I/O Streams

Prof. Seokin Hong

SUNGKYUNKWAN UNIVERSITY

Agenda

- Streams and Basic File I/O
- Tools for Stream I/O
- Character I/O

Streams and Basic File I/O

I/O Streams

- **A stream is a flow of data.**
  - **Input stream**: Data flows into the program
    - If input stream flows from **keyboard**
    - If input stream flows from a **file**
  - **Output stream**: Data flows out of the program
    - To the screen
    - To a file

cin And cout Streams

- **cin**
  - Input stream connected to the keyboard
- **cout**
  - Output stream connected to the screen

- **cin and cout defined in the iostream library**
  - Use include directive: **#include <iostream>**

File I/O

- **You can declare your own streams to use with files**

- **Reading from a file**
  - Taking input from a file
  - Just as done from the keyboard

- **Writing to a file**
  - Sending output to a file
  - Just as done to the screen

Streams is a kind of Objects

- **A stream is a special kind of variable called an object**
  - Streams use special functions instead of the assignment operator to change values

Objects : special variables that have their own special-purpose functions

- **Stream Variables**
  - Like other variables, a stream variable…
    - Must be declared before it can be used
    - Must be initialized before it contains valid data
      - Initializing a stream means connecting it to a file
      - The value of the stream variable can be thought of as the file it is connected to
  - Can have its value changed
    - Changing a stream value means disconnecting from one file and connecting to another

Declaring An Input-file Stream Variable

- **Input-file streams are of type ifstream**

- **Type ifstream is defined in the fstream library**
  - You must use the include and using directives

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

- **Declare an input-file stream variable using**

```
ifstream inStream;
```

Declaring An Output-file Stream Variable

- **Ouput-file streams of are type ofstream**

- **Type ofstream is defined in the fstream library**
  - You must use these include and using directives

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

- **Declare an output-file stream variable using**

```
ofstream outStream;
```

Connecting To A File

- **Once a stream variable is declared, connect it to a file**
  - Connecting a stream to a file is opening the file
  - Use the **open** function of the stream object

```
inStream.open("infile.dat");
```

Double quotes

File name on the disk

Using The Input Stream

- **input-stream variable can be used to produce input just as you would use cin with the extraction operator**
  - Example:

```
int oneNumber, anotherNumber;
inStream >> oneNumber
         >> anotherNumber;
```

Using The Output Stream

- **An output-stream works similarly to the input-stream**

```
ofstream outStream;
outStream.open("outfile.dat");

outStream << "one number = "
          << oneNumber
          << "another number = "
          << anotherNumber;
```

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

- **The system will automatically close files if you forget as long as your program ends normally**

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 inStream;
    ofstream outStream;

    inStream.open("infile.dat");
    outStream.open("outfile.dat");
    int first, second, third;
    inStream >> first >> second >> third;
    outStream << "The sum of the first 3\n"
              << "numbers in infile.dat\n"
              << "is " << (first + second + third)
              << endl;
    inStream.close( );
    outStream.close( );
    return 0;
}
```

infile.dat (Not changed by program.)	outfile.dat (After program is run.)
1 2 3 4	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.

Streams is a kind of Objects!

- **An object is a variable that has functions and data associated with it**
  - Functions and data associated with an object are called "**member**"

  - inStream and outStream each have a function named **open()** associated with them
  - inStream and outStream 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

Classes and Objects

- **A type whose variables are objects, is a class**
  - ifstream is a class
  - The class of an object determines its member functions
    - The class ifstream has an open function
    - Every variable (object) declared of type ifstream has that open 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: inStream.open("infile.dat");**

Calling object

Dot operator

Member function

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

- **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

Errors On Opening Files (Cont'd)

- **When a stream open function fails, it is generally best to stop the program**
  - The function exit, halts a program
    - exit() causes program execution to stop
    - exit() returns its argument to the operating system
- **Exit requires the include and using directives**

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

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

Appending Data

- **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);

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.
```

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

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 outStream with:**

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

precision()

- **precision() is a member function of output streams**
  - After outStream.precision(2),
    - Output of numbers with decimal points…
      - 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()

- **setf() is an abbreviation for set flags**
  - A flag is an instruction to do one of two options
    - Example: ios::fixed is a flag
  - Calls to setf() apply only to the stream named in the call

- **Example: Showing in fixed-point notation**
  - After outStream.setf(ios::fixed),
    - All further output of floating point numbers will be written in fixed-point notation

setf()

- **Example: Showing decimal point**
  - After outStream.setf(ios::showpoint),
    - Output of floating point numbers will show the decimal point even if all digits after the decimal point are zeroes

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

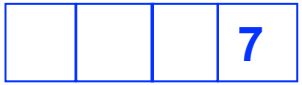
width()

- **Creating space in output**
  - The width() function specifies the number of spaces for the next item

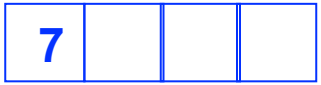
- **Example: Printing the digit 7 in four spaces use**

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

Three of the spaces will be blank



outStream.setf(ios::right);



outStream.setf(ios::left);

Unsetting Flags

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

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

causes the program to stop printing plus signs on positive numbers

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

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 makeNeat(ifstream& messyFile, ofstream&  neatFile);
```

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:**
  - Example : **inStream >> next;**
    - Reads a value from inStream 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

End of File Example (Cont.)

- **To calculate the average of the numbers in a file**

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

double average = sum / count;
```

Program Example

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.

