# Strings

- String is a sequence of characters, for e.g. “Hello” is a string of 5 characters.
- String is basically an object that is backed internally by a char array.
- The CharSequence interface is used to represent the sequence of characters.
- We can create strings in java by using three classes.
    - String
        - In java, string is an immutable object which means it is constant and can’t be changed once it has been created.
    - StringBuffer
        - Java StringBuffer class is used to create mutable (modifiable) string.
        - The StringBuffer class in java is same as String class except it is mutable i.e. it can be changed.
    - StringBuilder
        - Java StringBuilder class is used to create mutable (modifiable) string.
        - The Java StringBuilder class is same as StringBuffer class except that it is non-synchronized.
        - It is available since JDK 1.5.

**Program:**

```
public class CreateStringProg1 {

	public static void main(String[] args) {

		//create string using java string literal
		String s1 = "Java String using java string literal ";
		System.out.println(s1);
```

```
                //creating java string using new keyword
                String s2 = new String("Java String using new keyword");
                System.out.println(s2);

                //Create string by converting char array to string
                char ch[]={'H', 'e', 'l', 'l', 'o',' ','I','n','d','i', 'a'};
                String s3 = new String(ch);
                System.out.println(s3);

                //creating java string by using StringBuffer()
                StringBuffer s_buffer=new StringBuffer("Java String using StringBuffer()");
                System.out.println(s_buffer);

                //creating java string by using StringBuilder()
                StringBuilder s_build = new StringBuilder("Java String using StringBuilder()");
                System.out.println(s_build);
        }
}
```

**Output:**

```
Java String using java string literal
Java String using new keyword
Hello India
Java String using StringBuffer()
Java String using StringBuilder()
```

**The key methods in StringBuilder**

| The key methods in StringBuilder | | |
|---|---|---|
| **Method** | **Return Type** | **Description** |
| append(String s) | StringBuilder | The append() method is used to append the specified string with this string. |
| insert(int offset, String s) | StringBuilder | The insert() method is used to insert the specified string within a string at the specified position. |
| replace(int startIndex, int endIndex, String str) | StringBuilder | The replace() method is used to replace the string from specified startIndex and endIndex. |
| delete(int startIndex, int endIndex) | StringBuilder | The delete() method is used to delete the string from specified startIndex and endIndex. |
| reverse() | StringBuilder | The reverse() method is used to reverse |

| | | |
|---|---|---|
| | | the string. |
| charAt(int index) | char | The charAt() method is used to return the character at the specified position. |
| deleteCharAt(int index) | StringBuilder | The deleteCharAt () method is used to delete the character at specified index |
| length() | int | The length() method is used to return the length of the string i.e. total number of characters. |
| substring(int beginIndex) | String | The substring() method is used to return the substring from the specified beginIndex. |
| substring(int beginIndex, int endIndex) | String | The substring() method is used to return the substring from the specified beginIndex and endIndex. |
| toString() | String | The toString() method to get string representation of an object. |

**Program:**

```
public class KeyMethodsStringBuilderExample {

        public static void main(String[] args) {

                StringBuilder sb1 = new StringBuilder("Hello ");
                sb1.append("Java");                                    //now original string is changed
                System.out.println("append() example: " + sb1);       //prints Hello Java

                StringBuilder sb2 = new StringBuilder("Hello ");
                sb2.insert(1,"Java");                                  //now original string is changed
                System.out.println("insert() example: " + sb2);       //prints HJavaello

                StringBuilder sb3 = new StringBuilder("Hello");
                sb3.replace(1,3,"Java");
                System.out.println("replace() example: " + sb3);      //prints HJavalo

                StringBuilder sb4 = new StringBuilder("Hello");
                sb4.delete(1,3);
                System.out.println("delete() example: " + sb4);       //prints Hlo

                StringBuilder sb5 = new StringBuilder("Hello");
                sb5.reverse();
                System.out.println("reverse() example: " + sb5);      //prints olleH

                StringBuilder sb6 = new StringBuilder("Hello");
                char ch = sb6.charAt(0);
```

```
                    System.out.println("charAt() example: " + ch);        //prints H

                    StringBuilder sb7 = new StringBuilder("Hello");
                    sb7.deleteCharAt(1);
                    System.out.println("deleteCharAt() example: " + sb7);  //prints Hllo

                    StringBuilder sb8 = new StringBuilder("Hello World");
                    int len = sb8.length();
                    System.out.println("length() example: " + len);        //prints 11

                    StringBuilder sb9 = new StringBuilder("Hello World");
                    String st1= sb9.substring(6);
                    System.out.println("substring() with one argument example: " + st1);
                                                                            //prints World

                    StringBuilder sb10 = new StringBuilder("Hello World");
                    String st2= sb10.substring(2, 8);
                    System.out.println("substring() with two argument example: " + st2);
                                                                            //prints llo Wo

                    StringBuilder sb11 = new StringBuilder("Hello World");
                    String st3= sb11.toString();
                    System.out.println("toString() example: " + st3);          //prints Hello World
          }
}
```

**Output:**

```
append() example: Hello Java
insert() example: HJavaello
replace() example: HJavalo
delete() example: Hlo
reverse() example: olleH
charAt() example: H
deleteCharAt() example: Hllo
length() example: 11
substring() with one argument example: World
substring() with two argument example: llo Wo
toString() example: Hello World
```

## Palindrome String

A palindrome string is one which reads the same when it is reversed.

