Lesson 10 - Streams and Files

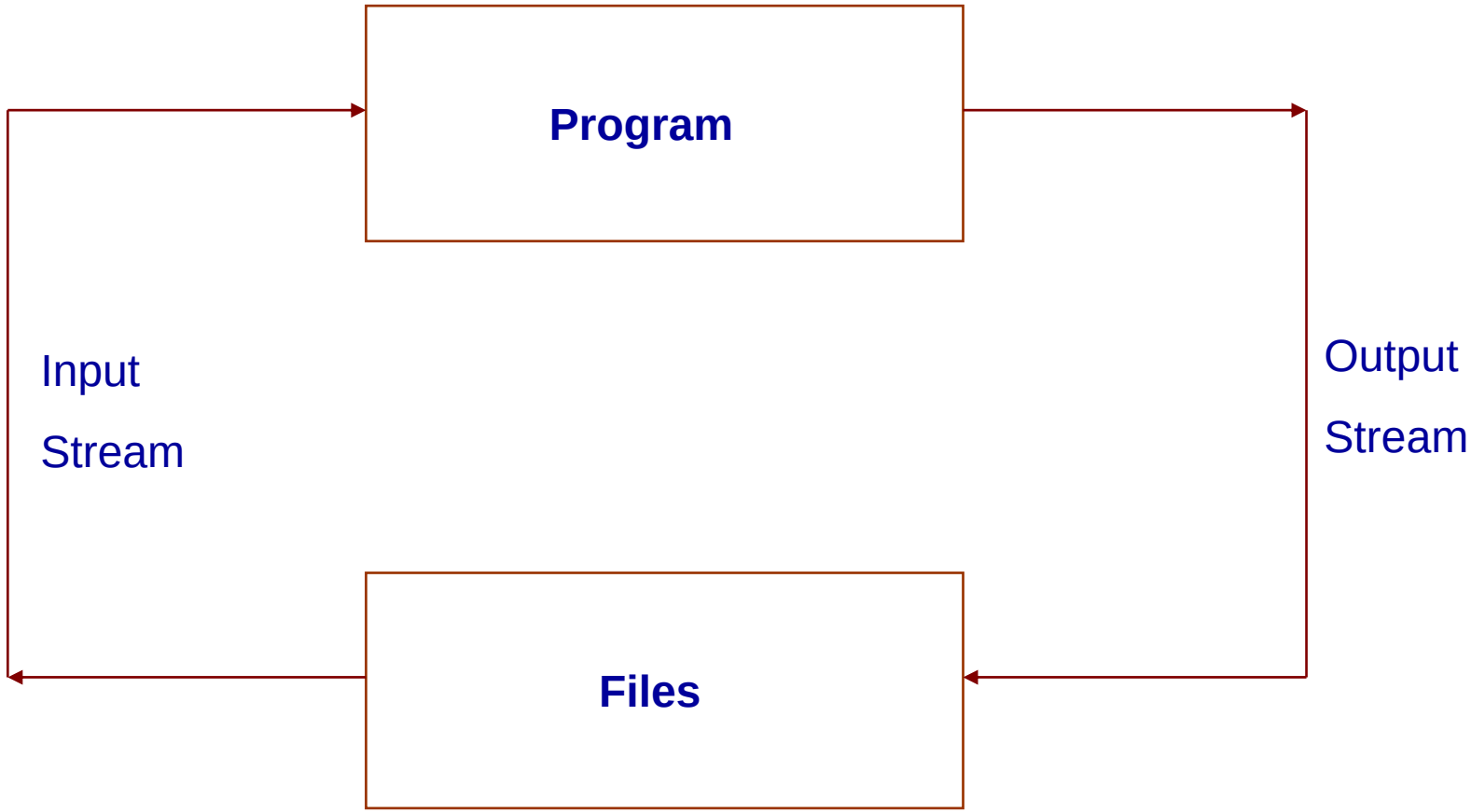
Outline

1. **Introduction**
2. **Input and Output**
3. **Classes for Stream I/O in C++**
4. **File Operations**
5. **Example**

