CS 4410: Operating Systems
Homework 1

Instructions for Homework 1:

- This is the first "k out of n" homeworks CS 4410.
- The homework may be done in pairs, or individually. If doing in pairs, you need to submit only one copy of the solutions on Gradescope. Please note down the name of your partner on the solutions.
- The deadline is Tuesday, the 20th of September at 11:59AM.
- No late submissions will be accepted.
- Expected time to complete this homework is less than 2 hours.
- You must attribute every source used to complete this homework.
- For each of the pseudo-codes in the homework, assume that each line of the pseudo-code is executed "atomically", that is, cannot be interleaved by any other operation.
- For some of the problems, you will need two integers. Here is the algorithm for computing these integers:
  - If you are working with a partner, let `var` be the lexicographically smaller of the two NetIDs.
  - Let `varInt` be the integral part of `var`. That is, if `var = rst12`, then `varInt = 12`.
  - If `varInt` is a single digit integer, let `varInt = 13 × varInt`
  - Let `Int1` be the first digit of `varInt`
  - Let `Int2` be the second digit of `varInt`

1 TA Game (3 marks)

At last, Aurora invited Ted to enjoy the beautiful view and breeze on Libe Slope. After sitting down, Aurora suddenly took out two decks of cards. "Let's play a card game, Ted!", said Aurora. "Sounds good, what is it?" Aurora showed Ted two deck of cards especially designed for this game. Each card in Ted's deck had the letter "T" on it and each card in Aurora's deck had the letter "A" on it. Aurora told Ted that there were so many of these cards in each deck that he could assume infinite number of them.

Then she explained the rules of the game. Actually, the rules turned out to be quite simple and straight forward: there is a designated area on picnic cloth where cards can be placed on top of each other. The area is initially empty. Each player, regardless of the other's action, does the following:

1. At first, unconditionally put one of his/her card onto the top of the pile.
2. Look at the card that is currently at the top of the pile and put one of his/her card on top *if and only if* the top card has a different initial from his/her own cards (put "T" only if there is an "A" card on the top and vice versa).

The "state" of the pile is defined as the list of cards in the pile when read from bottom of the pile to the top of the pile. For instance, if Ted puts a card first and then Aurora puts the card, then the state of the pile is "TA".

Ted laughed, "this sounds like a boring game!". He suggested there would only be two possible states — "TATA" or "ATAT". Aurora said "No, this game is simple but *not stupid* if the last step can be repeated, and two people *do not* necessarily put cards in turn. Think about it carefully", she twinkled with a mysterious smile. "Well, then…" Ted, being a strategist, wanted to figure out all the possible states before starting to play the game. He decided to reach out to one of his most nerdy friends — You — hoping that you could help him figure out all possible states. He knew you were a good programmer, so he decided to formulate the problem into some pseudo-code:

Algorithm 1 TA Game rule: same for each participant who individually may execute at different speed

```
▷ Player i (could be "A" or "T") play the game for k rounds.
function PLAY(i, k)
    Put one card with initial i onto the top of the pile
    for each round of k do
        Examine the top the of pile
        if the top card does not have initial i then
            Put a card with initial i onto the top of the pile
        else
            Do nothing
        end if
    end for
end function
```

The game starts by initiating two running instances of Play function by calling Play(`A`) and Play(`T`).

1. For $k = 1$, what are all possible outcomes of the final state of the pile (list all possible strings).
2. For $k = 2$, what are all possible outcomes of the final state of the pile (list all possible strings).
3. For $k = n$ (some fixed integer), what is the total number of possible states (write a closed-form expression in terms of n)?

2 Aurora's Addiction (14 marks)

Aurora loves mathematics. She is particular fan of the "modulus" operator. At one point, she got so addicted with the operator that she asked her best friend, Ted, to build a super-fast machine for calculating the addition of a positive integer x and the modulus of another integer δ:

$$y = x + |\delta|$$

Now, it is Ted's show time. He builds a fancy machine to satisfy Aurora's addiction. The machine runs into a main loop of function MainLoop() after booting up. After that, the function waits for a notification which is sent by another function Add(a, d). As a user, Aurora can call this function whenever she wants to satisfy her addiction. Both of two functions share the exact same set of global variables: x, y, and δ.