**Write the program to checks whether a string is palindrome or not.**

**Approach:**

- Traverses the input string forwards and backwards, and compare the each character.
    - thereby saving space.

**Program:**

```
import java.util.Scanner;

public class CheckPalindromicStringProg1 {

	public static void main(String[] args) {

		Scanner sc = new Scanner(System.in);
		System.out.print("Enter a string to check palindrome or not:  ");
		String str = sc.nextLine();

		str = str.toLowerCase();

		boolean flag = isPalindromic(str);

		if(flag)

			System.out.print("String " + str + " is palindrome.");

		else

			System.out.print("String " + str + " is not palindrome.");
	}

	public static boolean isPalindromic(String s){

		for (int i = 0, j = s.length() - 1; i < j; ++i, --j) {

			if (s.charAt(i) != s.charAt(j)) {

				return false;
			}
		}
```

```
                return true ;
        }
}
```

**Output:**

```
Case 1:
Enter a string to check palindrome or not:  Madam
String madam is palindrome.

Case 2:
Enter a string to check palindrome or not:  Sir
String sir is not palindrome.
```

**Time and Space Complexity:**

- The time complexity is O(n) and the space complexity is O(1), where n is the length of the string.

## INTERCONVERT STRINGS AND INTEGERS

- A string is a sequence of characters.
- A string may encode an integer, e.g., "123" encodes 123.
- Interconvert of strings and integer means
    - Convert a string number to integer
    - Convert an integer to string

**Write a program to convert a given string to an integer without using any in-built functions.**

**For example:**

- Case 1: If string is positive number
    - If user inputs "321", then the program should give output 321 as an integer number.
- Case 2: If string is negative number

- If user inputs "-321", then the program should give output -321 as an integer number.

**Program:**

```
import java.util.Scanner;

public class StringToIntConversionProg1 {

        public static void main(String[] args) {

                Scanner sc = new Scanner(System.in);
                System.out.print("Enter an Integer Number, Which Stores as String :  ");
                //store as string
                String strNum = sc.nextLine();

                //stringToInt() method convert the integer number to string number
                int intNum = stringToInt(strNum);

                //Display the string number
                System.out.println("The Entered Number in Integer : " + intNum);

        }

        public static int stringToInt(String strNum) {

                //initialize isNegative as false and result as zero
                boolean isNegative = false;
                int result = 0;

                //Iterate till length of the string to Extract
                //the each digit from given string number
                for (int i = 0; i < strNum.length(); i++) {

                        if (strNum.charAt(i) == '-') {

                                //if the first character is '-', set isNegative as true
                                isNegative = true;
                        }

                        else {

                            //converting each character into ASCII format and form the number.
                            //Minus the ASCII code of '0' to get the numeric value of the charAt(i)
                            final int digit = (int)( strNum.charAt(i) - '0');

                            //multiply the partial result by 10 and add the digit.
                            result = result * 10 + digit;
                }
            }

            //if the first character is '-', i.e., isNegative is true, then adds the
            // negative sign to the result and return it otherwise return result
            if (isNegative) {

                return - result;
            }

            else {

                return result;
            }
        }
}
```

**Output:**

**Case 1:**

```
Enter an integer number, which stores as string :  321
The entered number in integer : 321
```

**Case 2 :**

```
Enter an integer number, which stores as string :  -321
The entered number in integer : -321
```

**Time and Space Complexity:**

- The time complexity is O(n) and space complexity is O(1). Here n is the string length.

**Write a program to convert a given integer number to string, without using inbuilt method i.e., parseInt() and valueOf() in java.**

**For example:**

- Case 1: If positive integer
    - If user inputs 871, then the program should give output “871” as a string number.
- Case 2: If string is negative number

- If user inputs -871, then the program should give output “-871” as a string number.

