COMP-202 Midterm exam

1 Multiple Choice

Question 1.1 The Java compiler translates Java source code into a special representation called

a) bitcode
b) bytecode
c) Java hex code
d) machine code
e) none of the above

Solution b)

Question 1.2 343 in decimal (base 10) is equivalent to

a) 101010111 in binary (base 2)
b) 101001011 in binary (base 2)
c) 101010101 in binary (base 2)
d) 101011111 in binary (base 2)
e) None of the above

Solution a)

Explanation

$$\begin{aligned} decimal(101010111) &= 1 \cdot 2^8 + 0 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 \\ &= 256 + 64 + 16 + 4 + 2 + 1 \\ &= 343 \end{aligned}$$

Question 1.3 Programs developed by you in Assignments 1-3 typically compiled (give the most appropriate answer):

a) in approximately one second or less
b) in about fifteen to twenty seconds
c) approximately a minute
d) in an amount of time that varied considerably from at least a second to half a minute.
e) in several seconds and inversely proportional to the number of compilation errors

Question 1.4 What will be the values of `p`, `q`, `r` and `s` after executing the following program fragment?

```
int p = 4,q = 7,r,s;
r = p;
p = q;
s = p;
r = r - 1;
p = q + 1;
q = q - 1;
```

a) 4, 7, 4, 7
b) 4, 7, 4, 4
c) 8, 6, 3, 7
d) 6, 6, 3, 5
e) 8, 6, 7, 8

Solution c)

Explanation Trace the values of the variables. The table below shows the values of the variables after executing the corresponding statement:

After statement	p	q	r	s
`int p = 4,q = 7,r,s;`	4	7	-	-
`r = p;`	4	7	4	-
`p = q;`	7	7	4	-
`s = p;`	7	7	4	7
`r = r - 1;`	7	7	3	7
`p = q + 1;`	8	7	3	7
`q = q - 1;`	8	6	3	7

Question 1.5 Consider the following outline of a nested `if-else` structure which has more `if` clauses than `else` clauses. Which of the statements below is true regarding this structure?

```
if (condition1)
     if (condition2)
                statement1;
else  statement2;
```

a) syntactically it is invalid to have more `if` clauses than `else` clauses
b) `statement2` will only execute if `condition1` is false and `condition2` is false
c) `statement2` will only execute if `condition1` is true and `condition2` is false
d) `statement2` will only execute if `condition1` is false, it does not matter what `condition2` is
e) `statement2` will never execute

Solution c)

Explanation When an `if-else` statement does not use the brackets `{` and `}`, the `else` clause corresponds to the closest `if`, regardless of indentation. Hence the program above is the same as:

```
if (condition1) {
    if (condition2) {
        statement1;
    }
    else {
        statement2;
    }
}
```

Hence, the only way `statement2` is executed is if `condition1` is true and `condition2` is false.

Question 1.6 What will be the value of i after executing this code?

```
boolean x = false, y = true;
int i = 0;
if (x) {
    i = 1;
    x = false;
}
else {
    i = 2;
    if (y) {
        y = false;
        i++;
    }
    else {
        i = i + 2;
    }
}
```

a) 0
b) 1
c) 2
d) 3
e) 4

Solution d)

Explanation Since `x` is initialized to false, the first condition is not satisfied and therefore the `else` clause is executed. This means that `i` is assigned the value 2. Then the condition `y` is tested, and this is true, so the statements `y = false; i++;` are executed, and the `else` clause is not. This means that `i` is assigned the value 3.

Question 1.7 What will be the value of s after executing the following program fragment?

```
int s = 1, k = 13;
while (k > 1) {
   if (k % 3 == 0) {
      s = s + k;
   }
   k--;
}
```

a) 1
b) 13
c) 3
d) 32
e) 31

Solution e)

Explanation Trace the execution of the program. The following table has the values of the variables (as well as the loop's condition and the expression in the conditional,) at thebeginning of each iteration:

Iteration	k	s	k>1	k%3==0
0 (before the loop)	13	1	true	false
1	12	1	true	true
2	11	13	true	false
3	10	13	true	false
4	9	13	true	true
5	8	22	true	false
6	7	22	true	false
7	6	22	true	true
8	5	28	true	false
9	4	28	true	false
10	3	28	true	true
11	2	31	true	false
12	1	31	false	

What this loop is doing is adding the natural numbers less of equal to 13 which are divisible by 3 (proceeding backwards,) this is, 12+9+6+3 = 30, and adding 1 to this result (since s starts as 1.)

