

APS105: Lecture 15

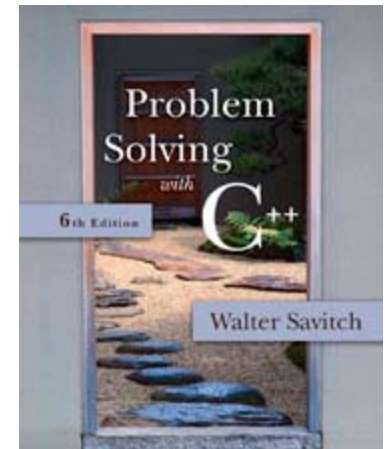
Wael Aboelsaadat

wael@cs.toronto.edu

<http://ccnet3.utoronto.ca/20079/aps105h1f/>

Acknowledgement: These slides are a modified version of the text book slides as supplied by Addison Wesley

Download the code shown in lecture from course website:
Handouts → Lectures Source Code - Wael



Few words about last lecture

- What are static variables?
- What is the difference between a library and a standalone program?
- Behavior of `in_stream`
- Revisiting namespaces

Static variables

- First, let's talk about variable life time

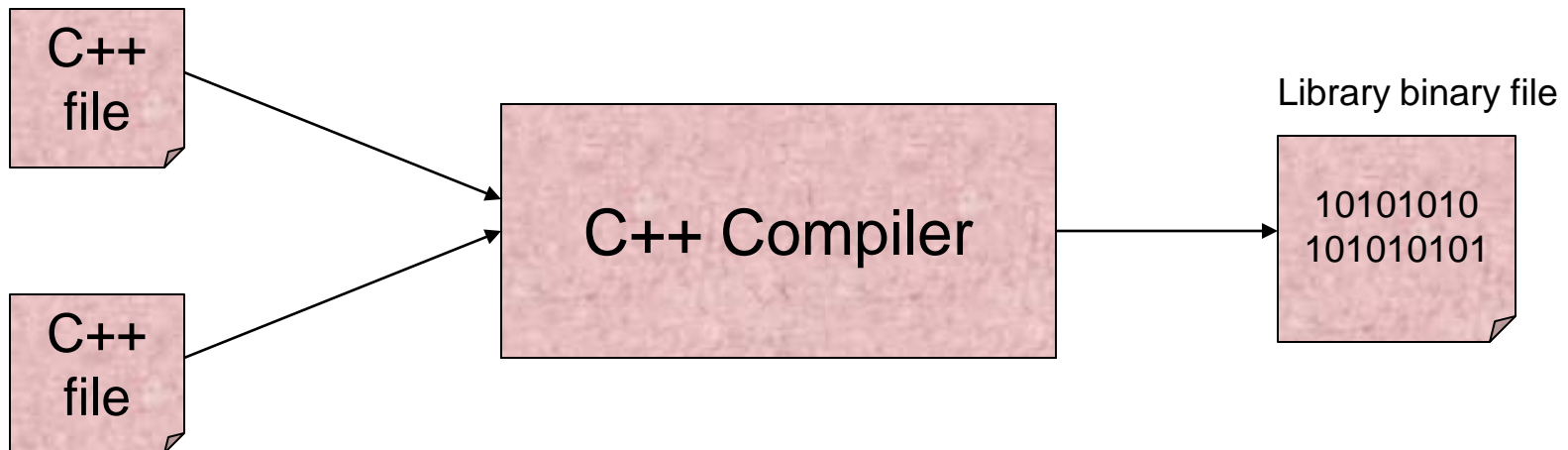
Static variables

- If used in a function, a static variable is a variable whose value is not lost between function calls (it is not deleted from memory!)
- If used in a class, a static variable is a variable whose value is not lost once the object is created and remains until it is deleted from memory
 - How to access it if inside a class? using `::` notation

