COEN70: FORMAL SPECIFICATION AND ADVANCED DATA STRUCTURES

Lecture 1 part 2

Instructor: Aikaterini (Katerina) Potika
Email: apotika@scu.edu

Slides adapted from Problem solving in C++, W. Savitch

Information

❑**Lecture:**
- MWF 11:45 a.m.-12:50, EC326

❑**Office Hours:**
- MWF: 10:30-11:30 a.m., EC 201B
- or by appointment

The End of The File

- Input files used by a program may vary in length
  - Programs may not be able to assume the number of items in the file
- A way to know the end of the file is reached:
  - The boolean expression *(in_stream >> next)*
    - Reads a value from in_stream and stores it in next
    - True if a value can be read and stored in next
    - False if there is not a value to be read (the end of the file)

The End of The File - Example

- Example: To calculate the average of the numbers in a file

```
double next, sum = 0;
int count = 0;
while(in_stream >> next)
{
        sum = sum + next;
        count++;
}

double average = sum / count;
```

How To Test End of File

- We have seen two methods
  - `while ( in_stream >> next)`
  - `while ( ! in_stream.eof( ) )`
- Which should be used?
  - In general, use `eof` when input is treated as text and using a member function `get` to read input
  - In general, use the extraction operator method when processing numeric data

Insertion & extraction operators

<< insertion operator, applied to an output stream
>> extraction operator, applied to an input stream
Example:

```
int n;
cout << "Enter a number: ";
cin >> n;
cout << "You have entered: " << n << '\n';
```

Where cout is the standard output (monitor) and cin is the standard input (keyboard)

Program Errors

- ***Syntax errors***
  - Violation of the grammar rules of the language
  - Discovered by the compiler
    - Error messages may not always show correct location of errors
- ***Run-time errors***
  - Error conditions detected by the computer at run-time
- ***Logic errors***
  - Errors in the program's algorithm
  - Most difficult to diagnose
  - Computer does not recognize an error

Fix bugs

- Compiler for syntax errors
- Debugger for the rest
  - gdb 1st Lab

The Standard `string` Class

- The `string` class allows the programmer to treat strings as a basic data type

- The `string` class is defined in the `string` library and the names are in the standard namespace
  - To use the `string` class you need these lines:

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

Assignment of Strings

- Variables of type string can be assigned with the = operator
  - Example:

```
string s1, s2, s3;
      …
 s3 = s2;
```

- Quoted strings (i.e. C-strings) are **type cast** to type string
  - Example: `string s1 = "Hello Mom!";`

Using + With strings

- Variables of type string can be concatenated with the + operator
  - Example:

```
string s1, s2, s3;
           …
s3 = s1 + s2;
```

  - If s3 is not large enough to contain s1 + s2, more space is allocated

Comparison of strings

- Comparison operators work with string objects
  - Objects are compared using lexicographic order (Alphabetical ordering using the order of symbols in the ASCII character set.)
  - == returns true if two string objects contain the same characters in the same order
  - <, >, <=, >= can be used to compare string objects

string Constructors

- The default string constructor initializes the string to the empty string
- Another string constructor takes a C-string argument
  - Example:

```
string phrase; // empty string
string noun("ants"); // a string version
                     //  of "ants"
```

Mixing strings and C-strings

- It is natural to work with strings in the following manner

```
string phrase = "I love" + adjective + " "
                                 + noun + "!";
```

  - It is not so easy for C++! It must either convert the null-terminated C-strings, such as "I love", to strings, or it must use an overloaded + operator that works with strings and C-strings

Member Function length

- The string class member function length returns the number of characters in the string object:
  - Example:

```
int n = string_var.length( );
```

String Processing

- The string class allows the same operations we used with C-strings…and more
  - Characters in a string object can be accessed as if they are in an array
    - last_name[i] provides access to a single character as in an array
    - Index values are **not** checked for validity!

