# C++ IO

For : COP 3330.
Object oriented Programming (Using C++)
`http://www.compgeom.com/~piyush/teach/3330`

*Piyush Kumar*

## C++ IO

- All I/O is in essence, done one character at a time
- Concept: I/O operations act on *streams* (sequences) of ASCII characters

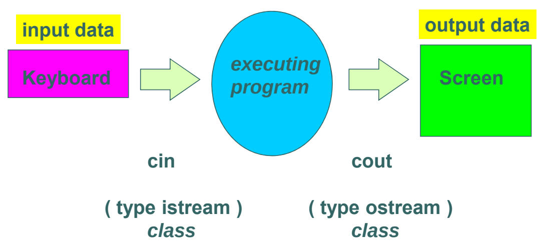
## C++ IO

| | |
|---|---|
| cout | standard output stream<br>sequence of characters printed to the monitor |
| cin | standard input stream<br>sequence of characters input from the keyboard |

- both cout and cin are data objects and are defined as classes

## Interactive I/O

#include <iostream>



## Example

Namespaces: They provide a way to avoid name collision. Be careful about using this.

```
#include <iostream>

using namespace std;

int main(void){

        cout  << "Hello World" ;
        cout  << endl;
        return 0;

}
```

Standard IO library for C++. Defines two fundamental types, istream and ostream.

Stream: A flow of characters (1 or 2 bytes long). Can flow in and out of Files, strings, etc.

## Example

Ostream object named cout.

```
#include <iostream>

using namespace std;

int main(void){

        cout  << "Hello World" ;
        cout  << endl;
        return 0;

}
```

Equivalent to:
  operator<< (cout, "Hello World");
Its calling a *friend* function of ostream with input data.

Uses function declaration (approx):
ostream& operator<<( ostream&, const char * )

## Example

invokes a manipulator function called endl. endl looks something like this:

```
ostream& endl( ostream& os)
{
        os << '\n';
        os.flush();
        return os;
}
```

```
#include <iostream>

using namespace std;

int main(void){

        cout  << endl;
        return 0;

}
```

Equivalent Compiler statement:

```
std::cout.operator<<(
     std::endl(std::cout)
);
```

Scope Operator for namespaces.

## Special Output Characters

| | |
|---|---|
| \n | new line |
| \t | tab |
| \b | backspace |
| \r | carriage return |
| \' | single quote |
| \" | double quote |
| \\ | backslash |

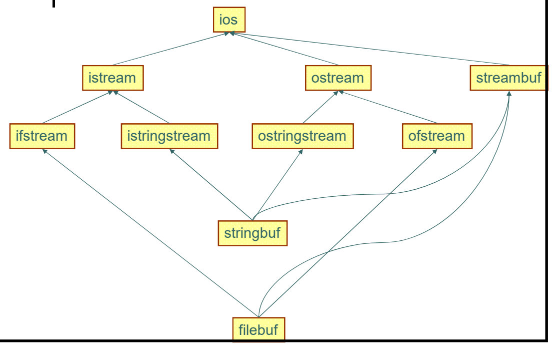
## Stream IO headers

- iostream -- contains basic information required for all stream I/O operations
- iomanip – used for performing formatted I/O with stream manipulators
- fstream – used for performing file I/O operations
- stringtream -- used for performing in-memory I/O operations (i.e., into or from strings in memory)

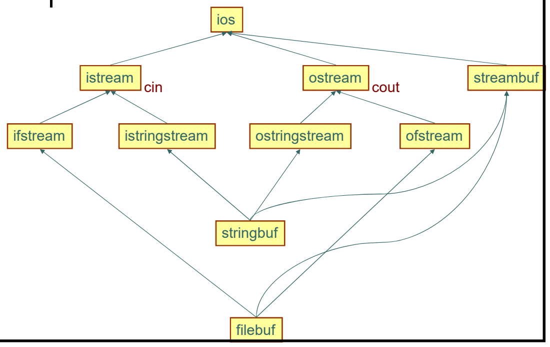
## A Stream

- A flow of characters.
- Buffers: IO to streams goes thru a buffer. C++ allows you change the default behavior of associated buffers.
- State: Each stream is associated with a state indicating various things like if an error has occurred or not…

## C++ IO Class hierarchy



## C++ IO Class hierarchy



## C++ IO Hierarchy

- The **ios** hierarchy defines the interface of the IO system.
- The **streambuf** hierarchy defines the implementation of the IO system, mostly provides the facilities of *buffering* and *byte-level I/O*

## Other Predefined Streams

- cerr - the standard destination for error messages (often the terminal window). Output through this stream is unit-buffered, which means that characters are flushed after each block of characters is sent.
- clog - like cerr, but its output is buffered.

