String 1: " << str1 << endl;
    str2 = str1;
    cout << "String 2: " << str2 << endl;
    str3 = "Hello";
    cout << "String 3: " << str3 << endl;
    str4 = 'A';
    cout << "String 4: " << str4 << endl;
    cout << "String 5: " << str5 << endl;
    return 0;
}   // main
```

```
/*      Results:
String 1: String 1
String 2: String 1
String 3: Hello
String 4: A
String 5: String 5
*/
```

Assignment vs. Copy Constructor

- String s1 = ”Hello”; // correct
- String s2 = ’a’; // error
  - Compare with String s2 = ’a’, which is correct

Example: Array of Strings

- Of course, strings can be used in an array

```
int main ()
{
        string daysAry[7];  // declaration of an array of strings

        daysAry[0] = "Sunday";
        daysAry[1] = "Monday";
        daysAry[2] = "Tuesday";
        daysAry[3] = "Wednesday";
        daysAry[4] = "Thursday";
        daysAry[5] = "Friday";
        daysAry[6] = "Saturday";

        cout << "\nThe days of the week\n";
        for (int daysIndex = 0; daysIndex < 7; daysIndex++)
           cout << daysAry[daysIndex] << endl;
        return 0;
} // main
```

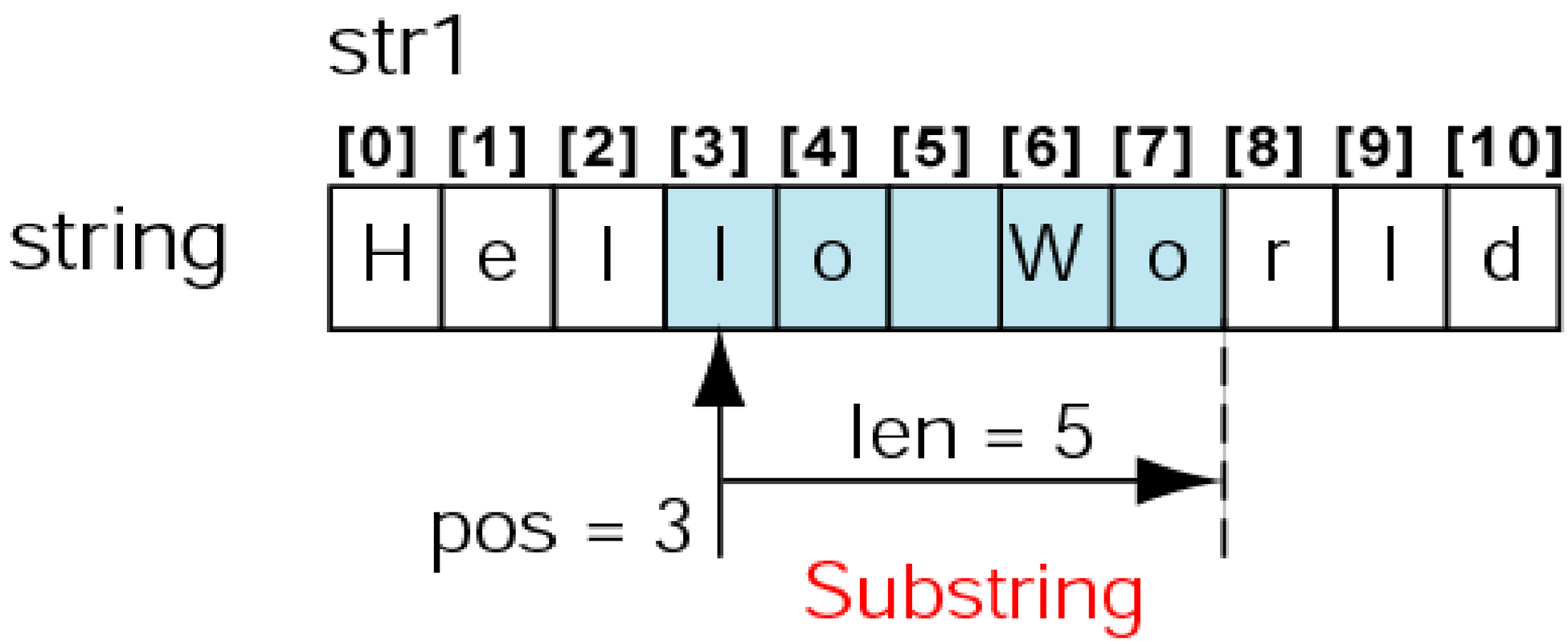
```
/*      Results

