**Separate Compilation**

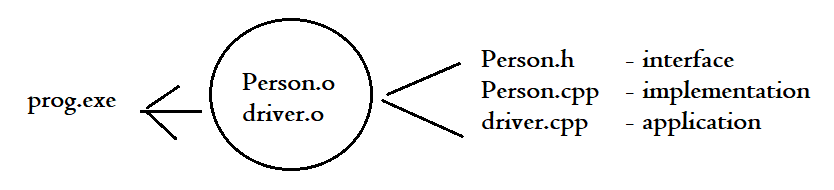
Separate Compilation is the separating and organization of code, and then compiling it together into an executable file.

The code in its most basic form is comprised of three files.   
An Interface file, Implementation file, and the Application file.

The interface file or files are where you declare the libraries the program will use, the functions that are used, as well as individual classes (Most commonly each class gets its own file). The interface file is also known as the header file, which it’s extension is .h instead of .cpp.

The implementation file defines the functions declared in the header file, and the application file, or also known as the driver file, is where you write the code found in main. Those two files keep the .cpp file, until in command line you compile them to become .o files.

Note: Each header file accompanies a implementation file, and you have multiple files of these to usually separate classes. Once you have your .h and .o files you can compile them together to make an .exe file and run it as a normal program.



**How to Write : The Interface/Header (.h):**

The header file will declare its individually necessary libraries, its class and relevant functions . This file should also explain what each function is set to do through standard notation.

#include <iostream>

#include <string>

using namespace std;

class Person {

public:

// initializes object's name to "" & age to 0

Person();

// @param n - set the value for person's name

// @param a - set the value for person's age

Person(string n, int a);

// @return - person's name as a string

string getName() const;

// @return - person's age as an integer

int getAge() const;

// @param n - set the value for the person's name

void setName(string n);

// @param a - set the value for the person's age

void setAge(int a);

// outputs person's name and age

void output() const;

// @return - the number of person objects

static int getNumPeople();

private:

string name;

int age;

static int numPeople;

};

**How to Write : The Implementation/Definition (.cpp -> .o):**

The Implementation file will define all functions from connected header file. It starts by using a #include to connect the header file to this file. Each class function has to be called upon as if it’s being defined outside of the class (Example – Person :: Person( ) )

#include "EX\_5.1\_Person.h"

Person::Person(): Person("", 0) {}

Person::Person(string n, int a): name(n), age(a) {}

string Person::getName() const { return name; }

int Person::getAge() const { return age; }

void Person::setName(string n) { name = n; }

void Person::setAge(int a) { age = a; }

void Person::output() const {

cout << "Name: " << name << "\nAge: " << age << endl;

}

int Person::numPeople = 0;

int Person::getNumPeople() { return numPeople; }

**How to Write : The Application/Driver (.cpp -> .o):**

The Application file contains the code and function calls found in main. This is what runs the application. You connect all h files used to it by using the #include library.

#include "EX\_5.1\_Person.h"

int main() {

Person p1{};

p1.setName("John Smith");

p1.setAge(20);

p1.output();

return 0;

}

**Combining the ‘Trinity’**

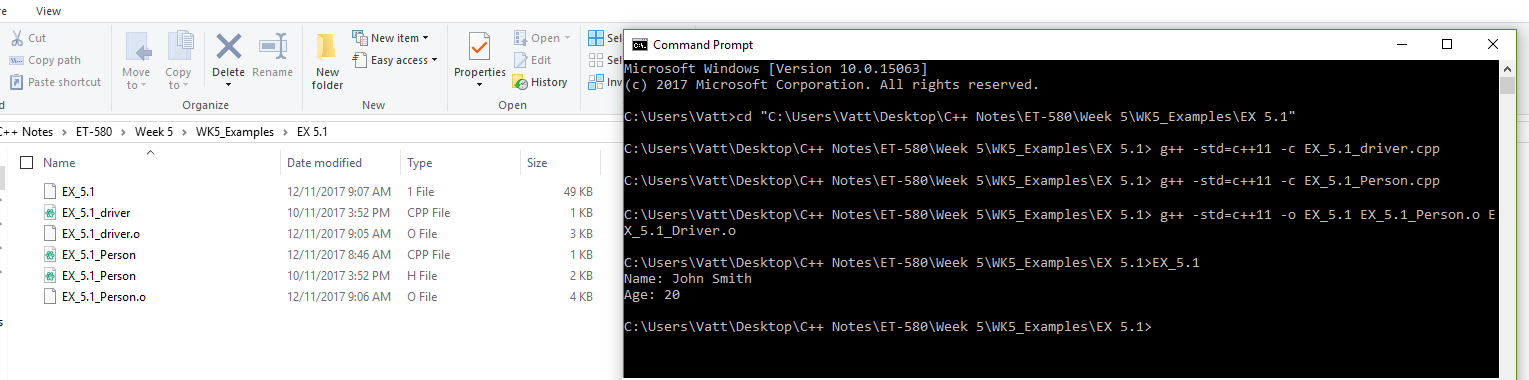
To combine these three files and create an executable, first you must compile and convert the two .cpp files into .o files. You can do this through Command Prompt.

1. First navigate command prompt to the file’s folder   
   ( You can do so by typing CD (Change Directory) and dragging the folder onto the command line screen )
2. Compile each .cpp file with the following code:

g++ -std=c++11 –c *filename*.cpp

1. Link the .o files together with the following code:

g++ -std=c++11 -o *program-name* filename.o driver.o

1. ****Run the program by running its name!

Command Prompt Basics

CD = Change Directory

CD .. = Go down a directory

DIR = Shows files in current folder (directory)

cls = Clears screen

prompt $g = Removes Path from view

prompt = Returns Path to view

**How to Create a Makefile**

A Makefile is a command prompt-read file that when written it compiles and cleans your c++ files automatically, without having to type a thing in command prompt, except for the Makefile.

Create a file called **makefile** without any extensions.

Keywords:

# = Commenting Tool

\ = Continue code to next line

CXXFLAGS: = Helps prevents compilation errors or warnings

**Code:**

CXXFLAGS := -std=c++11  
  
# executable file to create and its dependencies (Notice direct command prompt code)

Program-Name.exe : driver.o filename1.o filename2.o

g++ -std=c++11 -o Program-Name driver.o filename1.o \ filename2.o

# file dependencies

driver.o : filename1.h filename2.h

g++ -std=c++11 -c driver.cpp

filename1.o : filename1.h filename2.h

g++ -std=c++11 -c filename1.cpp

filename2.o : filename2.h

g++ -std=c++11 -c filename2.cpp

# files to remove

clean:

del driver.o filename1.o filename2.o

del prog.exe

To Run: In command prompt, type : mingw32-make

To Clean: In command prompt, type : mingw32-make clean