CS 162 Worksheet 6

1. Accessor and Mutators:
   Create a garage class that has a **dynamic array of vehicle structs**. Make sure you create an int variable to indicate the number of vehicles and follow the rules for encapsulation. Write the declarations for mutator, and accessor functions needed to access the members in the garage. Use const when necessary.

```
struct vehicle {
   string name;
   int num_wheels, num_seats;
   bool motor;
};

class garage {
      private:

      _____________________________;   //dynamic array of vehicles

      _____________________________;   //number of vehicles

   public:

      _____________________________;  //accessor for dynamic array

      _____________________________;  //mutator for dynamic array

      _____________________________;  //accessor for number of vehicles

      _____________________________;  //mutator for number of vehicles
};
```

2. Use of const:

Given the following class declaration, explain each use of const. For a-d, tell what is legal, what is not illegal, and why.

```
class MyClass {
private:
      int member1;
public:
      void fun1(const int x);
      int fun2() const;
};
```

a.
```
void Myclass::fun1 (const int x){
      int y = x;
}
```

b.
```
void Myclass::fun1 (const int x){
      x = member1;
}
```

c.
```
int Myclass::fun2() const{
      return this->member1;
}
```

d.
```
int Myclass::fun2() const{
      this->member1 = 2;
      return this->member1;
}
```

3. Classes and objects:

Read and trace the code from the following three files, and answer the following questions.

garage.h:

```
1  #include <iostream>
2  #include <string>
3
4  using namespace std;
5
6  class garage{
7  private:
8      int num_cars; //Number of cars in garage
9      string* cars; //An array of car names
10 public:
11     garage();
12     garage(int num_cars);
13
14     // A function that deletes the garage's
15     // dynamic memory. Make sure to call it
16     // before the garage falls out of scope, or else
17     // you'll have a memory leak. Note: there's a
18     // better way to do this via "destructors", but
19     // we may not have covered destructors in lecture
20     // by the time you see this worksheet.
21     void delete_memory();
22
23     // Mutator for individual car within array
24     void set_car(int index, string value);
25
26     // Accessor for individual car
27     string get_car(int index) const;
28
29     // Accessor for num_cars
30     int size() const;
31
32     // Some other member functions for
33     // educational purposes
34     garage fun1() const;
35     garage& fun2(const garage&);
36 };
37
38
```

garage.cpp:

```cpp
#include "garage.h"

// Default constructor implementation. Sets
// .num_cars = 0, and .cars = nullptr
garage::garage() : num_cars(0), cars(nullptr) {
    cout << "Garage()" << endl;
}

// Nondefault garage ctor. Sets .num_cars = n,
// and .cars = new string[n]
garage::garage(int n) : num_cars(n), cars(new string[n]) {
    cout << "Garage(int)" << endl;
}

void garage::delete_memory() {
    // Equivalently, if(cars) {...}
    if (cars != nullptr) {
        delete [] cars;
        cars = nullptr;
    }
}

void garage::set_car(int index, string value) {
    if (index < 0 || index >= num_cars){
        cout << "Error! set_car index out of bounds!" << endl;
    } else {
        cars[index] = value;
    }
}

string garage::get_car (int index) const {
    if (index < 0 || index >= num_cars){
        cout << "Error! get_car index out of bounds!" << endl;
        return "";
    } else {
        return cars[index];
    }
}

int garage::size() const {
    return num_cars;
}

garage garage::fun1() const {
    // Create an empty garage and return its
    // value (reminder: return values are copied,
    // unless you're returning a reference, and
    // you can't return a reference to a local variable)
    garage empty_garage;
    return empty_garage;
}

garage& garage::fun2(const garage& some_garage) {
    // Ignore some_garage and just return *this
    // (i.e., return THIS garage). However,
    // we're returning a garage, not a garage&
    // or a garage*, so it returns a copy.
    return *this;
}
```

main.cpp:

```cpp
#include <iostream>

#include "garage.h"

using namespace std;

int main(){
    garage g1;
    cout << g1.size() << endl;
    g1.set_car(0, "Tesla");
    g1.get_car(0);

    garage g2(5);
    cout << g2.size() << endl;
    g2.set_car(0, "Maserati");
    g2.set_car(1, "Jeep");
    cout << g2.get_car(0) << endl;
    g2.get_car(1);
    g2.get_car(5);

    g2.fun1();

    garage& g3 = g2.fun2(g1);

    // Don't forget to delete the
    // garages' dynamic memory, if
    // they have any!
    g1.delete_memory();
    g2.delete_memory();
    // Don't delete g3's memory! It's
    // just a reference to g2
    return 0;
}
```

