Functions
Oct. 13, Ch 4-5

- $e^x$
- $e^x \cdot \frac{d}{dx}$
- $e^x$

A wild Exponential Function appeared
You use Differentiate
It is not very effective
Announcements

Quiz back (today?)

Midterm next week, similar to quiz (longer)
- Covers Ch. 1-3 (up to and including loops)
Functions

Functions give chunks of code a name to...
- break down larger code into small simple parts
- allow re-usability across your program (even more reason not to copy-paste!)

Arguments (inside parenthesis after function name), allow information to be passed into a function and the return allows information to be passed back out
Functions

You can also use functions that return bool types in an if statement or loop.

This is commonly used if you have complex logic as it is normally easier to write a function that have a very complex bool expression.

(See: findPrime.cpp)
(See: SillySwap.cpp)

Typically the value of variables is copied and not given access to the real value.

This is similar to moodle, the score you see for grades cannot change the score I give you!
Blocks (inside `{ }`) of code can only see variables from their parent blocks.

You can also make global variables outside of all blocks (almost as if your whole program has a start and end brace around it).

(See: globalVariable.cpp)
We will talk more about the difference between a variable's memory location and value later.

For now, a memory location (or reference or pointer) will give a function full access to modify the value.
You can give away your memory location by using “call by reference” with functions. This will share the variable between the two functions, namely the function that is using the references (&) can modify the value.

(See: callByReferenceSwap.cpp)
Memory

Memory:

Code:
Memory (declaration)

Memory:
#0 (int) x

Code:

```c
int x;
```
Memory (initialization)

Memory:

#0 (int) x

2

Code:

```
int x;
x = 2;
```
Memory (re-assign)

Memory:
#0 (int) x

Code:
```c
int x;
x = 2;
x = 9;
```
Memory (copy value)

Memory:

#0 (int) x
9

#1 (double) y
9.00 * 10^0

Code:

```c
int x;
x = 2;
x = 9;
double y = x;
```
When memory does not actually hold the value of an object, but instead holds information about the actual location this is called a pointer
(See: memoryLocation.cpp)
Debugging

- Test small pieces of code at a time (much like the clock problem was split up for you)
- Add cout statements to find where error is in loops (and to localize error in general)
- Test code that you know the answer for
- Add comments (to functions!)
- Use assert (See assert.cpp)
Functions
Oct. 13, Ch 6

stream of consciousness
I'm aware.
Highlights

- text file output
  ```cpp
  ofstream out;
  out.open("output.txt");
  ```

- text file input
  ```cpp
  ifstream in;
  in.open("input.txt");
  ```
Download vs stream
A “stream” is information flow that is immediately processed

For example:

- Streaming video is watch as data arrives
- Downloading video stores it for later

For file input/output (file I/O), we will have to create a stream between file and code
“Opening” a file

File output is very similar to terminal output, except we have to open and close files.

To create a stream between a variable name and file name:

```cpp
ofstream out;  
out.open("output.txt");
```
“Opening” a file

Sometime you cannot open a file (don't have permission)

You can check if the file actually opened by calling fail() (returns true if did NOT open):

```c
if (out.fail())
{
    exit(1); // non-zero for an error state
}
```

exit() in `<cstdlib>`, causes program to terminate
After you have opened a file (stream), you can then write to it.

This is done in an almost as cout, except you use the your variable name for the file.

Terminal: `cout << "Hello!\n";`

File: `out << "Hello!\n";`
Closing a file

Once we are done writing to a file, we should close the stream:

```java
out.close();
```

If you don't close your stream, something might be left in the buffer.
File output imports

To use ofstream type, you need to include `<fstream>`

```cpp
#include <fstream>
```

This gives you ofstream (output file stream) and ifstream (input file stream), which we will see next

(See: helloWorldFile.cpp)
Where did this file go?

The default “path” for a file is where your cpp file is located

You can specify the path when you open the file:

```java
out.open("/home/park0580/PutItHere.txt");
```

You can also use relation operations:

```java
out.open("../PutItHere.txt");
```
Appending to files

What happens if I run HelloWorldFile multiple times?

Open file and override:

```cpp
out.open("output.txt");
```

Open file and append:

```cpp
out.open("output.txt", ios::app);
```

(See: helloWorldFileAppend.cpp)
File writing overview

- You need to open a file before writing to it
- You should close the file when you are done
- You can either override or append to files
- Use `.fail()` to see if file actually opened

- You **cannot** go backwards and “replace” or “undo”
- You **cannot** “preppend” to a file (must either append from end or override)
Input is similar to output, we need to open a stream then use it similar to cin

```cpp
string x;
ifstream in;
in.open("input.txt");
if(!in.fail())
{
    in >> x;
}
in.close();
```

What is a major difference between reading and writing to a file?
End of file (EOF)

When there is nothing left in a file to read, we call it **end of file**

C++ is fairly nice about handling EOF, and you can detect it in 3 ways:

```c++
while(getline(in,x))
while(in >> x)
while(!in.eof())
```

- **reads from file**
- **does not read from file (just tells if at end)**
You can use also use setf() on your streams, but you can also use setw() and setprecision() from <iomanip>

```cpp
ofstream outFile;
outFile.open("fancyOutput.txt");
outFile.setf(ios::showpoint);
outFile.setf(ios::fixed);
//outFile.unsetf(ios::fixed); //undoes above
outFile << "$" << setprecision(2) << setw(8) << 23.61 << endl;
```

(See readTable.cpp)