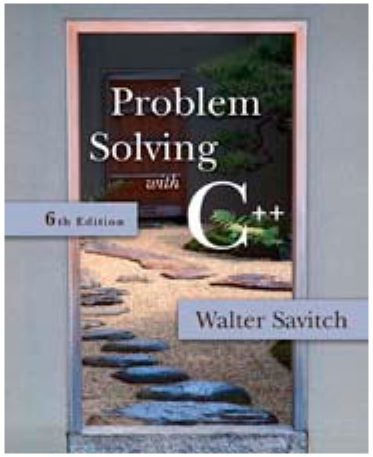
APS105: Lecture 14

Wael Aboelsaadat

wael@cs.toronto.edu

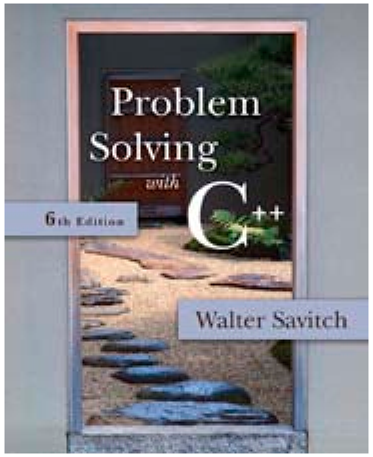
http://ccnet3.utoronto.ca/20079/aps105h1f/

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



Chapter 6

I/O Streams as an Introduction to Objects and Classes



Closing a File

- After using a file, it should be closed
  - This disconnects the stream from the file
  - Close files to reduce the chance of a file being corrupted if the program terminates abnormally
- It is important to close an output file if your program later needs to read input from the output file
- The system will automatically close files if you forget as long as your program ends normally

Display 6.1

Objects

- An object is a variable that has functions and data associated with it
  - in_stream and out_stream each have a function named open associated with them
  - in_stream and out_stream use different versions of a function named open
    - One version of open is for input files
    - A different version of open is for output files

Member Functions

- A member function is a function associated with an object
  - The open function is a member function of in_stream in the previous examples
  - A different open function is a member function of out_stream in the previous examples

Objects and Member Function Names

- Objects of different types have different member functions
  - Some of these member functions might have the same name
- Different objects of the same type have the same member functions

Classes

- A type whose variables are objects, is a class
  - ifstream is the type of the in_stream variable (object)
  - ifstream is a class
  - The class of an object determines its member functions
  - Example:

```
ifstream in_stream1, in_stream2;
```

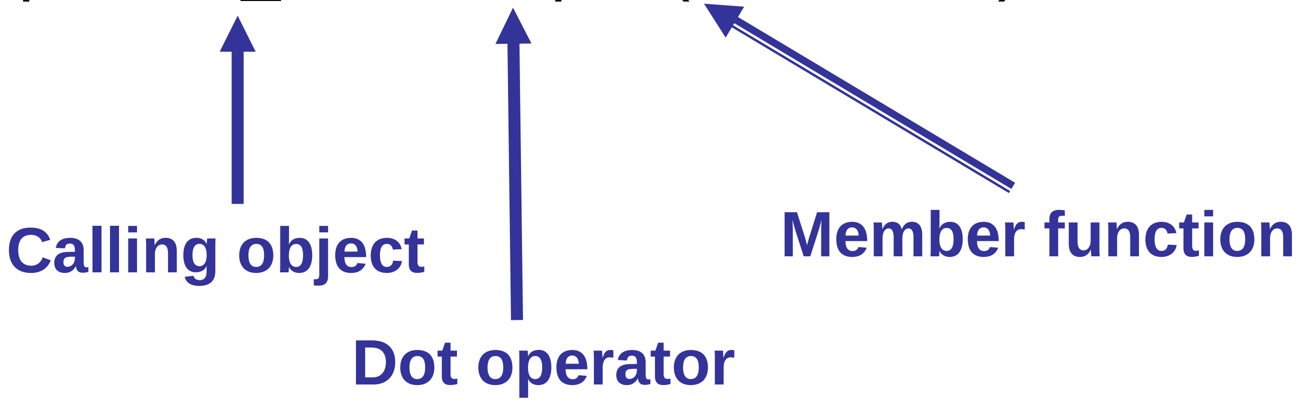
    - in_stream1.open and in_stream2.open are the same function but might have different arguments