e.g. `cout << classname::variablename;`

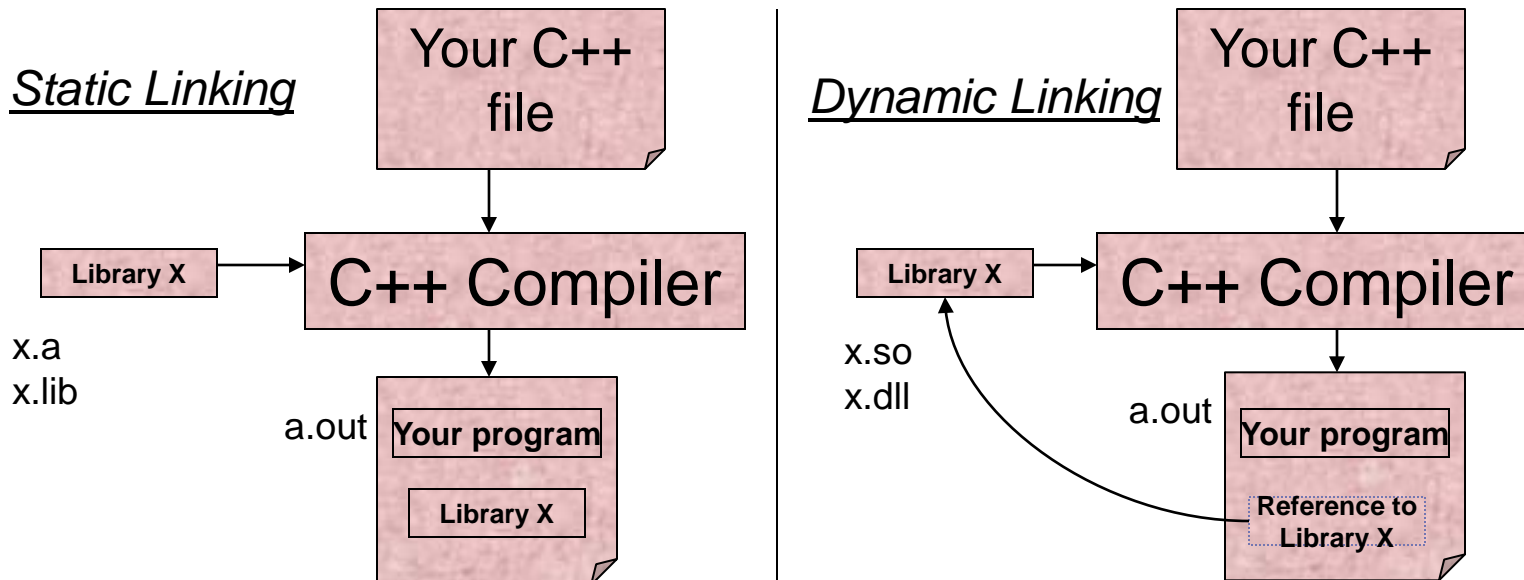
What's a library?

- A set of functions defined in 1 or more files without a main function...
 - You can't run it from command line.. It is not a program.. It is a library...!
 - It is a binary file that can only be used through programs
 - On Unix, they have extension .a or .so
 - On Windows, they have extension .dll or .lib



How to use a library from your program?

- If it is a static library, the compiler adds it to the output binary corresponding to your program
 - That's why the file size is bigger....
- If it is a dynamic library, a reference is put to the library but it is not copied to your executable



Behaviour of ifstream & ofstream

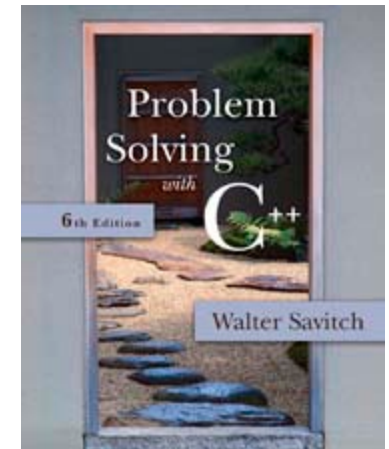
- Same as cin and cout

Namespaces

- A label for 1 or more variables...
- Why?
- How to reference a variable inside a name space?

6.3

Character I/O



First: Pass by reference vs. pass by value

- We have seen pass by value until now.
- What is pass by reference?

