CSE 143

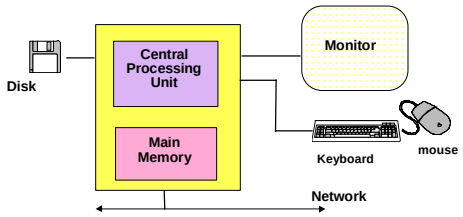
Basic Stream I/O

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?

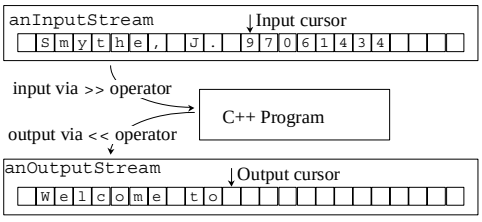


Stream I/O

- 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> :
  - **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 or from files and other devices.

<< Review

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

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

How Stream Input Works

Rule: With simple types: leading whitespace is skipped

```
int ID;
char Name[40];
char ch;

cin >> ID;   // interprets as integer
cin >> ch;   // reads a char
cin >> Name; // interprets as character string,
             // stopping at trailing whitespace
```


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

Stream States

- All streams are actually objects (instances of stream classes)
  - In particular, cout and cin are such objects
- All streams have a “state”.
- Once opened, the stream object should be in a "good" state
  - Remains thus until the end of file, or until any error occurs

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 Error State

- Stream input "fails" if the next thing in the input has the wrong format or there is no more data
  - Example: try to read an integer, but a letter is encountered instead
  - Example: trying to read something, but already at end of file

Testing the State

- The state can be very simply tested by treating the stream as a boolean(!)
  - This is a special property of streams, and doesn't work for most objects

```
cin >> k;
if (cin)
     cout << "new value for k read ok";
else
     cout << "input failed, or at EOF; "
```


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>
using namespace std;
int main() {
  int j;
  while (cin >> j)
     cout << j << '\n';

  return 0;
}
```


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 (type) *ifstream* for input text files, *ofstream* for output text files.

Stream Classes

- cin and cout are defined in <iostream>.
- Library <fstream> 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

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!

Opening a File

- The simplest way to open a file is to give the (disk) file name as a parameter when the file stream variable is created:

  ifstream infile ("testdata.txt");
- This does two things
  - Declares a variable named infile of type ifstream
  - Opens it so it accesses the file named testdata.txt in the current directory.

Opening & Closing Files

- The parameter giving the file name may be an array of characters containing a C null-terminated string (not, unfortunately a C++ string)

```
char filename[256];
cout << "enter file name: ";
cin >> filename;
ifstream infile (filename);
```

- Files are automatically closed when exiting the function that contains the file variable declaration

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 ("data.txt");

   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 Copy Example (1)

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

int main() {
  ifstream inFile("input.txt");     // open input
  ofstream outFile("output.txt");  // open output

  // quit if files not opened
  if (!inFile || !outFile) {
    cout << "file open failed" << endl;
    return 1;
  }
```

File Copy Example (2)

```
  string word;

  // copy words to output file, one word per line
  while (inFile >> word) {
      outfile << word << endl;
  }

  // files closed automatically when main exits
  return 0;
}
```