Class Member Functions

- Member functions of an object are the member functions of its class
- The class determines the member functions of the object
  - The class ifstream has an open function
  - Every variable (object) declared of type ifstream has that open function

Calling a Member Function

- Calling a member function requires specifying the object containing the function
- The calling object is separated from the member function by the dot operator
- Example: in_stream.open("infile.dat");



Member Function Calling Syntax

- Syntax for calling a member function:

  Calling_object
  .Member_Function_Name(Argument_list);

Errors On Opening Files

- Opening a file could fail for several reasons
  - Common reasons for open to fail include
    - The file might not exist
    - The name might be typed incorrectly

- May be no error message if the call to open fails
  - Program execution continues!

Catching Stream Errors

- Member function fail, can be used to test the success of a stream operation
  - fail returns a boolean type (true or false)
  - fail returns true if the stream operation failed

Halting Execution

- When a stream open function fails, it is generally best to stop the program
- The function exit, halts a program
  - exit returns its argument to the operating system
  - exit causes program execution to stop
  - exit is NOT a member function
- Exit requires the include and using directives

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

Using fail and exit

- Immediately following the call to open, check that the operation was successful:

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

Display 6.2

Techniques for File I/O

- When reading input from a file…
  - Do not include prompts or echo the input
    - The lines

```
cout << "Enter the number: ";
cin  >> the_number;
 cout << "The number you entered is "
        << the_number;
```

become just one line

```
in_file >> the_number;
```

  - The input file must contain exactly the data expected

Appending Data (optional)

- Output examples so far create new files
  - If the output file already contains data, that data is lost
- To append new output to the end an existing file
  - use the constant ios::app defined in the iostream library:

```
        outStream.open("important.txt", ios::app);
```

  - If the file does not exist, a new file will be created

Display 6.3

File Names as Input (optional)

- Program users can enter the name of a file to use for input or for output
- Program must use a variable that can hold multiple characters
  - A sequence of characters is called a string
  - Declaring a variable to hold a string of characters:

    ```
    char   file_name[16];
    ```

    - file_name is the name of a variable
    - Brackets enclose the maximum number of characters + 1
    - The variable file_name contains up to 15 characters

Using A Character String

-
  ```
  char file_name[16];
  cout << "Enter the file_name ";
  cin >> file_name;
  ifstream in_stream;
  in_stream.open(file_name);
  if (in_stream.fail( ) )
  {
          cout << "Input file opening failed.\n";
          exit(1);
  }
  ```

Display 6.4 (1)

Display 6.4 (2)

Section 6.1 Conclusion

- Can you
  - Write a program that uses a stream called fin which will be connected to an input file and a stream called fout which will be connected to an output file? How do you declare fin and fout? What include directive, if any, do you nee to place in your program file?
  - Name at least three member functions of an iostream object and give examples of usage of each?

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)

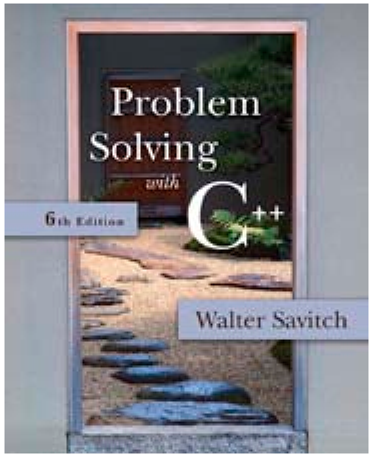
End of File Example

- 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;
    ```

6.3

Character I/O



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)

Checking Input: Check for Yes or No?

- get_int continues to ask for a number until the user responds 'Y' or 'y' using the do-while loop

```
do
  {
     // the loop body
  } while  ((ans !='Y') &&(ans != 'y') )
```

- Why not use ((ans =='N') | | (ans == 'n'))?
  - User must enter a correct response to continue a loop tested with ((ans =='N') | | (ans == 'n'))
    - What if they mis-typed "Bo" instead of "No"?
  - User must enter a correct response to end the loop tested with ((ans !='Y') &&(ans != 'y'))

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'

A Fix To Remove '\n'

-
  ```
  cout << "Enter a number:\n";
  int number;
  cin >> number;
  cout << "Now enter a letter:\n";
  char symbol;
  cin >>symbol;
  ```