Character I/O

- All data is input and output as characters
 - Output of the number 10 is two characters '1' and '0'
 - Input of the number 10 is also done as '1' and '0'
 - Interpretation of 10 as the number 10 or as characters depends on the program
 - Conversion between characters and numbers is usually automatic

Low Level Character I/O

- Low level C++ functions for character I/O
 - Perform character input and output
 - Do not perform automatic conversions
 - Allow you to do input and output in anyway you can devise

Member Function get

- Function get
 - Member function of every input stream
 - Reads one character from an input stream
 - Stores the character read in a variable of type char, the single argument the function takes
 - Does not use the extraction operator (>>) which performs some automatic work
 - Does not skip blanks

Using get

- These lines use `get` to read a character and store it in the variable `next_symbol`

```
char next_symbol;  
cin.get(next_symbol);
```

- Any character will be read with these statements
 - Blank spaces too!
 - `'\n'` too! (The newline character)

get Syntax

- `input_stream.get(char_variable);`

- Examples: `char next_symbol;`
 `cin.get(next_symbol);`

```
ifstream in_stream;  
in_stream.open("infile.dat");  
in_stream.get(next_symbol);
```

More About get

- Given this code:

```
char c1, c2, c3;  
cin.get(c1);  
cin.get(c2);  
cin.get(c3);
```

and this input:

AB

CD

- `c1 = 'A'` `c2 = 'B'` `c3 = '\n'`
 - `cin >> c1 >> c2 >> c3;` would place 'C' in c3
(the ">>" operator skips the newline character)

The End of The Line

- To read and echo a line of input

- Look for '\n' at the end of the input line:

```
cout<<"Enter a line of input and I will "  
    << "echo it.\n";  
char symbol;  
do  
{  
    cin.get(symbol);  
    cout << symbol;  
} while (symbol != '\n');
```

- All characters, including '\n' will be output

'\n' vs "\n"

- '\n'
 - A value of type char
 - Can be stored in a variable of type char
- "\n"
 - A string containing only one character
 - Cannot be stored in a variable of type char
- In a cout-statement they produce the same result

Member Function put

- Function put
 - Member function of every output stream
 - Requires one argument of type char
 - Places its argument of type char in the output stream
 - Does not do allow you to do more than previous output with the insertion operator and cout

put Syntax

- `Output_stream.put(Char_expression);`

- Examples: `cout.put(next_symbol);`
`cout.put('a');`

```
ofstream out_stream;  
out_stream.open("outfile.dat");  
out_stream.put('Z');
```

Member Function putback

- The putback member function places a character in the input stream
 - putback is a member function of every input stream
 - Useful when input continues until a specific character is read, but you do not want to process the character
 - Places its argument of type char in the input stream
 - Character placed in the stream does not have to be a character read from the stream

putback Example

- The following code reads up to the first blank in the input stream `fin`, and writes the characters to the file connected to the output stream `fout`

- ```
 fin.get(next);
 while (next != ' ')
 {
 fout.put(next);
 fin.get(next);
 }
 fin.putback(next);
```

- The blank space read to end the loop is put back into the input stream

# Program Example

## Checking Input

- Incorrect input can produce worthless output
- Use input functions that allow the user to re-enter input until it is correct, such as
  - Echoing the input and asking the user if it is correct
  - If the input is not correct, allow the user to enter the data again

# Checking Input:

## get\_int

- The `get_int` function seen in Display 6.7 obtains an integer value from the user
  - `get_int` prompts the user, reads the input, and displays the input
  - After displaying the input, `get_int` asks the user to confirm the number and reads the user's response using a variable of type character
  - The process is repeated until the user indicates with a 'Y' or 'y' that the number entered is correct



# Checking Input:

## new\_line

- The new\_line function seen in Display 6.7 is called by the get\_int function
  - new\_line reads all the characters remaining in the input line but does nothing with them, essentially discarding them
  - new\_line is used to discard what follows the first character of the the user's response to get\_line's "Is that correct? (yes/no)"
    - The newline character is discarded as well

**Display 6.7 (1)**

**Display 6.7 (2)**

# Mixing `cin >>` and `cin.get`

- Be sure to deal with the `'\n'` that ends each input line if using `cin >>` and `cin.get`
  - `"cin >>"` reads up to the `'\n'`
  - The `'\n'` remains in the input stream
  - Using `cin.get` next will read the `'\n'`
  - The `new_line` function from Display 6.7 can be used to clear the `'\n'`