The days of the week
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
*/
```

String Manipulation Function

- A rich set of methods that can be used to manipulate strings
- Make it easier for us to write programs

Substring

- A contiguous set of characters within a string
- Identfied by ***a start position and a length***



Extracting a Substring

- Creates a new string by extracting part of a string

str1.substr(starting_pos, len)

- str1.substr(starting_pos): length defaults to *string::npos*, a constant defining the maximum length

String Length (*length* and *size*)

- Returns the length of a string, which is defined as the number of characters in the string

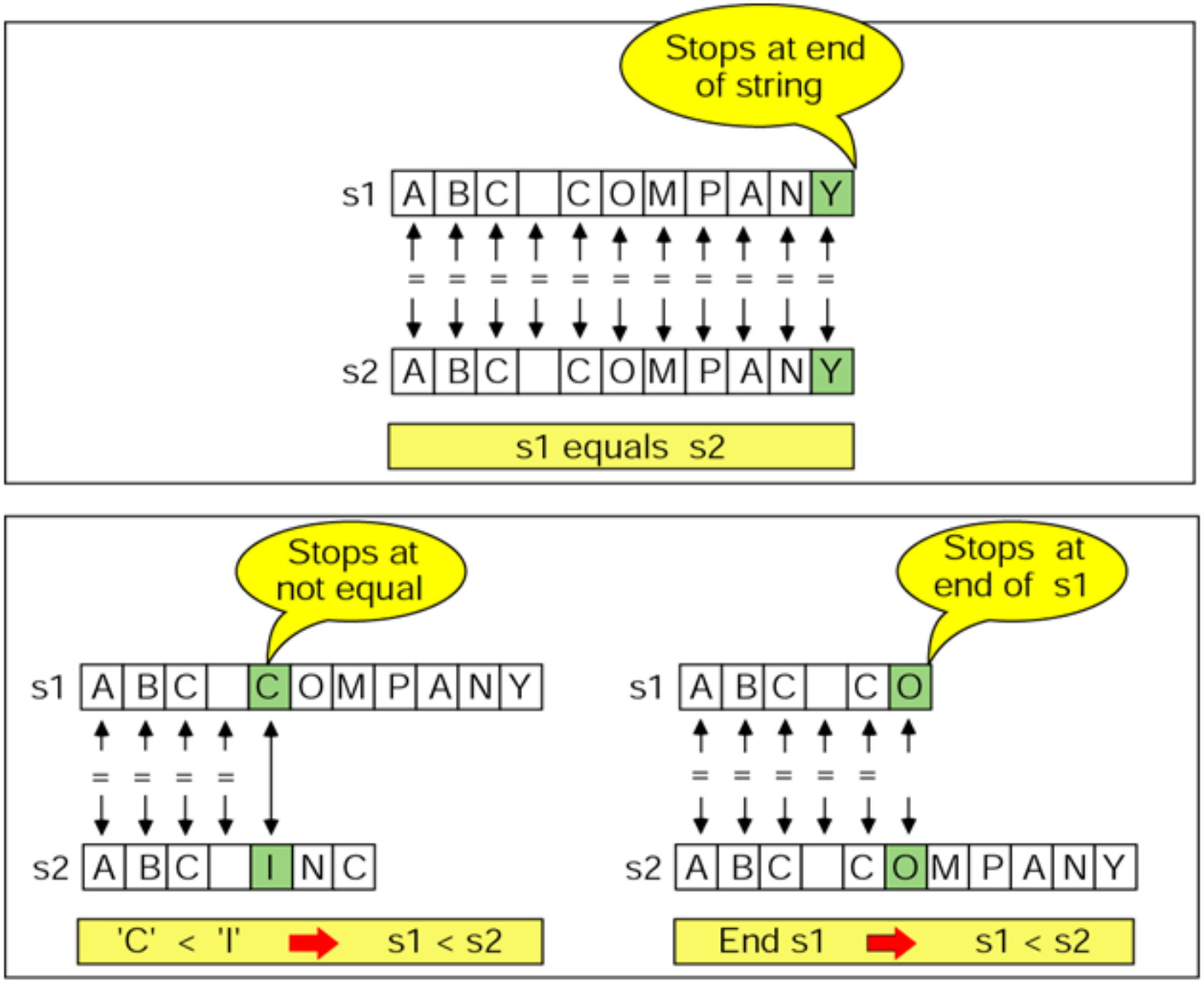
```
len = str1.length();
len = str1.size();
```

String Compare

- Two alternatives
  - compare operators: used for a boolean result
  - compare method: used for a ternary answer (less than, equal, or greater-than)

- The comparison can be between two string objects or between a string object and a C string

String compare concept



String Relational Operators

- All of the relational operators are overloaded for the string class
- Return a boolean value–*true* or *false*

```
str1 == str2     str1 <  str2     str1 >   str2
str1  != str2    str1 <= str2  str1 >= str2
```

(Either str1 or str2, but not both, can be a C string)

- Usually used in a *while* or *if* statement

String Compare Method

- Results are a negative number (less than), 0 (equal), or a positive number (greater than)
- Basic formats

```
str1.compare(str2);
str1.compare(pos1, len1, str2);
str1.compare(pos1, len1, str2, pos2, len2)
```

(str1 must be a C++ string while str2 can be a C string)

- Usually used for searching or sorting

Concatenating and Appending

- Places the contents of one string at the end of another
- Concatenation
  - uses the plus(+) operator
  - The result must be placed in another string object