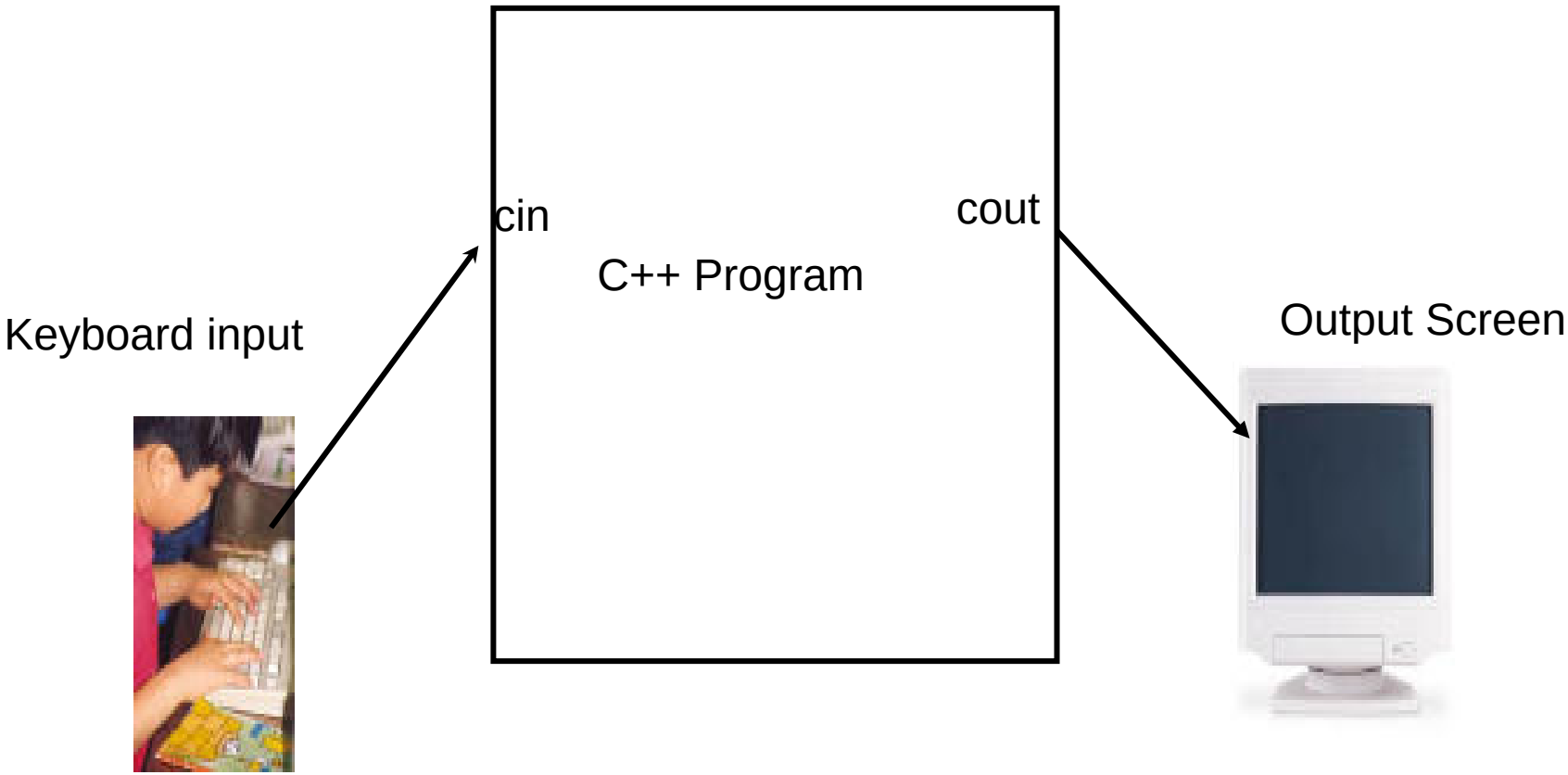
Introduction

- C++ provides:
  - A common interface for reading and writing output;
  - This is done by exploiting a hierarchy of objects;
  - Standard I/O + files seen as **streams**;
  - Once a stream has been created, it can be manipulated using the usual I/O operations.

