CS-2303 System Programming Concepts WPI, A-term 2017  
Professor Mike Ciaraldi Quiz #6 (20 points)  
Quiz date: Thursday, October 5, 2017 ANSWER KEY

|  |  |  |
| --- | --- | --- |
| Question | Possible | Points |
| 1 | 5 |  |
| 2 | 5 |  |
| 3 | 5 |  |
| 4 | 5 |  |
| Maximum | 20 |  |

NAME:

WPI E-mail ID:

READ THESE INSTRUCTIONS BEFORE STARTING THE QUIZ.

This is a closed-book quiz, open-notes quiz. You can consult any of the class handouts, and anything you wrote yourself. You cannot consult tests from previous versions of the course. You are not allowed to use any electronic or communications device during the quiz, without special permission.

This quiz is worth 20 points.

Answer questions in the spaces provided on the quiz itself. Take the number of points assigned to each question and the amount of space provided for your answer as a measure of the length and difficulty of the expected solution.

Write or print your answers, name, and ID legibly and large enough that we can easily read them.

Be sure to answer the question which is actually being asked, in a way which demonstrates that you understand the meaning of the question and the answer. For example:

* If the question asks you to say what happens and why, you have to tell both to get full credit.
* If the question asks you to explain something in general and to give an example, you have to do both to get full credit.
* If the question asks how many bytes, answer with the number of bytes, not the number of bits, kilometers, or Volts.

*Remember, the graders know the answers to these questions. What you write has to show that* ***you*** *understand the question and its answer.*

A *program fragment* is a section of code which is part of a complete program. You can make any reasonable assumptions about the rest of the program.

All questions apply to C and/or C++ unless otherwise indicated.

This quiz will end at 8:20 am. *It is* STRONGLY *suggested that you read the entire quiz before attempting to answer any questions. Also, re-read all the parts of any one question before you start answering it, so you put your answers in the right place.*

1. **const parameters and functions** [5 points]
   1. In C and C++, if you have a parameter which is a pointer and is declared **const**, what does that mean (i.e., what can you do and not do with that pointer, as compared to a pointer parameter which is not const)? Give an example which clearly shows that you understand this. [2 points]

*Possible answer: If the pointer parameter is declared const, this means that the pointer itself can be changed by the function so it points somewhere else, but the function cannot follow the pointer and use it to change the value stored in the calling function. For example, suppose the function prototype is* ***int foo (int\* x);*** *Inside function foo you could do* ***x++*** *, which changes the value of x, so it points to the next integer. But you could not do* ***\*x = 5*** *, which would go to the integer pointed to by x and change its value.*

* 1. In C++, what does it mean when a member function is declared **const**? [2 points]

*Answer: This means that the function will not change the state of the object (i.e., the value of data inside the object).*

* 1. In C++, if a member function is declared **const**, the compiler will only allow that function to call other functions of that class which are also declared **const**. Why does C++ have this rule? [1 point]

*Answer: Because if it did not have this rule, the other functions could change the state of the object, which means the first function would not really be const!*

1. **Function Overloading in C++** [5 points]
   1. Suppose you have a class with no base class or derived classes. It has two functions with the same name. When some program calls a function with that name, on an object of that class, how does the compiler know which one to actually call? Give an example which illustrates this.[3 points]

*Possible answer: It tries to match the signature; specifically, the number and type of the parameters. So, for example, if you had a function* ***foo(int x)*** *and another function* ***foo(double x, int y),*** *the compiler would see if you called it with one or two parameters, and use the version which matched.*

* 1. Why can the compiler not use the return type to make this decision? Give an example which illustrates this. [2 points]

*Possible answer: Because C and C++ allow you to “promote” the type of the result. For example, if you had a function int foo(int x), and you called it with double y = foo(z), the compiler will happily turn the returned int into a double for you. But if you also had float int foo(int x), the compiler could also promote. And if you had double foo(int x), the compiler could handle that without promoting. Since any of these would be legal, the language does not allow you to have this ambiguity.*

1. **Class Hierarchy in C++** [5 points]
   1. Suppose you have a function (method) in a base class, and you want the derived class to use the same function. What should you do while writing your program, so the compiler knows that this is what you want? [1 point]

*Answer: Do not do anything special in the derived class, and it will automatically inherit the function from the base class.*

* 1. Suppose you have a function in a base class, and you want the derived class to do something different when the function with that name is called. What should you do? [1 point]

*Answer: Define the function in the derived class, so it overrides the one in the base class.*

* 1. Suppose you have base class A and its derived class B. You have a function foo() defined in both classes. You have a variable p which is of type A\* and it points to an object of type B. When you execute p->foo(), which version of the function do you get, the one in A or the one in B? Why? [2 points]

*Answer: The version in A, because non-virtual methods are statically bound, so they depend on the type of the variable.*

* 1. Same as in part c, except the function is declared virtual. Which version do you get? Why? [1 point]

*Answer: The version in B, because virtual methods are dynamically bound, so they depend on the type of the object.*

1. **More Class Hierarchy in C++** [5 points]
   1. Suppose you have a class which has at least one pure virtual method (i.e. an abstract class). C++ has the rule that you cannot ever instantiate an object which is just of that class (it has to be one of the concrete subclasses). Why does C++ have that rule? [3 points]

*Answer: If C++ allowed this, you could have an object of that type, and then when you tried to invoke the pure virtual method on it, there would not actually be a method body to execute (because by definition, a pure virtual method does not have a body). In other words, an abstract class is not complete enough to allow you to have a usable object of that type. Note: Some students wrote that the abstract class can serve as the base of a hierarchy of classes; that is true, but so can a concrete class.*

* 1. Suppose you have a parent class A, with subclasses B, C, D, and E. You specify a pure virtual function in the parent class, and concrete ones with the same signature (calling sequence) in each of the subclasses. Why did you need to have the prototype of the function in the parent class? [2 points]

*Answer: Suppose you declare a variable of type A\*; even though you can never instantiate an object of type A, this variable can point to objects of type B, C, D, and E. Having the prototype in A notifies the compiler that it is safe to call this method on any object which this variable points to. That is because you can only instantiate objects defined by concrete classes, so any object which that variable points to is guaranteed to have a body for that method. Note: The question specifically says that the subclasses are concrete, so we know that the function bodies are defined there. If any of them did not have a function body, it would be an abstract class.*