**CS 162 Worksheet 5**

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:

vehicle\* v; //dynamic array of vehicles

int num\_vehicles; //number of vehicles

public:

vehicle\* get\_v() const; //accessor for dynamic array

void set\_v(const vehicle\* new v); //mutator for dynamic array

int get\_num\_vehicles() const; //accessor for number of vehicles

void set\_num\_vehicles(const int new\_num\_vehicles); //mutator for number of vehicles

};

1. 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;

};

1. void Myclass::fun1 (const int x){ // Legal. X isn’t being modified.

int y = x;

}

1. void Myclass::fun1 (const int x){ // Illegal. X can’t be modified.

x = member1;

}

1. int Myclass::fun2() const{

return this->member1; // Legal. Nothing is changing (X is being returned).

}

1. int Myclass::fun2() const{

this->member1 = 2;

return this->member1; // Illegal. member1 is changing but is a const.

}

3. Classes and objects:

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

|  |
| --- |
| garage.h: |
| garage.cpp:   |  |  | | --- | --- | |  |  | |  |  | |
| main.cpp: |

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

**The default constructor is line 11, and the non-default constructor is line 12.**

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

**Line 8 prints the string “Garage()”.**

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

**Line 9 prints the number of cars. In this case, it will print the string “0”.**

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

**Line 10 will print an error message since the index of zero at g1.set\_car(0, “Tesla”) is equal to the number of cars.**

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

**Line 11 will also print an error message since the index is out of bounds.**

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

**Line 13 will print the string “Garage(int)”.**

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

**Line 14 will print the string “14”.**

1. Is anything printed by lines 15 and 16 in main.cpp? If so, what?  
     
   **Nothing will be printed by line 15.**
2. What is printed by line 17 in main.cpp?

**Line 17 will print “Maserati”.**

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

**Line 18 will not print anything to the screen.**

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

**Line 19 will print an error message because the index is out of range.**

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

**Line 21 will print the string “Garage()”.**

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

**Line 23 will not print anything.**

1. What would happen if we additionally called g3.delete\_memory() at the end of main()?

**If we call g3.delete\_memory() at the end of main, we would get an error message since g3 is not dynamic memory.**

4. Understanding errors

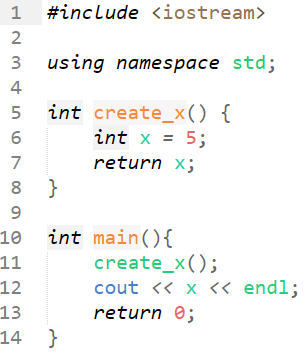
For each program and compiler / linker error shown below, answer the following questions: In what file and line of code does the error

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:  
   

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;

| ^  
**The error appears in file one.cpp at line 12, and means that the variable ‘x’ was not declared before attempting to use it. You would fix this by declaring ‘x’ before attempting to use it.**

1. two.cpp:  
   

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();

| ^~~~~~~~~~~ **// The error is in file two.cpp at line 6, and means that the function was not declared/defined before the main() function. You can fix this by moving this function above the main() function.**

|  |  |  |
| --- | --- | --- |
| three.h: | foo.cpp: | main.cpp: |

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!");

| ~~~~^~~~~~~~~~ **// This error occurs in the file three\_main.cpp at line 4, and means that the function being called has too many parameters compared to what it expected as inputs. This can be fixed by removing the unexpected parameter or updating the function to allow it.**

In file included from three\_main.cpp:1:

three.h:4:6: note: declared here

4 | void func();

| ^~~~ **// The error is at line 4 in the three.h header file. The error means that function func() was previously declared in a different file but doesn’t match the header prototype, and can be fixed by making the prototype for func() match the intended use for the function.**

|  |  |  |
| --- | --- | --- |
| hello.h: | hello.cpp: | main.cpp: |

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 **// This error occurs in file four\_main.cpp at line 4, and means that the function hello\_world() hasn’t been defined. This can be fixed by renaming the header file “four.h” or by renaming the file includes to match the current header file name.**