**Program:**

```
import java.util.Scanner;

public class IntToStringConversionProg1 {

	public static void main(String[] args) {

		Scanner sc = new Scanner(System.in);
		System.out.print("Enter an Integer Number :  ");
		int intNum = sc.nextInt();

		//intToString() method convert the integer number to string number
		String strNum = intToString(intNum);

		//Display the string number
		System.out.println("The Entered Number in String : " + strNum);
	}

	public static String intToString(int num) {

		//initialize isNegative as false
		boolean isNegative = false;

		//if the num is less than 0, set isNegative as true
		if (num < 0) {

			isNegative = true;
		}

		//Create object of mutable string by using StringBuilder()
		StringBuilder sb = new StringBuilder();

		//Extract the each digit from given number (i.e., num)
		//add it to the end
		do {
			//extract one digit at a time from the integer
			// by moving right to left order;
			//starting from the LSB towards MSB
			int digit = num % 10;

			//convert digit to char and append to object of mutable string
			sb.append ((char) ('0' + Math.abs(digit)));
```

```
                    //update the num by removing the digit you just extracted.
                    num /= 10;

                } while(num != 0);

                //if the num is negative number then adds the negative sign
                if (isNegative) {

                    sb.append('-') ;
                }

                //reverse the mutable string to get the computed result
                sb.reverse();

                // Convert the mutable string to immutable string and return
                return sb.toString () ;
            }
}
```

**Output:**

**Case 1:**

```
Enter an Integer Number :  7357
The Entered Number in String : 7357
```

**Case 2 :**

```
Enter an Integer Number :  -7357
The Entered Number in String : -7357
```

**Time and Space Complexity:**

- The time complexity is O(n) and space complexity is O(n). Here n is the number of digits in input number and/or output string length.

## BASE CONVERSION

In Computer Science, quite often, we are required to convert numbers from one base to another. The commonly used number systems are binary numbers system (i.e., bases 2), octal numbers system (i.e., bases 8), decimal numbers system (i.e., bases 10) and hexadecimal numbers system (i.e., bases 16).

Binary numbers include digits 0 and 1, and so they have base 2. Octal numbers have digits from 0 to 7, thus having base 8. Decimal numbers have digits from 0 to 9, thus have base 10. And hexadecimal numbers have digits from 0 to 9 and then A to F, thus having base 16.

In the decimal number system, the position of a digit is used to signify the power of 10 that digit is to be multiplied with. For example, "314" denotes the number 3 X 100 +1 X 10 + 4 X 1. The base *b* number system generalizes the decimal number system: the string "$a_{n-1}\, a_{n-2} \ldots a_1\, a_0$", where $0 < a_i < b$, denotes in base-b the integer $a_{n-1} X\, b^{n-1} + a_{n-2} X\, b^{n-2} \ldots a_1 X\, b^1 + a^0\ X\, b^0$.

**Write a program in** Java **to convert a number in any base into another base, limiting the base value up to 16. The input is a string, an integer $b_1$, and another integer $b_2$. The string represents be an integer in base $b_1$. The output should be the string representing the integer in base $b_2$. Assume $2 \le b_1, b_2 \le 16$. Use "A" to represent 10, "B" for 11, ..., and "F" for 15. (For example, if the string is "615", $b_1$ is 7 and $b_2$ is 13, then the result should be "1A7", since $6 \times 7^2 + 1 \times 7 + 5 = 1 \times 13^2 + 10 \times 13 + 7$).**

**Approach:**

- First convert a string in base $b_1$ to decimal integer value using a sequence of multiplications and additions.
- Then convert that decimal integer value to a string in base $b_2$ using a sequence of modulus and division operations.

**Example:**

**Program:**

```
import java.util.Scanner;

public class BaseConversionProg1 {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        System.out.print("Enter an Integer Number, Which Stores as String :  ");
        //store as string
        String strNumb1 = sc.nextLine();

        System.out.print("Enter Source base (between 2 and 16) : ");
         int b1 = sc.nextInt();

        System.out.print("Enter Destination base (between 2 and 16) : ");
        int b2 = sc.nextInt();

        String strNumb2 = convertBase(strNumb1, b1, b2);
        System.out.println("The base " + b1 + " equivalent of " + strNumb1 + " in base "
                                                  + b2 + " is " + strNumb2);
    }

    public static String convertBase(String numAsString , int b1, int b2) {

        //startsWith(), tests the string starts with the specified prefix.
        boolean isNegative = numAsString.startsWith("-");

        //initialization
        int numAsInt = 0;
        int digit = 0;

        //Iterate till length of the string to Extract the each digit
        // the given string number is converted to decimal integer value.
        //convert each character into ASCII format and form the decimal value.
        for (int i = (isNegative ? 1 : 0); i < numAsString.length(); ++i) {

            //Check the specified character in the string is a digit or not
            if (Character.isDigit(numAsString.charAt(i))) {

                //if character is a digit, then minus the ASCII code
                //of '0' to get the decimal integer value of the charAt(i)
                 digit = (int)(numAsString.charAt(i) - '0');
            }

            else {

                //if character is not a digit, then minus the ASCII code
                //of 'A' to get the decimal integer value of the charAt(i)
                digit = (int)(numAsString .charAt(i) - 'A' + 10);

            }

            //multiply the partial result by b1 and add the digit.
            numAsInt = numAsInt * b1 + digit;
        }

        //Check the entered number is zero or not
        if (numAsInt == 0)    {

            return ("0");
        }

        else {
```

```
				//if the first character is '-', i.e., isNegative is true then adds
				//negative sign to the result and return it; otherwise return result.
				//The result is found by Converting the decimal value to base b2
				if (isNegative) {

					return ("-" + constructFromBase(numAsInt , b2));
				}

				else  {

					return ("" + constructFromBase(numAsInt , b2));
				}
			}
		}

		// Converting the decimal value to base b2 conversion using recursion:
		private static String constructFromBase(int numAsInt, int base) {

			if (numAsInt == 0) {

				return "";			// base case
			}

			else {
				//If the remainder is >= 10 or not
				if(numAsInt % base >= 10) {

				   //If the remainder is >= 10, add a character with the
				   //corresponding value (i.e., A = 10, B = 11, C = 12, ...)
				  return (constructFromBase(numAsInt / base, base)
								+ (char)('A' + numAsInt % base - 10));
				}

				else {
					 // If the remainder is a digit < 10, simply add it
					return (constructFromBase(numAsInt / base, base)
								+ (char)('0' + numAsInt % base));
				}
			}
		}
	}
```

