File IO/Command Line (+streams, pointers, *this* pointer, argc and argv review)

Streams (PT 1):

What's actually happening:

- a sequence of bytes (containing information) "flowing" into or out of our program (like a stream)
    - For example, when we are typing in words, a sequence of bytes is "flowing" into our program
    - From the keyboard to your program
    - From your program to the screen

- I will show what *stringstream* does in the class code below

How C++ handles it:

- C++ puts all of this input/output work into classes
- C++ has the following standard streams:
    - cin
        - standard input stream (defaulted to the keyboard)
    - cout
        - standard output stream (defaulted to the console)

    - cerr *We will talk more about this later in the semester*
        - standard error stream (defaulted to the console)
        - unbuffered (meaning each character is flushed as you write it)
    - clog *We will talk more about this later in the semester*
        - standard log stream (defaulted to the console)
        - buffered

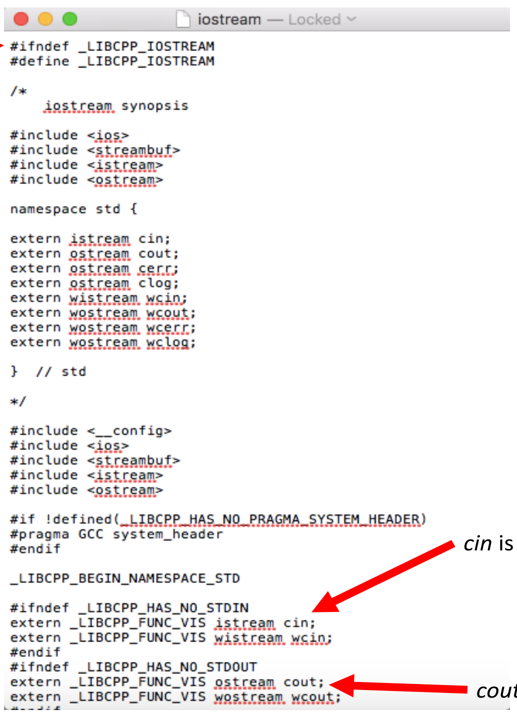
We can also have streams between a program and file (we will see this today).

We've been using *cin* and *cout* along with << (insertion operator) and >> (extraction operator). But what are they exactly?

cin and *cout* are both objects (meaning they come from classes) declared in the iostream header that we include at the top of our programs. *cin* is an object from the istream class and *cout* is an object from ostream class:

Header guards

iostream header



```
#ifndef _LIBCPP_IOSTREAM
#define _LIBCPP_IOSTREAM

/*
    iostream synopsis

#include <ios>
#include <streambuf>
#include <istream>
#include <ostream>

namespace std {

extern istream cin;
extern ostream cout;
extern ostream cerr;
extern ostream clog;
extern wistream wcin;
extern wostream wcout;
extern wostream wcerr;
extern wostream wclog;

}  // std

*/

#include <__config>
#include <ios>
#include <streambuf>
#include <istream>
#include <ostream>

#if !defined(_LIBCPP_HAS_NO_PRAGMA_SYSTEM_HEADER)
#pragma GCC system_header
#endif

_LIBCPP_BEGIN_NAMESPACE_STD

#ifndef _LIBCPP_HAS_NO_STDIN
extern _LIBCPP_FUNC_VIS istream cin;
extern _LIBCPP_FUNC_VIS wistream wcin;
#endif
#ifndef _LIBCPP_HAS_NO_STDOUT
extern _LIBCPP_FUNC_VIS ostream cout;
extern _LIBCPP_FUNC_VIS wostream wcout;
```

