CS 280
Programming Language Concepts

Spring 2025

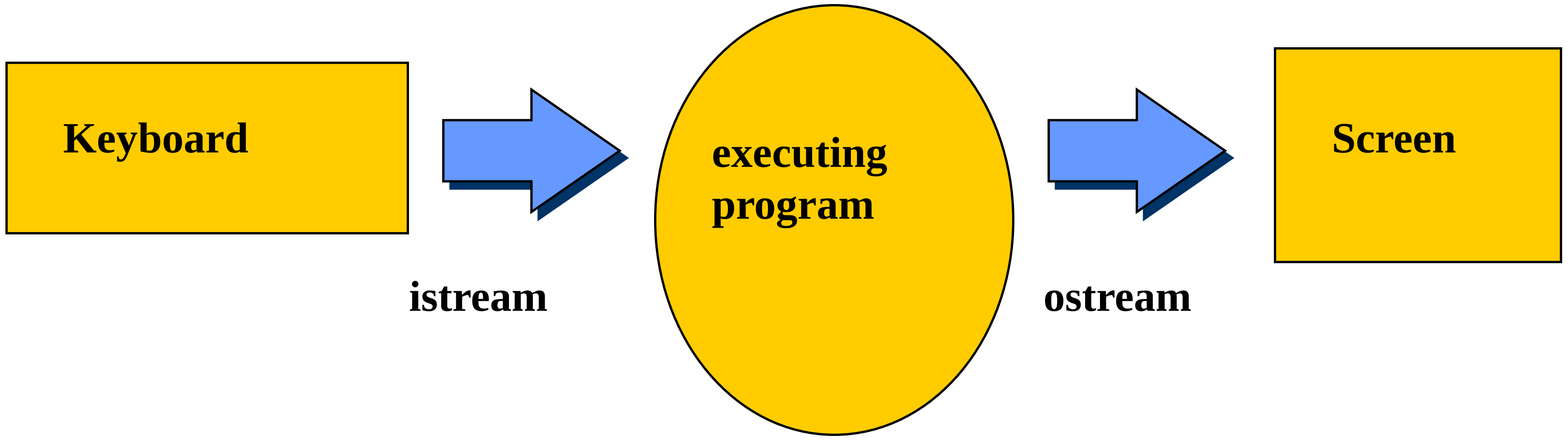
C++ Streams and Files

Topics

- C++ Input/Output
- Files
- String Stream Processing
- Examples
- Recitation Assignment 2

C++ Input/Output

- **No built-in I/O in C++**
- **A library provides input stream and output stream**