# '\n' Example

- The Code:  

```
cout << "Enter a number:\n";
int number;
cin >> number;
cout << "Now enter a letter:\n";
char symbol;
cin.get(symbol);
```

The Dialogue:  
Enter a number:  
21  
Now enter a letter:  
A

The Result:  
number = 21  
**symbol = '\n'**

# Detecting the End of a File

- Member function eof detects the end of a file
  - Member function of every input-file stream
  - eof stands for end of file
  - eof returns a boolean value
    - True when the end of the file has been reached
    - False when there is more data to read
  - Normally used to determine when we are NOT at the end of the file
    - Example: `if ( ! in_stream.eof( ) )`

# Using eof

- This loop reads each character, and writes it to the screen

- ```
                in_stream.get(next);
while (! in_stream.eof( ) )
{
    cout << next;
    in_stream.get(next);
}
```

- (! In_stream.eof()) becomes false when the program reads past the last character in the file

The End Of File Character

- End of a file is indicated by a special character
- `in_stream.eof()` is still true after the last character of data is read
- `in_stream.eof()` becomes false when the special end of file character is read

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

Program Example: Editing a Text File

- The program of Display 6.8...
 - Reads every character of file cad.dat and copies it to file cplusplus.dat except that every 'C' is changed to "C++" in cplusplus.dat
 - Preserves line breaks in cad.dat
 - get is used for input as the extraction operator would skip line breaks
 - get is used to preserve spaces as well
 - Uses eof to test for end of file

Display 6.8 (1)

Display 6.8 (2)

Character Functions

- Several predefined functions exist to facilitate working with characters
- The `cctype` library is required
 - `#include <cctype>`
using namespace `std`;

The toupper Function

- toupper returns the argument's upper case character
 - toupper('a') returns 'A'
 - toupper('A') return 'A'

toupper Returns An int

- Characters are actually stored as an integer assigned to the character
- toupper and tolower actually return the integer representing the character
 - `cout << toupper('a');` //prints the integer for 'A'
 - `char c = toupper('a');` //places the integer for 'A' in c
`cout << c;` //prints 'A'
 - `cout << static_cast<char>(toupper('a'));` //works too

The isspace Function

- isspace returns true if the argument is whitespace
 - Whitespace is spaces, tabs, and newlines
 - isspace(' ') returns true
 - Example:

```
if (isspace(next) )  
    cout << '-';  
else  
    cout << next;
```
 - Prints a '-' if next contains a space, tab, or newline character
- See more character functions in

Display 6.9 (1)

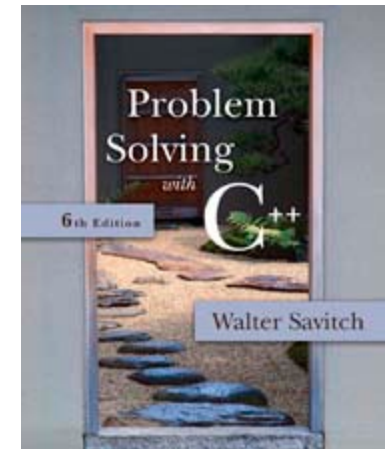
Display 6.9 (2)

Section 6.3 Conclusion

- Can you
 - Write code that will read a line of text and echo the line with all the uppercase letters deleted?
 - Describe two methods to detect the end of an input file:
 - Describe whitespace?