Question 1.8 What will be the value of c in terms of the input in this program fragment?

```
int a,b,c,p;
a = Keyboard.readInt();
b = Keyboard.readInt();
c = 0;
p = 1;
while (p < b) {
   p = p * a;
   c++;
}
System.out.println(c);
```

a) c is a^b (a to the power b)
b) c is b^a (b to the power a)
c) c is $floor(log_a b)$ (the largest integer smaller than the logarithm of b to the base a)
d) c is $floor(log_b a)$ (the largest integer smaller than the logarithm of a to the base b)
e) c is $ceil(log_a b)$ (the smallest integer larger than the logarithm of b to the base a)

Note: the logarithm of x to the base y is the number z such that y to the power z is x.

Solution e)

Explanation The variable c starts being 0 and each iteration is incremented by 1, so its value is equal to the number of times that the loop is executed. The loop stops whenever $p \geq b$. The variable p starts being 1, and in each iteration it is multiplied by a, so p is some power of a, namely $p = a^c$:

a	b	c	p
any	any	0	1
		1	$1 \cdot a = a$
		2	$a \cdot a = a^2$
		3	$a^2 \cdot a = a^3$
		4	$a^3 \cdot a = a^4$
		5	$a^4 \cdot a = a^5$
		...	...
		n	$a^{n-1} \cdot a = a^n$

Hence, $p = a^c$, and saying that the loop stops when $p \geq b$ is the same as saying that it stops when $a^c \geq b$, which is the same as saying that it stops when $c \geq log_a b$. Hence c is *at least* $log_a b$. Since c is an integer, and the loop stops as soon as $c \geq log_a b$, then c must be the smallest integer greater or equal to the logarithm of b in base a.

Here is a sample run: Suppose that a is 2 and b is 35.

a	b	c	p	p<b
2	35	0	1	true
		1	$1 \cdot 2 = 2$	true
		2	$2 \cdot 2 = 2^2 = 4$	true
		3	$2^2 \cdot 2 = 2^3 = 8$	true
		4	$2^3 \cdot 2 = 2^4 = 16$	true
		5	$2^4 \cdot 2 = 2^5 = 32$	true
		6	$2^5 \cdot 2 = 2^6 = 64$	false

Since we know that $2^5 = 32$ and $2^6 = 64$, which is the same as saying that $log_2 32 = 5$ and $log_2 64 = 6$ then we can conclude that $5 \leq log_2 35 \leq 6$ (because $32 \leq 35 \leq 64$.) And since c ends up being 6, we have that it is indeed a number greater than the logarithm of 35 in base 2, and the smallest.

2 Short Problems

Question 2.1 Using the following `Die` class:

```
public class Die
{
  private int numFaces;   // number of sides on the die
  private int faceValue;  // current value showing on the die
  public Die()
  {
    numFaces = 6;
    faceValue = 1;
  }
  public int roll()
  {
    faceValue = (int)(Math.random() * numFaces) + 1;
    return faceValue;
  }
  public int getFaceValue()
  {
    return faceValue;
  }
}
```

Write a method that will use two `Die`, roll them 100 times, and return the number of times that a total of 7 or 11 is rolled on the two `Die`.

Solution

```
public int roll7or11()
{
    Die d1 = new Die();
    Die d2 = new Die();
    int j, num = 0, sum = 0;
    j = 1;
    while (j <= 100) {
        sum = d1.roll() + d2.roll();
        if (sum == 7 || sum == 11) {
            num++;
        }
        j++;
    }
    return num;
}
```

Question 2.2 A data verification loop is a loop that prints a line asking the user to input a value within a restricted range repeatedly until the user

has input a value within the requested range. Write a data verification loop that inputs an int value x within the range 0 and 100. Assume cs1.Keyboard has been imported.

Solution

```
int x = -1;
while (x < 0 || x > 100) {
    System.out.println("Enter an int value between 0 and 100");
    x = Keyboard.readInt();
}
```

Question 2.3 Does this fragment stop? What will it print out?

```
int j = 9, k = 2;
while (j > k) {
   if ((j - k) % 2 == 1) {
      j++;
   }
   else {
      j = j - 2;
      k = k + 1;
   }
   System.out.println(j + "," + k);
}
```

Solution Yes, it does stop. It will print:

```
10,2
8,3
9,3
7,4
8,4
6,5
7,5
5,6
```