```
str3 = str1 + str2;
```

  - str1 & str2 remains unchanged
- Append
  - when using the overloaded plus-assign operator

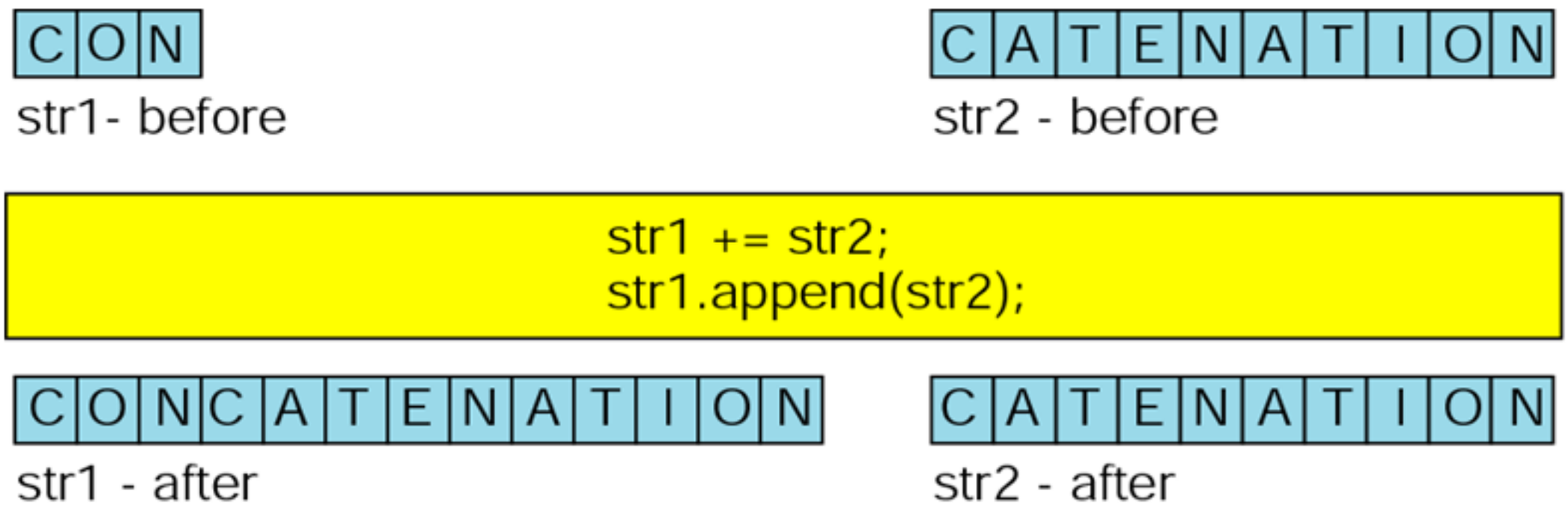
```
str1 += str2;
```

  - when using the *append* method in the string class

```
str1.append(str2);
str1.append(str2, pos2, len2);
```

  - str1 contains the result while str2 remains unchanged

String append concept



Substring Searching Forward: *find*

- Searches for a substring anywhere in a string

```
where = str1.find (str2);          // from the beginning
where = str1.find (str2, pos1); // starting at pos1
```

- Returns the *index location* within the string for the substring it located; *string::npos* if not found
- Examples
  - to locate the first occurrence `where = str.find("ten");`
  - to determine if the find was successful

```
if (where != string::npos) // test for success
   // Found processing
else
   // Not found processing
```

  - To find the next occurrence `where = str.find("ten", where+1);`

Substring Searching Backward: *rfind*

- Search for a substring starting at the end of a string and searching toward the beginning of the string

```
where = str1.rfind (str2, pos1);
```

Character Search Forward

- Find first matching character