6.2

Tools for Streams I/O



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);
```


out_stream.precision(2);

- precision is a member function of output streams

- After out_stream.precision(2);

Output of numbers with decimal points...

- will show a total of 2 significant digits

23. 2.2e7 2.2 6.9e-1 0.00069

OR

- will show 2 digits after the decimal point

23.56 2.26e7 2.21 0.69 0.69e-4

- Calls to precision apply only to the stream named in the call

setf(ios::fixed);

- setf is a member function of output streams
 - setf is an abbreviation for set flags
 - A flag is an instruction to do one of two options
 - ios::fixed is a flag
 - After `out_stream.setf(ios::fixed);`
All further output of floating point numbers...
 - Will be written in fixed-point notation, the way we normally expect to see numbers
- Calls to setf apply only to the stream named in the call

setf(ios::showpoint);

- After `out_stream.setf(ios::showpoint);`

Output of floating point numbers...

- Will show the decimal point even if all digits after the decimal point are zeroes

Display 6.5

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

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) << 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

Manipulator Definitions

- The manipulators `setw` and `setprecision` are defined in the `iomanip` library
 - To use these manipulators, add these lines

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

Stream Names as Arguments

- Streams can be arguments to a function
 - The function's formal parameter for the stream must be call-by-reference
- Example: `void make_neat(ifstream& messy_file, ofstream& neat_file);`

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)

End of File 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;
```

# Stream Arguments and Namespaces

- Using directives have been local to function definitions in the examples so far
- When parameter type names are in a namespace
  - A using directive must be outside the function so C++ will understand the parameter type names such as ifstream
- Easy solution is to place the using directive at the beginning of the file
  - Many experts do not approve as this does not allow using multiple namespaces with names in common

# Program Example

- The program in Display 6.6...
  - Takes input from rawdata.dat
  - Writes output to the screen and to neat.dat
    - Formatting instructions are used to create a neater layout
    - Numbers are written one per line in a field width of 12
    - Each number is written with 5 digits after the decimal point
    - Each number is written with a plus or minus sign
  - Uses function `make_neat` that has formal parameters for the input-file stream and output-file stream

**Display 6.6 (1)**

**Display 6.6 (2)**

**Display 6.6 (3)**

# Section 6.2 Conclusion

- Can you
  - Show the output produced when the following line is executed?

```
cout << "*" << setw(3) << 12345 << "*" endl;
```

- Describe the effect of each of these flags?  
ios::fixed      ios::scientific      ios::showpoint  
ios::right      ios::right      ios::showpos



## Formatting Flags for set f

| Flag                         | Meaning                                                                                                                                                                                                                                                                                                                                                                 | Default |
|------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------|
| <code>ios::fixed</code>      | If this flag is set, floating-point numbers are not written in e-notation. (Setting this flag automatically unsets the flag <code>ios::scientific</code> .)                                                                                                                                                                                                             | Not set |
| <code>ios::scientific</code> | If this flag is set, floating-point numbers are written in e-notation. (Setting this flag automatically unsets the flag <code>ios::fixed</code> .)<br>If neither <code>ios::fixed</code> nor <code>ios::scientific</code> is set, then the system decides how to output each number.                                                                                    | Not set |
| <code>ios::showpoint</code>  | 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 |
| <code>ios::showpos</code>    | If this flag is set, a plus sign is output before positive integer values.                                                                                                                                                                                                                                                                                              | Not set |
| <code>ios::right</code>      | If this flag is set and some field-width value is given with a call to the member function <code>width</code> , then the next item output will be at the right end of the space specified by <code>width</code> . In other words, any extra blanks are placed <i>before</i> the item output. (Setting this flag automatically unsets the flag <code>ios::left</code> .) | Set     |
| <code>ios::left</code>       | If this flag is set and some field-width value is given with a call to the member function <code>width</code> , then the next item output will be at the left end of the space specified by <code>width</code> . In other words, any extra blanks are placed <i>after</i> the item output. (Setting this flag automatically unsets the flag <code>ios::right</code> .)  | Not set |

# Display 6.5



# Display 6.6 (1/3)



## Formatting Output (part 1 of 3)

```
//Illustrates output formatting instructions.
//Reads all the numbers in the file rawdata.dat and writes the numbers
//to the screen and to the file neat.dat in a neatly formatted way.
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <iomanip>
using namespace std;

void make_neat(ifstream& messy_file, ofstream& neat_file,
 int number_after_decimalpoint, int field_width);
//Precondition: The streams messy_file and neat_file have been connected
//to files using the function open.
//Postcondition: The numbers in the file connected to messy_file have been
//written to the screen and to the file connected to the stream neat_file.
//The numbers are written one per line, in fixed-point notation (that is, not in
//e-notation), with number_after_decimalpoint digits after the decimal point;
//each number is preceded by a plus or minus sign and each number is in a field of
//width field_width. (This function does not close the file.)

int main()
{
 ifstream fin;
 ofstream fout;

 fin.open("rawdata.dat");
 if (fin.fail())
 {
 cout << "Input file opening failed.\n";
 exit(1);
 }

 fout.open("neat.dat");
 if (fout.fail())
 {
 cout << "Output file opening failed.\n";
 exit(1);
 }
}
```