**Output:**

**Case 1:**

```
Enter an Integer Number, Which Stores as String :  615
Enter Source base (between 2 and 16) : 7
Enter Destination base (between 2 and 16) : 13
The base 7 equivalent of 615 in base 13 is 1A7
```

**Case 2:**

```
Enter an Integer Number, Which Stores as String :  -1A7
Enter Source base (between 2 and 16) : 13
Enter Destination base (between 2 and 16) : 7
The base 13 equivalent of -1A7 in base 7 is -615
```

**Case 3:**

```
Enter an Integer Number, Which Stores as String :  0
Enter Source base (between 2 and 16) : 3
Enter Destination base (between 2 and 16) : 8
The base 3 equivalent of 0 in base 8 is 0
```

**Time and Space Complexity:**

- The time complexity is $(n(1 + \log_{b_2} b_1))$ , where *n* is the length of string. The reasoning is as follows. First, we perform *n* multiply-and-adds to get d*ecimal integer*, *i.e.*, *x* from string. Then we perform $\log_{b_2} x$ multiply and adds to get the result. The value *x* is upper-bounded by $b_1^n$, and $log_{b_2}\,(b_1^n) = n\; log_{b_2}\, b_1$

## COMPUTE THE SPREADSHEET COLUMN ENCODING

Spreadsheets or excel sheet often use an alphabetical encoding of the successive columns. Specifically, columns are identified by "A", "B", "C", . . ., "X", "Y", "Z", "AA", "AB", ..., "ZZ", "AAA", "AAB",.... and so on.

**Write a program that converts a spreadsheet column id to the corresponding *column number, i.e.*, integer. For Example, if the input (i.e., column id) to the program is A, the output (i.e., column number) is 1. Similarly, if the input is D the output is 4. Suppose the input is AB the program should return output as 28.**

**Approach:**

- This problem is basically the problem of converting a string representing a base-26 number to the corresponding integer, except that "A" corresponds to 1 not 0.

- Each alphabet has a numeric equivalent, i.e., A = 1, B = 2, ... Z = 26.

Example -1: Convert AB,

Use the system where an A = 1, B = 2, ... Z = 26.

Therefore,

$1 \text{ X } 26^1 + 2 \text{ X } 26^0$
$= 1 \text{ X } 26 + 2 \text{ X } 1$
$= 26 + 2$
$= 28$

Example -2: Convert CDA

Use the system where an A = 1, B = 2, ..., Z=26.

Therefore,

$3 \text{ X } 26^2 + 4 \text{ X } 26^1 + 1 \text{ X } 26^0$
$= 3 \text{ X } 676 + 4 \text{ X } 26 + 1 \text{ X } 1$
$= 2028 + 104 + 1$
$= 2133$

**Program:**

```
import java.util.Scanner;

public class SSColIdtoColNumConversion {

        public static void main(String[] args) {

                Scanner sc = new Scanner(System.in);
                System.out.print("Enter a spreadsheet column id (Upper case Chracter(s) only):  ");
                String colId = sc.nextLine();

                //stringToInt() method convert the integer number to string number
                int colNum = ssDecodeColID(colId);

               //Display the column number
               System.out.println("The equivalent column number is " + colNum);

        }

        public static int ssDecodeColID(final String colId) {

                //Initialization
                int result = 0;
                //Iterate till length of the string to Extract the each character
                // and compute base-26 number: multiply the partial result by 26
                //and add the digit.
                for (int i = 0; i < colId.length () ; i++) {

                        char ch = colId.charAt (i) ;

                        result = result * 26 + ch - 'A' + 1;
                }

                return result;
        }
}
```

**Output:**

```
Enter a spreadsheet column id (Upper case Chracter(s) only):  CDA
The equivalent column number is 2133
```

**Time and Space Complexity**

- The time complexity is O(n) and the space complexity is O(1).

