What we have begun, we shall finish

Challenge Questions

Please refer to the Course Outline for guidelines on these questions

Challenge Question 1. Finance (11)

Develop a closed formula for calculating the remaining balance of a mortgage

Level of difficulty: 4/10

Scenario. A mortgage is a loan from a bank to allow the purchase of a property. The borrower pays this loan back to the bank by making regular, constant payments (usually monthly) over a period of time, with certain agreed upon conditions.

For example, a person might borrow $400,000, at 3% yearly interest rate, to be paid back with monthly payments over a period of 25 years (the amortization period). The money returned will necessarily be more than $400,000 since there is an interest that is applied over time. The monthly payment is calculated as follows:

$$Pmt = \frac{P \times i}{12(1-(1+\frac{i}{12})^{-12t})}$$

where *Pmt* is the resulting monthly payment, *P* is the amount borrowed, *i* is the yearly interest rate, and *t* is the amortization period.

Starting with the first month after the money has been borrowed, the bank applies the monthly interest to the amount owing, and thereafter the borrower makes the monthly payment. Because the monthly payment is larger than the monthly interest generated, the money owed by the borrower will decrease over time.

The table on the next page shows a sample of what would happen in the first few months, if someone borrowed $400,000 at a yearly interest rate of 2.99% over a 25 year amortization period:

The first three months of an amortization table

Month	Balance at start of month	Interest	Owing at end of month	Payment
1	400,000.00	996.67	400,996.67	1894.77
2	399,101.90	994.43	400,096.33	1894.77
3	398,201.56	992.18	399,193.74	1894.77

Note that the interest rate applied changes every month because the amount owing changes every month.

Task. Develop a closed formula that calculates the balance at the start of month *m* from a given starting mortgage loan, a yearly interest rate, and a term of repayment. Assume that the first month is 1. Your work must show clear steps that ultimately arrive at the formula.

Challenge Question 2. Computer Graphics (11)

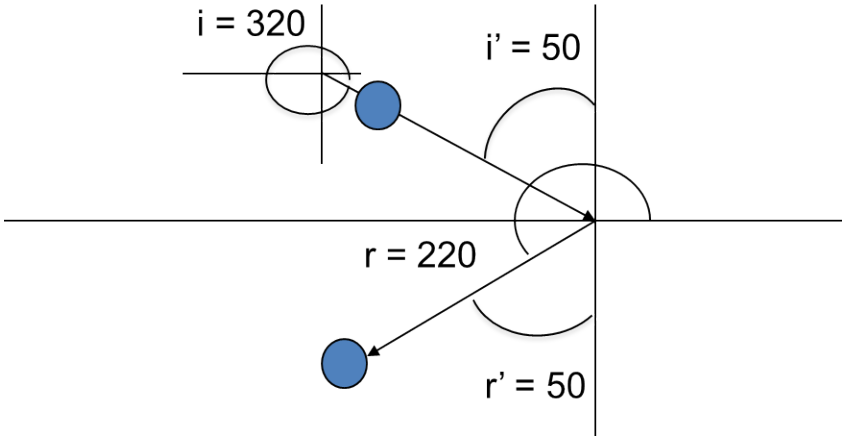
Write an algorithm and program that return the angle of reflection in standard form

Level of difficulty: 4/10

Scenario. When an object such as a ball bounces off a surface, it bounces with an angle of reflection that is equal to the angle of incidence. In the example below a ball is hitting a right wall with an angle of incidence of 50 degrees. The ball bounces off with an angle of reflection that is 50 degrees with respect to the wall.

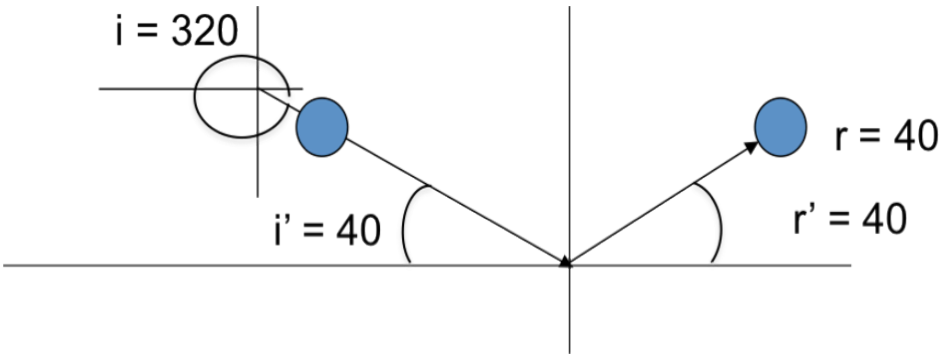
If we wish to simulate a bouncing ball on the computer screen, we need to make graphics realistic. Objects need to bounce on the computer screen in the same way that they bounce in reality. To make the movements of the ball easy to program, it is best to use angles in standard position.

