CSE 143

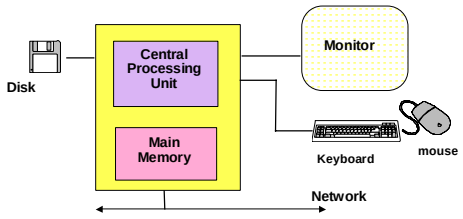
Stream I/O

Appendix A

Input/Output Concepts

- Concepts should be review!
  - New syntax, but same fundamental concepts
- input vs. output, read vs. write
- conversion between characters in a stream and C/C++ data values (types) in a program
- File concepts
  - what is a file?
  - file name vs. file variable
  - open, close
  - end-of-file

What's a Computer?

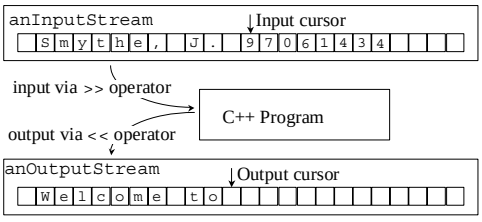


Stream I/O

- The basic C++ I/O library is built around the concept of streams.
  - both for keyboard/monitor and for files
- Old C-style **`printf`**, **`scanf`**, etc. library is still available, **but**….
  - Mixing the two is bad news
  - You **must** use only stream I/O in CSE143

What is a Stream?

A stream is just a sequence of *characters*, nothing else:



Only characters?!

- Wait a minute... if the stream is only characters, how can we read or write integers, or doubles, or strings?
- Answer: the library functions *convert* other types to and from characters.

Example: the stream contains

```
45
```

That is two characters, not a number!

`cin >> i;` converts the two characters into an integer and stores it in the integer variable i.

Well-Known Streams

- Global streams defined in **`iostream.h`** :
  - **`cin`**: standard *input* stream (usually keyboard)
  - **`cout`**: standard *output* stream (usually screen)
  - **`cerr`**: standard *error* stream (also usually directed to the screen)
- Programs can open other streams to/from files and other devices.

<< Review

For output streams, << is the “put to” or "insertion" operator

```
#include <iostream.h>
…
int count = 23;
cout << “Hello, World!” << ‘\n’;
          // endl: same as ‘\n’, but flushes output
cout << “The count is ” << count << endl;
```


>> Review

For input streams, >> is the “get from” or "extraction" operator

```
#include <iostream.h>
…
int x, ID;
char Name[40];
cin >> x;
cin >> Name >> ID;
// Can read multiple items on one line
// Note: no &’s as with scanf
```

- **`<< and >> are aware of the types of the data`**

Two Rules for Input

Rule 1 is about “when does the scan start?”

Rule1 for int, double, char, string:

leading whitespace is skipped

Rule 2 is about “when does the scan stop?”

Rule 2 for int, double: scan stops on 1st char that can’t be part of the value for that type

Rule 2 for char: one character only is read

Rule 2 for strings (char arrays): scan stops on 1st whitespace

Example

```
int i; char ch; char buffer[BUF_SIZE];
cin >> ch >> buffer >> i;
```

User types

_hello\t\n15w

_	h	e	l	l	o	\t	\n	1	5	w

Built-in vs other types

- cin and cout understand the basic C++ types, including strings
- They do not understand other arrays or user-defined types (structs, classes, enums, etc)
- But... it is possible to “overload” << and >> to understand your classes!
- Eventually you will be able to write

  cout << myFavoriteBook
- and have it do something reasonable

What Are cin and cout?

- Actually, cin and cout are objects
  - They are global variables whose types are the classes istream and ostream.
- As classes, they have methods that can be called
  - Use regular **.** notation
  - Some methods can be used to check the status of the stream, or change the status.
  - Other methods are used for I/O other than what << and >> provide

Stream States

- Several methods are available to check or set state.

```
cin.eof();   // true if cin eof reached
cin.clear(); // set state to "good"
```

- The stream itself can be used in an expression to check its state

```
if (!cin)
  cerr << "error or eof on cin" << endl;
```


End-Of-File State

- Means there is no more input in the stream
- *eof is a state; it's not a special value in the stream*
- eof is most often used with files
- eof with keyboard input?
  - User signals by typing a special key combination
  - CNTL-Z, CNTL-D, etc. depends on operating system
  - The special key is NOT sent to the program. The eof status is what is detected.