**Write a program that converts a spreadsheet column id to the corresponding *column number, i.e.,* integer. For Example, if the input (i.e., column id) to the program is A, the output (i.e., column number) is 1. Similarly, if the input is D the output is 4. Suppose the input is AB the program should return output as 28.**

Given an integer `columnNumber`, return *its corresponding column title as it appears in an Excel sheet*.

Implement
a function that converts an integer to the spreadsheet column id. For example, you
should return "D" for 4, "AA" for 27, and "ZZ" for 702.

**Approach:**

- This problem is basically the problem of converting a string representing a base-26 number to the corresponding integer, except that "A" corresponds to 1 not 0.

- Each alphabet has a numeric equivalent, i.e., A = 1, B = 2, ... Z = 26.

Example:

1. start n = 26, result = ""
   r = 26 % 26 = 0
   n = (26 / 26) - 1 = 0, result += 'Z'
   n = 0, stop,
   output 'Z'

2. start n = 676, result = ""
   r = 676 % 26 = 0
   n = (676 / 26) - 1 = 25, result += 'Z'

   r = 25 % 26 = 25
   n = (25 / 26) = 0, result += 'A' + (25 - 1) = 'Y'
   n = 0, stop,
   output reverse("ZY") = "YZ"

3. start n = 1000, result = ""
   r = 1000 % 26 = 12
   n = (1000 / 26) = 38, result += 'A' + (12 - 1) = 'L'

   r = 38 % 26 = 12
   n = (38 / 26) = 1, result += 'A' + (12 - 1) = 'L'

   r = 1 % 26 = 1
   n = (1 / 26) = 0, result += 'A' + (1 - 1) = 'A'
   n = 0, stop,
   output reverse("LLA") = "ALL"

## REPLACE AND REMOVE

**Write a program which takes as input an array of characters, and removes each entry containing 'b' and replaces each 'a' by two 'd's. Specifically, along with the array, you are provided an integer-valued size. Size denotes the number of entries of the array that the operation is to be applied to. You do not have to worry preserving about subsequent entries. Furthermore, the program returns the number of valid entries.**

**Example, if the array is <** ***a, b, a, c,' ',' ',' '*** **> and the size is 4, then the program should return the array <** ***d, d, d, d, c*** **>.**

**Approach:**

- First, delete 'b's and compute the final number of valid characters of the string, with a forward iteration through the string.

- Then replace each 'a' by two 'd's, iterating backwards from the end of the resulting string.
- If there are more 'b's than 'a's, the number of valid entries will decrease.
- If there are more 'a's than 'b's the number will increase.

**Program:**

```
public class ReplaceAndRemoveProg1 {

        public static void main(String[] args) {

                char arr[] = {'a', 'b', 'a', 'c', ' ' , ' ',' ' };

                int size = 4;

                System.out.print("Before processing the array elements : ");
                for (int i=0; i < arr.length; i++) {

                        System.out.print(arr[i] + " ");
                }

                System.out.println(" ");

                int finalSize = replaceAndRemove(size, arr);

                System.out.print("After processing the array elements : ");
                for (int i=0; i < arr.length; i++) {

                        System.out.print(arr[i] + " ");
                }
                System.out.println(" ");
                System.out.println("The number of valid entries : " + finalSize);
        }

        public static int replaceAndRemove(int size, char[] s) {

                // Forward iteration: remove "b"s and count the number of "a"s.
                int writeldx = 0; //index variable
                int aCount = 0;

                for (int i = 0; i < size; ++i){

                        if (s[i] != 'b') {
```

```
                        s[writeldx++] = s[i];

                    }

                    if (s[i] == 'a') {

                        ++aCount ;

                    }
                }

                // Backward iteration: replace "a"s with "d d"s starting from the end.
                int curldx = writeldx - 1;

                writeldx = writeldx + aCount - 1;

                final int finalSize = writeldx + 1;

                while (curldx >= 0) {

                    if (s[curldx] == 'a') {

                            s[writeldx --] = 'd';

                            s[writeldx --] = 'd';
                    }

                    else {

                        s[writeldx --] = s[curldx];

                    }

                    --curldx ;

                }

                return finalSize;
            }
        }
```

**Output:**

```
Before processing the array elements : a b a c
After processing the array elements : d d d d c
The number of valid entries : 5
```

**Time and Space Complexity:**

- The forward and backward iteration search take O*(n)* time, so the total time complexity is *O(n).* No additional space is allocated.

## TEST PALINDROMICITY

- A sentence can be defined as to be a palindrome string that reads the same thing forward and backward, after removing all the non-alphanumeric and ignoring case.