>> Operator

>> is called the **input or extraction operator**

>> is a binary operator

>> is left associative

```
int age, weight;
```

Expression	**Has value**
`cin >> age`	`cin`

Statement

```
cin >> age >> weight;
```

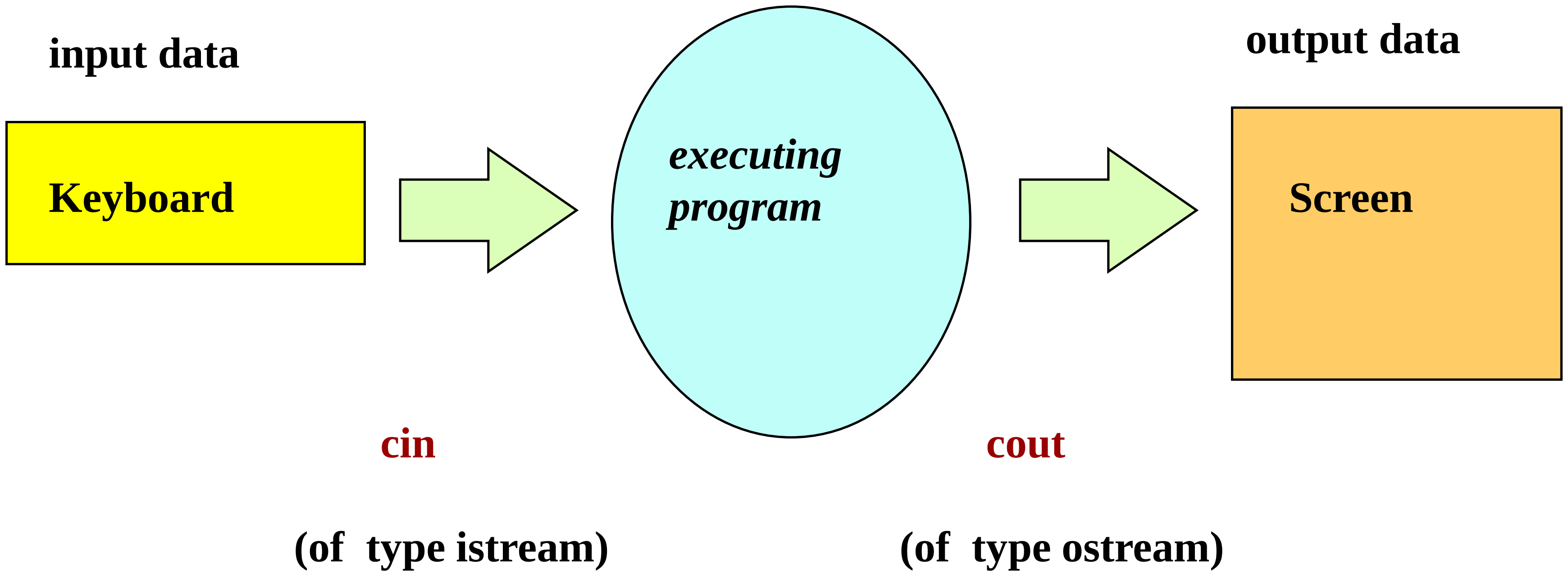
- Reading an integer
- What happens if the user enters a string of non digit characters?

Extraction Operator (>>)

- "skips over" (actually reads but does not store anywhere) leading **white space characters** as it reads your data from the input stream (either keyboard or disk file).
  - Returns **`0`** when EOF encountered
    - Otherwise, returns reference to the istream object, **`cin`**.
    - **`cin >> grade`**
  - State bits set if errors occur.
- End-of-file
  - Indicates end of input
    - *ctrl-z* on IBM-PCs/Windows
    - *ctrl-d* on UNIX and Macs
  - **`cin.eof()`**
    - Returns **`1`** (**`true`**) if EOF has occurred

Keyboard and Screen I/O

#include <iostream>



Reading Data Using get() Function

- The `get()` function can be used to read a single character.
- `get()` obtains the very next character from the input stream without skipping any leading whitespace characters
  - `cin.get()`

    Returns one character from stream (even whitespace)

    Returns `EOF` if end-of-file encountered
- The **ignore()** function is used to skip (read and discard) characters in the input stream. The call
  - `cin.ignore(howMany, whatChar);`
  - will skip over up to **howMany** characters or until **whatChar** has been read, whichever comes first
- The **put()** function can be used to display a single character on the output stream.

Reading Data Using get() Function

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

int main()
{
    int character; // use int, because char cannot represent EOF

    // prompt user to enter line of text
    cout << "Before input, cin.eof() is " << cin.eof() << endl
         << "Enter a sentence followed by end-of-file:" << endl;

    // use get to read each character; use put to display it
    while ( ( character = cin.get() ) != EOF )
        cout.put( character );

    // display end-of-file character
    // use int, because char cannot represent EOF
    cout << "\nEOF in this system is: "<< character << endl;
    cout << "After input, cin.eof() is " << cin.eof() << endl;
    return 0;
}
```

Reading Data Using get() Function

- You can test a character if it is a decimal digit, alphabetic, alphanumeric, or a space by including <cctype> header and using the functions:
  - `isdigit()`
  - `isalpha()`
  - `isalnum()`
  - `isspace()`

```
Before input, cin.eof() is 0
Enter a sentence followed by end-of-file:
testing the eof on this system.
testing the eof on this system.
^Z

EOF in this system is: -1
After input, cin.eof() is 1
```

getline() Function

- Because the extraction operator stops reading at the first trailing whitespace, >> cannot be used to input a string with blanks in it.
- Use the `getline` function with 2 arguments to overcome this obstacle.
  - `getline(inFileStream, str)`
- First argument is an input stream variable, and second argument is a string variable
- **Example**