Member Function at

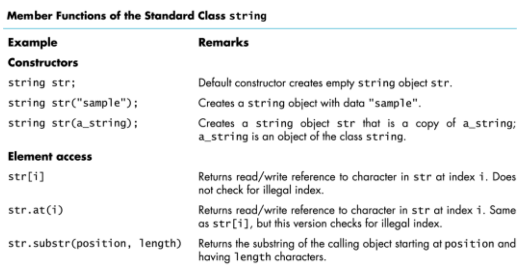
- at is an alternative to using [ ]'s to access characters in a string.
  - at checks for valid index values
  - Example: string str("Mary");
  - **Equivalent**

```
cout << str[6] << endl;
cout << str.at(6) << endl;
```

  - **Equivalent**

```
str[2] = 'X';
str.at(2) = 'X';
```

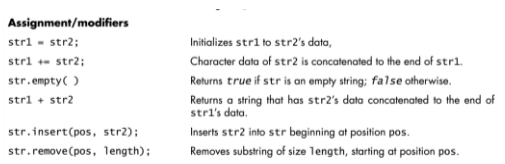
Other string class functions are found in next slide



Member Functions of the Standard Class string

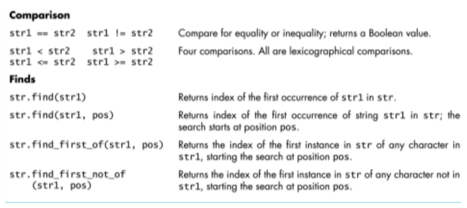
Example	Remarks
Constructors	
string str;	Default constructor creates empty string object str.
string str("sample");	Creates a string object with data "sample".
string str(a_string);	Creates a string object str that is a copy of a_string; a_string is an object of the class string.
Element access	
str[i]	Returns read/write reference to character in str at index i. Does not check for illegal index.
str.at(i)	Returns read/write reference to character in str at index i. Same as str[i], but this version checks for illegal index.
str.substr(position, length)	Returns the substring of the calling object starting at position and having length characters.

Member functions of the standard Class string



Assignment/modifiers	
`str1 = str2;`	Initializes `str1` to `str2`'s data,
`str1 += str2;`	Character data of `str2` is concatenated to the end of `str1`.
`str.empty( )`	Returns *true* if `str` is an empty string; *false* otherwise.
`str1 + str2`	Returns a string that has `str2`'s data concatenated to the end of `str1`'s data.
`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.

Member functions of the standard Class string



Comparison	
`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.

Finds	
`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 the search at position pos.

I/O With Class string

- The insertion operator << is used to output objects of type `string`
  - Example: `string s = "Hello Mom!";`
    `cout << s;`
- The extraction operator >> can be used to input data for objects of type string
  - Example: `string s1;`
    `cin >> s1;`
    - >> skips whitespace and stops on encountering more whitespace

`string` Objects to C-strings

- Recall the automatic conversion from C-string to string:
```
char a_c_string[] = "C-string";
string_variable = a_c_string;
```
- `strings` are not converted to C-strings
- Both of these statements are **illegal**:
  - `a_c_string = string_variable;`
  - `strcpy(a_c_string, string_variable);`

Converting strings to C-strings

- The string class member function `c_str` returns the C-string version of a string object
  - Example:
```
strcpy(a_c_string, string_variable.c_str( ) );
```
- This line is still **illegal**
```
      a_c_string = string_variable.c_str( ) ;
```
  - Recall that operator = does not work with C-strings

Tools for Stream I/O

- To control the format of the program's output
  - We use commands that determine such details as:
    - The spaces between items
    - The number of digits after a decimal point
    - The numeric style: scientific notation for fixed point
    - Showing digits after a decimal point even if they are zeroes
    - Showing plus signs in front of positive numbers
    - Left or right justifying numbers in a given space

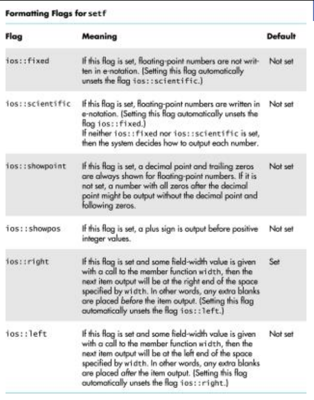
Formatting Output to Files

- Format output to the screen with:

```
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
```

- Format output to a file using the out-file stream named out_stream with:

```
out_stream.setf(ios::fixed);
out_stream.setf(ios::showpoint);
out_stream.precision(2);
```