- **Example:** palindrome sentence

    - The sentence "A man, a plan, a canal, Panama." is palindrome, because after removing all the non-alphanumeric and ignoring case (here making lowercase) the sentence is turned as “amanaplanacanalpanama”; when you read it forward and backward, it is same.

- **Example:** not palindrome sentence

    - The sentence "Ray a Ray" is not a palindrome, because after removing all the non-alphanumeric and ignoring case (here making lowercase) the sentence is turned as "RayaRay"; when you read it forward and backward, it is not same.

**Write a program to check if a sentence is a palindrome or not. You can ignore white spaces and other characters to consider sentences as palindrome.**

**Approach:**

- Use two indices to traverse the string.

- One index is pointing to the 1st character (i.e., i = 0) of the sentence to move forward and another index is pointing to the last character (i.e., j = string length - 1) of the sentence to move backward, skipping non-alphanumeric characters, performing case-insensitive comparison on the alphanumeric characters.

- Return false as soon as there is a mismatch, means sentence is not palindrome. If the indices cross, return true, means sentence is palindrome.

**Program:**

```
import java.util.Scanner;

public class TestPalindromcityProg1 {
```

```
public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a Sentence : ");
        String str = sc.nextLine();

        boolean  flag = isPalindrome(str);

        if(flag)

                System.out.println("Sentence is palindrome");
        else

                System.out.println("Sentence is not palindrome");
}

// To check sentence is palindrome or not
public static boolean isPalindrome(String str) {

        // i and j are index variable
        // i pointing to the 1st character (i.e., i = 0) of the sentence
        // j pointing to the last character (i.e., i = 0) of the sentence
        int i = 0, j = str.length() - 1;

        // Compares character until they are equal
        while (i < j) {

                // i moves forward; skip non-alphanumeric characters.
                while (!Character.isLetterOrDigit (str.charAt(i)) && i < j) {

                        ++i ;
                }

                // j moves backward; skip non-alphanumeric characters.
                while (!Character.isLetterOrDigit(str.charAt (j)) && i < j) {

                        --j ;
                }

                // If characters are not equal then sentence is not palindrome
                if (Character.toLowerCase(str.charAt(i++)) !=
                                        Character.toLowerCase(str.charAt(j--))) {

                        return false;
                }
```

```
                    }

                    // Returns true if sentence is palindrome
                    return true ;
          }
}
```

**Output:**

```
Enter a Sentence : A man, a plan, a canal, Panama.
Sentence is palindrome
```

**Time and Space Complexity:**

- Spend 0(1) per character and the sentence has n characters, so the time complexity is O(n). The space complexity is O(1).

## REVERSE ALL THE WORDS IN A SENTENCE

A sentence is given which containing a set of words and the words are separated by whitespace. Reverse all the words in a sentence means transform that sentence in such a way so that the words appear in the reverse order.

**Example,**

Let the sentence is "Alice likes Bob", transforms to "Bob likes Alice".

**Approach:**

- Reverse the whole sentence first.
  - By reversing sentence the words appeared correctly in their relative positions.
  - However, the words with more than one character, their letters appear in reverse order.
- Finally to get the correct words, reverse the individual word.

**Example:**

- Let the given sentence is “Alice likes Bob”.
- Reversing the whole sentence, the obtained reverse sentence is “boB  sekil ecilA”.

- By reversing each word in reverse sentence, obtain the final result, “Bob likes Alice”.

**Program:**

```
import java.util.Scanner;

public class ReverseAllWordsInSentenceProg1 {

	public static void main(String[] args) {

		Scanner sc = new Scanner(System.in);
		System.out.print("Enter a Sentence : ");
		String str = sc.nextLine();

		// toCharArray() method converts a string
		// to a new character array.
		char[] chArray = str.toCharArray();

		//Calling function which reverse all words in a sentence
		reverseWords(chArray);

		// valueOf() method returns the string representation
		//of the char array argument.
		String Output = String.valueOf(chArray);

		// Display the resultant String
		System.out.println("After reversing all words, the sentence : " +  Output);
	}

	//reverse all words in a sentence
	public static void reverseWords(char[] input) {

		// Reverses the whole string first.
		reverse(input, 0, input.length);

		int start = 0;

		//In the reverse string, it find the
		// index of first word boundary
		int end = find(input, ' ' , start);

		//reverses each individual word in the
		// reverse string, except last word
		while (end != -1) {

			// Reverses each word in the reverse string.
```

```
                        reverse( input , start, end);
                        start = end + 1;

                        //it find the index of next word boundary
                        end = find(input, ' ' , start);
                }

                // Reverses the last word of reverse string.
                reverse(input, start, input.length);
        }

        public static void reverse(char[] array, int start, int stoplndex) {

                if (start >= stoplndex) {

                        return ;
                }

                int last = stoplndex - 1;

                for (int i = start; i <= start + (last - start) / 2; i++) {

                        char tmp = array[i];

                        array[i] = array[last - i + start];

                        array[last - i + start] = tmp;
                }
        }

        public static int find(char[] array, char c, int start){

                for (int i = start; i < array.length ; i++) {

                        if (array[i] == c) {

                                return i ;
                        }
                }

                return -1;
        }
}
```

**Output:**

```
Enter a Sentence : Alice likes Bob
After reversing all words, the sentence : Bob likes Alice
```

**Time and Space Complexity:**

- The time complexity is O*(n)*, where *n* is the length of the of the string. The additional space complexity is *O(1)*.