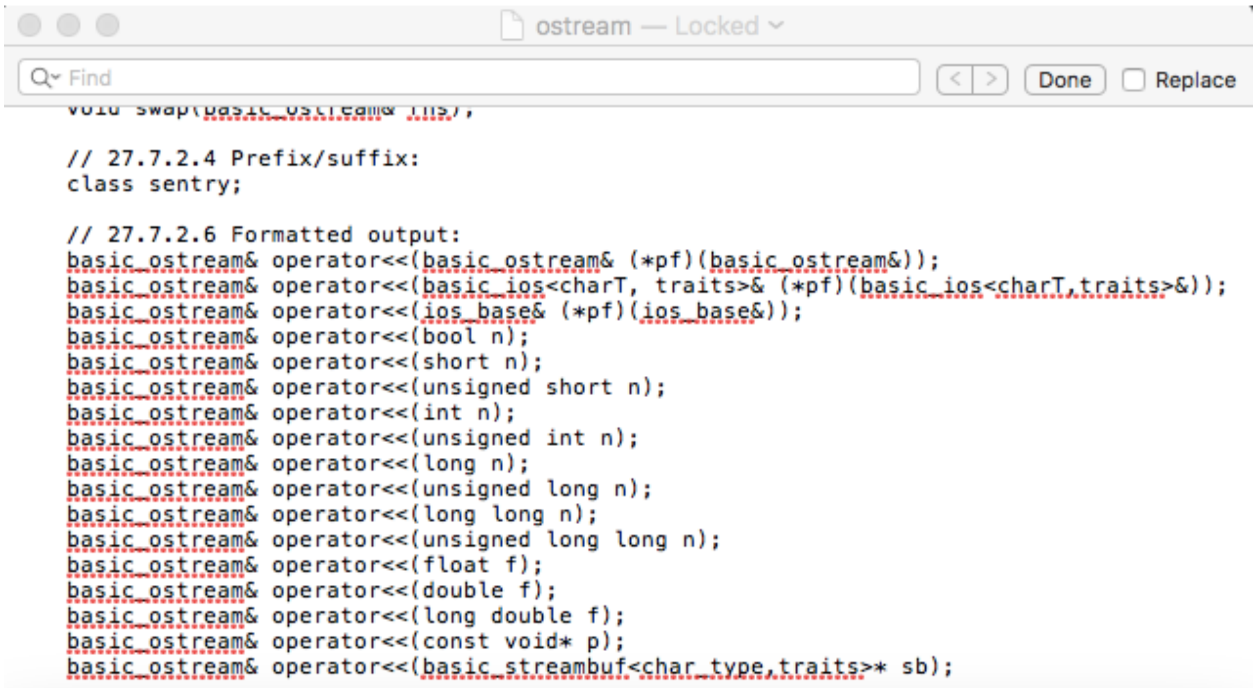
Notice we are including the istream and ostream headers for our *cin* and *cout* objects

cin is an istream object (from the istream class)

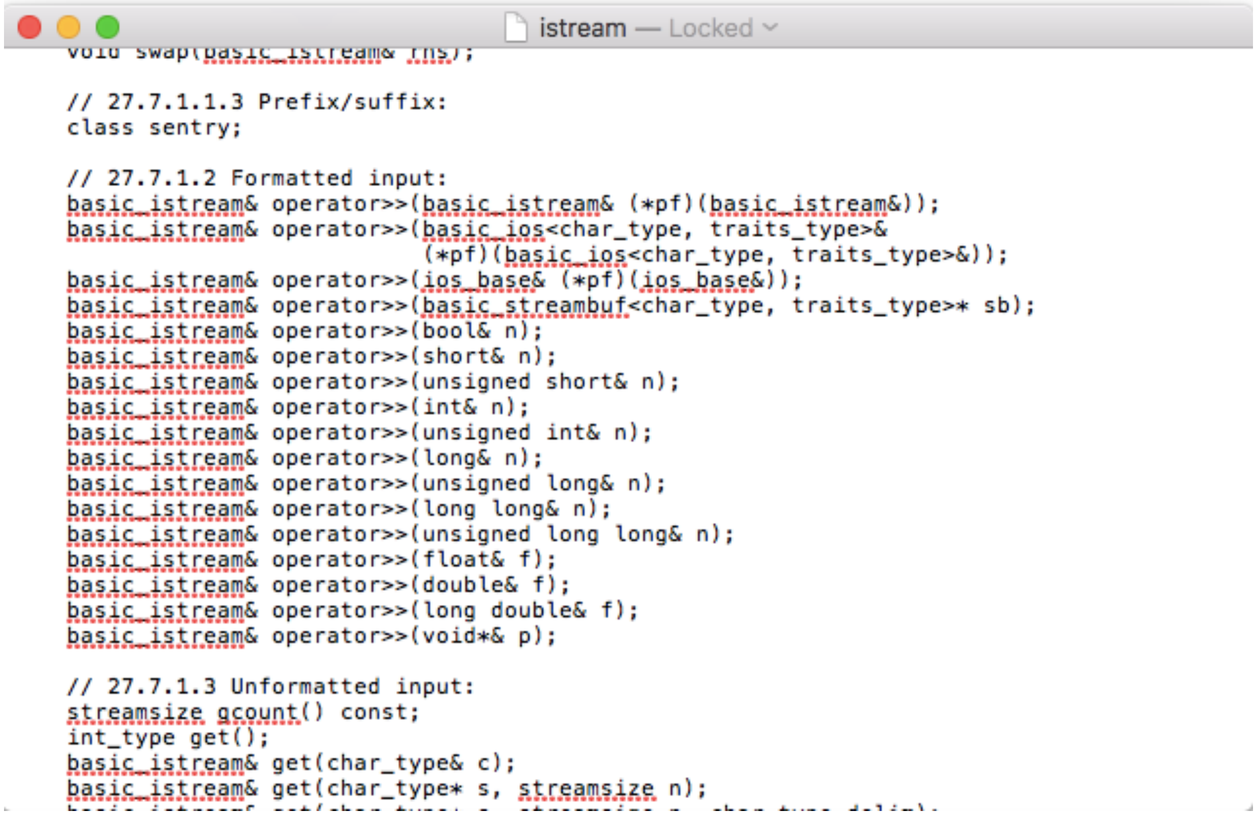
cout is an ostream object (from the ostream class)

