CS-2303 System Programming Concepts WPI, A-term 2017  
Professor Mike Ciaraldi Quiz #4 (20 points)  
Quiz date: Thursday, September 21, 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 an 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.

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 unless otherwise indicated.

This quiz will end at 8:20 pm. *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. **Incrementing.** [5 points]

C has both a pre-increment operator and a post-increment operator.

* 1. What is the difference between pre-incrementing and post-incrementing? Explain this in general, then give an example which clearly illustrates the difference. [3 points]

*Answer: In pre-incrementing, the value of the variable gets changed (increased), and then the updated value is used. In post-incrementing, the value of the variable is used, and only after that does the value in the variable gets changed.*

*A possible example:*

*Pre-increment: If j == 1, and you then do k = ++j, then j becomes 2, and k is also set to 2.*

*Post-increment: If j == 1, and you then do k = j++, the original value of j, which is 1, gets copied into k, and then the value in j gets updated to 2.*

* 1. Sometimes using the ++ operator does not increase the numeric value of a variable by 1; the value increases by a different amount. In what case would this happen? How does the compiler decide how much to add? Give an example. [2 points]

*Answer: This can happen when the variable is a pointer. In that case, the numeric value increases by the number of bytes in the type of variable being pointed at. For example, a pointer to int on x86 would be incremented by 4.*

1. **Characters and Strings** [5 points]
   1. What is the difference between writing ‘R’ and “R” in your program? Hint: How much memory does each occupy, and why? [2 points]

*Answer: When you write ‘R’, you are specifying a single character, which occupies one byte. When you write “R”, you’re a specifying a string of length one; this occupies two bytes, because the system needs one byte for the letter and a second byte for the null terminator character.*

* 1. Consider the following program fragment. By “program fragment” I mean that this is just part of a bigger program which compiles correctly.

**char\* s = “Ahoy, matey!”;**

**int i; // Counts the length of a string**

**i = 0;**

**while (\*s) { s++; i++; }**

Why will this correctly calculate the length of the string? Be sure your answer is clear, and demonstrates that you understand how these things work in C: Loop conditions (what changes each time through the loop, and what causes the loop to continue or end), pointer dereferencing, and logical tests. Note: An answer which basically says “the program counts the characters in the string until it gets to the end” is not sufficient. [3 points]

*Answer: The pointer s starts at the first character in the string, and i starts at zero. Each time through the loop, s gets moved to point to the next character, and i gets incremented. At the start of each iteration of the loop, the character pointed to by s gets tested. Since C considers the value zero to be false and any non-zero value to be true, and because the end of a string is marked by a null terminator (which has the value zero), the loop continues until s is pointing at the terminator.*

*Note: To get full credit, you needed to include enough of this explanation to demonstrate your understanding of each of the topics mentioned:*

* *Dereferencing: Pointer s points into the character string. \*s is the character currently being pointed to.*
* *Logical test: Zero value is considered false, any non-zero is considered true.*
* *Loop condition: Continues as long as condition is true.*
* *Null terminator has the numeric value zero.*

1. **Structs** [5 points]

Suppose you have the following statements at the start of a program:

**struct foo { double x; double y; }; // Define the struct called foo**

**struct foo a; // Declare a local variable**

**struct foo\* c = &a; // Declare another local variable; initialize it.**

Remember that on an x86-64 machine like our VM, a double and a pointer are each 64 bits long.

* 1. You now have a variable **a** which contains a struct of type foo. Write a C statement which would put the value 7.5 into data field **x** in that struct. [1 point]

*Answer:* ***a.x = 7.5;*** *Note that there is no need to mention foo again.*

* 1. You now also have a variable **c** which contains a valid pointer to an existing struct of type foo. Write a C statement which would put the value 6.2 into data field **y** in that struct. [1 points]

*Answer:* ***c->y = 6.2;*** *Note:* ***(\*c).y = 6.2;*** *would also work. So would* ***\*c.y = 6.2;*** *because the star binds with higher priority than the dot. Still, the arrow is usually clearer.*

* 1. How many bytes of memory does variable **a** occupy on our VM, and why? [1 point]

*Answer: 16 bytes, 8 bytes for each of the doubles. Note: Remember that each byte contains 8 bits. A few students wrote “24 because you need 8 more bytes for the pointer,” but there is no pointer inside the struct.*

* 1. How many bytes of memory does variable **c** occupy on our VM, and why? [2 points]

*Answer: 8 bytes, because it is a pointer.*

1. **Tree traversal** [5 points total]

Consider the following binary tree. Every time a node is visited, we will print the number in that node. What numbers would be printed, in what order, if the tree is traversed in each of the following orders?

**50**

**37**

**15**

**75**

**20**

**45**

* 1. Pre-order [2 points]

*Answer: 50, 15, 20, 45, 37, 75*

* 1. Post-Order [1 point]

*Answer: 20, 45, 15, 75, 37, 50*

* 1. In-order [2 points]

*Answer: 20, 15, 45, 50, 37, 75*