Program Example (Cont.)

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;
    }
}
```

Program Example (Cont.)

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.
```

Character I/O

Character I/O

- **All inputs and outputs are 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 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
  - Does not skip blanks and '\n'
- **get() Syntax**
  - inputStream.get(charVariable)
- **Examples:**

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

ifstream  inStream;
inStream.open("infile.dat");
inStream.get(nextSymbol);
```

Member Function get() – Cont.

- **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

Member Function get() – Cont.

- **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

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
- **put() Syntax**
  - outputStream.put(Char_expression);
- **Examples:**

```
cout.put(nextSymbol);
cout.put('a');
ofstream outStream;
outStream.open("outfile.dat");
outStream.put('Z');
```

Program Example: Checking Input

DISPLAY 6.7 Checking Input *(part 1 of 2)*

```cpp
 //Program to demonstrate the functions newLine and getInput.
#include <iostream>
using namespace std;

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

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


int main( )
{
    int n;

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


//Uses iostream:
void newLine( )
{
    char symbol;
    do
    {
        cin.get(symbol);
    } while (symbol != '\n');
}
//Uses iostream:
void getInt(int& number)
{
    char ans;
    do
    {
        cout << "Enter input number: ";
        cin >> number;
        cout << "You entered " << number
             << ". Is that correct? (yes/no): ";
        cin >> ans;
        newLine( );
    } 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.
```

Inheritance and Output

- **ostream is the class of all output streams**
  - cout is of type ostream

- **ofstream is the class of output-file streams**
  - The ofstream class is a child class of ostream
  - This function can be called with ostream or ofstream arguments

```
 void sayHello(ostream& anyOutStream)
{
        anyOutStream << "Hello";
}
```

Program Example: Another newline() Function

- **This version works for any input stream**

```
void newLine(istream& inStream)
{
        char symbol;
        do
         {
                inStream.get(symbol);
        } while (symbol != '\n');
}
```

```
newLine(cin);   // newLine();
ifstream fin;
fin.open("test.txt");
newLine(fin);    // called with an input-file stream
```

- **A default value can be specified in the parameter list**
  - The default value is selected if no argument is available for the parameter
  - The newLine header can be written as
    newLine (istream & inStream = cin)
  - If newLine is called without an argument, cin is used

Multiple Default Arguments

- **When some formal parameters have default values and others do not,**
  - All formal parameters with default values must be at the end of the parameter list
  - Arguments are applied to the all formal parameters in order
  - The function call must provide at least as many arguments as there are parameters without default values
- **Example**

```
void defaultArgs(int arg1, int arg2 = -3)
 {
                cout << arg1 << ' ' << arg2 << endl;
 }

defaultArgs(5);          //output is   5   -3
defaultArgs(5, 6);       //output is   5   6
```

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'
- **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;
```

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

```
//Uses iostream:
void newLine( )
{
    char symbol;
    do
    {
        cin.get(symbol);
    } while (symbol != '\n');
}
//Uses iostream:
void getInt(int& number)
{
    char ans;
    do
    {
        cout << "Enter input number: ";
        cin >> number;
        cout << "You entered " << number
             << ". Is that correct? (yes/no): ";
        cin >> ans;
        newLine( );
    } 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.
```

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

  - Normally used to determine when we are NOT at the end of the file
    - **Example:**

```
if ( ! inStream.eof( ) )
```

Detecting the End of a File (Cont.)

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

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

- **We have seen two methods**
  - while (inStream >> next)
  - while (! inStream.eof( ))

Program Example: Editing a Text File

- **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 to preserve line breaks
  - get() is used to preserve spaces as well
- **Uses eof to test for end of file**

Program Example: Editing a Text File

DISPLAY 6.8 Editing a File of Text

```
//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 addPlusPlus(ifstream& inStream, ofstream& outStream);
//Precondition: inStream has been connected to an input file with open.
//outStream has been connected to an output file with open.
//Postcondition: The contents of the file connected to inStream have been
//copied into the file connected to outStream, 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( ))
    {
```

Program Example: Editing a Text File (Cont.)

```
23          cout << "Input file opening failed.\n";
24          exit(1);
25      }
26      fout.open("cplusad.dat");
27      if (fout.fail( ))
28      {
29          cout << "Output file opening failed.\n";
30          exit(1);
31      }
32      addPlusPlus(fin, fout);
33      fin.close( );
34      fout.close( );
35      cout << "End of editing files.\n";
36      return 0;
37  }
38
39  void addPlusPlus(ifstream& inStream, ofstream& outStream)
40  {
41      char next;
42      inStream.get(next);
43      while (! inStream.eof( ))
44      {
45            if (next == 'C')
46               outStream << "C++";
47            else
48               outStream << next;
49            inStream.get(next);
50      }
51  }
```

More member function

- **read(char* s, streamsize n)**
  - Extracts *n* characters from the stream and stores them in the array pointed to by *s*.

- **write(const char* s, streamsize n)**
  - Inserts the first *n* characters of the array pointed by *s* into the stream.

https://www.cplusplus.com/reference/istream/istream/read/

https://www.cplusplus.com/reference/ostream/ostream/write/

Copyright Notice

- The contents of this slide deck are taken from the textbook (Problem Solving with C++, Walter Savitch).
- See your textbook for more details.
- Redistribution of this slide deck is not permitted.

NEXT ?

Array, String, Vectors