We don't have to worry about the inner workings of input and output (that is all hidden from us)-we can just use these objects (the same idea is used with file io in the fstream class that we will see today).

The << and >> we use with them are overloaded operators (we will learn about overloaded operators in a future lecture):



```
    void swap(basic_ostream& rhs);

    // 27.7.2.4 Prefix/suffix:
    class sentry;

    // 27.7.2.6 Formatted output:
    basic_ostream& operator<<(basic_ostream& (*pf)(basic_ostream&));
    basic_ostream& operator<<(basic_ios<charT, traits>& (*pf)(basic_ios<charT,traits>&));
    basic_ostream& operator<<(ios_base& (*pf)(ios_base&));
    basic_ostream& operator<<(bool n);
    basic_ostream& operator<<(short n);
    basic_ostream& operator<<(unsigned short n);
    basic_ostream& operator<<(int n);
    basic_ostream& operator<<(unsigned int n);
    basic_ostream& operator<<(long n);
    basic_ostream& operator<<(unsigned long n);
    basic_ostream& operator<<(long long n);
    basic_ostream& operator<<(unsigned long long n);
    basic_ostream& operator<<(float f);
    basic_ostream& operator<<(double f);
    basic_ostream& operator<<(long double f);
    basic_ostream& operator<<(const void* p);
    basic_ostream& operator<<(basic_streambuf<char_type,traits>* sb);
```



```
    void swap(basic_istream& rhs);

    // 27.7.1.1.3 Prefix/suffix:
    class sentry;

    // 27.7.1.2 Formatted input:
    basic_istream& operator>>(basic_istream& (*pf)(basic_istream&));
    basic_istream& operator>>(basic_ios<char_type, traits_type>&
                              (*pf)(basic_ios<char_type, traits_type>&));
    basic_istream& operator>>(ios_base& (*pf)(ios_base&));
    basic_istream& operator>>(basic_streambuf<char_type, traits_type>* sb);
    basic_istream& operator>>(bool& n);
    basic_istream& operator>>(short& n);
    basic_istream& operator>>(unsigned short& n);
    basic_istream& operator>>(int& n);
    basic_istream& operator>>(unsigned int& n);
    basic_istream& operator>>(long& n);
    basic_istream& operator>>(unsigned long& n);
    basic_istream& operator>>(long long& n);
    basic_istream& operator>>(unsigned long long& n);
    basic_istream& operator>>(float& f);
    basic_istream& operator>>(double& f);
    basic_istream& operator>>(long double& f);
    basic_istream& operator>>(void*& p);

    // 27.7.1.3 Unformatted input:
    streamsize gcount() const;
    int_type get();
    basic_istream& get(char_type& c);
    basic_istream& get(char_type* s, streamsize n);
```

Note that there is a *getline()* function in the istream class (since it deals with input):

```
#include <iostream>

int main (int argc, char **argv) {
  char sentence[256];

  std::cout << "Enter a sentence:";
  std::cin.getline (sentence,256); //we can use our cin object to access this function (since it is in the same class)
  std::cout << "Sentence entered: "<< sentence<<std::endl;
  return 0;
}
```

