Exam 2: Alternative Version

1. Which statement below tests if letter holds W. (letter is a char)
   a. if (letter == "W") → W is in double quotes so the system will interpret it as a String. This code will cause an error
   b. if (letter >= "W") → See A
   c. if (letter == W) → W is not in quotes so the system will interpret it as a variable. This code will likely cause an error
   d. if (letter = 'W') → Only one = is used, so it's an assignment statement within the if, not a comparison. This code will cause an error
   e. if (letter == 'W') → The correct comparator is used, ==, and W is in single quotes so the system knows to interpret it as a literal char

2. What are if statements used for in programs?
   a. Repeating commands
   b. Storing data
   c. Numeric calculations
   d. Numeric casts
   e. Making decisions → if statements are used for making decisions

3. What is output?

```
int x = 31 % 8;
if ( x > 10)
      System.out.println(1);
else if ( x > 8 )
      System.out.println(2);
else if ( x > 6 )
      System.out.println(3);
else if ( x > 4 )
      System.out.println(4);
else
      System.out.println(5);
```

   a. 1 b. 2 **c. 3** d. 4 e. 5

4. Short circuit evaluation means that in the code:

```
if ( y != 0 && x/y > 10 )
```

   a. If x/y > 10 is false it evaluates y != 0 → The second statement is never evaluated first
   b. If y != 0 is false it doesn't evaluate x/y > 10 → This is an 'and' statement, so if the first statement is false, then the whole thing is false. The system doesn't need to evaluate the second half
   c. If x/y > 10 is false it doesn't evaluate y != 0 → See A
   d. If y != 0 is true it doesn't evaluate x/y > 10 → Both halves need to be true for the statement to be true
   e. If y != 0 is false it evaluates x/y > 10 → If the first half is true, it won't evaluate the second half

5. The following truth table matches which boolean condition?

A	B	?
T	T	T
T	F	F
F	T	F
F	F	F

a. A && (A || B) → This would be true for only lines 1,2
b. A || (!A && !B) → This would be true for lines 1,2,4
c. **A && (A && B) → This is the same as A && B, only true for line 1**
d. !A && (A || !B) → This would be true only for line 4
e. A || (A || B) → This would be true for lines 1,2,3

6. Consider the following code segment:

```
if( x < 200 || x > 299 )
      System.out.println("Not in the 200's");
else
      System.out.println("In the 200's");
```

Which of the following code segments produces the exact same output?

I.

```
if ( x < 200 )
      System.out.println("Not in the 200's");
else if ( x > 299 )
      System.out.println("Not in the 200's");
else
      System.out.println("In the 200's");
```

II.

```
if ( x < 200) {
      if ( x > 300)
            System.out.println("In the 200's");
      else
            System.out.println("Not in the 200's");
} else
      System.out.println("Not in the 200's);
```

III.

```
if( x >= 200 )
      System.out.println("In the 200's");
else if ( x <= 299 )
      System.out.println("In the 200's");
else
      System.out.println("Not in the 200's");
```

a. **I only → This takes the original code and makes it 3 cases instead of 2, but handles them all appropriately**
b. II only → This will print ' not in the 200s' for all numbers
c. III only → This will print 'in the 200s' for every number
d. I and II → II is incorrect, see B
e. II and III → Both II and III are incorrect

7. To test if a grade is not a C (not between 70 and 79 inclusive) you would do:

if (g __ 70 __ g __ 79)

a. >=, &&, <=
b. >=, ||, <=
c. >, ||, <
d. <, &&, >

e. <, ||, > This uses the correct comparisons (<70, >79) and joins them correctly with OR

8. Assume that x and y are properly initialized boolean values. Which option best describes the result?

```
(x || y) && ! (x || y)
```

a. Always true
b. Always false → The first half is true whenever either x and y is true and the second half is true only when both are false. Because this is an AND statement and it is never possible to have both halves be true, it is always false
c. true only when x is true and y is true
d. true only when x and y have the same value
e. true only when x and y have different values

9. Assume that x and y are properly initialized boolean variables. Which option best describes the result?

```
!(x && y) || (x && y)
```

a. Always true → The second half is true when both x and y are true and the first half is true whenever one or both of them is false. Because this is an OR statement, it is always true
b. Always false
c. true only when x is true and y is true
d. true only when x and y have the same value
e. true only when x and y have different values

10. `!( x >= y || w == z )` Simplifies to:

a. `x <= y && w == z`
b. `x >= y || w != z`
c. `x <= y || w != z`
d. `x <= y && w != z`
e. `x < y && w != z` → The statement inside the parentheses is an Or, so it's only false when both of its halves are false, which is when x<y AND w!=z

11. What is output to the screen by the following code?

```
int c = 0;
while( c < 6) {
      c++;
      System.out.print((int)Math.pow(-1, c) + " ");
}
```

a. -1 1 -1 1 -1 1 -1 → The loop will only increment 6 times
b. 1 -1 1 -1 1 -1 → The loop increments before it prints, so it will start with an odd power
c. -1 1 -1 1 -1 1 → The loop increments before the print statement, so it will have an odd power first, and iterate 6 times
d. 1 1 1 1 1 1 → We are using Math.pow(-1,c) which returns -1^c, meaning that on odd values of c, the result will be negative
e. -1 -1 -1 -1 -1 -1 → We are using Math.pow(-1,c) which returns -1^c, meaning that on even values of c, the result will be positive