Another '\n' Fix

-
  ```
  cout << "Enter a number:\n";
  int number;
  cin >> number;
  new_line( ); // From Display 6.7
  cout << "Now enter a letter:\n";
  char symbol;
  cin.get(symbol);
  ```

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 cplusad.dat except that every 'C' is changed to "C++" in cplusad.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)

Display 6.1

Back Next

Simple File Input/Output

```
//Reads three numbers from the file infile.dat, sums the numbers,
//and writes the sum to the file outfile.dat.
//(A better version of this program will be given in Display 5.2.)
#include <fstream>

int main( )
{
    using namespace std;
    ifstream in_stream;
    ofstream out_stream;

    in_stream.open("infile.dat");
    out_stream.open("outfile.dat");

    int first, second, third;
    in_stream >> first >> second >> third;
    out_stream << "The sum of the first 3\n"
               << "numbers in infile.dat\n"
               << "is " << (first + second + third)
               << endl;

    in_stream.close( );
    out_stream.close( );

    return 0;
}
```

infile.dat
(Not changed by program.)

```
1
2
3
4
```

outfile.dat
(After program is run.)

```
The sum of the first 3
numbers in infile.dat
is 6
```

There is no output to the screen and no input from the keyboard.

Display 6.2

Back Next

File I/O with Checks on open

```
//Reads three numbers from the file infile.dat, sums the numbers,
//and writes the sum to the file outfile.dat.
#include <fstream>
#include <iostream>
#include <cstdlib>

int main( )
{
    using namespace std;
    ifstream in_stream;
    ofstream out_stream;

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

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

    int first, second, third;
    in_stream >> first >> second >> third;
    out_stream << "The sum of the first 3\n"
               << "numbers in infile.dat\n"
               << "is " << (first + second + third)
               << endl;

    in_stream.close( );
    out_stream.close( );

    return 0;
}
```

Screen Output (If the file `infile.dat` does not exist)

```
Input file opening failed.
```

Display 6.3

Back Next

Appending to a File (*Optional*)

```
//Appends data to the end of the file data.txt.
#include <fstream>
#include <iostream>

int main( )
{
    using namespace std;

    cout << "Opening data.txt for appending.\n";
    ofstream fout;
    fout.open("data.txt", ios::app);
    if (fout.fail( ))
    {
        cout << "Input file opening failed.\n";
        exit(1);
    }

    fout << "5 6 pick up sticks.\n"
         << "7 8 ain't C++ great!\n";

    fout.close( );
    cout << "End of appending to file.\n";

   return 0;
}
```

Sample Dialogue

data.txt
(Before program is run.)

```
1 2 bucket my shoe.
3 4 shut the door.
```

data.txt
(After program is run.)

```
1 2 bucket my shoe.
3 4 shut the door.
5 6 pick up sticks.
7 8 ain't C++ great!
```

Screen Output

```
Opening data.txt for appending.
End of appending to file.
```

Display 6.4 (1/2)

Back Next

Inputting a File Name (*Optional*) (*part 1 of 2*)

```
//Reads three numbers from the file specified by the user, sums the numbers,
//and writes the sum to another file specified by the user.
#include <fstream>
#include <iostream>
#include <cstdlib>

int main( )
{
    using namespace std;
    char in_file_name[16], out_file_name[16];
    ifstream in_stream;
    ofstream out_stream;

    cout << "I will sum three numbers taken from an input\n"
         << "file and write the sum to an output file.\n";
    cout << "Enter the input file name (maximum of 15 characters):\n";
    cin >> in_file_name;
    cout << "Enter the output file name (maximum of 15 characters):\n";
    cin >> out_file_name;
    cout << "I will read numbers from the file "
         << in_file_name << " and\n"
         << "place the sum in the file "
         << out_file_name << endl;

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

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

Display 6.4 (2/2)

Back

Next

Inputting a File Name (*Optional*) (*part 2 of 2*)

```
    int first, second, third;
    in_stream >> first >> second >> third;
    out_stream << "The sum of the first 3\n"
               << "numbers in " << in_file_name << endl
               << "is " << (first + second + third)
               << endl;

    in_stream.close( );
    out_stream.close( );

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

numbers.dat
(Not changed by program.)

```
1
2
3
4
```

sum.dat
(After program is run.)

```
The sum of the first 3
numbers in numbers.dat
is 6
```