There is also a function called get() that can take one char at a time:

```
 computer$ ./a.out
hello world!
Only the first letter: h
```

```
#include <iostream>

int main (int argc, char **argv) {

  char c=std::cin.get(); //getting the first letter
  std::cout<<"Only the first letter: "<<c<<std::endl;
}
```

----
ignore() function with get() function:

```
computer$ ./a.out
Enter the day and mood (separated by a comma): Tuesday,happy
Shortened:T h
```

```
#include <iostream>

int main (int argc, char **argv) {
  char day, mood;

  std::cout << "Enter the day and mood (separated by a comma): ";

  day = std::cin.get();
  std::cin.ignore(10,',');   // ignore until comma OR 10 letters out

  mood= std::cin.get();      // get one character

  std::cout << "Shortened:" << day <<" "<< mood << '\n';

  return 0;
}
```

Stringstream:

Stringstream allows us to treat strings as streams (the same idea of streams as above). Instead of an actual stream of data flowing into your program from user input, for example, the stream of information is coming from a string (so the string is treated like a stream):

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

int main (int argc, char** argv)
{
  std::stringstream ss; //create a stringstream object

  ss << "hello there buddy!"; //"putting" values into the string stream (space to separate values)

  std::string word1, word2, word3;
  ss >>word1>>word2>>word3;

  std::cout << "Word1: " << word1 << std::endl;
  std::cout << "Word2: " << word2 << std::endl;

  return 0;
}
```

You can also call a stringstream constructor with a string argument (the string to use as our stream):

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

int main (int argc, char** argv)
{
  std::stringstream ss("hello there buddy!");   //using a stringstream constructor

  std::string word1, word2, word3;
  ss >>word1>>word2>>word3;

  std::cout << "Word1: " << word1 << std::endl;
  std::cout << "Word2: " << word2 << std::endl;

  return 0;
}
```

I used stringstream on the first day of class:

Variation 1:

```
computer$ g++ friends_variation1.cpp
computer$ ./a.out
Enter all 3 friends for the group:
Jon Jane Jill
Jon
Jane
```

```
Jill
All friends added! :)
```

```
#include <iostream>
#include <vector>
#include <sstream>
using namespace std;


int main(int argc, char **argv)
{
  string all_friends;

  cout<<"Enter all 3 friends for the group: "<<endl;
  getline(cin,all_friends);
  stringstream ss(all_friends);

  string friend1;
  string friend2;
  string friend3;

  ss>>friend1>>friend2>>friend3;  SAME AS THE FIRST EXAMPLE

  cout<<friend1<<endl;
  cout<<friend2<<endl;
  cout<<friend3<<endl;

  cout<<"All friends added! :)"<<endl;
}
```

```
computer$ g++ friends_variation2.cpp
computer$ ./a.out
Enter friends:
Jon Jill Will Bill
Total friends: 4
Jon
Jill
Will
Bill
```

Variation 2:

```
#include <iostream>
#include <vector>
#include <sstream>
```

```
using namespace std;



int main(int argc, char **argv)
{
  string all_friends;
  string single_name;
  vector<string> friend_vector;

  cout<<"Enter friends: "<<endl;
  getline(cin,all_friends);
  stringstream ss(all_friends);

  //Here, the loop stops when there is nothing left in our stringstream
  while(ss>>single_name)
  {
    friend_vector.push_back(single_name);
  }

  cout<<"Total friends: "<<friend_vector.size()<<endl;

  for(int i=0;i<friend_vector.size();i++)
  {
    cout<<friend_vector.at(i)<<endl; //you can use [] instead of the at() function
  }

}
```