```
whereFwd = str1.find_first_of (str2, pos1);
```

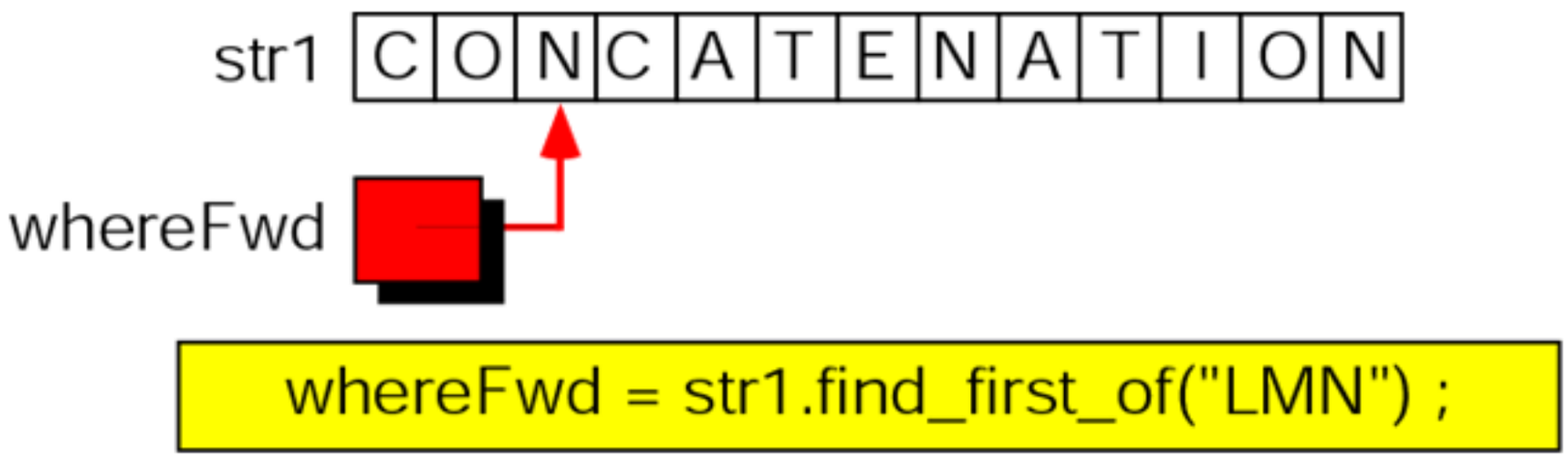
  - Search for the first character in the string that matches any of the characters in the input set (str2)
  - Returns the *position of the matching character*; if no matching characters are found, it returns *string::npos*

- Find first nonmatching character

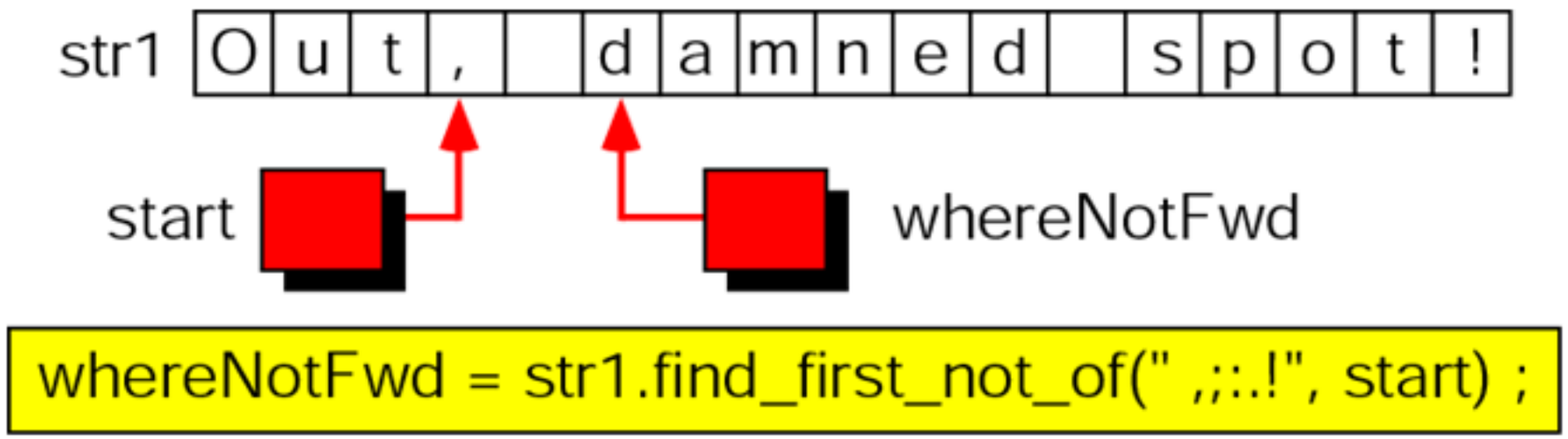
```
whereNotFwd = str1.find_first_not_of (str2, pos1);
```

  - find the first character that does not match the input set

Find matched character in the string (forward direction)



Find non-matched character in the string (forward direction)

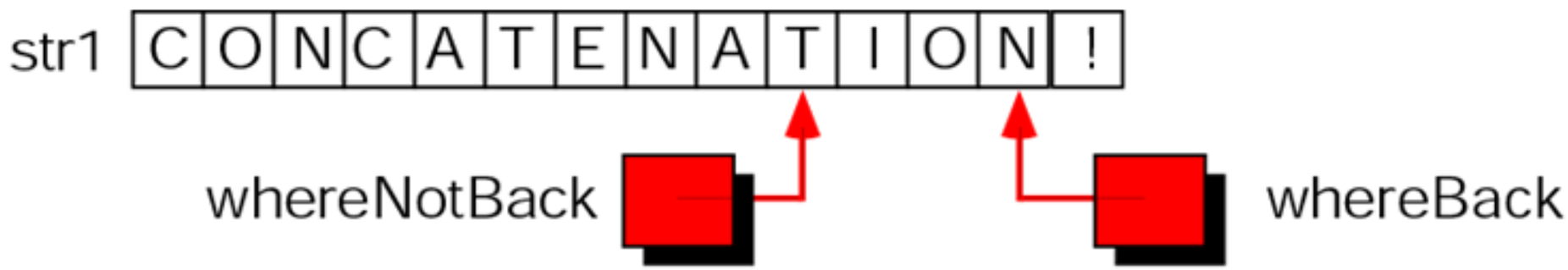


Character Search Backwards

- Search for a character in a string starting at the end of the target string
- Search toward the beginning of the string stopping at
  - the first matching character for a *find last* or
  - the first nonmatching character for a *not-find last*