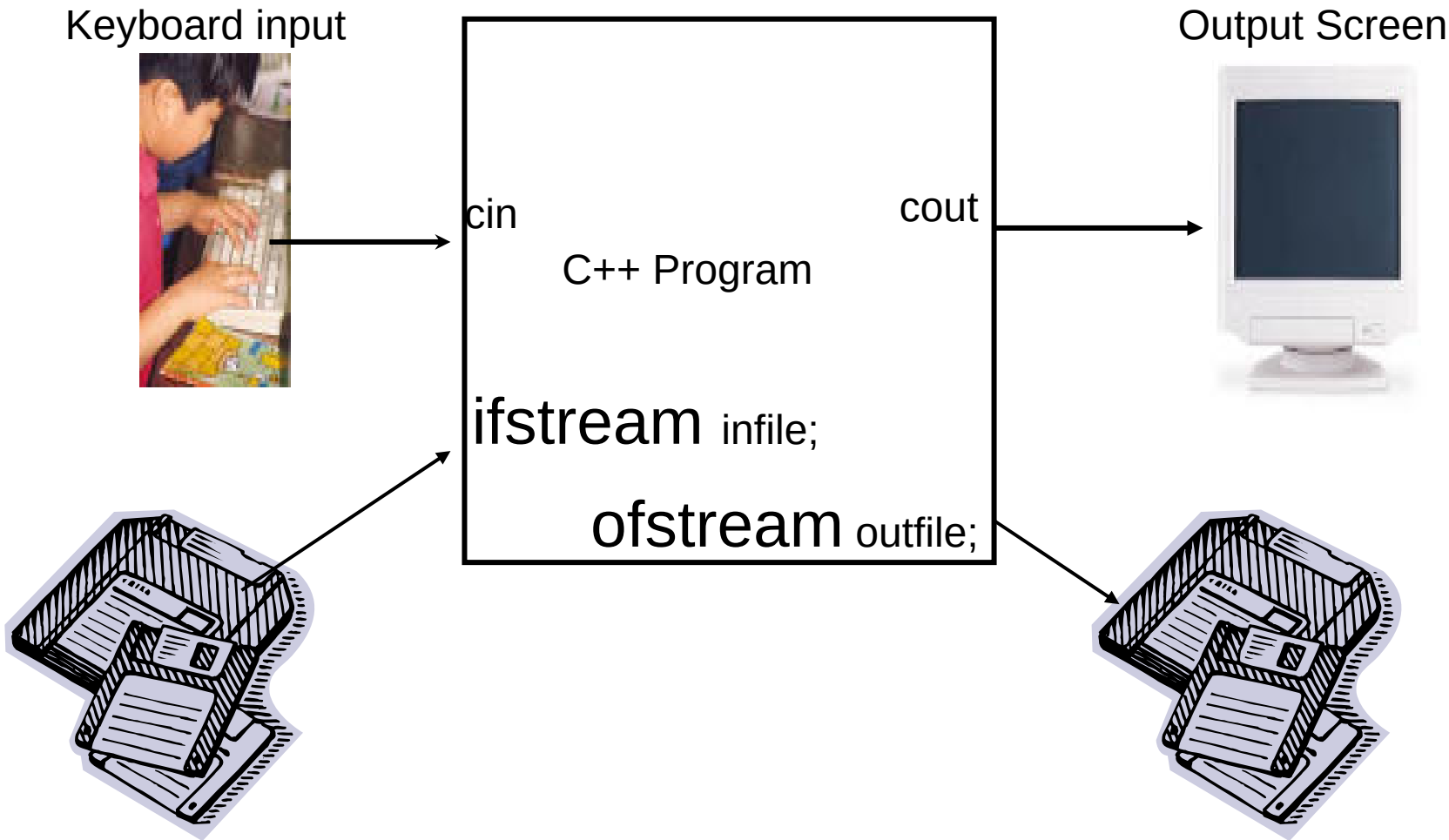
Input and Output



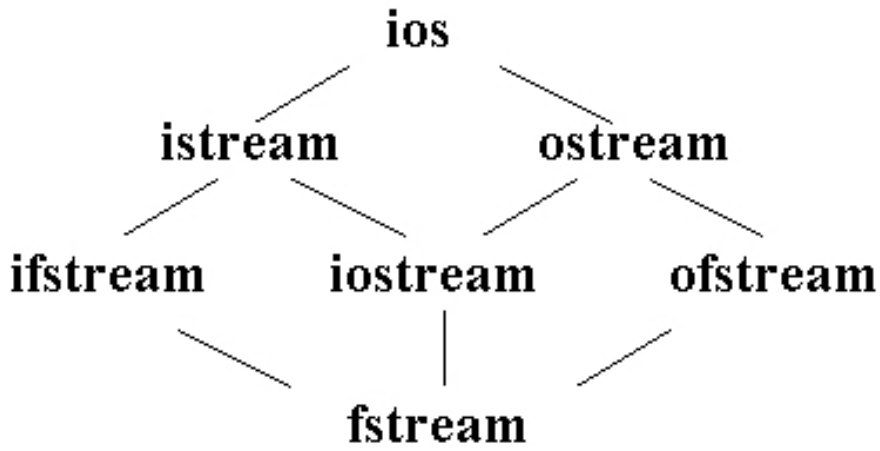
Standard Input Output



File Input / Output features



Classes for Stream I/O in C++



- ios is the base class.
- istream and ostream inherit from ios
- ifstream inherits from istream (and ios)
- ofstream inherits from ostream (and ios)
- iostream inherits from istream and ostream (& ios)
- fstream inherits from ifstream, iostream, and ofstream

I/O Streams

Stream	Description
cin	**Standard input stream**
cout	**Standard output stream**
cerr	**Standard error stream**

File I/O Streams

Stream Classes required for File I/O :

- ifstream
- ofstream
- fstream

Ifstream / ofstream

Input

- Input file stream Class
- open() is a member function of the class **ifstream**
- Inherited functions of **ifstream** class, from the class **istream** are
  - **`get(); getline(); read(); seekg(); tellg();`**

Output

- Output file stream Class
- open() is a member function of the class **ofstream**
- Inherited functions of **ofstream** class, from the class **ostream** are
  - **`put(); write(); seekp(); tellp();`**

fstream

- It supports files for simultaneous input and output
- fstream is derived from
  - ifstream
  - ofstream
  - iostream

- They are parent classes and fstream is the child class

- Member functions of the class fstream
  - **`open(); close(); seekg(); seekp(); tellg(); tellp();`**

Stream I/O Library Header Files

Note: There is no ".h" on standard header files.

Be careful about "*using namespace std*"