Manipulators

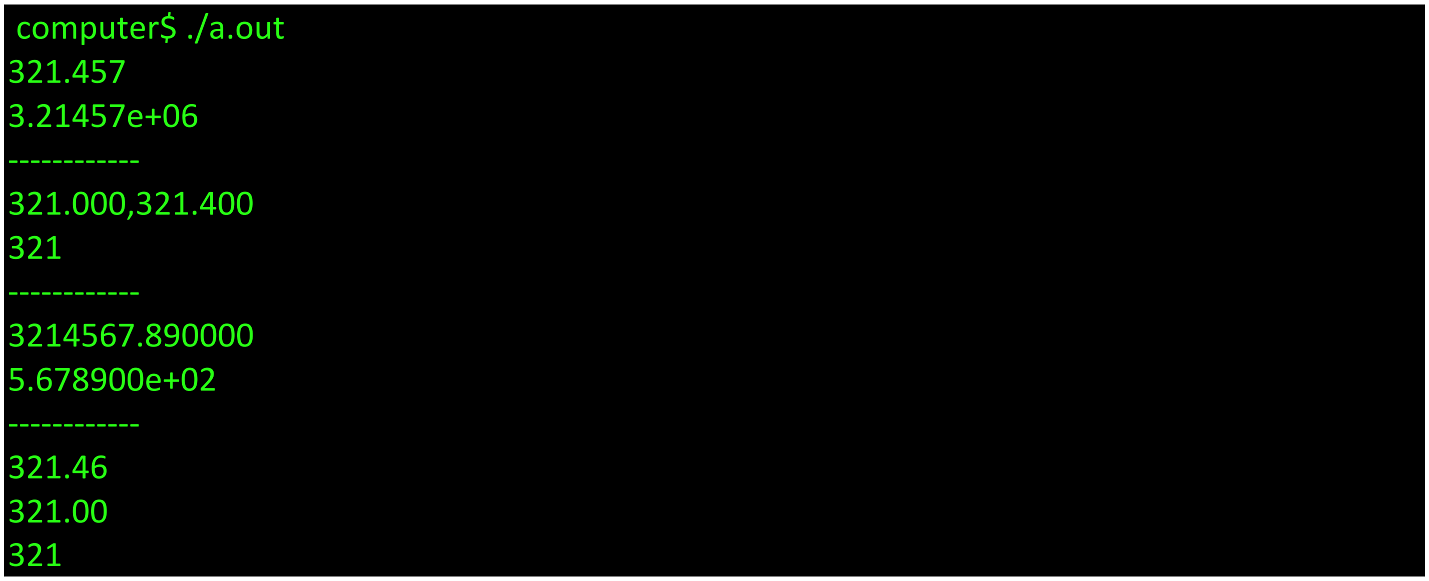
A manipulator controls the formatting of input/output values

Changing bases:

```
computer$ ./a.out
128
80
200
128
128,0x80,0200
+128
```

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

int main () {
        cout << 128 << endl;  //default is decimal
        cout << hex << 128 << endl; //show in hex

        cout << oct << 128 << endl;
        cout << setbase(10) << 128 << endl;  // set base

        //show hex with 0x prefix and oct with 0 prefix
        cout << showbase << 128 << "," << hex << 128 << "," << oct << 128 << endl;
        cout << noshowbase << dec;

        // show with + sign
        cout << showpos << 128 << endl;
}
```

<u>Floating point output (fixed, scientific, default):</u>



```
 computer$ ./a.out
321.457
3.21457e+06
------------
321.000,321.400
321
------------
3214567.890000
5.678900e+02
------------
321.46
321.00
321
```

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

int main () {
        // default floating-point format...fixed-point format
        cout << 321.456987 << endl;

        // default precision is 6...6 digits before and after the decimal point
        cout  << 3214567.89 <<  endl;    //becomes scientific-notation when more than 6 before decimal

        cout<<"------------"<<endl;
        //show trailing 0s
        cout << showpoint << 321. << "," << 321.4 << endl;
        cout << noshowpoint << 321. << endl; //dont show trailing 0s
```

```
            cout<<"------------"<<endl;
            // fixed-point formatting
            cout << fixed <<  3214567.89 << endl;

            // set scientific formatting
            cout << scientific << 567.89 << endl;

            cout<<"------------"<<endl;
            // set precision
            cout << fixed << setprecision(2) << 321.456987 << endl;
            cout << 321. << endl;

            cout << setprecision(0) << 321.456987  << endl;
        }
```

----

setw

```
computer$ ./a.out
            106
```

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

int main () {
  std::cout << std::setw(15);  //moves output over 15 spaces. declared in iomanip header
  std::cout << 106 << std::endl;
  return 0;
}
```

Note: *endl* is a manipulator. We will learn more about it in a future lecture.

Pointers:

```
computer$ ./a.out
Hotel
Hospital
```

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

class Building{
  string name;

public:
  Building(string name){
    this->name=name;
}

  void print_name()
  {
    cout << name << endl;
  }


};

int main (int argc, char **argv) {

  Building b1("Hotel");
  Building b2("Hospital");

  //just like an int variable can change the value, a pointer can change value
  //int i=3;
  //i=4;

  //int *ptr=&n;
  //ptr=&n1;

  Building *ptr_b=&b1; //this pointer is now pointing at the first building
  ptr_b->print_name();

  ptr_b=&b2; //now it's pointing at the second building
  (*ptr_b).print_name(); //remember ptr_b->print_name() can also be written (*ptr_b).print_name()

}
```

We will learn about something called references in a future lecture.

this pointer:

```
computer$ ./a.out
Value of this: 0x7fff5211c9d8
Value of this: 0x7fff5211c998

~Address of b1:0x7fff5211c9d8
~Address of b2:0x7fff5211c998
```

this is a pointer that holds the address of an object we create from the class.

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

class Building{
  string name;
public:
  Building(string name){
    (*this).name=name; //this is a pointer, it points at a specific object (created from this class).  Note that
this->name can be rewritten the way I have written it here: (*this).name
    cout << "Value of this: "<< this << endl; //we will get the address of the object
  }

  void print_name()
  {
    cout << name << endl;
    cout << "Value of this: "<< this << endl;
  }

};

int main (int argc, char **argv) {

  Building b1("Hotel");  // 0x7fff5211c9d8 is the address for b1
  Building b2("Hospital"); //0x7fff5211c998 is the address for b2

  cout<<"\n~Address of b1:"<<&b1<<endl;  // 0x7fff5211c9d8 is the address for b1
  cout<<"~Address of b2:"<<&b2<<endl;  //0x7fff5211c998 is the address for b2

}
```

argc and argv review:

(Check slides first)

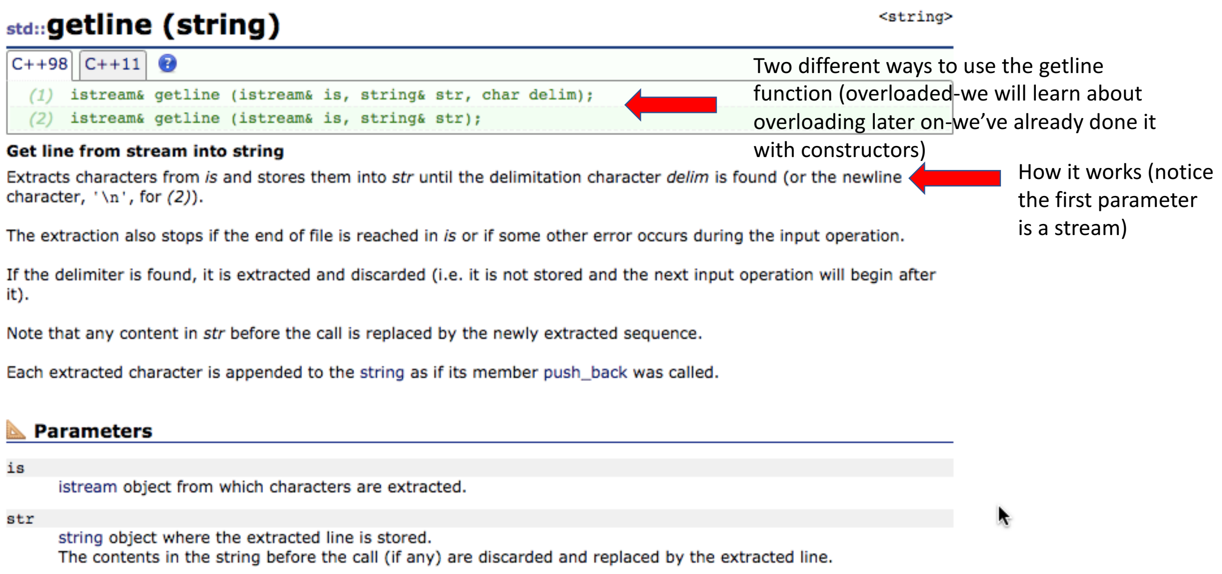
```
computer$ g++ practice.cpp
computer$ ./a.out hello
Address of argv pointer: 0x7fff5e4629e0
Value in argv[0]: ./a.out
Address of argv[0]: 0x7fff5e462aa8
First letter: .

