Purpose of this handout:

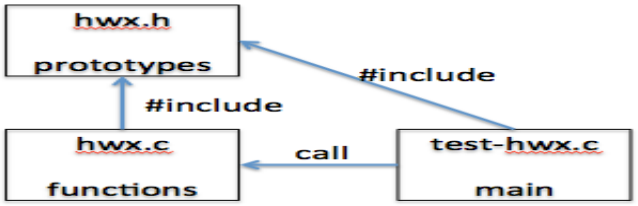
1) Testing methods
2) Simple conditionals
3) Splitting a program into three files

Basic C Program 5:

```
/* convert_celsius_to_fahrenheit
 * purpose: Converts a temperature given in Celsius into Fahrenheit.
 *  input parameters:
 *   float – the temperature in Celsius
 *  return value:
 *   float – the temperature in Fahrenheit
 */
float convert_celsius_to_fahrenheit(float cel)
{
      return (cel*9.0)/5.0+32;              // must explicitly return the calculated number
}


/* test_convert
 * purpose: Helper function to test convert_celsius_to_fahrenheit
* inputs:
 *       double Celsius – input Celsius that is to be converted
 *       double expected – the expected result (temperature in Fahrenheit)
 *       double accuracy – how close to the value to be considered correct
 * outputs:
 *      0 if incorrect, 1 if correct
 */
unsigned int test_convert(double celsius, double expected, double accuracy)
{
      double fahr;
      // call the function
      fahr =  convert_celsius_to_fahrenheit(celsius);
      // check whether it matched expected results.
      // Note the range check for accuracy.
      if ((fahr >= expected - accuracy) && (fahr <= expected + accuracy))
      {
            printf("Test passed: Test celsius %lfF, expected %lfC, calculated %lfC.\n",
                  celsius, expected, fahr);
            return 1;
      }
      //OR: if (!((fahr >= expected - accuracy) && (fahr <= expected + accuracy)))
      if ((fahr < expected - accuracy) || (fahr > expected + accuracy))
      {
            printf("FAILED TEST: Test celsius %lfF, expected %lfC, calculated %lfC.\n",
                  celsius, expected, fahr);
            return 0;
      }
}
```

```
int main()
{
      double fahr, celsius;
      unsigned int total_tests = 0;
      unsigned int passed_tests = 0;

      // print out user information
      printf("This program converts from Celsius to Fahrenheit\n");

      // call a series of tests
      // fill in the text_convert blanks with well thought-out tests!
      passed_tests = passed_tests + test_convert(          ,          );
      total_tests++;
      passed_tests +=  test_convert(          ,          );
      total_tests++;
      passed_tests +=  test_convert(          ,          );
      total_tests++;
      passed_tests +=  test_convert(          ,          );
      total_tests++;

      printf("Passed %u out of %u tests\n",passed_tests, total_tests);

      // return success
      return (0);

}
```

Basic Program 5b: Showing the same code, placed in multiple files
All programs in this class will be placed in a minimum of 3 files.
test-hwx.c – contains main code that starts execution. Contains function *calls*.
hwx.c – contains function *implementations* that are called by main
hwx.h – contains function *prototypes* of all functions implemented in hwx.c



********** **hwx.h** **********

```
#ifndef HWX_H                   // always at top of hwx.h. Compiler reads file only 1x.
#define HWX_H                   // this makes prior statement false next time read.

/* convert_celsius_to_fahrenheit        // this is the required function header
 * complete function header here….
 */
float convert_celsius_to_fahrenheit(float);                     // function prototype

#endif                          // closes if stmt starting with #ifndef in 1st line
```

********** **hwx.c** **********

```
#include <stdio.h> // these are the first two lines of all .c files
#include <stdlib.h>
#include "hwx.h"
/* convert_celsius_to_fahrenheit
*  complete function header here…..
 */
float convert_celsius_to_fahrenheit(float cel)
{
      return (cel*9.0)/5.0+32;          // must explicitly return the calculated number
}
```

********** **test-hwx.c** **********

```
/* test-hwx.c
 * purpose: illustrate how all homework will be structured (3 files)
 * same code as basic program 5, just different file structure and testing
 */
#include <stdio.h> // these are the first two lines of all .c files
#include <stdlib.h>
#include "hwx.h"    // this allows us to call functions from hwx.c
/* test_convert
*  complete function header here…..
 */
unsigned int test_convert(double celsius, double expected, double accuracy)
{
      double fahr;
      // calculate the temperature
      fahr =  convert_celsius_to_fahrenheit(celsius);
      // check whether it matched. Note the range check for accuracy.
      if ((fahr >= expected - accuracy) && (fahr <= expected + accuracy))
      {
            printf("Test passed: Test celsius %lfF, expected %lfC, calculated %lfC.\n",
                  celsius, expected, fahr);
            return 1;
      }
      if ((fahr < expected - accuracy) || (fahr > expected + accuracy))
      {
            printf("FAILED TEST: Test celsius %lfF, expected %lfC, calculated %lfC.\n",
                  celsius, expected, fahr);
            return 0;
      }
}
int main()
{
      double fahr, celsius;
      unsigned int total_tests = 0, passed_tests = 0;
      printf("This program converts from Celsius to Fahrenheit\n");
      // call a series of tests
      passed_tests = passed_tests + test_convert(       ,       );
      total_tests++;
      passed_tests +=  test_convert(       ,       );
      total_tests++;
      passed_tests +=  test_convert(       ,       );
      total_tests++;
      passed_tests +=  test_convert(       ,       );
      total_tests++;
      printf("Passed %u out of %u tests\n",passed_tests, total_tests);
      return (0);
}
```

****

To compile the code (go from human-readable code to an executable for the computer):

The executable will be named a.out:	`$ clang hwx.c test-hwx.c`
To run:	`$ ./a.out`
To make an executable with the name blah	`$ clang hwx.c test-hwx.c –o blah`
To run:	`$ ./blah`

****

To automate the compilation process, unix has the program "make" that uses a file named makefile
Contents of makefile for this program:

```
myexe: test-hwx.c hwx.c hwx.h
      clang test-hwx.c hwx.c –o myexe
```

Notes:

1) test-hwx.c hwx.c hwx.h on first line after ':' means that if any of those files have changed, recompile. List all files there for which it should compile in case of change.
2) Only the .c files are on the compile line, not the .h file
3) Must be a tab before "clang", not spaces.
4) myexe: is the name of the target.
5) You may have multiple targets in one makefile.
    a. Typing make compiles the 1st target in file.
    b. If you have multiple targets, use "make myexe" to compile