1. Fibonacci

In mathematics, the Fibonacci numbers or Fibonacci series or Fibonacci sequence are the numbers in the following integer sequence: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144...

By definition, the first two numbers in the Fibonacci sequence are 0 and 1, and each subsequent number is the sum of the previous two.

Given an integer n, return the nth Fibonacci number.

For example, if n = 3, return 2.

import java.util.Scanner;

public class Solution {

public static int fibo(int number){

if(number == 0) return 0;

if(number == 1) return 1;

return fibo(number - 1) + fibo(number - 2);

}

public static void main(String[] args){

Scanner scan = new Scanner(System.in);

int number = scan.nextInt();

System.out.println(fibo(number));

}

};

codes starts from line 13

line 15 requires an input number

at line 16 calls static fibo method which will print out the result

i will be defining the nth value as 9th because 3rd value feels inefficient for proper definition

at fibo method(line 5), it's designed using recursive call

when designing a recursive method, there are two things to consider

1. what to return

2. when the recursive call stops

to solve for factor 1, we need to consider the value of fibonacci values.. starts from 0 and adds the next and then the next

0 1 1 2 3 5 8 13 21 34 55 and so on

for example, if user wants the 9th fibonacci value which is 34, we need to add fibonacci's 8th value(21) and 7th value(13)

so that's how i solves the 1st factor on line 10, fibo(number - 1) + fibo(number -2)

now to solve for factor 2, when the recursive call stops

if we don't solve for factor 2 properly, recursive call functions as a infinite loop or return a result that's unwanted

on line 10 is same as fibo(8) + fibo(7)

fibo(7) + fibo(6)

fibo(6) + fibo(5)

fibo(5) + fibo(4)

and so on. we need to stop at fibo(0) and fibo(1) or fibo(1) and fibo(2), it really depends on the return value. i decided to go down all the way to fibo(0) and returned 0 and fibo(1) returning 1

the first fibo(number -1) ends up as 0 + 1 + 1 + 2 .. till 21

the second fibo(number -2) ends up as ... 13

and then add two numbers together resulting in 34 as the 9th value of Fibonacci

Write a function that takes a string as input and reverse only the vowels of a string.

Example 1:

Given s = "hello", return "holle".

Example 2:

Given s = "kongstudios", return "kongstiduos".

Note:

The vowels does not include the letter "y".

// Template for C++

class Solution {

public:

string reverseVowels(string s) {

}

};

To be honest, this one I had the help of geeksforgeeks site

starts at line 35, Scanner receives the inputed value and is saved at str line 37

line 38 prints out the value of partialWordReverse with str parameter

the key point of this algorithm lies within two factors

1. String's values must switch from the first value with the last value

2. must check if it's a value

partialWordReverse starts at 10

- need to convert string to char array to compare one char at a time

- the comparing placement could be assigned as 0 for i and length of the string as j. -1 is required for j because array size and placement is different

- performs the while loop while i is less than j

this is because when i has reached j, it has nothing to be compared to

- first if statement compares if str1[i] is a vowel or not. if it's not a vowel i value increments and by using continue, loop starts over from the beginning.

- when first if statement finds a vowel in which is o in kongstuidos, continues for theh second if statement, which compares for the vowel from the back

- when two if statements are passed without a problem, temp variable temporary holds the str1[i] value and i and j values are switched.

- i++ and j-- occurs because those placement values have been switched and this while loop continues till i becomes greater than j

isVowel at 5th line returns a boolean

if the parameter of isVowel is same as a e i o u or A E I O U it retuns true else returns false

Implement the following operations of a queue using stacks.

push(x) -- Push element x to the back of queue.

pop() -- Removes the element from in front of queue.

peek() -- Get the front element.

empty() -- Return whether the queue is empty.

Notes:

You must use only standard operations of a stack -- which means only push to top, peek/pop from top, size, and is empty operations are valid.

Depending on your language, stack may not be supported natively. You may simulate a stack by using a list or deque (double-ended queue), as long as you use only standard operations of a stack.

You may assume that all operations are valid (for example, no pop or peek operations will be called on an empty queue).

// Template for C++

class Queue {

public:

// Push element x to the back of queue.

void push(int x) {

}

// Removes the element from in front of queue.

void pop(void) {

}

// Get the front element.

int peek(void) {

}

// Return whether the queue is empty.

bool empty(void) {

}

};

Queue is fifo while stack is lifo structure.

The problem didn't specifically require me to print anything so did not create a main method.

example : 1 , 2 , 3 , 4 , 5

I managed to solve the situation of having to implement queue using stack by these methods:

For fields

- Two stacks s1 and s2 to for lifo structure

- int first to temporary hold the value

push method

- the parameter x is pushed to s1 using a fifo, having a 5 4 3 2 1 order. since s1 was empty we will save the first pushed value, which is 1(will explain at peek)

- since im using two stacks to solve for queue methods, pushing values in lifo does not cause any errors

pop method

- we have the value 1 2 3 4 5 in lifo order so it's 5 4 3 2 1 currently

- we need to get the value 1 if it's fifo method of the value 1 2 3 4 5

- we can accomplish this by putting the top s1 value which is 5 and popping it and putting the value on to s2 stack.