## Formatting with predefined streams.

- Remember: Due to inheritance, anything you learn about formatting IO with predefined streams (cin, cout, clog, cerr) also applies to file IO and string IO.
- Anything available or defined in the ios class is available everywhere in the IO subsystem.

## Stream IO

Inside ios

- **<<** (left-shift operator)

  Overloaded as *stream insertion operator*
- **>>** (right-shift operator)

  Overloaded as *stream extraction operator*
- Both operators used with **`cin`**, **`cout`**, **`cerr`**, **`clog`**, and with user-defined stream objects

## Example

- cin >> Variable;
- cout << Variable;
- clog << Variable;
  - Buffered
- cerr << Variable;
  - Unbuffered, prints Variable immediately.
- Note: Variable types are available to the compiler.

## << operator

- << is overloaded to work on built-in types of C++.
- Can also be used to output user-defined types.
- Other interesting examples:
  - cout << '\n'; // newline.
  - cout << "1+2=" << (1+2) << endl;
  - cout << endl; // newline.
  - cout << flush; // flush the buffer.

## << operator

- Associates from left to right, and returns a reference to its left-operand object (i.e. **`cout`**). This enables cascading.
- Outputs "char *" type as a string.
- If you want to print the address, typecast it to (void *).
- Example:
  - char name[] = "cop3330";
  - cout << name << static_cast<void *>( name ) << endl;
  - static_cast<void *>( name ) equivalent to ((void *) name) in C except that it happens at compile time.

## Stream insertion: One char.

- **`put`** member function
  - Outputs one character to specified stream
    **`cout.put('C');`**
  - Returns a reference to the object that called it, so may be cascaded
    **`cout.put( 'C' ).put( '\n' );`**
  - May be called with an ASCII-valued expression
    **`cout.put( 65 );`**
    - Outputs **A**

## Input Stream

- >> (stream-extraction)
  - Used to perform stream input
  - Normally ignores whitespaces (spaces, tabs, newlines)
  - Returns zero (**`false`**) when **`EOF`** is encountered, otherwise returns reference to the object from which it was invoked (i.e. **`cin`**)
    - This enables cascaded input
    **`cin >> x >> y;`**
- >> controls the state bits of the stream
  - **`failbit`** set if wrong type of data input
  - **`badbit`** set if the operation fails

## Input Stream : Looping

```
while (cin >> fname)
```

">>" returns **`0`** (**`false`**) when **`EOF`** encountered and loop terminates.

## Example Program

```
#include <iostream>

using std::cout;
using std::cin;
using std::endl;

int main(void) {
       int height = 0, maxheight = 0;

       cout << "Enter the heights: (enter end of file to end): ";
       while(cin >> height)
         if( height > maxheight)
           maxheight = height;

       cout << "Tallest person's height = "
            << maxheight << endl;
       return 0;
}
```

## Output

```
$ ./a.exe
Enter the heights: (enter end of file to end): 72
89
54
33
68
66
Tallest person's height = 89
```

## istream member function: get

- **`char ch = cin.get();`**
  - Inputs a character from stream (even white spaces) and returns it.
- **`cin.get( c );`**
  - Inputs a character from stream and stores it in **c**

## istream member function: get
## get (array_name, max_size) ;

char **fname[256]**
cin.get (**fname, 256**);

- Read in up to 255 characters and inserts a null at the end of the string “fname". If a delimiter is found, the read terminates. The array acts like a buffer. The delimiter is not stored in the array, but is left in the stream.

## istream member function:
## getline (array_name, max_size)

char **fname[256]**
cin.getline (**fname, 256**);

- Same as get, except that getline discards the delimiter from the stream.

## istream member functions: ignore()

- cin.ignore ( ) ;
  - Discards one character from the input stream.
- cin.ignore (10) ;
  - Discards 10 characters.
- cin.ignore(256,’\n’);
  - Discards 256 characters or newline, whichever comes first.

## istream member functions: peek(), putback()

- char ch = cin.peek ( ) ;
  - Peeks into the stream’s next character.
- cin.putback (‘A’) ;
  - Puts ‘A’ back in the stream.