Question 2.4 What does the following program print?

```
class A
{
    void f(int n)
    {
        int i = 0;
        while (i < n) {
            System.out.print(i);
            i++;
```

```
            }
            System.out.println();
        }
    }
    class B
    {
        public static void main(String[] args)
        {
            A obj = new A();
            int i = 1;
            while (i < 6) {
                obj.f(i);
                i++;
            }
        }
    }
```

Solution It will print

```
0
01
012
0123
01234
```

Explanation The method `f` in class `A` prints the digits from `0` up to `n-1`, where `n` is its parameter. The `main` program creates an instance of class `A`, called `obj`, and applies to it the `f` method 5 times. Each time it passes as parameter the value of the local variable `i` (local to the `main` method.) Hence, each iteration of the main loop results in invoking `obj.f(1)`, `obj.f(2)`, `obj.f(3)`, `obj.f(4)`, and `obj.f(5)`, each of which prints each of the lines above.

3 Programming

Question 3.1 A common way to compute the square root of a number is to use the following method: suppose that we want to compute the square root of x. We start by assuming that the square root of x (say y) is 1, and proceed to repeatedly improve the approximation by computing a new value for y, taking the average between the old value of y and x divided by the old value of y. For example if x is 2, the computation proceeds as follows:

Guess #	Guess (y)	Quotient (x/y)	Average ((y+x/y)/2)
1	1	2/1=2	(2+1)/2=1.5
2	1.5	2/1.5=1.333...	(1.5+1.333...)/2=1.4167...
3	1.4167...	2/1.4167...=1.4118...	(1.4167...+1.4118...)/2=1.4142...
4	1.4142...		

The exact solution is reached then the square of the guess is exactly the same as x, but for many numbers, this might never be the case and the process can continue ad infinitum. Therefore we need to stop when we have an approximation which we consider "close enough" with respect to some "tolerance" value, usually a very small number. The computed guess is close enough when the absolute difference between x and the square of the guess is less than the tolerance.

Write a program that asks the user to enter a number x and a tolerance epsilon, and prints the square root of x computed using this algorithm. (Note: Do not use the Math.sqrt method or the Math.abs method.)

Solution

```
import cs1.Keyboard;
public class SquareRoots {
  public static void main(String[] args)
  {
    double x, guess, tolerance, diff;
    boolean good_enough;

    System.out.print("Enter a number = ");
    x = Keyboard.readDouble();
    System.out.print("Enter a tolerance = ");
    tolerance = Keyboard.readDouble();

    good_enough = false;
    guess = 1;
    while (!good_enough) {
      guess = (guess + (x / guess)) / 2;
      diff = guess * guess - x;
      if (diff < 0 ) diff = -diff;
      good_enough = (diff <= tolerance);
    }
    System.out.println("Result = "+guess);
  }
}
```

Question 3.2 In this problem you are asked to write a simple class to represent elevators. An elevator has a current floor, a number of floors, a current number of passengers, and a maximum capacity of passengers.

An elevator :

- is constructed by specifying:
  - the total number of floors in the building
  - the maximum elevator capacity
  - that the elevator initially doesn't have any passengers,
  - and that the elevator is initially located on the bottom floor
- can move one floor up if it not on the top floor
- can move one floor down if it is not on the bottom floor
- can accept a certain number of passengers (up to its maximum capacity)
- can drop off a certain number of passengers (no more then it actually has)
- can tell us which floor it's on.

Solution

```
public class Elevator
{
    int currentFloor;
    int numFloors;
    int currentPassengers;
    int maxPassengers;
    public Elevator(int maxFloors, int capacity)
    {
        numFloors = maxFloors;
        maxPassengers = capacity;
        currentFloor = 1;
        currentPassengers = 0;
    }
    public void moveUp()
    {
        if (currentFloor < numFloors)
            currentFloor++;
    }
    public void moveDown()
    {
        if (currentFloor > 0)
            currentFloor--;
    }
    //returns true if it accepted some passengers
    public boolean acceptPassengers(int num)
    {
        boolean result = true;
        int difference = maxPassengers - currentPassengers;
        if ( difference > 0 && num > 0)
```

```
            if (num < difference)
                currentPassengers += num;
            else
                currentPassengers += difference;
        else
            result = false;
        return result;
    }
    //returns true if it dropped off passengers
    public boolean dropOffPassengers(int num)
    {
        boolean result = true;
        if ( num <= currentPassengers && num > 0)
            currentPassengers -= num;
        else
            if ( num > currentPassengers )
                currentPassengers = 0;
            else
                result = false;
        return result;
    }
    public int getCurrentFloor()
    {
        return currentFloor;
    }
}
```