Angles in standard position in an XY plane are measured counterclockwise, starting on the positive x-axis from 0 degrees to 360 degrees. When an object is moving, we place the XY axis system and its origin in such a way so that it is always behind the moving object. Then the angle is measured with respect to that XY system.



In this example the angle of incidence (i') is 50 degrees with respect to the wall or 320 degrees in standard position (i). The angle of reflection (r') is also 50 degrees with respect to the wall or 220 degrees in standard position (r)

However, when an object moves in the computer screen, in most cases it will be moving towards two possible surfaces for every angle of incidence. In the example above, the ball is moving at an angle of 320 degrees towards the right wall. Another possible scenario is that it moves at an angle of 320 degrees towards the floor. The angle of reflection will be different depending on the surface on which the object bounces. Here is what would happen if it bounced off the floor:



The angle of reflection in standard position in this case would be 40 degrees.

Task. Develop an algorithm, and then write a function (C, C++, or Turing) or method (Java) that implements the algorithm. The algorithm takes two pieces of information:

1. The angle of incidence of the moving object in standard position
2. An integer that specifies the surface where the object is bouncing. Use the following values: 0 for the floor, 1 for the right wall, 2 for the ceiling, 3 for the left wall

The function or method needs to return the angle of reflection in standard position. All calculations must be done by your algorithm/code. You may not use any predefined graphics libraries that perform the calculations.

All that is required from the function/method is a number: The angle of reflection in standard position. No printing or drawing is required.

Challenge Question 3. System's Programming (11)

Implement a natural number equality operator using only bitwise operations

Level of difficulty: 5/10

Scenario. The operations that we perform on a computer or a calculator are highly abstract. We are able to ask for the sum or the multiplication or the square root of numbers, but these operations are made of smaller blocks at their foundation. In this question we explore how equality between two natural numbers is implemented.

Task. Write a Java method with the signature

```
public static int equal(int x, int y)
```

that returns

0 if x is equal to y
1 if x is not equal to y

For simplicity, x and y are assumed to be Natural numbers (positive integers) in the range [0, 4095]

Because our tools are very primitive, we cannot:

1. Use any arithmetic operators or the equality operator ==
2. Change the parameters x and y nor assign them to any other types, nor perform any casting
3. Change the method signature
4. Use any equality predefined Java methods
5. Introduce any new variables in the method

Bitwise operations are the only tools that we have available.

Challenge Question 4. Clock synchronization (11)

Preventing a clock from accumulating time error

Level of difficulty: 4/10

Scenario. When simulating a stop watch or a clock in the computer with the use of loops, we need to slow down the display of time with a delay process. The simplest way is to delay the computer by one second every time the display needs to be updated, for example from 00:00:12 to 00:00:13. However, since any instruction takes itself time to execute, this mechanism will accumulate all the small fractions of time that it takes to execute instructions, and over time the clock will be off. For example, in this approach

```
loop
     display stop watch
     delay for one second
end loop
```

extra unwanted time will be accumulated after the delay in the instruction `end loop` and then in the instruction `loop`. This will happen every time around the loop. Furthermore, any other code that we include in the loop will also take time. Eventually the instruction `display stop watch` will be lagging behind and the time will be incorrect.

Task. Write a Java program that implements an infinite digital stop watch, displaying hours, minutes and seconds in the standard format `hh:mm:ss`, without accumulating error. We have the following guidelines for this problem:

1. We do not have access to methods in the `Calendar` class except for `Thread.sleep()`. The only available time tools are

   `System.currentTimeMillis()` and/or
   `Thread.sleep()`

2. We cannot make assumptions about the speed of the computer. This program could potentially run on a machine that takes considerable time to execute instructions.

3. The stop watch does not necessarily have to display every second in sequence. Amounts of time may be skipped in order to readjust the clock so that time error is not accumulated. It is possible that the display of the clock itself may be slightly off, since we have no control over the speed of the screen. But it's the *accumulation* of error that we wish to prevent from happening.