```
string    message;
getline(cin,  message);
```

getline() Function

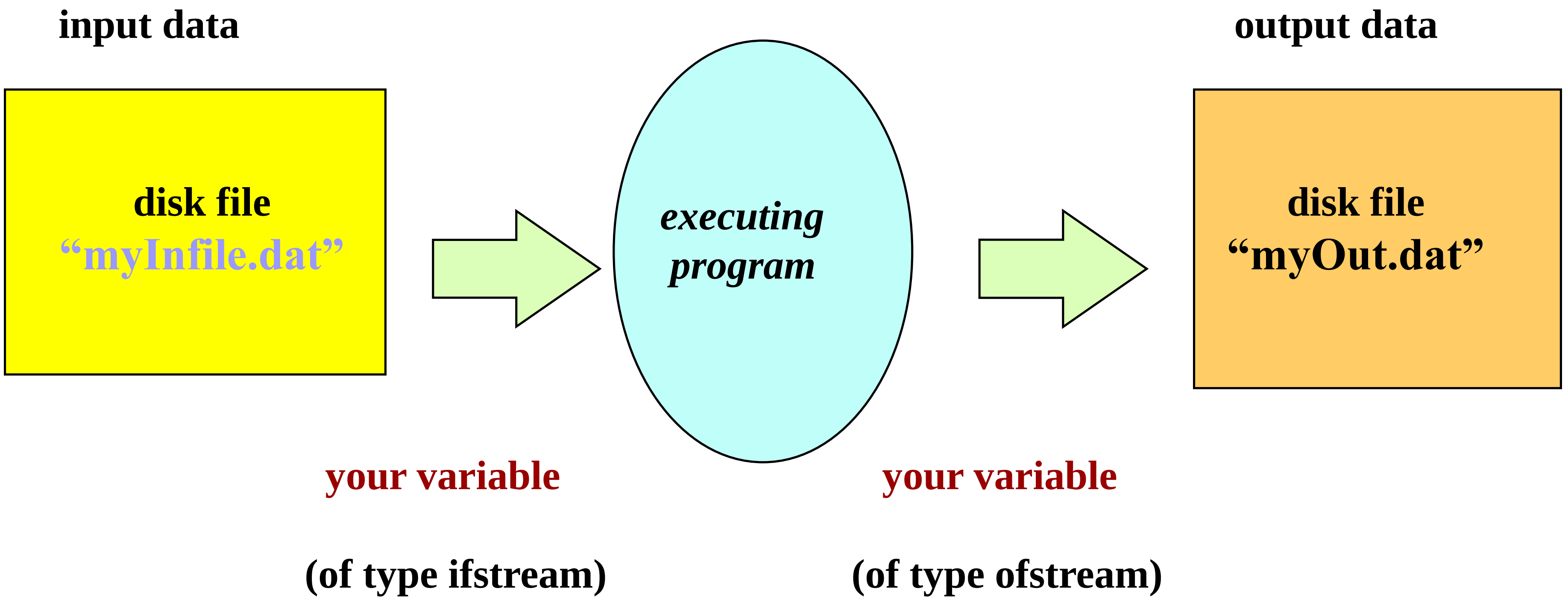
- `getline` reads successive characters (including blanks) into the string, and stops when it reaches the newline character ‘\n’

- The newline is consumed by `getline`, but is not stored into the string variable

- Notice the difference between the newline marker on Windows and Linux.
  - On Windows it is the combination of **carriage return (ASCII 0x0d or \r)** and a newline(\n), also referred to as CR/LF.
  - On Linux it is just a newline (\n)

Disk Files for I/O

#include <fstream>



Disk I/O

To use disk I/O

- **Access #include <fstream>**
- **Choose valid identifiers for your filestreams and declare them**
- **Open the files and associate them with disk names**
- **Use your filestream identifiers in your I/O statements (using >> and << , manipulators, get, ignore)**
- **Close the files**

Disk I/O Statements