```
whereBack = str1.find_last_of (str2, pos1);

whereNotBack = str1.find_last_not_of (str2, pos1);
```

Find matched and non-matched character in the string (reverse direction)




```
whereBack    = str1.find_last_of("CION") ;
whereNotBack = str1.find_last_not_of("CION") ;
```

Access and Modify Characters

- The ***at*** method can be used to access a character in a string
  - The *at* function tests for an invalid index and may abort the program if it is out of range

```
oneChar = str.at (where);
```

- The ***index*** location (***brackets***) can be used to access and modify a character in a string
  - Bracket access does not check for a out-of-range error

```
oneChar = str[where];
```

String Insertion

- Insert a character, a character a specified number of times, a string, or a substring at a specified position in a string object

```
str1.insert (pos1, str2);
str1.insert (pos1, str2, pos2, len2);
str1.insert (pos1, char);
str1.insert (pos1, numchar, char);
```

- str2 can be a string object or a C string

Replace String

- Replace all or part of a string with another string

```
str1.replace (pos1, len1, str2);
str1.replace (pos1, len1, str2, pos2, len2);
```

- The replacement string value can be a string object or a C string

- While the replace method can be used to replace the entire string, the assignment is faster

Erase String

- The *erase* method can be used to erase the entire string or to erase from a specific index position

```
str.erase (pos, num);
```

- The *clear* method erases the entire contents of the string

```
str.clear ();
```

Swap String

- Swap two string objects

```
swap (str1, str2);
```

- Notice it is a standalone function (not a class member)

Convert to C String

- Converts a C++ string object to a C string
- Returns a character pointer constant

```
string str("Hello");
char* cString = str.c_str();
```

C String

- A variable-length array of characters that is delimited by the null character ('\0')
- Described in C string header file <cstring>

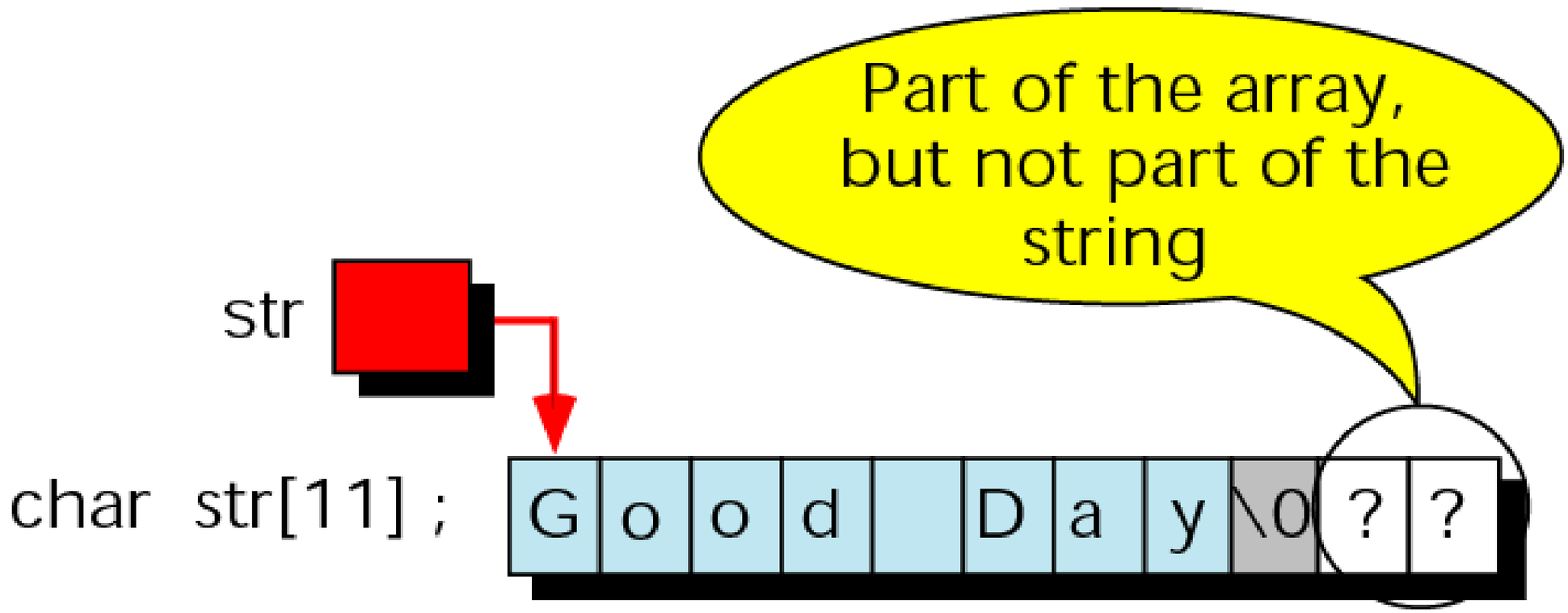
C string

H	**e**	**l**	**l**	**o**		**W**	**o**	**r**	**l**	**d**	**\0**

Storing C Strings

- Must provide enough room for the maximum-length string that will be stored *plus one for the delimiter*
- An array may have the null character in the middle when the stored string is less than the array size
  - The part of the array from the beginning to the null character is considered as the string

Strings in Arrays



Initializing Strings: Two Alternatives

<u>Using an array of characters</u>

- Define the string as an array of characters and assigns a value to it