Input Errors

- Stream input "fails" if the next thing in the input has the wrong format or if there is no more data (end of file).
- If an input operation fails, the variable involved is not changed.

```
if (cin >> k)
  cout << "new value for k read ok";
else
  cout << "input failed; "
       << "k not changed";
```


Input Errors (cont)

- Once a stream input operation has failed, any further operations will also fail until the stream state is cleared.

```
// suppose next input is "xyz"
cin >> k;  // fails (why?); k unchanged
cin >> j;  // cin state not good, so
           // nothing happens
cin.clear();  // cin can be used for
              // input again
```


Example: Copy Integers

- This program copies integers from cin to cout until an input operation fails. Each integer is written on a separate output line.

```
#include <iostream.h>
int main() {
  int j;
  while (cin >> j)
     cout << j << '\n';

  return 0;
}
```

Reading a Whole Line

- Reading
  Seattle Rain
- vs
  Seattle-Rain
- cin >> stringvar won't do the former -- why?
- Need an additional function: getline
  cin.getline (stringvar, len);
- Dot notation! What's happening here??
  - Answer: Remember, cin and cout are really objects

Unformatted Stream I/O

- >> and << provide formatted I/O.
  - There are methods which provide unformatted (character-level) I/O.
- Examples:

```
char ch; char s[100];
cin.get(ch);  // read 1 character into ch
cin.getline(s,n); // read next line into s
cout.put(ch); // write 1 character ch
```

- Variations available to limit how many characters are read, specify end-of-line characters, etc.

Next Step: Files

- Review: File is a named collection of data on disk
- Basic idea of using files in C++: Attach a file to a stream!
  - Then the characters of that file become the characters of the stream.
- Use class *ifstream* for input text files, *ofstream* for output text files.
- You can attach (open) the file by giving its name to the constructor:
  - ifstream myfile ("c:\\testdata.txt"); // why "\\" here?

What is a file?

- A collection of data stored on a disk
- Text file
  - A sequence of characters
- Binary file
  - stores data in an efficient, non-human-readable, form
- "File name": a way of naming a file
  - OS rules such as DOS: 8 chars . 3 chars

a	_	f	i	l	e	\n	e	n	d	EOF

File Operations (Abstract)

- "open"
  - Creating a variable to represent the file
  - Allows you to access the file's contents
- "read"
  - getting data from the file, similar to `cin >> var;`
- "write"
  - storing data to a file, similar to `cout << var;`
- "close"
  - Tells the OS you're finished with a file
  - Can't do any more reading/writing
  - Might lose data if you forget to close!

More Stream Classes

- cin and cout are defined in <iostream.h>.
- Library <fstream.h> contains similar classes for file I/O
- Input stream classes:
  - istream: console input (cin)
  - ifstream: file input
- Output stream classes
  - ostream: console output (cout, cerr)
  - ofstream: file output

Streams as C++ Classes

- Streams are C++ classes
- Streams have lots of built-in methods
- We use the "." syntax to access member functions, as usual.

```
inFile.get(ch);            // get a character
outFile.put(ch);     // put a character
outFile.getline(str, len); //get a whole line
outFile.close();     // close the stream
inFile.eof();        // end of File??
```


Testing the Stream

- The stream can be tested as if it were a boolean
  if (mystream)...
- Two typical occasions for testing:

1. Right after opening, to see if the open worked
   ifstream dfile ("c:data");
   if (dfile) cout << "OK"; else cout << "bad";
2. While processing, to see if end of file
   while (dfile) //is the stream still good?
   { keep reading data}

File Stream Example

```
#include <iostream.h>
#include <fstream.h>

void main() {
     // open file--ios:: flags needed in MSVC++
  ifstream inFile("input.txt",
                  ios::nocreate | ios::in);
  ofstream outFile("output.txt");  // open output
  char ch;

  // should test for successful opening here..

  while (inFile.get(ch)) { // while more input
      outFile.put(ch);     // write it to output
  }
  inFile.close();          // close the files...
  outFile.close();
}
```


Another File Example

```
#include <fstream.h>

  // multiply every int in a file by a factor
void multiplyFile(char in[], char out[],
                  int factor) {
     // open file--ios:: flags needed in MSVC++
  ifstream inFile(in, ios::nocreate | ios::in);
  ofstream outFile(out);  // open output
  int i;

  // should test for successful opening here..

  while (inFile >> i)) {    // while more input
     outFile << i * factor << ' ';
  }
  inFile.close();          // close the files...
  outFile.close();
}
```


Stream Class Relationships

- Every ifstream (file) is also a istream.
  - An ifstream is an "enhanced" istream that has extra capabilities to work with disk files
    `An ifstream object can be used wherever an istream object is needed (function parameter, for example)`
  - But the reverse is not true. An istream is not also an ifstream.
    `So if an ifstream is explicitly called for, cin can't be used`
- A similar relationship holds between ofstreams and ostreams.
- This is an example of "inheritance"
  - An important object-oriented concept we will study later

Notes and Advice

- File and stream processing can get VERY baroque
  - Many details, gotchas, exceptions, etc. in the C++
  - File formats are often complex
- Learn the basics
- Try to keep it simple (not always possible)
- You can't memorize it all
- Buy a good C++ book and keep it handy when programming!
  - Bookstore has lots to choose from. Browse and buy one you like