```
#include <fstream>

ifstream   myInfile;          // Declarations
ofstream  myOutfile;

myInfile.open(“myIn.dat”); // Open files
myOutfile.open(“myOut.dat”);

myInfile.close();             // Close files
myOutfile.close();
```

Opening a File

Opening a file

- **Associates the C++ identifier for your file with the physical(disk) name for the file**
  - **If the input file does not exist on disk, open is not successful**
  - **If the output file does not exist on disk, a new file with that name is created**
  - **If the output file already exists, it is erased**
- **Places a file reading marker at the very beginning of the file, pointing to the first character in the file**

Stream Fail State

- **When a stream enters the fail state,**
  - **Further I/O operations using that stream have no effect at all.**
  - **The computer does not automatically halt the program or give any error message.**

- **Possible reasons for entering fail state include**
  - **Invalid input data (often the wrong type).**
  - **Opening an input file that doesn’t exist**
  - **“End of file” is reached**
  - **Opening an output file on a disk that is already full or is write-protected.**

Stream Fail State

- You need to check for errors!
- **Stream methods for checking errors**
  - `good()` is true if there are no errors.
  - `eof()` is true if the end of file was reached.
  - `fail()` is true if there was a logical error or a read/write error on the stream.
  - `bad()` is true if there is a read/write error on the stream.

Run Time File Name Entry

```
#include <string>
// Contains conversion function c_str

ifstream  inFile;
string    fileName;

cout << "Enter input file name: " << endl; // Prompt
cin    >>  fileName;

// Convert string fileName to a C string type
inFile.open(fileName.c_str());
```

Example 1: Reading from a File

```
#include <iostream>
#include <fstream>
#include <cstdlib>  // exit prototype
#include <iomanip>
using namespace std;
int main(){
  ifstream inClientFile( "clients.dat", ios::in );
  // exit program if unable to create file
  if ( !inClientFile ) {  // overloaded ! Operator
     cerr << "File could not be opened" << endl;
     exit( 1 );
   }
   cout << left << setw( 10 ) << "Account" << setw( 13 )
        << "Name" << "Balance" << endl << fixed << showpoint;
   int account;
   char name[ 30 ];
   double balance;
```

Example 1

```
   // display each record in file
   while ( inClientFile >> account >> name >> balance ){
        cout << left << setw( 10 ) << account << setw( 13 )
        << name << setw( 8 ) << setprecision( 2 ) << right
        << balance << endl;
    } // end while

    return 0;  // ifstream destructor closes file
} // end main
```

Example 2: Writing to a File

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

int main() {
  ofstream outClientFile( "clients.dat", ios::out );

   // exit program if unable to create file
   if ( !outClientFile ) {  // overloaded ! operator
      cerr << "File could not be opened" << endl;
      exit( 1 );
   }
```

Example 2

```
   cout << "Enter the account, name, and balance." << endl
        << "Enter end-of-file to end input.\n? ";
   int account;
   char name[30];
   double balance;

   while ( cin >> account >> name >> balance ) {
     outClientFile << account << ' ' << name
        << ' ' << balance << endl;
     cout << "? ";
     } // end while
   return 0;  // ofstream destructor closes file
 } // end main
```

String Stream Processing

- Allows you to use streams that read and write from strings instead of reading and writing from files. These are called "string streams"
- I/O of strings to and from memory
  - Called in-memory I/O or string stream processing
  - Classes
    - **`istringstream`** (input from string)
    - **`ostringstream`** (output to a string)
    - **`<sstream>`** and **`<iostream>`** headers
  - Use string formatting to save data to memory

String Stream Processing

- String output
  - **`Ostringstream outputString;`**
  - **`outputString << s1 << s2;`**
  - Member function **`str`**
    - Returns **`string`** that was output to memory
    - **`outputString.str()`**

- String input
  - **`istringstream inputString ( myString );`**
  - **`inputString >> string1 >> string2`**
  - Like reading from **`cin`**

String Stream Processing

- **Example**

```
#include <sstream>

istringstream mystring("help 10 times");
string w1, w2;
int ival;

mystring >> w1; // w1 will be "help"
mystring >> ival;// ival will be 10
mystring >> w2; // w2 will be "times"
```

String Stream Processing

- **Example**

```
#include <sstream>
istringstream mystring(“read    a   word”);
string w1;
while( mystring >> w1 ) cout << w1 << endl;
```

// NOTE: the spaces will be skipped

- **Example**

```
#include <sstream>
ostringstream os;
os << “There are “;
os << 101 << “ boxes”;
string combo = os.str();
```