*Needed for setw*

*Stream parameters must be call-by-reference.*



```
make_neat(fin, fout, 5, 12);

fin.close();
fout.close();

cout << "End of program.\n";
return 0;
}

//Uses iostream, fstream, and iomanip:
void make_neat(ifstream& messy_file, ofstream& neat_file,
 int number_after_decimalpoint, int field_width)
{
 neat_file.setf(ios::fixed); ← Not in e-notation
 neat_file.setf(ios::showpoint); ← Show decimal point
 neat_file.setf(ios::showpos); ← Show + sign
 neat_file.precision(number_after_decimalpoint);
 cout.setf(ios::fixed);
 cout.setf(ios::showpoint);
 cout.setf(ios::showpos);
 cout.precision(number_after_decimalpoint);

 double next;
 while (messy_file >> next) ← Satisfied if there is a
 { next number to read
 cout << setw(field_width) << next << endl;
 neat_file << setw(field_width) << next << endl;
 }
}
```

# Display 6.6

## (3/3)



### Formatting Output (part 3 of 3)

---

#### **rawdata.dat**

(Not changed by program.)

```
10.37 -9.89897
2.313 -8.950 15.0

 7.33333 92.8765
-1.237568432e2
```

#### **neat.dat**

(After program is run.)

```
+10.37000
 -9.89897
 +2.31300
 -8.95000
+15.00000
 +7.33333
+92.87650
-123.75684
```

#### **Screen Output**

```
+10.37000
 -9.89897
 +2.31300
 -8.95000
+15.00000
 +7.33333
+92.87650
-123.75684
End of program.
```

## Checking Input (part 1 of 2)

---

```
//Program to demonstrate the functions new_line and get_input.
#include <iostream>
using namespace std;

void new_line();
//Discards all the input remaining on the current input line.
//Also discards the '\n' at the end of the line.
//This version only works for input from the keyboard.

void get_int(int& number);
//Postcondition: The variable number has been
//given a value that the user approves of.

int main()
{
 int n;

 get_int(n);
 cout << "Final value read in = " << n << endl
 << "End of demonstration.\n";
 return 0;
}

//Uses iostream:
void new_line()
{
 char symbol;
 do
 {
 cin.get(symbol);
 } while (symbol != '\n');
}
```

# Display 6.7 (1/2)



# Display 6.7

## (2/2)



### Checking Input (part 2 of 2)

---

```
//Uses iostream:
void get_int(int& number)
{
 char ans;
 do
 {
 cout << "Enter input number: ";
 cin >> number;
 cout << "You entered " << number
 << " Is that correct? (yes/no): ";
 cin >> ans;
 new_line();
 } while ((ans != 'Y') && (ans != 'y'));
}
```

### Sample Dialogue

```
Enter input number: 57
You entered 57 Is that correct? (yes/no): No
Enter input number: 75
You entered 75 Is that correct? (yes/no): yes
Final value read in = 75
End of demonstration.
```

```
//Program to create a file called cplusplus.dat that is identical to the file
//cad.dat, except that all occurrences of 'C' are replaced by "C++".
//Assumes that the uppercase letter 'C' does not occur in cad.dat except
//as the name of the C programming language.
```

```
#include <fstream>
#include <iostream>
#include <cstdlib>
using namespace std;
```

```
void add_plus_plus(ifstream& in_stream, ofstream& out_stream);
//Precondition: in_stream has been connected to an input file with open.
//out_stream has been connected to an output file with open.
//Postcondition: The contents of the file connected to in_stream have been
//copied into the file connected to out_stream, but with each 'C' replaced
//by "C++". (The files are not closed by this function.)
```

```
int main()
{
 ifstream fin;
 ofstream fout;

 cout << "Begin editing files.\n";

 fin.open("cad.dat");
 if (fin.fail())
 {
 cout << "Input file opening failed.\n";
 exit(1);
 }

 fout.open("cplusplus.dat");
 if (fout.fail())
 {
 cout << "Output file opening failed.\n";
 exit(1);
 }

 add_plus_plus(fin, fout);
```