- first round

s1 = 5 4 3 2 1

s2 = null

pop

s1 = 4 3 2 1

s2 = 5

repeat till s1 is not empty

s1 = 3 2 1

s2 = 4 5(s1 and s2 are stacks, so lifo occurs, having 5 on the bottom while 4 stacks on top)

repeat till

s1 = null

s2 = 1 2 3 4 5

- return the s2(1 2 3 4 5) stack pop value is 1, the value we wanted to get considering if inputed data is 1 2 3 4 5, queue method of pop is getting the first value pushed which is 1.

empty method

- we need to check if s1 and s2 both are empty because if either one of them has a value inside, it means "queue" still has value

peek method

- we need to see the top value of the queue, assuming inputed value is 1 2 3 4 5, which will be 1

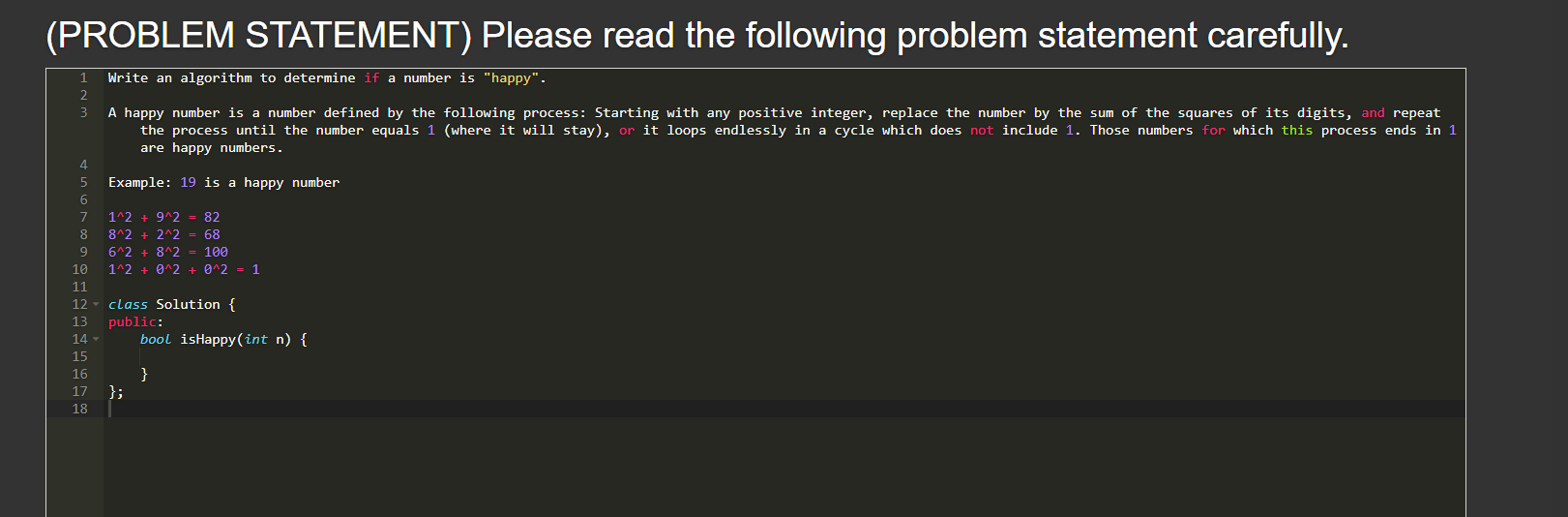
- since we are working with two stacks s1 and s2, we need to do the following

- if s2 is not empty, it means s2 stack is sorted in queue fifo method. so we simply need to use stack's peek which will be 1

- if s2 is empty, it means s1 has the value 5 4 3 2 1 so s1.peek will simply return the value 5 which is not the value we want to get. so this is where the front integer comes in handy. we stated if s1 is empty, we save the value to front variable. since it's the first data that has been inputed to the stack, it will act same as peek function.

- if we peek first, which will result 1 and pop and peek again, front does not matter because s2 is not empty, resulting the value 2 if it has been popped.

At first I did not understand the quesiton properly because i have never solved this type of algorithm before. so i looked it up online, source leetcode. although i did get the code from leetcode, I understood the whole process. all the explanation is in my own words and thoughts.

if a number is given, we need to find out if each placeholder of the number's sqaure adds and repeats till it results 1 or goes infinitely.

- if it is a happy number, result is always 1

- if it is not a happy number, it loops infinitely because the same number appears as a result.

starts from line 26, int n stores the number and on line 29 uses isHappyNumber method to see if n is a happy number or not.

isHappyNumber

- we need to find out if the same number appears or not, this is a perfect situation to use the set collection because in which order the value stores is not important as long as the same value is not stored

- first while loop runs as long as set.add holds true

- second while loop runs till n is not 0.

- inner while loop, squareNum adds the value to itself to add the first loop and seoncd loop and if required third loop

- we need to get the first the ones value and this could be accessible by getting the remainder if divided by 10

19 % 10 is 9 and we need to square 9 which results to 81 and adds to squareNum is is 0 + 81 = 81

- n is divided by 10 and since it's int it does not store decimal.

1 is not 0 so while loop repeats and 1 % 10 is 1 and 1 \* 1 is 1 + 81 is 82

- 81 is stored at set and while loop repeats

1^2 + 9^2 = 82

8^2 + 2^2 = 68

6^2 + 8^2 = 100

1^2 + 0^2 + 0^2 = 1

1 is stored and 1 % 10 is 1 \* 1 is 1 and + squareNum is 1;

1 is already stored inside set and while loop finally stops and if n is 1, which is 1 currently so it returns true.

at line 29 it prints happy if it is true and not happy if the result is false.

As I'm writing this down, i realized i wrote my explanation for using hashmap on previous question wrong, it was a description meant for this problem. Hashmap is to store the i value of the number. not because of it's unqiue key value.

Thank you for giving the me the opportunity to take the coding interview because there were parts I did not understand this gave me the opportunity to study parts i did not completely understand, giving more opportunity to grow as a programmer. I typed all the answers in english because the questions were in english. Thank you once again for giving me the opportunity. Hope to here you from soon