Challenge Question 5. Encryption (11)

The Enigma encoder prevents a letter from encrypting to itself

Level of difficulty: 3/10

Scenario. The Enigma machine was initially designed for encrypting financial information that needed to be transmitted via telegraph and radio signals at the beginning of the 1900's. Enigma had a series of improvements made to it after Polish mathematicians had been able to crack its coding mechanism during the 1920's. After these improvements, Enigma became practically impossible to crack. Designers believed that preventing a character from encrypting to itself would make Enigma harder to crack. This modification turned out to be a small glitch in the system which helped Alan Turing in breaking its mechanism.

In the Enigma design of 1938, which is the machine that was used the most during World War II, a plaintext character is encrypted once through the *plugboard*, then encrypted three times through *three rotors*, encrypted once again and reflected back by the *reflector* into the rotors which perform yet another three encryptions.

The following are very helpful in understanding the mechanism of Enigma.

1. From the Perimeter Institute for Theoretical Physics

   https://www.youtube.com/watch?v=mcX7iO_XCFA

2. Details of Enigma Rotors

   https://en.wikipedia.org/wiki/Enigma_rotor_details

3. How Enigma Works (includes an Enigma emulator)

   http://enigma.louisedade.co.uk/howitworks.html

4. Technical details of Enigma

   http://users.telenet.be/d.rijmenants/en/enigmatech.htm

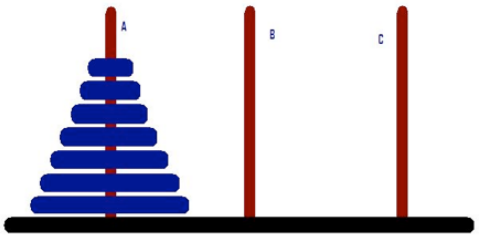
Task. Explain why the reflector guarantees that a letter does not encrypt to itself and why this is a flaw in the system rather than an aid to more secure encryption.

Challenge Question 6. Recursion (11, 12)

Computational intractability of a game

Level of difficulty: 4/10

Scenario. The legend of the Towers of Hanoi speaks of a group of Eastern monks whose meditation consists of moving 64 disks from peg A to peg C as shown below. The disks must be moved one at a time, using peg B as an intermediary step when necessary, with no larger disk ever on top of a smaller disk. The legend claims that the world will come to an end once all 64 disks have been moved:



Task. The program below uses a method recursively to solve the Towers of Hanoi

```
import hsa.*;
public class TowersOfHanoi {
   public static void solve(int n, String start, String aux, String end) {
       if (n == 1) {
           System.out.println(start + " -> " + end);
       } else {
           solve(n - 1, start, end, aux);
           System.out.println(start + " -> " + end);
           solve(n - 1, aux, start, end);
       }
   }

   public static void main(String[] args) {
       System.out.print("Enter number of discs: ");
       int discs = Stdin.readInt();
       solve(discs, "A", "B", "C");
   }
}
```

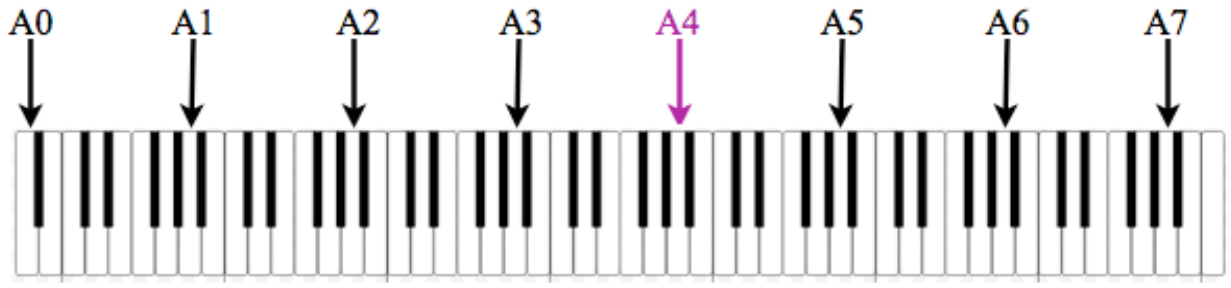
Perform an accurate and detailed trace of the program with an input of 4 disks, showing the values of all variables, parameters, and state of the stack as method calls are made. Then, develop an expression that determines the number of moves necessary to move *n* disks. Finally, explain why this problem is intractable as defined in computational complexity and estimate the time it would take for a computer to complete the solution with 64 disks if the computer could make one million moves per second.