# Display 6.8 (1/2)



```
 fin.close();
 fout.close();

 cout << "End of editing files.\n";
 return 0;
}

void add_plus_plus(ifstream& in_stream, ofstream& out_stream)
{
 char next;

 in_stream.get(next);
 while (! in_stream.eof())
 {
 if (next == 'C')
 out_stream << "C++";
 else
 out_stream << next;

 in_stream.get(next);
 }
}
```

### **cad.dat**

(Not changed by program.)

C is one of the world's most modern programming languages. There is no language as versatile as C, and C is fun to use.

### **cplusplus.dat**

(After program is run.)

C++ is one of the world's most modern programming languages. There is no language as versatile as C++, and C++ is fun to use.

### **Screen Output**

Begin editing files.  
End of editing files.

# Display 6.8 (2/2)





# Display 6.9

## (1/2)



### Some Predefined Character Functions in `cctype` (*part 1 of 2*)

---

| Function                       | Description                                              | Example                                                                      |
|--------------------------------|----------------------------------------------------------|------------------------------------------------------------------------------|
| <code>toupper(Char_Exp)</code> | Returns the uppercase version of <code>Char_Exp</code> . | <pre>char c = toupper('a');<br/>cout &lt;&lt; c;<br/><b>Outputs:</b> A</pre> |
| <code>tolower(Char_Exp)</code> | Returns the lowercase version of <code>Char_Exp</code> . | <pre>char c = tolower('A');<br/>cout &lt;&lt; c;<br/><b>Outputs:</b> a</pre> |

---

## Some Predefined Character Functions in ctype (part 2 of 2)

| Function                       | Description                                                                                                                                     | Example                                                                                                                                                                                    |
|--------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>isupper(Char_Exp)</code> | Returns <i>true</i> provided <i>Char_Exp</i> is an uppercase letter; otherwise, returns <i>false</i> .                                          | <pre>if (isupper(c))     cout &lt;&lt; c &lt;&lt; " is uppercase."; else     cout &lt;&lt; c         &lt;&lt; " is not uppercase.";</pre>                                                  |
| <code>islower(Char_Exp)</code> | Returns <i>true</i> provided <i>Char_Exp</i> is a lowercase letter; otherwise, returns <i>false</i> .                                           | <pre>char c = 'a'; if (islower(c))     cout &lt;&lt; c &lt;&lt; " is lowercase."; <b>Outputs:</b> a is lowercase.</pre>                                                                    |
| <code>isalpha(Char_Exp)</code> | Returns <i>true</i> provided <i>Char_Exp</i> is a letter of the alphabet; otherwise, returns <i>false</i> .                                     | <pre>char c = '\$'; if (isalpha(c))     cout &lt;&lt; c &lt;&lt; " is a letter."; else     cout &lt;&lt; c         &lt;&lt; " is not a letter."; <b>Outputs:</b> \$ is not a letter.</pre> |
| <code>isdigit(Char_Exp)</code> | Returns <i>true</i> provided <i>Char_Exp</i> is one of the digits '0' through '9'; otherwise, returns <i>false</i> .                            | <pre>if (isdigit('3'))     cout &lt;&lt; "It's a digit."; else     cout &lt;&lt; "It's not a digit."; <b>Outputs:</b> It's a digit.</pre>                                                  |
| <code>isspace(Char_Exp)</code> | Returns <i>true</i> provided <i>Char_Exp</i> is a whitespace character, such as the blank or new-line symbol; otherwise, returns <i>false</i> . | <pre>//Skips over one "word" and //sets c equal to the first //whitespace character after //the "word": do {     cin.get(c); } while (! isspace(c));</pre>                                 |

# Display 6.9 (2/2)

