Strings

Strings are Immutable

```
String str = "abcd";

String str = new String("abcd");
```

Input / Output:

Syntax:

```
Scanner sc = new Scanner(System.in);
String name, name2;
name = sc.nextLine(); → Takes all in
name = sc.next(); → Takes only one word.
```

String length

```
int length = name.length();
```

* In strings, length is a fn that's why use parenthesis.
* It also counts spaces.

String concatenation (दो चीजों को जोड़ना)

simply add '+' between two strings.

String CharAt() -

gives us the character at that index.

eg: name.charAt(2);

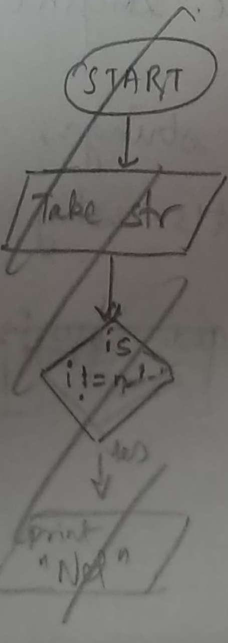
Q) Check palindrome

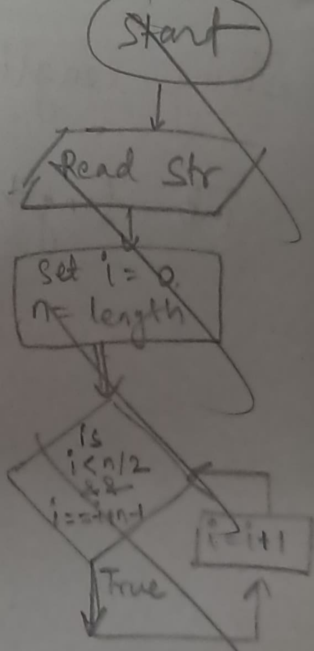
Steps
1) for loop till n/2 , n is length

2) Check char at i != charat n-i-1

3) If yes return false

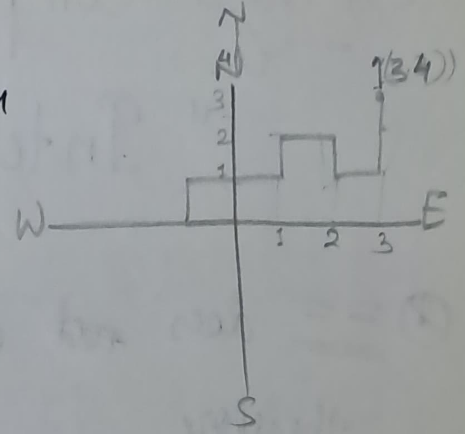
4) else return true.





Q) Shortest Path

"WNEENE SENNN"



Steps:

1) Read str

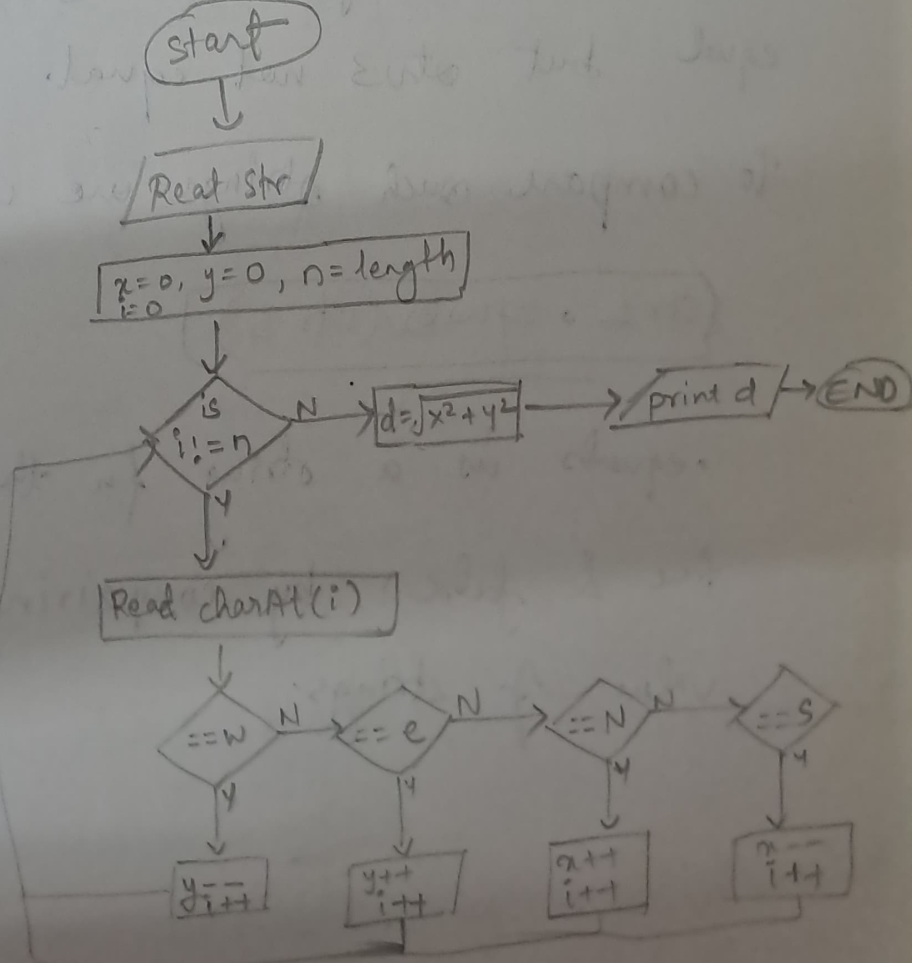
2) If charAt(i) == w go y = y - 1

== e → y++

== n → x++

== s → x--

3) At end, calculate sqrt of $x^2 + y^2$



Strings Compare

.equals();

"Interlink"

* == does ~~not~~ compare strings origin therefore,

```
String str1= "Tony";
String str2 = "Tony";
String str3 = new String("Tony");
```

there == will gives us str 1 & str 2 equal but str3 not equal.

To compare such strings we use

```
(str1.equals(str3))
```

.equals is a string fn that returns true & false by compairing only value of strings.

String functions - Substrings

str.substring(0,5);

Takes twoparameteres si and ei and returns a string from si to ei-1.

String functions - Largest string

~~str.compare to~~

(I) str.compareTo(str1)

↓

0 : equal

-ve : str < str 1

+ve : str > str 1

(II) str.comparetoIgnoreCase(str 1)

It compares strings based on lexicographical order.

a is smallest

z is largest

eg: a b c $\underline{d}$ → small

a b c $\underline{e}$ → big

Interning ~~Interlink~~ ning

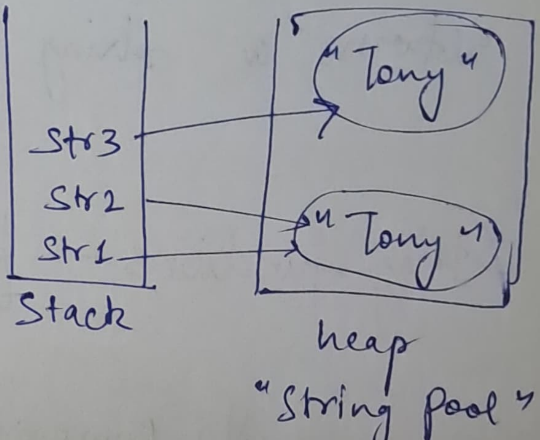
There are 2 types of memory, heap & stack

eg:

String str1 = "Tony"

String str2 = "Tony"

String str3 = new String("Tony"



"String Pool"

"Intern pool"

This causes string to be Immutable, coz,

if we change str1 from "Tony" to "Stark"

then str1 points to new string in heap memory, i.e., Stark, Tony remains there only.

That's why here comes

da-da-da-da-da

String Builder

```
StringBuilder sb = new StringBuilder("A");
```

However this is not string data type, that's why,

```
sb.toString();
```

We can do all programs with sb as we do them with string.

String builder is time & space efficient.

Q) Convert to Upper case each word's first letter.

```
Step 1) Read str
  2) ~~If charat(0)=~~ Make empty sb
  3) sb. append ( Character. toUpperCase (str. charAt (0)));
  4) for i=0 to n-1
  5) If (charAt(i) == ' '  && i<n-1)
     {
       sb. append (charAt (i))
       i++
       sb. append (Character. toUpperCase (str. charAt (i)));
     }
  6) else sb. append (charAt (i));
  7) return sb. toString ( );
```

Q) String Compression.

"aaa bb cccdd" $\longrightarrow$ "a3b2c3d2"

<u>Steps</u>)

```
for ( i=0 to str.length)
        ch(i)
        count = 1
        while (repeat)
              count ++
        count > 1 → no
              = 1 → X
```

O(n)