|  |  |  |
| --- | --- | --- |
| **APCS Exposure Java** | **Exercises 19.01-06** | **Date:** |
| **Name:** | | **Period:** |

01. Define *Recursion*.

02. Is recursion covered on the AP Exam?

03. What *BASIC* command is frowned upon by computer scientists?

04. Look at program ***Java1901.java***. What is wrong with this program?

05. What happens to the computer if it is executing a recursive method that has no planned method

of stopping?

06. What does *recursion* simulate?

07. What are the 3 *iterative* control structures?

08. Refer to the previous question. What stops the looping process in all 3 of these *control structures*?

09. Look at program ***Java1902.java***. Explain the purpose of the **if (k < 100)** statement.

10. Look at the output of program ***Java1902.java***. Why do the numbers not line up nicely?

11. All recursive methods must have an **exit** that stops the recursive process.

What is the special case or condition that stops the recursion called?

12. What are the 2 goals the author is trying to make with the first 2 program examples?

13. Look at program ***Java1903.java***. What is the *base case* for the recursive *skip2* method?

14. Look at program ***Java1904.java***. What is the *base case* for the recursive *count* method?

15. Look again at program ***Java1904.java***. How is the *counting* accomplished?

16. Look at program ***Java1905.java***. What is the difference between the 2 recursive *count* methods?

17. Look at the output of program ***Java1905.java***. Why is the second set of numbers in reverse order?

18. Define a *stack*.

19. What does *LIFO* stand for?

20. Where is every program statement stored?

21. What handles program execution sequence?

22. Besides *Internet Protocol*, what else does *IP* stand for?

23. Finish this sentence:

*When the method execution is finished, program execution needs to resume with the ...*

24. When you add information to a *stack*, what is that officially called?

25. When a recursive call interrupts the execution of a method, what happens to the "unfinished business"

at the end of the method?

26. What data structure controls *recursion*?

27. Look at program ***Java1907.java***. What is the *base case*?

28. Look at program ***Java1908.java***. What is the *base case*?

**For all values of n > 0**

**n! = n(n-1)!**

**and**

**0! = 1**

29. Compare the recursive **fact2** method from program

***Java1908.java*** to the mathematical definition of factorial

on the right. What do you notice?

30. What does a computer scientist mean when he/she refers to an *elegant solution*?

31. Look at program ***Java1909.java***. What is the *base case*?

32. Look again at program ***Java1909.java***.

The iterative **gcf1** require the statements **n1 = n2;** and **n2 = rem;**

How is this handled in the recursive **gcf2** method?

33. Look at program ***Java1910.java***. What is the *base case*?

34. Look again at program ***Java1910.java***. What is wrong with the recursive method **fibo2**?

35. If an iterative **fibo** method takes 10 seconds to find the *1,000,000th* Fibonacci number?

About how long will it take to find the *2,000,000th* Fibonacci number?

36. If a recursive **fibo** method takes 30 seconds to find the *100th* Fibonacci number?

About how long will it take to find the *101st* Fibonacci number?