Sample Dialogue

```
I will sum three numbers taken from an input
file and write the sum to an output file.
Enter the input file name (maximum of 15 characters):
numbers.dat
Enter the output file name (maximum of 15 characters):
sum.dat
I will read numbers from the file numbers.dat and
place the sum in the file sum.dat
End of Program.
```

Display 6.5

Back Next

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

Display 6.6 (1/3)

Back Next

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;
```

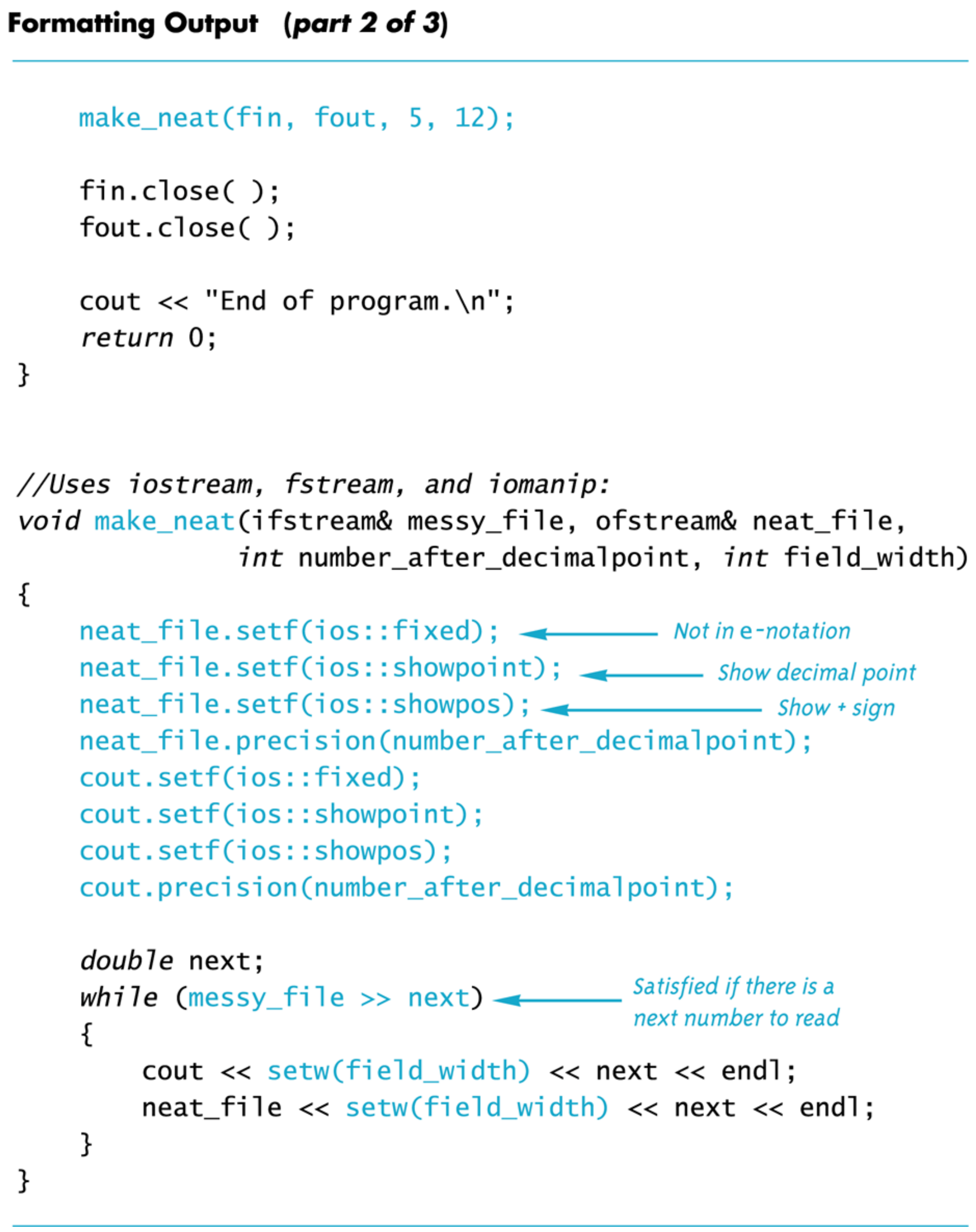
Needed for setw

Stream parameters must be call-by-reference.