Formatting Flags for `setf`

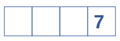
Flag	Meaning	Default
`ios::fixed`	If this flag is set, floating-point numbers are not written in e-notation. (Setting this flag automatically unsets the flag `ios::scientific`.)	Not set
`ios::scientific`	If this flag is set, floating-point numbers are written in e-notation. (Setting this flag automatically unsets the flag `ios::fixed`.) If neither `ios::fixed` nor `ios::scientific` is set, then the system decides how to output each number.	Not set
`ios::showpoint`	If this flag is set, a decimal point and trailing zeros are always shown for floating-point numbers. If it is not set, a number with all zeros after the decimal point might be output without the decimal point and following zeros.	Not set
`ios::showpos`	If this flag is set, a plus sign is output before positive integer values.	Not set
`ios::right`	If this flag is set and some field-width value is given with a call to the member function `width`, then the next item output will be at the right end of the space specified by `width`. In other words, any extra blanks are placed before the item output. (Setting this flag automatically unsets the flag `ios::left`.)	Set
`ios::left`	If this flag is set and some field-width value is given with a call to the member function `width`, then the next item output will be at the left end of the space specified by `width`. In other words, any extra blanks are placed after the item output. (Setting this flag automatically unsets the flag `ios::right`.)	Not set

Creating Space in Output

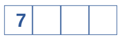
- The width function specifies the number of spaces for the next item
  - Applies only to the next item of output
- Example: To print the digit 7 in four spaces use

```
out_stream.width(4);
out_stream << 7 << endl;
```

  - Three of the spaces will be blank



(ios::right)



(ios::left)

Not Enough Width?

- What if the argument for width is too small?
  - Such as specifying

```
cout.width(3);
```

  when the value to print is 3456.45
- The entire item is always output
  - If too few spaces are specified, as many more spaces as needed are used

Unsetting Flags

- Any flag that is set, may be unset
- Use the unsetf function
  - Example:

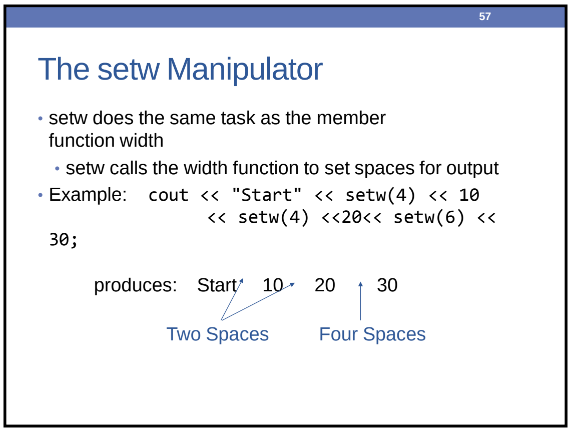
```
cout.unsetf(ios::showpos);
```

  causes the program to stop printing plus signs on positive numbers

Manipulators

- A manipulator is a function called in a nontraditional way
  - Manipulators in turn call member functions
  - Manipulators may or may not have arguments
  - Used after the insertion operator (<<) as if the manipulator function call is an output item
- To use these manipulators, add these lines

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



The setw Manipulator

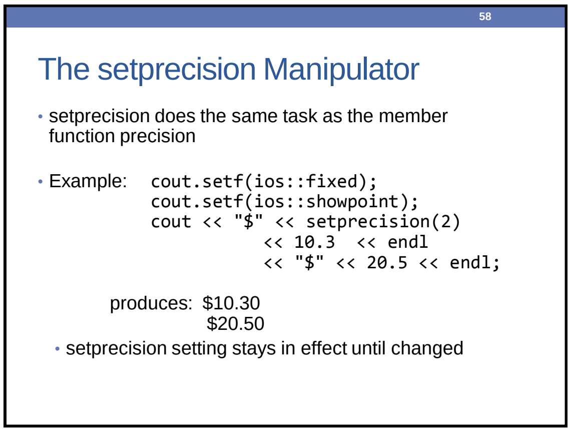
- setw does the same task as the member function width
  - setw calls the width function to set spaces for output
- Example:

```
cout << "Start" << setw(4) << 10
         << setw(4) <<20<< setw(6) <<
30;
```

produces:

```
Start  10  20    30
```

Two Spaces Four Spaces



The setprecision Manipulator

- setprecision does the same task as the member function precision
- Example:

```
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout << "$" << setprecision(2)
           << 10.3  << endl
           << "$" << 20.5 << endl;
```

produces:

```
$10.30
$20.50
```

  - setprecision setting stays in effect until changed