```
char str[11] = "Good Day";
```

  - Creates an 11-byte array and fills the first 9 positions with the string value and a delimiter

<u>Using a character pointer</u>

- Define the string as a character pointer and assigns a value to it

```
char* str = "Good Day";
```

  - A constant string is created that consumes the minimum memory space needed to hold the string

Review: Lvalue and Rvalue

- Every C++ expression has a value, but the value in an expression (after evaluation) can be used in two different ways: lvalue and rvalue
- An **lvalue** expression can be used to access, modify, examine, or copy its data

  a = ... **a[5] = ...**

  (a) = ... ***p = ...**
- An **rvalue** expression can be used only to supply a value for an expression

  5 **a+2** **a*6**

  a[2]+3 **a++**
- Some operators need an lvalue as their operand
  - E.g., the left operand of the assignment operator

Strings and the Assignment Operator

- The name of the string is a pointer constant
- As a pointer constant, it is an rvalue and therefore cannot be used as the left operand of the assignment operator

```
char str1[11] = “Hello”;
char str2[11];
str2 = str1;                // Compile error
str1 = “Hello”;             // Compile error
```

Copy C strings

We cannot use the assignment operator to copy C strings.
We must use the strcpy function.

Reading C Strings: Extraction Operator

- Simple and natural way for reading strings

```
char month[10];
cin >> month;    or    fsIn >> month;
```

- The extraction operator does not read whitespace (similar to reading C++ string)
  - It skips any leading whitespace
  - Once it finds a character, it reads until it finds whitespace, putting each character in the array in order
  - When it finds a white space character, it stores the string with a null delimiter character
  - The whitespace character is left in the input stream

Protect against entering too much data

Always use set width when reading C strings.

- If the array is not large enough to store all the input data, then whatever follows the array in memory will be destroyed
- Set the width with the set-width manipulator

```
char month[10];
cin >> setw(10) >> month;
```

Reading C Strings: getline()

- Extracts text (including whitespaces) from an input stream and makes a null-terminated string out of it
- Three parameters
  - 1st: the string area into which the string is to be read
  - 2nd: the maximum number of characters that are to be transferred, including the generated string delimiter character (use the sizeof operator)
  - 3rd: an optional terminating character
- Examples

```
cin.getline (inArea, sizeof(inArea));     // stop at \n
fsIn.getline (inArea, sizeof(inArea), ';'); // stop at ;
```

Writing C Strings: Insertion Operator

- String output is provided by the insertion operator (<<)

  cout << month; or fsOut << month;

- The width option sets the *minimum* print area for the string in the output
- The justification option specifies the orientation of data in a field
  - left-justified vs. right-justified

```
cout << "*" << "Hi there!" << "*" << endl;
cout << "*" << setw(20) << "Hi there!" << "*" << endl;
cout << right;
cout << "*" << setw(20) << "Hi there!" << "*" << endl;
```

```
/*   Results:
*Hi there!*
*Hi there!           *
*           Hi there!*
*/
```

String Function Library

- A rich set of string functions are in the C string library (<cstring>)

  - String length (strlen)
  - String copy (strcpy, strncpy)
  - String compare (strcmp, strncmp)
  - String concatenate (strcat, strncat)
  - Search for a character (strchr, strrchr)
  - Search for a substring (strstr)
  - Search for characters in a string (strspn, strcspn)

String Length (*strlen*)

- Returns the length of a string, specified as the number of characters in the string excluding the null character

```
length = strlen (str1);
length = strlen ("Hello World");
```

String Copy

- ***strcpy*** copies the contents of one string to another string

  strcpy (toStr, fromStr);

  - toStr: a pointer to the array that is to receive the string
  - fromStr: the string being copied

- ***strncpy*** (string number copy) contains a size parameter that specifies the *maximum* number of characters than can be moved at a time

  strncpy (toStr, fromStr, size);

  - If the sending string is longer than size, the destination variable will not have a delimiter

- Both functions return the new string's address, which may be stored or discarded

Example: String Copy

- Make the destination valid after strncpy

```
strncpy (s1, s2, sizeof(s1) – 1);
*(s1 + (sizeof(s1) – 1)) = '□0';
```

String Compare

- ***strcmp*** compares two strings until unequal characters are found or until the end of the string is reached

  result = strcmp (str1, str2);

- ***strncmp*** compares until unequal characters are found, a specified number of characters have been tested, or until the end of a string is reached

  result = strncmp (str1, str2, size);

- Result
  - 0 if two strings are equal
  - a negative number if str1 is less than str2
  - a positive number if str2 is greater than str2

Examples: String Compare

- Example 1

```
if (strcmp(str1, str2) == 0)
    // strings are equal
else
    // strings are not equal
```

- Example 2

```
if (strcmp(string1, string2) < 0)
    // string1 is less than string2
```

- Example 3

