Input/Output

CSci 588: Data Structures, Algorithms and Software Design

http://www.cplusplus.com/doc/tutorial/files/

http://www.cplusplus.com/doc/tutorial/basic_io/

Reading a variable from keyboard input.

input from keyboard

```
// i/o example

#include <iostream>
using namespace std;

int main ()
{
  int i;
  cout << "Please enter an integer value: ";
  cin >> i;
  cout << "The value you entered is " << i;
  cout << " and its double is " << i*2 << ".\n";
  return 0;
}
```

input from keyboard

```
// i/o example

#include <iostream>
using namespace std;

int main ()
{
  int i;
  cout << "Please enter an integer value: ";
  cin >> i;
  cout << "The value you entered is " << i;
  cout << " and its double is " << i*2 << ".\n";
  return 0;
}
```

cin takes input from the keyboard and moves it into a variable.

input from keyboard

```
// i/o example

#include <iostream>
using namespace std;

int main ()
{
  int i;
  cout << "Please enter an integer value: ";
  cin >> i;
  cout << "The value you entered is " << i;
  cout << " and its double is " << i*2 << ".\n";
  return 0;
}
```

cin works similarly to cout, except in reverse. Data is coming from the keyboard and moved into variables.

input from keyboard

```
// i/o example

#include <iostream>
using namespace std;

int main ()
{
  int i;
  cout << "Please enter an integer value: ";
  cin >> i;
  cout << "The value you entered is " << i;
  cout << " and its double is " << i*2 << ".\n";
  return 0;
}
```

You have to press enter on the keyboard for cin to register the input and move it into the variable (or variables specified).

Reading multiple variables from keyboard input.

input from keyboard

```
// i/o example

#include <iostream>
using namespace std;

int main ()
{
  int i, j;
  cout << "Please enter two integer values: ";
  cin >> i >> j;
  cout << "The values you entered are " << i << " and “ << j << “.\n”;
  cout << " the product of them is " << i*j << ".\n";
  return 0;
}
```

You can read multiple variables from the same line of keyboard input. They can be separated by any whitespace: a space, a tab or a newline.

Again, for all the variables to be parsed, enter must be pressed.

Reading strings from keyboard input.

cin and strings

```
// cin with strings
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string mystr;
  cout << "What's your name? ";
  cin >> mystr;
  cout << "Hello " << mystr << ".\n";
  cout << "What is your favorite team? ";
  cin >> mystr;
  cout << "I like " << mystr << " too!\n";
  return 0;
}
```

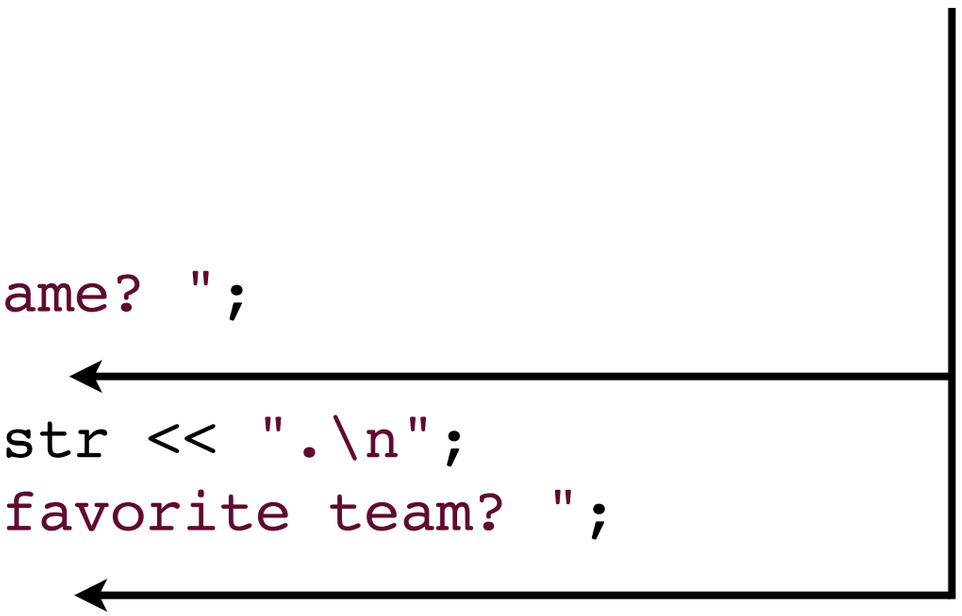
cin separates values with whitespace, so if you try to enter your full name this won't work.

cin and strings

```
// cin with strings
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string mystr;
  cout << "What's your name? ";
  getline(cin, mystr);
  cout << "Hello " << mystr << ".\n";
  cout << "What is your favorite team? ";
  getline(cin, mystr);
  cout << "I like " << mystr << " too!\n";
  return 0;
}
```

You can use the getline function to read the entire line entered before the user presses enter.



stringstream

stringstream

```
// stringstreams
#include <iostream>
#include <string>
#include <sstream>
using namespace std;

int main ()
{
  string mystr;
  float price=0;
  int quantity=0;

  cout << "Enter price: ";
  getline (cin,mystr);
  stringstream(mystr) >> price;
  cout << "Enter quantity: ";
  getline (cin,mystr);
  stringstream(mystr) >> quantity;
  cout << "Total price: " << price*quantity << endl;
  return 0;
}
```

You can use a stringstream to move data into variables (just like cin).

