C++ Stream I/O

Streams

• A stream is a sequence of bytes
  - Input: flow from a device to memory
  - Output: flow from memory to a device

Stream I/O Classes and Objects

• cin: an instance of istream tied to stdin
• cout: an instance of ostream tied to stdout
• cerr, clog: instances of ostream tied to std error. Outputs to cerr are unbuffered, to clog is buffered
Stream Output

- Stream-insertion operator `<<` is overloaded to output data items of built-in types, strings, and pointer values.

```cpp
cout << "hello\n";
cout << "hello" << endl;
```

- The stream manipulator `endl` flushes the output buffer, while `\n` does not.

```cpp
cout << "abc" << (20+30) << endl;
```

- Cascading is allowed because `<<` returns a reference to `cout`.

```cpp
cout.put('A'); cout.put(65); // both output 'A'
```

- The `put` member function outputs one character.

Stream Input

- The stream extraction operator `>>` returns 0 if end-of-file is encountered; otherwise returns a reference to the object that received the message (e.g., `cin`).

- Cascading is allowed.
- end-of-file: `<ctrl>-d` (UNIX, Mac), `<ctrl>-z` (PC)
istream Member Functions

- `get()`, `getline()`
- `ignore()`: skips over a designed number of characters or delimiter
- `putback()`: places the previous character obtained by `get()` back onto that stream
- `peek()`: returns next character from an input, but doesn’t remove it from the stream

Unformatted I/O

- Inputs or outputs some number of bytes to or from a character array in memory

```cpp
void main()
{
    char buffer[80];
    cout << "Enter: \n";
    cin.read(buffer, 20);
    cout << "Entered was: \n";
    cout.write(buffer,
               cin.gcount());
    cout << endl;
}
```

```
Enter:
Using the read, write, gcount
Entered was:
Using the read, write
```

Integral Stream Base

- dec(base 10), oct(base 8), hex(base 16), `setbase()`(parameter 10, 8, or 16)

```cpp
#include <iomanip>

void main()
{
    int n;
    cout << "Enter a num: \n";
    cin >> n;
    cout << hex << n << " ";
    cout << oct << n << " ";
    cout << dec << n << " ";
    cout << setbase(10) << n;
    cout << endl;
}
```

```
Enter a num:
20
14 24 20 20
```
Floating-Point Precision

- Use member function `precision()`, or stream manipulator `setprecision`

```cpp
#include <iomanip>
void main()
{
    double root2 = sqrt(2.0);
    for (int i = 1; i < 5; i++)
        cout << setprecision(i) << root2 << " ";
    cout << endl;
    for (int i = 5; i < 7; i++)
        cout << root2 << " ";
}
```

User-Defined Manipulators

- User may create their own stream manipulators

```cpp
#include <iostream>
ostream& bell(ostream& output){return output<<'\a';}
ostream& tab(ostream& output){return output<<'\t';}
ostream& endLine(ostream& output){return output<<'\n'<<flush;}
void main()
{
    cout << 'a'<<tab<<'b'<<bell<<endLine;
}
```

Stream Format States

- Trailing zeros

```cpp```
```
cout<<9.00; //output 9
cout.setf(ios::showpoint);
cout<<9.00; //output 9.00
```

- Justification

```cpp```
```
int x = 12345;
cout<<setw(10)<<x; //output |
12345 |
cout.setf(ios::left, ios::adjustfield);
cout<<setw(10)<<x; //output | 12345 |
cout<<setw(10)<<setfill(' ')<<x; //output | 12345 |
Stream Format States

- Integral Stream Base

```cpp
int x = 100;
cout << setiosflags(ios::showbase) << x // output 100
    << oct << x // output 0144
    << hex << x; // output 0x64
```

- Floating-Point Numbers

```cpp
double x = 0.001234567, y = 1.946e9;
cout << x << y; // output 0.001234567 1.946e+009

cout.setf(ios::scientific, ios::floatfield);
cout << x << y; // output 1.234567e-003 1.946000e+009

cout.unsetf(ios::scientific);
cout << x << y; // output .001234567 1.946e+009

cout.setf(ios::fixed, ios::floatfield);
cout << x << y; // output 0.001235 1946000000.000000
```

Stream Error States

- Floating-Point Numbers

```cpp
int x;
cout << "Before a bad input operation"
    << cin.rdstate() // output 0
    << cin.eof() // output 0
    << cin.fail() // output 0
    << cin.bad() // output 0
    << cin.good(); // output 1

cin >> x; // input a character 'A'
cout << "After a bad input operation"
    << cin.rdstate() // output 2
    << cin.eof() // output 0
    << cin.fail() // output 0
    << cin.bad() // output 0
    << cin.good(); // output 0
```