Value in argv[1]: hello
Address of argv[1]: 0x7fff5e462ab0
First letter: h

Value in argv (address-same as addy of argv[0]): 0x7fff5e462aa8
Value in argv (address-same as addy of argv[1]): 0x7fff5e462ab0
```

```
#include <iostream>
```

```
using namespace std;

int main(int argc, char ** argv)
{
        cout<<"Address of argv pointer: "<<&argv<<endl;

        cout<<"Value in argv[0]: "<<argv[0]<<endl;
        cout<<"Address of argv[0]: "<<&argv[0]<<endl;
        cout<<"First letter: "<< *argv[0]<<endl;

        cout<<"\nValue in argv[1]: "<<argv[1]<<endl;
        cout<<"Address of argv[1]: "<<&argv[1]<<endl;
        cout<<"First letter: "<< *argv[1]<<endl;

        cout<<"\nValue in argv (address-same as addy of argv[0]): "<< argv<<endl;
        argv++; //pointer arithmetic, move from argv[0] to argv[1]

        cout<<"Value in argv (address-same as addy of argv[1]): "<< argv<<endl;
}
```

Program 1: Song list

A note about getline:



std::getline (string) <string>

C++98 | C++11

```
(1) istream& getline (istream& is, string& str, char delim);
(2) istream& getline (istream& is, string& str);
```

Two different ways to use the getline function (overloaded-we will learn about overloading later on-we've already done it with constructors)

Get line from stream into string

Extracts characters from *is* and stores them into *str* until the delimitation character *delim* is found (or the newline character, '\n', for *(2)*).

How it works (notice the first parameter is a stream)

The extraction also stops if the end of file is reached in *is* or if some other error occurs during the input operation.

If the delimiter is found, it is extracted and discarded (i.e. it is not stored and the next input operation will begin after it).

Note that any content in *str* before the call is replaced by the newly extracted sequence.

Each extracted character is appended to the string as if its member push_back was called.

Parameters

is
istream object from which characters are extracted.

str
string object where the extracted line is stored.
The contents in the string before the call (if any) are discarded and replaced by the extracted line.

http://www.cplusplus.com/reference/string/string/getline/

```
computer$ g++ -std=c++11 practice.cpp
computer$ ./a.out Fad songs.txt
Enter number on list:
1
Fad's Song: Redbone *** Artist: Childish Gambino
Enter number on list:
3
Fad's Song: a Lot *** Artist: 21 Savage
Enter number on list:
6
Fad's Song: Low life *** Artist: Future
Enter number on list:
8
Fad's Song: Safe to Say *** Artist: Yella Beezy
Enter number on list:
0
```

```cpp
#include <iostream>
#include <map>
#include <string>
#include <fstream>
#include <sstream>

class Listener{

public:
   std::string name;
   std::map<int, std::string> current_songs; //song, favorite list

   Listener(std::string name, std::string filename)
   {
          this->name=name;
          std::ifstream inFile; //file object created from the ifstream class
          std::string line;
          int cals;
          inFile.open(filename);

          if (!inFile.is_open())
          {
                    std::cout << "Unable to open file";
                    exit(1); // terminate with error
          }

          while (!inFile.eof())
          {
                    std::string intermediate;
                    int number;
                    std::string songname_artist;
                    std::string line_from_file;
```

```
        getline(inFile,line_from_file); //using getline to read in whole line (otherwise will
split at space).  notice the first argument is the file object, not cin because cin is for standard
input from the keyboard
        std::stringstream delimt_line(line_from_file); //create a stringstream object-the
whole line I just read from the file

        getline(delimt_line,intermediate, '.'); //get number (before the .)
        number=stoi(intermediate);

        getline(delimt_line,intermediate, '(');
        songname_artist="Song: "+intermediate;

        getline(delimt_line,intermediate, ')');
        songname_artist=songname_artist+" *** Artist: "+intermediate;
        current_songs.insert({number,songname_artist});
    }
  }
};

//just type in the number of the song to see the song on the list.  0 means exit (also notice the
program doesn't account if you enter an number that doesn't have a song)
int main(int argc, char ** argv)
{
    int response;
    Listener l1(argv[1],argv[2]);

    while(true)
    {
        std::cout<<"Enter number on list: "<<std::endl;
        std::cin>>response;

        if(response==0) //exit is 0
        {
            break;
        }

        std::cout<<l1.name<< "'s "<<l1.current_songs.at(response)<<std::endl; //can
also say: l1.current_songs[response]
    }

}
```