## FILE IO Example.

Copy File “first.txt” into “second.txt”.

```
#include <iostream>
#include <fstream>

using namespace std;

int main(void)
{
   ifstream source("first.txt");
   ofstream destin("second.txt");

   char ch;
   while (source.get(ch))
      destin<<ch;
        return 0;
}
```

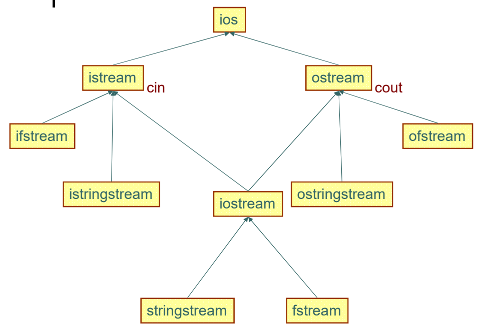
## Slightly modified

```
#include <iostream>
#include <fstream>

using namespace std;

int main(void)
{
  ifstream source("first.txt");
  ofstream destin("second.txt");

  char ch;
  while (source.peek() != EOF){
      source.get(ch);
      destin.put(ch);
  }
      return 0;
}
```

## More IO functions

- read()
  - cin.read(fname, 255);
    - Reads 255 characters from the input stream. Does not append '\0'.
- cout.write(fname,255);
    - Writes 255 characters.
- gcount: returns the total number of characters read in the last input operation.

## C++ IO Class Hierarchy Revisited



## Another example

```
#include <iostream>
#include <sstream>
#include <string>


using namespace std;

int main() {
        int i;
        string line;
        while(getline(cin,line)){
            stringstream sfstream(line);
            while (sfstream >> i){
                    cout << i << endl;
            }
        }
        return 0;
}
```

What if I replace this with istringstream?

## stringstream operations

- stringstream strm;
- stringstream strm(mystring);
  - Initializes strm with a copy of mystring.
- strm.str();
  - Returns the content of strm in string format.
- strm.str(s);
  - Copies the string s into strm. Returns void.

#include <iomanip>

## Stream Manipulators

## dec, hex, oct, setbase

- oct, hex, dec
  - Cout << hex << 15;
    - Prints 'F'
- cout << setbase(16) << 15;
  - Prints 'F' again.

## Formatting Output - Integers

- int numstdts = 35533;
  cout << "FSU has" << numstdts
  << "students."
  prints
  FSU has35533students.

- default field width == minimum required
  *default*: what happens when explicit formatting is not specified

## Formatting Output - Integers p.2

- we can specify the field width, or number of spaces used to print a value

  cout << "FSU has" << setw(6)
  << numstdts << " students."
  prints

```
FSU has 35533 students.
```

function call

prints in field width 6, right-justified

## Formatting Output - Integers p.3

- cout << "FSU has" << setw(10)
  << numstdts << " students."
  prints

```
FSU has      35533 students.
```

prints in field width 10, right-justified

## Formatting Output - Integers p.4

- cout << left; // flip to left justification

  cout << "FSU has " << setw(10)
  << numstdts << "students."
  prints

```
FSU has 35533     students.
```

prints in field width 10, left-justified

## Using the default - Integers p.5

- Note on field widths: if a field width specified is too small, or is not specified, it is automatically expanded to minimum required

- numstdts = 100;
  cout << "FSU has "
  << numstdts << " students."
  prints

```
FSU has 100 students.
```

and works for any value of numstdts

## General Rule of Thumb

- When you are printing numeric values in sentences or after a verbal label, the default field width usually works well
- When you are printing numeric values lined up in columns in a table, it is usually necessary to call setw to generate well-formatted output (we will see examples of this later in the course)

## Formatting Output - Reals

-
  ```
  float cost = 5.50;
  cout << "Cost is $" << cost
              << "today."
  ```
  prints
  ```
  Cost is $5.5today.
  ```
- default
  - large values printed in scientific notation
  - if number is whole, no decimal point
  - numbers of digits not under your control

## Formatting Output - Reals p.2

- Setting up real formatting

  ```
  // use fixed point notation
  cout  <<  fixed;

  // print a decimal point (with whole numbers)
  cout  <<  showpoint;  ( noshowpoint )
  ```

  *these remain in effect until changed explicity, as does setprecision. setw only changes next value printed.*

## Formatting Output - Reals p.3

-
  ```
  float cost = 5.50;
  cout << "Cost is $" << setw(5)
       << setprecision(2) << cost
       << " today."
  ```
  prints
  ```
  Cost is $ 5.50 today.
  ```
- if no field width is specified, minimum is used, just as for integers