Challenge Question 7. Physics of Sound (11)

Tuning systems in Western Music

Level of difficulty: 3/10

Scenario. The musical note that is used to tune an orchestra is A4, with a frequency of 440 Hz. For every octave that we move up the scale, the frequency is multiplied by 2 and for every octave that we move down the scale, the frequency is divided by 2.



27.5Hz 55Hz 110Hz 220Hz 440Hz 880Hz 1760Hz 3520Hz

This modern Western tuning system has been in use since the 1700's and was proven effective in J.S. Bach's collection of 48 preludes and fugues entitled *The Well Tempered Klavier*. However, prior to the 1700's and for over 2,000 years, the irregular tuning system known as *Mean Tone* or *Pythagorean* was used for music composition and is now becoming increasingly popular as it seems to resonate more naturally with the human ear. The modern system (*Equal Tempered*) is made up of a geometric sequence of frequencies where the common ratio between any two adjacent notes is $\sqrt[12]{2}$. This neat ratio between notes does not exist in *Mean Tone*.

Task. Write a Java program that asks the user for any fundamental frequency such as 440 Hz and, using this frequency as the first note of a scale, the program determines and outputs all the frequency values for a *Major* scale in the irregular *Mean Tone System.*

Challenge Question 8. Combinatorics (11, 12)

Generating all possible permutations

Level of difficulty: 4/10

Scenario. A jar contains *n* number of different coloured balls. We are interested in generating all the possible permutations that can be generated by choosing sets of balls from the jar. Each set can be of any size from 1 to *n*. For example, if the set contains 1 red ball, 1 blue ball and 1 yellow ball, then all the possible permutations that can be generated are:

R		B		Y			
RB	RY	BR	BY	YR	YB		
RBY	RYB	BRY	BYR	YRB	YBR		

Task. Write a Java program that asks the user for a value of *n* where $1 \le n \le 26$. Each value of *n* can then be represented by a letter of the alphabet. The program generates (by way of printing) all the possible permutations as explained above. We may not use built-in Java methods that generate these permutations, but must be done with primitive constructs. Use of recursion is recommended.

Challenge Question 9. Matrices (11)

Calculating the determinant of a matrix

Level of difficulty: 4/10

Task. Write a Java program that calculates the determinant of an *n by n* matrix. The matrix needs to be represented using a two dimensional array. We may not use built-in Java methods but compute the determinant by using basic constructs, our own methods and such. Recursion is recommended for this problem.

Challenge Question 10. Computability Theory (11, 12, and beyond)

Proof of an algorithm coming to an end: The Collatz Conjecture

Level of difficulty: 10/10

Scenario. Alan Turing proved that it is impossible to write a computer program that will determine whether or not another computer program's execution finishes. In doing so he extended on the uncertainty that Kantor introduced into Mathematics. Where mathematicians and scientists devoted their lives to the pursuit of certainty, Kantor made many people very nervous by saying that Mathematics is not as clear cut as they believed, and that there are problems that cannot be solved by Mathematics.

The *Halting Problem* asks whether or not an algorithm can be developed to determine if a computer program finishes running or continues to run forever. Turing proved that it is impossible to write such an algorithm.

In this question we are going to attempt to prove whether a particular program ends for a particular set of data.

Task. Consider the following program (written in pseudo-Turing):

```
let x be any natural number larger than 1
loop
    exit when x = 1

    if x is even then
        x := x / 2
    else
        x := 3x + 1
    end if
end loop
```

Prove or disprove that this program always ends for any natural number x, $x > 1$.

Due to the difficulty of this question, serious attempts at deducing possible reasons why the program ends (or not) will be accepted even if the problem is not solved.

Challenge Question 11. Graph Theory (12)

Find the minimum cost to travel from one city to another

Level of difficulty: 6/10

Scenario. A new airline is offering low rates to travel to major cities around the world. Below is the table of the one way flights offered along with their rates. Traveling from city A to city B costs the same as traveling from city B to city A.

	Boston	Paris	Tokyo	London	New York	Toronto	Moscow	Beijing	Manila
Boston	-	225	575	200	70	85	340	675	975
Paris	225	-	225	50	125	220	120	435	550
Tokyo	575	225	-	510	650	675	735	225	200
London	200	50	510	-	-	-	-	-	-
New York	70	125	650	-	-	-	-	-	-
Toronto	85	220	675	-	-	-	-	-	-
Moscow	340	120	735	-	-	-	-	-	-
Beijing	675	435	225	-	-	-	-	-	-
Manila	975	550	200	-	-	-	-	-	-

Because this is a new airline, there are many trips that are not yet offered as direct flights. Stop overs are necessary. For example, there is no direct flight from Moscow to New York, but it is still possible to accomplish the trip by traveling from Moscow to Boston and from Boston to New York.

