CS 162 Worksheet 5

1. Work with multiple files – dos and don'ts
   - NEVER compile .h files in a g++ command
     - Simply compile the .cpp files and the #include'd header files will be compiled as part of those.
   - DO #include header files in each source (.cpp) file that uses those functions (or classes).
   - DO recompile any .cpp files that #include the header file WHEN the header file changes.

2. Concept of Structs:
   Structs (originated in the C) are the predecessors of classes (C++)
   - a user-defined data type of grouping related data together to model some 'object'.

   Ex. Create a struct that represents a vehicle. Explain why you listed your member variables the way you did.

   Now, suppose you are given the following struct:

```
struct Garage {
      int num_of_vehicles;
      Vehicle* v;
};
```

   Explain the relationship between the Vehicle and the Garage struct.

   Once a Garage object g is defined, how would you populate the array of vehicles within g?

   How would you use these structs in a program that allows users to search for a vehicle of a specific model, rent a vehicle, and return a vehicle?

3. Describe what each line of the makefile does and answer the following questions.

```
CC = g++
exe_file = mult_div

$(exe_file): mult_div.o prog.o
$(CC) mult_div.o prog.o –o $(exe_file)

mult_div.o: mult_div.cpp
$(CC) –c mult_div.cpp

prog.o: prog.cpp
$(CC) –c prog.cpp


 clean:
   rm –f *.out *.o $(exe_file)
```

(1). What happens if you put the clean target at the top above the first target to make the executable?

(2). What happens if you define your .o targets before your executable target?

(3). What happens if you do not use tabs to indent?

4. Understand Program Errors
   a. Compiling Errors - The compiler GIVES you a line number and error message.
      E.g., syntax, missing a semicolon, use a variable before declaring it, etc.
      What is your debug strategy for compiling errors?

   b. Runtime Errors - produce the wrong output or crash, leaving no clues as to why.
      E.g., Segmentation Fault
      What is your debug strategy for compiling errors?

5. Pointer with structs:

```
1     struct Pokemon {
2           string name;
3           int size_move;
4           string *moves;
5     };
6
7     struct Pokedex {
8           int num;
9           Pokemon* p_arr;
10    };
11
12    void test( ___①___ var){
13          // code
14    }
15
16    int main () {
17          Pokedex p;
18          p.num = 5;
19          p.p_arr = new ____④____ [p.num];
20          for (int i = 0; i < p.num; i++) {
21                ...
22                p.p_arr[i]_③_moves = new ___⑤___ [p.p_arr[i].size_move];
23          }
24          test (___②___);
25          return 0;
26    }
```

a. From line 17, is **p** a data type or an object?

b. What should be filled in ③ at line 22, a . or ->? What's the difference? Can we replace it with:
`p.p_arr->moves = ...`
Why or why not?

c. What should be filled in ④ at line 19?

d. What should be filled in ⑤ at line 22?

e. If ② is **p**, what should be filled in ①?

f. If ② is **p.p_arr**, what should be filled in ①?

g. If ② is **p.p_arr[0]**, what should be filled in ①?

h. If ② is **`p.p_arr[0].moves`**, what should be filled in ①?

i. If ② is **`p.p_arr[0].moves[1]`**, what should be filled in ①?