```
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);
    }
```

Display 6.6 (2/3)

Back Next



Formatting Output *(part 2 of 3)*

```
    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)

Back

Next

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.
```

Display 6.7 (1/2)

Back Next

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 (2/2)

Back Next

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.
```

Display 6.8 (1/2)

Back Next

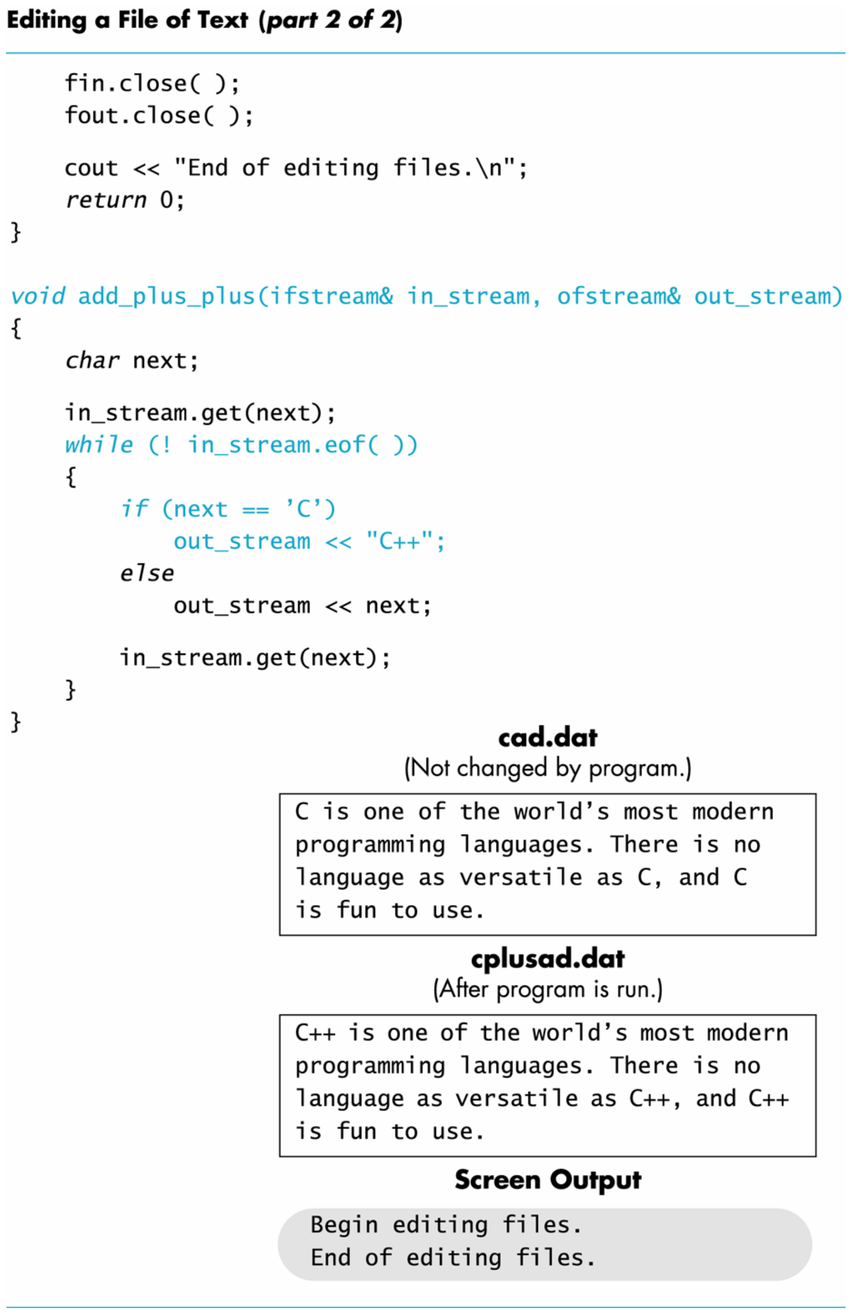
Editing a File of Text ***(part 1 of 2)***

```
//Program to create a file called cplusad.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("cplusad.dat");
    if (fout.fail( ))
    {
        cout << "Output file opening failed.\n";
        exit(1);
    }

    add_plus_plus(fin, fout);
```

Display 6.8 (2/2)

Back

Next



Editing a File of Text (*part 2 of 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.
```

cplusad.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.
```