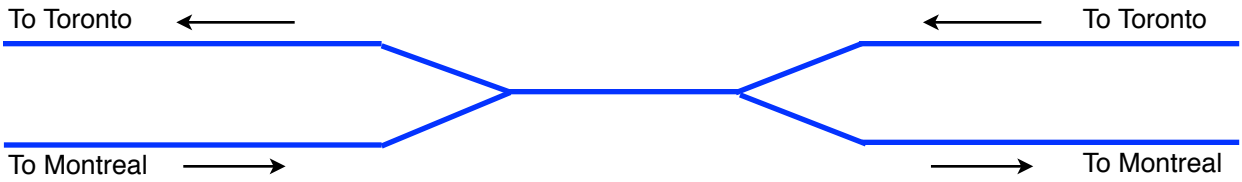
Task. Write a Java program that determines the cheapest way to travel between any two given cities. The program must work for *any given table of routes*. The data may be hard coded in the declaration parts of the program and this portion of the program may be modified to enter a new table, but the algorithm cannot be modified once implemented in Java. The program must run with upper limit complexity $O(n^2)$, where n is the number of cities. The presentation must also include a trace of the program with a smaller sample that contains only five cities.

Challenge Question 12. Processes (12)

Preventing two or more processes from colliding with each other

Level of difficulty: 5/10

Scenario. Portions of tracks between Toronto and Montreal are shared by trains that travel in opposite directions. This is done to save in construction costs. A portion of the journey can look like this:



The shared portion of the track must be monitored so that at most one train is using it at any given time. If the shared track is in use by a train, then any other train that attempts to enter the shared track must be immediately stopped and kept waiting until the shared track becomes free, at which point this latter train may proceed into the shared track. For simplicity, no two trains can be in the shared track even if they travel in the same direction.

Task. Write a Java program that simulates this situation with two trains traveling in opposite directions. The program must use concurrent processes to simulate the trains. The program must implement an explicit binary semaphore. Java mechanisms can be used to guarantee indivisible instructions. As the program runs, simple text output must be provided that periodically displays where each train is.

Challenge Question 13. Search Algorithms (12)

Increasing data volume significantly while increasing search time insignificantly

Level of difficulty: 7/10

Scenario. A company keeps inventory information on file. At the present moment the company has twenty thousand items and the current computer system is able to retrieve the information quickly enough to keep the users happy. When a user requests an item to be retrieved, it takes the system 0.002 milliseconds to locate the item in the file. Since the system is on-line, there are normally many users retrieving items. In the worst-case scenario, there could be one thousand users in a queue requesting items. Because of this, a user could potentially wait up to 2 seconds in order to receive the information.

The company has requested the Human Resources department to do a study to determine how long a user would be willing to wait for an item to be retrieved, possibly over a period of a workday, without becoming terribly frustrated. Human Resources has done an extensive survey and they have estimated that a user will tolerate a maximum wait time of 2.5 seconds.

The concern is that the company is about to acquire another company and the inventory file will become 40 times larger. This means that the file will grow from the original 20,000 items to 800,000 items. An added complication is that the new file will keep growing because items will be constantly added. Therefore the file cannot be kept sorted because sorting would slow the system intolerably. Off-line sorting is not possible because by the time the file is sorted off-line, the original file may have changed as new items are added. The file needs to maintain information in real time.

The company executives are terribly worried that the wait time is going to increase drastically and that the current computer system will not be able to handle the increase. The executives are also worried because they are being told by the computer sellers that a new computer system is needed, one that will need to be 40 times faster in order to keep up with the demand of the users. They essentially believe that they need to spend the money for the new system but this new system runs into the hundreds of thousands of dollars. Furthermore, they worry that as the inventory file grows they will need to be getting more and more expensive systems.

Task. You claim that you have a solution that allows for the drastic file size increase but with an insignificant increase in search time, i.e., you claim that the complexity of the search time can remain constant. You would be presenting your solution to the executives and telling them that their present computer system can handle the new file size, that *it is not necessary to buy a new system*, that users would retrieve information well within the 2.5 second limit, and furthermore, that the present system can handle even much larger file increases. In presenting this solution you would be saving the company a lot of money, and thus they will pay you handsomely for your solution. Your task is to present such a solution.