## CONVERT FROM ROMAN TO DECIMAL

- The Romans used Latin letters for writing numbers.
- Roman numerals are a number system developed in ancient Rome where letters represent numbers.
- The modern use of Roman numerals involves the letters I, V, X, L, C, D, and M.
    - In Roman numeral system I is 1, V is 5, X is 10, L is 50, C is 100, D is 500, M is 1000.
- Roman numeral system is classic example non-positional numeral systems
    - The value of a figure is independent of its positions.
        - For example, number 3, is written as III in Roman numerals.
    - Although, it's a non-positional numeral system but there is an additional rule that modify the value of a digit according to its place.
        - That rule forbids the use of the same digit 3 times in a row. That's why 3 is III
- The Roman number symbols to be a valid Roman Number, if symbols appear in non increasing order with the following exceptions allowed:
    - I placed before V or X indicates one less, so 4 is IV (one less than 5) and 9 is IX (one less than 10).
    - X placed before L or C indicates ten less, so 40 is XL (10 less than 50) and 90 is XC (ten less than a hundred).
    - C placed before D or M indicates a hundred less, so 400 is CD (100 less than 500) and 900 is CM (100 less than 1000).
- **Note:** Back-to-back exceptions are not allowed, e.g., IXC and CDM are invalid.

- A valid complex Roman number string represents the integer which is the sum of the symbols that do not correspond to exceptions; for the exceptions, add the difference of the larger symbol and the smaller symbol.

**Write a program which takes as input a valid Roman number string and returns the corresponding integer.**

**Approach:**

- Performs the right-to-left iteration, and if the symbol after the current one is greater than it, then subtract the current symbol.

**Program:**

```
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

public class ConvertRomanNumberToDecimalProg1 {

        public static void main(String[] args) {

                Scanner sc = new Scanner(System.in);
                System.out.print("Enter a Roman Number : ");
                String RomNum = sc.nextLine();

                int DecNum =  romanToInteger(RomNum);

                System.out.print("Roman Number " + RomNum + " is equivalent to Decimal
                                                              Number " + DecNum);
        }

        public static int romanToInteger(String RomNum){

                // Convert String to Upper Case
                RomNum = RomNum.toUpperCase();

                Map<Character , Integer> HM = new HashMap<Character, Integer>();

                        HM.put('I', 1);
                        HM.put('V', 5);
                        HM.put('X', 10);
                        HM.put('L', 50);
                        HM.put('C', 100);
                        HM.put('D', 500);
                        HM.put('M', 1000);

                // get() returns the value to which the specified key is mapped,
                // otherwise return null, if no mapping for the key.
                // initialize the sum
                int sum = HM.get(RomNum.charAt(RomNum.length() - 1));

                // Performs the right-to-left iteration
                for (int i = RomNum.length() - 2 ; i >= 0; --i) {

                        // if the symbol after the current one is greater than it,
                        // then subtract the current symbol. otherwise add.
                        if (HM.get(RomNum.charAt(i)) < HM.get(RomNum.charAt(i + 1))) {

                        sum -= HM.get(RomNum.charAt(i));

                        } else {

                        sum += HM.get(RomNum.charAt(i));

                        }
                }

                return sum;
        }
}
```

## Output:

```
Enter a Roman Number : XC
Roman Number XC is equivalent to Decimal Number 90
```

**Time and Space Complexity:**

- The overall time complexity is O(n), because for each character of string is processed in O(1) time, where n is the length of the string. The Space complexity is O(1).

## WRITE A STRING SINUSOIDALLY

- Write a string "Hello World!" in sinusoidal fashion, which as shown as below:

```
          e           ␣           l
      H       l   o       W   r       d        Here ␣ denotes
                l           o           !        a blank
```

- From the sinusoidal fashion of the string, it is possible to write the snakestring.
- To write the snakestring of the string, write the sequence from left to right and from top to bottom in which the characters appear in the sinusoidal fashion of the string.
- Example :
  - The snakestring for "Hello␣World!" is "e␣lHloWrdlo!".

**Write a program which takes as input a string and returns the snakestring of the inputted string.**

**Approach:**

- It can be observed that, write the given string in a sinusoidal manner in the array, which is shown below. Finally, for snakestring read out the non-null characters in row-manner.