## You can just do this, once:

-
  ```
  cout << fixed << showpoint
                      << setprecision(2);
  ```

  and these settings will remain in effect throughout your program run

## Formatting Output - char

- default field width == 1
  note: setw does have effect on
  char type data too.

  ```
  char ch = 'Q';
  cout << '*' << ch << setw(3) << '*';
  ```

  *prints*
  ```
  *Q  *
  ```

## Formatting Output - Strings

- default field width == number of characters in the string

  can use setw

  ```
  cout  <<  setw(10)  <<  "Hello";
  ```
  prints
  ```
       Hello
  ```

## Useful Output Spacer

- const string BLANK = " ";

  cout  <<  setw(10)  <<  BLANK;
  prints 10 blanks

- *consider this:*
  const char BLANK = ' ';
  cout  <<  setw(10)  <<  BLANK;
  prints 10 blanks!

## Unitbuf manipulator

- If you want to flush every output
  - cout << unitbuf

    ```
    << "first"
    << "second"
    << nounitbuf;
    ```

## Example Quiz

```
#include <iostream>
#include <iomanip>

using namespace std;

int main() {
        const double tenth = 0.1;
        const float one = 1.0;
        const float big = 1234567890.0;
        cout << "A. " << tenth << ", " << one << ", " << big << endl;
        cout << "B. " << fixed << tenth << ", " << one << ", " << big << endl;
        cout << "C. " << scientific << tenth << ", " << one << ", " << big << endl;
        cout << "D. " << fixed << setprecision(3) << tenth << ", " << one << ", " << big << endl;
        cout << "E. " << setprecision(20) << tenth << endl;
        return 0;
}
```

```
A.  0.1, 1, 1.23457e+009
B.  0.100000, 1.000000, 1234567936.000000
C.  1.000000e-001, 1.000000e+000, 1.234568e+009
D.  0.100, 1.000, 1234567936.000
E.  0.10000000000000000555
```

## Manipulators: Rolling your own.

•How to create our own stream manipulators?

**bell**
**ret** (carriage return)
**tab**
**endLine**

An Example.

Copy not allowed on ostreams.

```
#include <iostream>
#include <ostream>

using namespace std;

ostream& myendl( ostream& os) {
      os << "test\n";
      os.flush();
      return os;
}

int main(void) {
      cout   << myendl;
      return 0;
}
```

## Stream Error States

## Error states

- strm::eofbit
  - if (cin.eof() == true) break; // stream end of file.
- strm::failbit
  - if ( cin.fail() == true) break; // stream format error.
- strm::badbit
  - If (cin.bad() == true) break; // data lost!
- Goodbit?
  - cin.good()  = ((!eofbit) && (!failbit) && (!badbit))
  - All eofbit, failbit and badbit should be false.
- cin.clear() // makes cin good.

## Error States Example

```
int ival;

while ( cin >> ival, !cin.eof() ){
        Assert( !cin.bad() , “IO stream corrupted”);
        if (cin.fail()){  //bad input
                cerr << “Bad data, try again.”;
                cin.clear(istream::failbit);   // reset the stream
                continue;
        }
        // ok to process ival now
} //end of while.
```

## Operators for testing.

- **`operator!`**
  - Returns **`true`** if **`badbit`** or **`failbit`** set
- Useful for file processing
  - if ( ! readmefile ) cerr << “Error”;

## Interactive Input

- Write a prompt
  make it friendly and informative

  prompt typically contains prefix
  character to signal point at which to
  enter input
- Read value(s)
  user types data at keyboard

## Interactive Input: Example

```
int num;
char response;                    prefix

cout  <<  "Enter a number -> ";
cin  >>  num;

cout  <<  "Enter Y or N -> ";
cin  >>  response;
```

## Interactive Input: Contents of Output Window

```
Enter a number -> 17<return>

Enter Y or N -> Y<return>
```

*the program will not process the input until the return key is struck*

## Arguments: Count, Vector

```
#include <iostream>

int main(int argc, char** argv) {
    std::cout << “Argument Count: " << argc << std::endl;

    // Print Argument Vector
    for (int i = 0; i < argc; ++i) {
        std::cout << argv[i] << std::endl;
    }
}
```

## Another C++ Program (Hello argv[1])

```
#include <iostream>
#include <stdlib.h>
using namespace std;

int main(int argc, char *argv[])  {
  if (argc != 2) {
       cout << "Usage: hi.exe <name>" << endl;
       exit (1);
  }

  cout << "Hello " << argv[1] << endl;
  return 0;
}
```

## Control structures

- Statements you should already know :
  - While
  - For
  - If

Recommended Assignments: 1.17, 1.25