12. How many times will the following loop run?

```
int num = 49;
while(num > 0) {
      if( num%9 == 0 )
            num = num + 3;
      else
            num -= 4;
}
```

a. 20 → The loop will execute on num=49 before the subtractions begin
b. 21 → See trace below
c. 22 → The value of num will be 0 at the end of the 21st loop, so a 22nd interation will not be performed
d. 23 → See trace below
e. Infinite loop → The value of num decreases by 12 in between times when it increases by 3, so the loop will terminate

Note: the iterations will look like this: 49, 45, 48, 44, 40, 36, 39, 35, 31, 27, 30, 26, 22, 18, 21, 17, 13, 9, 12, 8, 4

13. What is output to the screen by the following code?

```
int num = 1987;
while( num > 0 ) {
      System.out.print(num%10 + " ");
      num = num/10;
}
```

a. 8 9 1 0 → This forgets about the 7 being printed first
b. 198 19 1 0 → This would be the result if it printed the results of the division instead of the results of the mod operation
c. 19 1 0 0 → Incorrect
d. 7 8 9 1 → The code prints the digits in reverse order by moding by 10, then dividing by 10 until it divides 1 by 10, which truncates to zero
e. The loop will not terminate → The loop will terminate because in Java 1/10=0

14. What is output to the screen by the following code?

```
int f = 0;
while( f < 8 ) {
      f++;
      System.out.print( f%3 + " " );
}
```

a. 2 0 1 2 0 1 2 → f starts as 0 and increments within the loop, so will have 8 iterations
b. 0 1 2 0 1 2 → f increments before the print statement, so it will be 1 at the first output
c. 1 2 0 1 2 0 1 2 → This has the correct number of iterations and the correct pattern
d. 1 2 0 1 2 0 1 2 0 → f increments in the loop, so it will be 8 at the end of the 8th loop and will not iterate a 9th time
e. 2 0 1 2 0 1 2 0 → This is the correct number of iterations but f is 1 at the first print statement and not 2

15. What is output to the screen by the following code?

```
System.out.println("The answer is: " + 5 + 19);
```

a. The answer is: 519 → The system will read these literal ints and concatenate them with the String output
b. The answer is: 19 → The 5 will also be printed
c. The answer is: 24 → The '5 + 19' is not in parentheses so the '+' is interpreted as concatenation, not addition
d. The answer is: 5 19 → There is no space between them
e. Error – Strings cannot do calculations. → Not true, this will print both numbers, and if they were in the parentheses it would print 24

16. What is output to the screen by the following code?

```
System.out.println( Math.sqrt(26));
```

a. 5
b. 5.0
c. 5.09901951359278 → Math.sqrt returns a double which is as close to the actual square root as possible
d. 6
e. Error – Possible loss of precision

17. Does the following code need a cast? If so, what should you type to cast?

```
double val = 13;
```

a. no, none → ints are valid double values and are smaller data type, so 'autoboxing' occurs. If you printed val you would get 13.0
b. yes, (decimal) → Decimal is not a primitive type
c. yes, (double) → This would not cause an error but it is unecessary
d. yes, (int) → This would cause an error. 13 is already an int, but the system will 'autobox' it into a double
e. yes, (String) → This would cause an error. Strings cannot be stored as doubles (but doubles can be stored as Strings)

18. What are the first and last numbers output by the following code?

```
int count = 4;
while ( count <= 3 ) {
      count++;
      System.out.println(count + " ");
}
```

a. 4 7
b. 4 8
c. 5 7
d. 5 8
e. Nothing is output. → The loop will never execute because count starts at 4 and the loop condition requires count to be at most 3

19. Of the following if statements, which correctly execute exactly two commands when the condition is true and does nothing if it is false?

I.
```
if (y == 99)
      System.out.println("A");
      System.out.println("B);
```

II.
```
if(y == 99)
      System.out.println("A");
      System.out.println("B");
System.out.println("C");
```

III.
```
if(y == 99) {
      System.out.println("A");
      System.out.println("B");
}
```

a. I only → This will cause an error if the statement is correct because there is not a second " after the B
b. II only → This will print C regardless of whether the condition is true
c. III only → If y is 99 it will print A and B, if y is not 99 it will do nothing
d. II and III but not I → II is not correct, See B
e. I and III but not II → I is not correct, see A

20. Consider the following code segment:

```
int c = 1;
while(c <= 35) {
      c++;
      if(c%5 == 0)
            System.out.print(c + " ");
}
```

Which of the following produces the exact same output?

I.
```
int c = 1;
while(c <= 35) {
      c++;
      if( c%5 == 4)
            System.out.print( (c + 1) + " ");
}
```

II.
```
int c = 0;
while(c <= 35) {
      c += 5;
      System.out.print(c + " ");
}
```


III.

```
int c = 0;
while(c < 35) {
      c += 5;
      System.out.print(c + " ");
}
```

a. I only → this is essentially the same as the code above, except that when c = 4 it prints 4+1, etc. c will be 4 when the first print happens, 34 when the last print happens, and 35 at the end of the last loop
b. II only → because c increments before the print statement and the loop uses <= this will also print 40
c. III only → c increments inside the loop, so it will have a value of 5 before the first print statement and of 35 at the end of the last loop
d. **I and III only → These are the correct answers**
e. I, II and III → II is not correct, see B

