CS2141 – Software Development using C/C++

Stream I/O

iostream

- Two libraries can be used for input and output: **`stdio`** and **`iostream`**
- The **`iostream`** library is newer and better:
  - It is object oriented
  - It can make use of C++ features such as references, function overloading, and operator overloading
  - It is type safe
  - It is extensible - just overload the **<<** and **>>** operators for your class

Output Streams

- An **`ostream`** converts values into sequences of characters which are then output somewhere
- Usually the **`<<`** operator is used to send values to an ostream, but there are also other functions:
  - put – Sends a single character
    ```
    cout.put( 'c' );
    ```
  - write – Sends a string of characters
    ```
    cout.write( buf, size );
    ```
  - flush – Makes sure characters are actually written to wherever the output is going
    ```
    cout.flush( );
    ```

Output Streams cont.

- The predefined **`ostream`** object **`cout`** sends text to the standard output device (**`stdout`**)
- The **`ostream`** object **`cerr`**, which is unbuffered, sends text to the standard error device (**`stderr`**)

Input Streams

- An **`istream`** reads sequences of characters from somewhere and converts them to values
- The predefined **`istream`** object **`cin`** reads from the standard input device (**`stdin`**)
- The **`>>`** operator is usually used to read values from an **`istream`**. By default it skips over white space before reading a value
- The **`eof( )`** function is used to detect the end of the input file

Input Streams cont.

- Other functions for reading:
  - getline – Read an entire line of characters
    ```
    cin.getline( buffer, size );
    ```
  - read – Read a string of characters
    ```
    cin.read( buffer, size );
    ```
  - get – Read a single character or a string
    ```
    char c = cin.get( );
    cin.get( c );
    cin.get( buffer, size );
    cin.get( buffer, size, delimiter );
    ```

Stream Errors

- There are several functions for detecting errors in an input or output stream:
  - fail – True if an error occured.

```
if( cin.fail( ) ) ...
```

  - bad – True if a fatal error occured.

```
if( cin.bad( ) ) ...
```

  - good – True if no errors have occured.

```
while( cin.good( ) ) ...
```

Stream File I/O

- File stream stuff is in the fstream header:

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

- The header defines several file i/o classes:
  - ofstream – Write out to a file
  - ifstream – Read in from a file
  - fstream – Generic reading/writing with files

Stream File I/O cont.

- There are two ways to open a file:
  - Pass the file name to the constructor:
    ```
    fstream in( "mydata" );
    ```
  - Use the open function:
    ```
    ofstream out;
    out.open( "names" );
    ```
- When opening a file, a second argument can also be used to specify the open mode:
  ```
  // Open file for appending
  ofstream fout( "data", ios::app );
  ```
  Other modes include **`ios::trunc`** and **`ios::binary`**

Stream File I/O cont.

- There are also two ways to check if a file was opened successfully:
  - The overloaded boolean conversion operator:
    ```
    if( !in ) { /* handle error */ }
    ```
  - The fail function:
    ```
    if( out.fail( ) ) ...
    ```
- Once opened, ofstreams can be used just like cout, and ifstreams just like cin
- Use the close function to close a file:
  ```
  in.close( );
  out.close( );
  ```

Example (ofstream)

- An **ofstream** is used to write to a file:

```
#include <fstream>
using namespace std;
int main( ) {
   char * filename = “outfile”;
   ofstream out;
   out.open( filename );
   if( !out )
      exit( 1 );
   out << “Cows go moo” << endl;
   out.close( );
   return 0;
}
```

Example (ifstream)

- An **ifstream** is used to read from a file:

```
#include <fstream>
using namespace std;
int main( ) {
   ifstream in( “infile” );
   string s;
   int n;
   in >> s;  // Read a word from in
   in >> n;  // Read an integer from in
   in.close( );
   return 0;
}
```

Formatted Stream Output

- Output streams have several member functions that are used for formatting
- The most commonly used ones are:
  - `precision( int n )`
  - `width( int n )`
  - `setf( ios::flag )`
- *Manipulators* can also be used for formatting

precision

- Depending on the compiler, **`precision`** will set the number of significant digits, or the the number of digits after the decimal point
  - With g++ it is significant digits
  - Output is converted to exponential notation if needed

Code:

```
float a = 3.084, b = 26.818;
cout.precision( 2 );
cout << “a=” << a << “ b=” << b << endl;
```

Output:

```
“a=3.1 b=26”
```

width

- **`width`** sets the minimum width of the column
  - If more space is needed to print a value, the entire value is printed
  - Calling **`width`** affects only the next item that is sent to the output stream
  - Code:

```
int count = 53;
cout.width( 10 );
cout << "Count=" << count << endl;
```

  - Output (using _ as space):

```
____Count=53
```

setf

- **`setf`** is short for set flag
- **`unsetf`** unsets a flag
- Some useful ios flags:
  - **`ios::fixed`** and **`ios::scientific`**
    - If **`fixed`** is set, floating point values are not converted to exponential notation
    - If **`scientific`** is set, floating point values are always converted to exponential notation
    - Neither of these are set by default

setf cont.

- Useful ios flags continued:
  - **`ios::showpoint`**
    - If set, forces the decimal point and trailing zeros to be shown for floating point values
    - If not set, a float with all zeros after the decimal point will look like an integer
    - By default, showpoint is not set
  - **`ios::showpos`**
    - If set, forces the + sign to be printed with positive values
    - It is not set by default

setf cont.

- Useful ios flags continued:
  - **`ios::right`**
    - When used with width, forces the value to be right justified within the column
    - Automatically unsets left
    - Set by default
  - **`ios::left`**
    - When used with width, forces the value to be left justified
    - Automatically unsets right
    - Default is not set

Manipulators

- Manipulators are functions that are called in an unusual way:

```
#include <iomanip>
...
cout << setw( 10 ) << setprecision( 5 ) <<
3.14159265979 << endl;
```

- The **`setw( n )`**manipulator works the same as **`ostream.width( n )`**
- Calling **`setprecision( n )`** is the same as calling **`ostream.precision( n )`**

Manipulators cont.

- The familiar manipulator **`endl`** is just a function:

```
ostream & endl( ostream & out ) {
   out << '\n';     // write an end-of-line
   out.fflush( );   // flush the buffer
   return out;      // return the buffer
}
```

- A second function is used to call it:

```
ostream & operator<<( ostream & out,
ostream & (*fun)(ostream &) ) {
   return fun( out );  // execute the function
}
```