1. Between lines 11 and 12 in garage.h, which one is the default constructor, and which one is the non-default constructor?

2. What is printed by line 8 in main.cpp?

3. What is printed by line 9 in main.cpp?

4. Is anything printed by line 10 in main.cpp? If so, what?

5. Is anything printed by line 11 in main.cpp? If so, what?

6. What is printed by line 13 in main.cpp?

7. What is printed by line 14 in main.cpp?

8. Is anything printed by lines 15 and 16 in main.cpp? If so, what?

9. What is printed by line 17 in main.cpp?

10. Is anything printed by line 18 in main.cpp? If so, what?

11. Is anything printed by line 19 in main.cpp? If so, what?

12. Is anything printed by line 21 in main.cpp? If so, what?

13. Is anything printed by line 23 in main.cpp? If so, what?

14. What would happen if we additionally called `g3.delete_memory()` at the end of `main()`?

4. Understanding errors

For each program and compiler / linker error shown below, answer the following questions: In what file and line of code does the error appear? In your own words, what does the error mean? How would you fix this error?

Hint: compiler errors are described in great detail, and there are only a couple of common linker errors—you shouldn't even need to look at the code to understand, at least superficially, what's causing the problem (though you may need to see the code to fully understand the issue).

1. one.cpp:

```cpp
#include <iostream>

using namespace std;

int create_x() {
    int x = 5;
    return x;
}

int main(){
    create_x();
    cout << x << endl;
    return 0;
}
```

error:

```
$ g++ one.cpp
one.cpp: In function ‘int main()’:
one.cpp:12:17: error: ‘x’ was not declared in this scope
   12 |         cout << x << endl;
      |                 ^
```

2. two.cpp:

```cpp
#include <iostream>

using namespace std;

int main(){
    my_function();
}

void my_function() {
    cout << "Hello, world!" << endl;
}
```

error:

```
$ g++ two.cpp
two.cpp: In function ‘int main()’:
two.cpp:6:9: error: ‘my_function’ was not declared in this scope
    6 |         my_function();
      |         ^~~~~~~~~~~
```

3.

three.h:

```
1  #ifndef THREE_H
2  #define THREE_H
3
4  void func();
5
6  #endif
```

foo.cpp:

```
1  #include <iostream>
2
3  #include "three.h"
4
5  using namespace std;
6
7  void func(string value) {
8      cout << value << endl;
9  }
```

main.cpp:

```
1  #include "three.h"
2
3  int main () {
4      func("Hello!");
5  }
```

Error:

```
$ g++ three.cpp three_main.cpp
three_main.cpp: In function ‘int main()’:
three_main.cpp:4:13: error: too many arguments to function ‘void func()’
    4 |         func("Hello!");
      |         ~~~~^~~~~~~~~~
In file included from three_main.cpp:1:
three.h:4:6: note: declared here
    4 | void func();
      |      ^~~~
```

4.

hello.h:

```
#ifndef FOUR_H
#define FOUR_H

//This function should simply
//print "Hello, world!" to the
//termial
void hello_world();

#endif
```

hello.cpp:

```
#include <iostream>

#include "four.h"

using namespace std;

void hello_world(string some_string) {
    cout << "Hello, world!" << endl;
}
```

main.cpp:

```
#include "four.h"

int main () {
    hello_world();
}
```

Error:

```
$ g++ four.cpp four_main.cpp
/bin/ld: /tmp/ccVopdR9.o: in function `main':
four_main.cpp:(.text+0x5): undefined reference to `hello_world()'
collect2: error: ld returned 1 exit status
```