**Week 1**

How to use headers:

Use #ifndef HEADERNAME\_H

#define HEADERNAME\_H

//header declarations

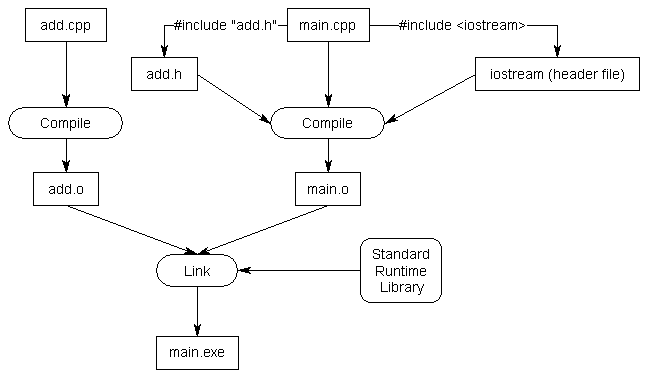
#endif

This is a header guard, it stops the computer from defining implementations twice.

Header files are used to hold declarations

Cpp files are used to hold implementations

The computer will find the cpp file with the main method to run that and when the header files are called included beforehand, the implementations will be found as well.



**Debugging Code**

-Compile the Code

-Press F10 to start debugging, keep pressing F10 to go line by line

-Press F11 when you’re line is at a function call to look into the function call

-Press F5 to run the rest of the program with no debugging

**Breakpoints**

-Highlight the line you want to set a breakpoint at, press F9

-press F5 to run the program until the breakpoint

**Try not to use (using namespace std;) in header files**

**Defining**

**-**#define can be used to define code

-#ifdef <code> #endif will compile the code if it was defined

- #ifndef <code> #endif will only compile the code if it wasn’t defined before

#ifndef class #define class <code> #endif works nicely in this manner

**Including**

-in a cpp file, only include the header file of another class if you must use that object explicitly

-otherwise, simply state class someclass; to hint that it’s going to be used (i.e. you have a someclass pointer or it’s passed in a parameter, etc)

--to avoid cyclical class calling, (A include B and B includes A) have A contain a pointer to a B object instead of an actual B object.

**Compiling Under G++**

**-**download MinGW

-search for your project folder

-type g++ -std=c++11 \*.cpp \*.h

-to run the program, simply type a.exe OR ./a.out

-use cd.. to navigate up in a folder, cd (foldername) to navigate into a folder

-use command “dir” to see the contents of your current folder

**Constructors**

-recall that classes of one type cannot call the private variables of another class

-^that doesn’t stop one class from accessing the private variables of the same class

-Circle a; Circle b; 🡪 b can access a’s private variables even if they’re different objects

-Copy Construction implements this function, Circle::Circle(const Circle &a){}

This allows us to create a new object based on the old one

A copy constructor must have const, pass by reference, and same object type

Circle b = a; will call the copy constructor 🡪 same as calling Circle b(a);

C++ provides us a default copy constructor, but this is how you write your own

Pitfall: If you use a default copy constructor and it involves pointers. If with the copied object you delete something a pointer is pointing to, the original object can’t access it either now

Anytime your class allocates dynamic memory (new / delete) or opens system resources,

Define your own copy constructor that dynamically allocates its own data (instead of using the original object’s dynamic memory allocation)

**Assignment Operator**

Same kind of use as copy constructor (there’s a predefined version but you should define your own if your class dynamically allocates or opens system reserve)

Takes in a const reference to an object

Circle & operator= (const Circle &source) //this is correct syntax

--if you try to set a= a, you might run into trouble

if(&source==this) {return (\*this)} //this fixs the bug

Must return (\*this) which would be the object itself after it has been worked on

Syntactic sugar: bar.operator=(foo) is the same as bar = foo;

**Linked Lists**

A class designed such that it holds information and a pointer variable of the same class. This allows the pointer to point to the next element hence a linked list of that class.

First item is called the head. Class is usually defined as Node.

-be sure to set the last pointer to nullptr to indicate the end of the list

-you can define a head and tail pointer and use p->next != nullpointer to traverse

-doubly linked lists have a next and prev pointer to Nodes