- iostream -- contains basic information required for all stream I/O operations
- iomanip -- contains information useful for performing formatted I/O with parameterized stream manipulators
- fstream -- contains information for performing file I/O operations
- strstream -- contains information for performing in-memory I/O operations (i.e., into or from strings in memory)

File Operations

- Open, close, << and >> operators
- **`eof()`** operation on an input file object returns a true or false (Boolean)
- **`get()`** reads a single character from an input file and put(char) writes a single character into an output file.
- **`fail()`** operation indicates if the opening of a file is successful or failure. Return Boolean type (true or false)

Defining File Streams

1. Include **fstream (#include <fstream>)**
2. declare file stream variable (object)
    1. **ifstream fin;**
    2. **ofstream fout;**
3. use open() to initialize file stream variable
4. use input file stream variable as you would use **cin** and use output file stream variable as you would use **cout**
5. use close() to close the file when finished with it

Writing Output to a File

- Similar to writing to screen
- Use object connected to output file
- Need the fstream header

```
#include <fstream>
```

- Open file for writing
  - Declare object of *ofstream* class

```
ofstream outfile;
```

Opening Files

- General form

  outfile.open(“*file_name*”);

- Choose *object_name* like variable name
- *object_name* is object of class ofstream
- Filename is where output will be stored

  Ex: outfile.open(“grades.out”);