```
if (strcmp(string1, string2) > 0)
    // string1 is greater than string2
```

- Example 4

```
if (strcmp(string1, string2) >= 0)
    // string1 is greater than or equal to string2
```

String Concatenation

- Append one string to the end of a second string
- Return the address pointer to the destination string
- The size of the destination string array is assumed to be large enough to hold the resulting string

destination

↑

```
strcat (str1, str2);
```

str2 is copied to the end of str1

```
strncat (str1, str2, size);
```

↓

maximum number of characters to be copied

Examples: String Concatenation

- Example 1

  char str1[20] = “Hello”;

  strcat (str1, “World”); ➔ str1: “HelloWorld”

- Example 2

  char str1[20] = “Hello”;

  char str2[20] = “World”;

  strcat (str1, str2); ➔ str1: “HelloWorld”, str2 remains unchanged

- Example 3

  char str1[8] = “Hello”;

  strcat (str1, “World”); ➔ str1 is destroyed because of lack of space

- Example 4

  char str1[8] = “Hello”;

  strncat (str1, “World”, 2); ➔ str1 becomes “HelloWo”

Searching for Characters

- String character (*strchr*)

  newStrPtr = strchr (str, ch);

  - Searches for the first occurrence of a character from the beginning of a string

- String rear character (*strrchr*)

  newStrPtr = strrchr (str, ch);

  - Searches for the first occurrence beginning at the end and working toward the beginning

- They return a pointer to it. (a null pointer if not found)

Searching for a Substring

- Locates a substring in a string
- Returns a pointer to the beginning of the substring in the string

```
newStrPtr = strstr (str, subStr);
```

- There is no function to locate a substring starting at the rear

Searching for Characters in a String

- Locate one of a set of characters in a string
- String span *strspn*

```
numChars = strspn (str1, charSet);
```

  - Searches the string, spanning characters that are in the set
  - Stop at the first character that is not in the set
  - Returns the number of characters that matched those in the set

- String complement span *strcspn*
  - Stop at the first character that matches one of the characters in the set

Searching for Tokens (*strtok*)

- Locate substrings, called *tokens*, in a string
  - token: a sequence of characters separated by *delimiters*

```
charPtr = strtok (str1, delimiters);
```

- Returns the pointer to the first token overwriting the delimiter followed by a null character
  - NULL is returned when there are no more tokens to be found.
- Use successive calls to *strtok()*, to extract all the tokens
  - When called with NULL as the first parameter, it will follow by where the last call to strtok found a delimiter.
  - *delimiters* may vary from a call to another.

Example: Searching for Tokens

d	o	g	,	c	a	t	,	p	i	g	\0

↑ start

↑ found a delimiter

d	o	g	\0	c	a	t	,	p	i	g	\0

↑ start

↑ delimiter changed to end-of-data null character

d	o	g	\0	c	a	t	,	p	i	g	\0

↑ return this

↑ the next to start

Example: Searching for Tokens (*strtok*)

```
#include <iostream>
#include <cstring>
int main ()
{
  char str[] ="This is a sample string,just testing.";
  char * pch;
  cout << “String: \” “<< str <<“\”” << endl;
  cout << "Splitting string in tokens:“ << endl;
  pch = strtok (str," ");
  while (pch != NULL) {
    cout << pch << endl;
    pch = strtok (NULL, " ,.");
  }
  return 0;
}
```

```
/* Output:
String: "This is a sample string,just
testing."
Splitting the string in tokens:
This
is
a
sample
string
just
testing
*/
```

Converting C String to C++ String

- Method 1: Assigns the C string to the C++ string
- Method 2: Use the C string as the copy constructor value