File Streams

Writing to a file

```
// basic file operations
#include <iostream>
#include <fstream>
using namespace std;

int main () {
  ofstream myfile;
  myfile.open ("example.txt");
  myfile << "Writing this to a file.\n";
  myfile.close();
  return 0;
}
```

Writing to a file

```
// basic file operations
#include <iostream>
#include <fstream>
using namespace std;

int main () {
  ofstream myfile;
  myfile.open ("example.txt");
  myfile << "Writing this to a file.\n";
  myfile.close();
  return 0;
}
```

We use an ofstream (output file stream) to read from a file.

Writing to a file

```
// basic file operations
#include <iostream>
#include <fstream>
using namespace std;

int main () {
  ofstream myfile;
  myfile.open ("example.txt");
  myfile << "Writing this to a file.\n";
  myfile.close();
  return 0;
}
```

This opens the file for writing.

Writing to a file

```
// basic file operations
#include <iostream>
#include <fstream>
using namespace std;

int main () {
  ofstream myfile;
  myfile.open ("example.txt");
  myfile << "Writing this to a file.\n";
  myfile.close();
  return 0;
}
```

Now instead of writing to the screen, we can write into that file.

Writing to a file

```
// basic file operations
#include <iostream>
#include <fstream>
using namespace std;

int main () {
  ofstream myfile;
  myfile.open ("example.txt");
  myfile << "Writing this to a file.\n";
  myfile.close();
  return 0;
}
```

When we're done, we need to close the file.

Writing to a file

```
// basic file operations
#include <iostream>
#include <fstream>
using namespace std;

int main () {
  ofstream myfile;
  myfile.open("example.txt");
  myfile << "Writing this to a file.\n";
  myfile.close();
  return 0;
}
```

However there is a problem. What if the "example.txt" file didn't exist?

Writing to a file 2

```
// writing on a text file
#include <iostream>
#include <fstream>
using namespace std;

int main () {
  ofstream myfile ("example.txt");
  if (myfile.is_open())
  {
    myfile << "This is a line.\n";
    myfile << "This is another line.\n";
    myfile.close();
  }
  else cout << "Unable to open file";
  return 0;
}
```

We can check to see if a file was opened with the is_open() method.

Reading from a file

```
// reading a text file
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main () {
  string line;
  ifstream myfile ("example.txt");
  if (myfile.is_open())
  {
    while ( myfile.good() )
    {
      getline(myfile,line);
      cout << line << endl;
    }
    myfile.close();
  }

  else cout << "Unable to open file";

  return 0;
}
```

Reading from a file

```
// reading a text file
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main () {
  string line;
  ifstream myfile ("example.txt");
  if (myfile.is_open())
  {
    while ( myfile.good() )
    {
      getline(myfile,line);
      cout << line << endl;
    }
    myfile.close();
  }

  else cout << "Unable to open file";

  return 0;
}
```

Reading from a file works just like reading from cin.

Reading from a file

```
// reading a text file
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main () {
  string line;
  ifstream myfile ("example.txt");
  if (myfile.is_open())
  {
    while ( myfile.good() )
    {
      getline(myfile,line);
      cout << line << endl;
    }
    myfile.close();
  }

  else cout << "Unable to open file";

  return 0;
}
```

We use an ifstream (input file stream) to open a file for reading.

Reading from a file

```
// reading a text file
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main () {
  string line;
  ifstream myfile ("example.txt");
  if (myfile.is_open())
  {
    while ( myfile.good() )
    {
      getline(myfile,line);
      cout << line << endl;
    }
    myfile.close();
  }

  else cout << "Unable to open file";

  return 0;
}
```

We can also check and see if the file continues to be good to read from (for example, we haven't reached the end of the file).

Checking State Flags

In addition to good(), which checks whether the stream is ready for input/output operations, other member functions exist to check for specific states of a stream (all of them return a bool value):

bad() - Returns true if a reading or writing operation fails. For example in the case that we try to write to a file that is not open for writing or if the device where we try to write has no space left.
fail() - Returns true in the same cases as bad(), but also in the case that a format error happens, like when an alphabetical character is extracted when we are trying to read an integer number.
eof() - Returns true if a file open for reading has reached the end.
good() - It is the most generic state flag: it returns false in the same cases in which calling any of the previous functions would return true.

In order to reset the state flags checked by any of these member functions we have just seen we can use the member function clear(), which takes no parameters.

File Streams

C++ provides the following classes to perform output and input of characters to/from files:

- **ofstream:** Stream class to write on files
- **ifstream:** Stream class to read from files
- **fstream:** Stream class to both read and write from/to files.

Setting flags for file streams

It is possible to set specific flags to change the way file streams read from and write to a file:

ios::in - Open for input operations.
ios::out - Open for output operations.
ios::binary - Open in binary mode.
ios::ate - Set the initial position at the end of the file. If this flag is not set to any value, the initial position is the beginning of the file.
ios::app - All output operations are performed at the end of the file, appending the content to the current content of the file. This flag can only be used in streams open for output-only operations.
ios::trunc - If the file opened for output operations already existed before, its previous content is deleted and replaced by the new one.

For example:

```
ofstream myfile;
myfile.open ("example.bin", ios::out | ios::app | ios::binary);
```

Default flags:

class	default mode parameters
ofstream	ios::out
ifstream	ios::in
fstream	ios::in | ios::out