Writing to Files

- General form

  object_name << ***variable_name*;**

- Use ofstream object to write to file like cout was used

  outfile << “Salary for week was ” << money;

- Additional writing appended to file

Closing Files (input and output)

- General form

  ***object_name*.close ( );**

- Use C++ library function **close**
- Use both object and function name separated by a period
- Example: outfile.close( );

Open File for Reading

- Need fstream header

  #include <fstream>

- Declare object of ifstream

  ifstream infile;

- **Open the file:**

  infile.open(“points.dat”);

- Use ifstream object to read file like cin

Reading From a File

- Use ifstream object to read file like cin used for keyboard

```
infile >> salary1 >> salary2;
```

- C++ looks for whitespace between numbers
  - Newline character treated as whitespace
- Additional reading continues sequentially

Example: General Behaviour

- We want to access a file containing numbers in order to compute the average of this numbers. In particular our program should:
  - display a prompt for the name of the input file;
  - read this file;
  - open a connection from itself to that input file;
  - read the numbers contained in the input file;
  - count them and compute their sum;
  - close the connection;
  - compute and display the average of the numbers.

Example: Algorithm

0. Display a prompt for input file name.
1. Read name of input file from *cin* into *inFileName*.
2. Open connection named *fin* to file named in *inFileName*.
3. Initialize *sum*, *count* to zero.
4. Loop:
    a. Read a value from *fin* into *number*;
    b. If no values were left, terminate repetition.
    c. Add *number* to *sum*.
    d. Increment *count*.
    End loop.
5. Close *fin*.
6. If count > 0: display *sum / count*.
    Else display error message.
    End if.

Example: Notes

- To establish connections to an input file, the fstream library provides the *ifstream* class.
- Always check using the ifstream **`is_open()`** function member.
- Once an ifstream is created, it can be read from using **`>>`**, like an istream.
- The ifstream function member **`eof()`** returns true if the last attempted read found no data remaining in the file.
- The fstream function member **`close()`** destroys the connection between a program and a file.

Example: Coding (1)

```
/* average.cpp
 * ...
 */
#include <iostream>                    // cin, cout, ...
#include <fstream>                     // ifstream, ofstream, ...
#include <string>                      // string
#include <cassert>                     // assert()
using namespace std;

int main()
{
  cout << "\nTo average the numbers in an input file,"
       << "\n enter the name of the file: ";
  string inFileName;
  cin >> inFileName;

  ifstream fin(inFileName.data()); // open the connection

  assert(fin.is_open());           // verify it opened

  double number, sum = 0.0;        // variables for
  int count = 0;                   //  computing average
```

Example: Coding (2)

```
  while (true)                  // input loop
  {
    fin >> number;              //  read number

    if (fin.eof()) break;       //  if none were left, quit

    sum += number;              //  add it to sum
    count++;                    //  bump count
  }                             // end loop

  fin.close();                  // close fstream

  if (count > 0)
    cout << "\nThe average of the values in "
         << inFileName << " is " << sum/count << endl;
  else
    cout << "\n*** No values found in file "
         << inFileName << endl;
}
```

Example: Further Observations (1)

If a program tries to open an ofstream to a file that doesn't exist, the open operation creates a new, empty file for output.

If a program tries to open an ofstream to a file that does exist, the open operation (by default) empties that file of its contents, creating a clean file for output.

Once an ifstream (or ofstream) has been opened, it can be read from using the usual input (or output) operations:

- input: >>, get(), getline(), ...
- output: <<, put(), ...

In general, anything that can be done to an istream (or ostream) can also be done to an ifstream (or ofstream).

Example: Further Observations (2)

When the most recent input operation found no data remaining in the file, the input operation is said to *fail*.

This can be detected using the ifstream function member **`eof()`** (or **`fail()`**), which returns true if the last input operation encountered the end of the file, and returns false otherwise.

Once we are done using an ifstream (or ofstream), it can be closed using the **`close()`** function member:

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

Most systems limit the number of files a program can have open simultaneously, so it is a good practice to close a stream when you are finished using it.