```
char* cStr = "Hello";
string str1;
str1 = cStr;              // Assignment
string str2 (cStr);       // In copy constructor
```

Comparison between C and C++ Strings 1/3

Action	C++ string	C string
Input	<<, getline	<<, getline
Output	>>	>>
Copy	=	strcpy, strncpy
Compare	Relational operators, compare	strcmp, strncmp
Concatenation	+, +=, append	strcat, strncat

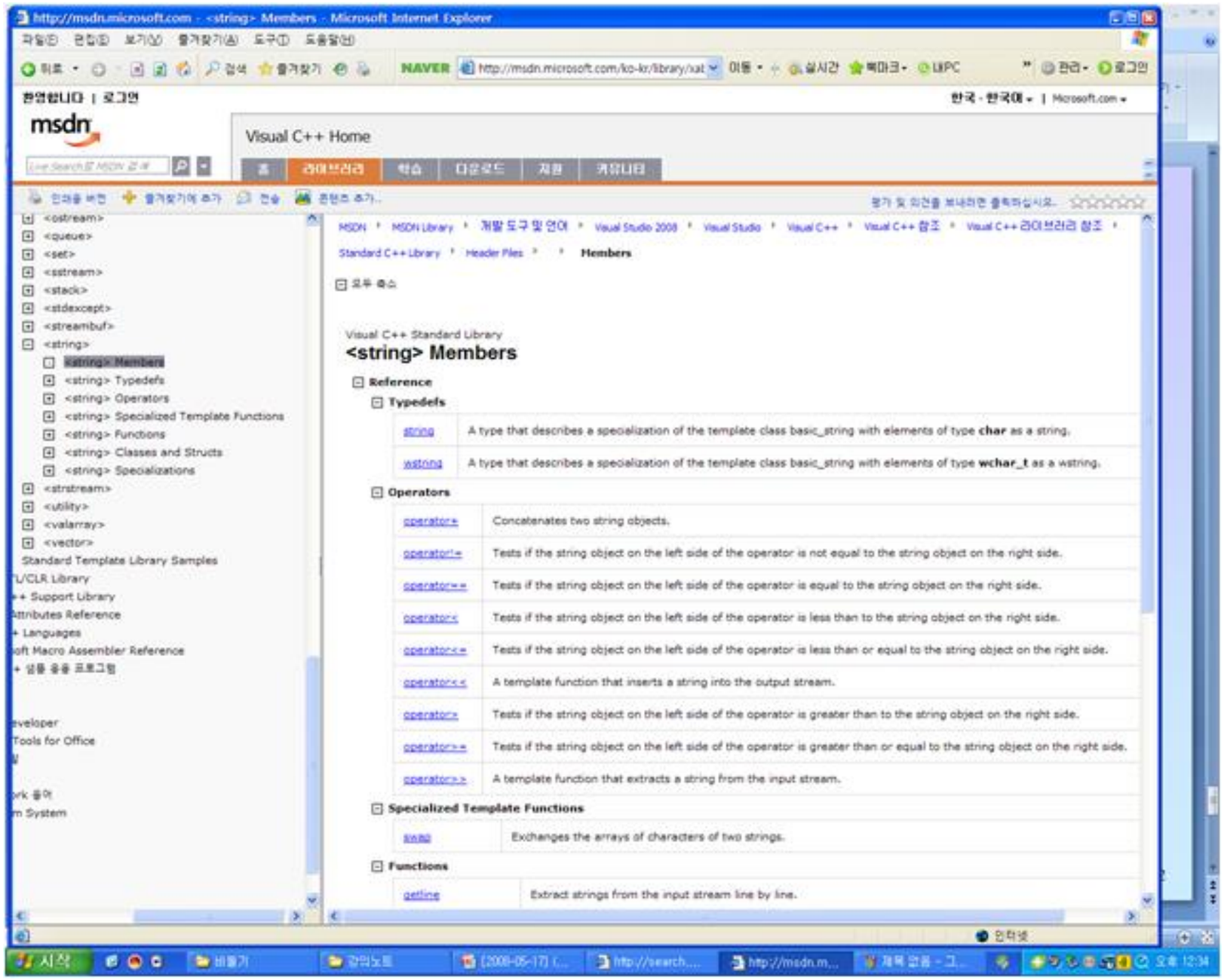
Comparison between C and C++ Strings 2/3

Action	C++ string	C string
Extraction	substr	strstr
Search for substring	find, rfind	strstr
Search for character	find, rfind	strchr, strrchr
Search for character in set	find_first_of, find_last_of	strspn
Search for character not in set	find_first_not_of, find_last_not_of	strcspn

Comparison between C and C++ Strings 3/3

Action	C++ string	C string
Access character	at, []	strchr, []
Insert	insert	N/A
Erase	erase, clear	N/A
Swap	swap	N/A
Convert to other format	c_str	assign or copy constructor

Search ‘String Class’ and its member functions from MSDN (1/2)



Search ‘String Class’ and its member functions from MSDN (2/2)

Search ‘swap’ functions from MSDN

Visual C++ Standard Library

swap (<string>)

Exchanges the arrays of characters of two strings.

```
template<class Traits, class Allocator>
   void swap(
      basic_string<CharType, Traits, Allocator>& _Left,
      basic_string<CharType, Traits, Allocator>& _Right
      );
```

Parameters

_Left

One string whose elements are to be swapped with those of another string.

_Right

The other string whose elements are to be swapped with the first string.

Remarks

The template function executes the specialized member function *_Left*.swap(*_Right*) for strings, which guarantees constant complexity.

Example

```
// string_swap.cpp
// compile with: /EHsc
#include <string>
#include <iostream>

int main( )
{
   using namespace std;
   // Declaring an object of type basic_string<char>
   string s1 ( "Tweedledee" );
   string s2 ( "Tweedledum" );
   cout << "Before swapping string s1 and s2:" << endl;
   cout << "The basic_string s1 = " << s1 << "." << endl;
   cout << "The basic_string s2 = " << s2 << "." << endl;

   swap ( s1 , s2 );
   cout << "\nAfter swapping string s1 and s2:" << endl;
   cout << "The basic_string s1 = " << s1 << "." << endl;
   cout << "The basic_string s2 = " << s2 << "." << endl;
}
```

```
Before swapping string s1 and s2:
The basic_string s1 = Tweedledee.
The basic_string s2 = Tweedledum.

After swapping string s1 and s2:
The basic_string s1 = Tweedledum.
The basic_string s2 = Tweedledee.
```

Requirements

Header: <string>

Namespace: std

See Also

Concepts

<string> Members

Questions?