Algorithm 2 Ted's super-fast machine: calculates Aurora's addition for her addiction

```
function MAINLOOP
    Let the global variable x = 0, y = 0, δ = −6
    while True do
        Wait for a notification from Add(a, d)
        if δ < 0 then
            y = x − δ
        else
            y = x + δ
        end if
        Print the value of y
    end while
end function

function ADD(a, d)
    Send a notification to the running MainLoop
    x = a
    δ = d
end function
```

1. Now, suppose the machine is started by executing MainLoop(). After entering the while loop and reaching the line of waiting for a notification, Aurora called Add($a1, d1$), where $a1$ is `Int1` and $d1$ is `Int2`. Write down all possible values of y that could be printed by function MainLoop().

2. Now suppose Aurora is feeling so addicted that as soon as she sees the printed value from Part 1 above, she immediately calls the function Add(a2, d2) again to notify the MainLoop(). But this time, she uses `Int1` as $a2$ and uses the negation of `Int2` as $d2$ (*e.g.*, if `Int2 = 2`, then $d2 = -2$). Write down all possible values of y that could be printed by function MainLoop().

3 To Be or Not To Be There (3 marks)

Believe it or not, there was a time in human civilization when people did not use phones or emails. Alice and Bob are a lovely couple from that civilization. Both Bob and Alice are very successful and busy, but fun people. Naturally, they get invited to a lot of parties (they have the same set of friends, and thus get invitations from exactly the same set of people). Given that they only get to see each other after work, they try to attend the same party among many possible parties that they get invited to. To choose the most fun party, they wait until the morning of the party day to decide which party to go to.

However, Bob and Alice have different schedules. Since they are both busy during the day, they can only tell each other which party they are going to by writing on a whiteboard before they go to work. If Alice decides and writes on the whiteboard first, Bob will agree to go to whichever party she decided on and vise versa.

Consider each person behaves as described in the following pseudo-code:

Algorithm 3 To Be or Not To Be There: simple case

```
function GoParty(i)
    if whiteboard = ∅ then
        p[i] = select_party(i)
        whiteboard = p[i]
    else
        p[i] = whiteboard
    end if
    go_to_party(p[i])
end function
```

Suppose both GoParty(A) and GoParty(B) ("A" for Alice and "B" for Bob) are executed simultaneously. Suppose select_party(i) returns `Int1` for Alice and `Int2` for Bob. For each of the three questions below, write down your answer as pairs of integers (a, b), where a denotes the party Alice finally goes to and b denotes the party that Bob finally goes to:

1. Sequential case, where Alice executes to completion before Bob actually gets to the first line. Which parties do they go in the end?

2. Interleaved case, where Alice executes the first line, then they execute following lines in turn.

3. Interleaved case, where Alice executes up till select_party(i)), then Bob executes the entire function, and then Alice continues to execute the remaining part of the function.

Now consider a moderately complex version in algorithm 4 given by Emmanuel.

Algorithm 4 To Be or Not To Be There: moderately complex case

```
function GoParty(i = Alice)
    alice_busy = True
    while bob_busy = True do
        Do nothing
    end while
    if whiteboard = ∅ then
        p[i] = select_party(i)
        whiteboard = p[i]
    else
        p[i] = whiteboard
    end if
    alice_busy = False
    go_to_party(p[i])
end function
```

```
function GoParty(i = Bob)
    bob_busy = True
    while alice_busy = True do
        Do nothing
    end while
    if whiteboard = ∅ then
        p[i] = select_party(i)
        whiteboard = p[i]
    else
        p[i] = whiteboard
    end if
    bob_busy = False
    go_to_party(p[i])
end function
```

Suppose again that both GoParty(A) and GoParty(B) ("A" for Alice and "B" for Bob) are executed simultaneously. Also, suppose select_party(i) returns `Int1` for Alice and `Int2` for Bob. For each of the three questions below, write down your answer as pairs of integers (a, b), where a denotes the party Alice finally goes to and b denotes the party that Bob finally goes to:

1. Sequential case, where Alice executes to completion before Bob starts executing his function. Which parties do they go in the end?

2. Interleaved case, where Alice executes the first line, then they execute following lines in turn.

3. Interleaved case, where Alice executes up till select_party(i)), then Bob executes the entire function, and then Alice continues to execute the remaining part of the function.