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | **e** | | | | ␣ | | | | **l** | | |
| 1 | **H** | | **l** | | **o** | | **W** | | **r** | | **d** | |
| 2 | | | | **l** | | | | **o** | | | | **!** |
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |

Here ␣ denotes a blank

- However, it observe that the result begins with the characters s[l], s[5], s[9],..., followed by s[0],s[2], s[4],..., and then s[3], s[7], s[11],…
- Hence, create snakestring directly (without writing string in a sinusoidal manner) with 3 iterations through given input string.

**Program:**

```
import java.util.Scanner;

public class SinusoidalStringProg1 {

        public static void main(String[] args) {

                Scanner sc = new Scanner(System.in);
                System.out.print("Enter a string : ");
                String str = sc.nextLine();

                String output = snakeString(str);

                System.out.println("The sinsoida structure of given string : " + output);
        }
        public static String snakeString(String s) {

                //Create object of mutable empty string by using StringBuilder()
                StringBuilder result = new StringBuilder ();

                // Outputs the first row, i.e., s[l], s[5], s[9], ...
                for (int i = 1; i < s.length(); i += 4) {

                        result.append (s . charAt (i) ) ;
                }

                // Outputs the second row, i.e., s[<9], s[2], s[4], ...
                for (int i = 0; i < s.length(); i += 2) {

                        result . append (s . charAt (i) ) ;
                }

                // Outputs the third row, i.e., s[3], s[7], s[ll], ...
                for (int i = 3; i < s.length(); i += 4) {

                        result . append (s . charAt (i) ) ;
                }

                return result.toString ();
        }
}
```

**Output:**

```
Enter a string : Hello World!
The sinsoida structure of given string : e lHloWrdlo!
```

**Time and Space Complexity:**

- Let *n* be the length of s. Each of the three iterations takes*0(n)* time, implying an*0(n)*time complexity.

**THE LOOK-AND-SAY PROBLEM**

- The look-and-say sequence starts with 1.
- Subsequent numbers are derived by describing the previous number in terms of consecutive digits.

- Specifically, to generate an entry of the sequence from the previous entry, read off the digits of the previous entry, counting the number of digits in groups of the same digit.

- The look-and-say sequence is the sequence of integers beginning as follows:

  1, 11, 21, 1211, 111221, 312211, 13112221, 1113213211, …

  - The first term is "1"
  - Second term is "11", generated by reading first term as "One 1"
  - Third term is "21", generated by reading second term as "Two 1"
  - Fourth term is "1211", generated by reading third term as "One 2 One 1"
  - Fifth term is "111221" generated by reading the forth term as "one 1 one 2 two 1"
  - and so on.

**Write a program that takes as input an integer *n* and returns the nth integer in the look-and-say sequence. Return the result as a string.**

**Example:**

- Let n = 4, then return the 4th term of the look and say sequence. Hence the sequence is 1211.

**Program:**

```
import java.util.Scanner;

public class LookAndSayProg1 {

	public static void main(String[] args) {

		Scanner sc = new Scanner(System.in);
		System.out.print("Enter an Integer : ");
		int num = sc.nextInt();

		String result = lookAndSay(num);

		System.out.println(" ");

	    System.out.println("The " + num + " term of the look and say sequence is " + result);
}

public static String lookAndSay(int n) {

        String str = "1";

        System.out.print("The look and say sequence are " + str + ' ');

        for (int i = 1; i < n; ++i) {

                str = nextNumber(str);

                System.out.print(str + ' ');

        }

        return str;
}

private static String nextNumber(String str) {

        //Create object of mutable empty string by using StringBuilder()
        StringBuilder result = new StringBuilder();

        for (int i = 0; i < str.length(); ++i) {

                // initialize the count as 1
                // count store how many times a digit occurred
                int count = 1 ;

                // Compares current digit and the next digit and counts
                // the number of times the current digit is repeated
                while (i + 1 < str.length() && str.charAt(i) == str.charAt(i + 1)) {

                        ++i ;
                        ++count ;
                }

                // Once a different digit is encountered,
                // the count and the current digit are appended to result.
                result.append(count);
                result.append(str.charAt(i));
        }

        return result.toString();
}
```

```
}
```

**Output:**

```
Enter an Integer : 4
The look and say sequence are 1 11 21 1211
The 4 term of the look and say sequence is 1211
```

**Time and Space Complexity:**

- Since there are *n* iterations and the work in each iteration is proportional to the length of the number computed in the iteration, a simple bound on the time complexity is $0(n^2)$.

## IMPLEMENT RUN-LENGTH ENCODING

- Run Length encoding (RLE) is a lossless data compression algorithm.
- RLE works by reducing the physical size of a repeating string of characters.
- This repeating string, called a run, is typically encoded into two bytes.
- The first byte represents the number of characters in the run and is called the run count.
- The second byte is the value of the character in the run and is called the run value.
- **Example:**
  - Let consider a string "BBBBBBBBBBBBBBB" and its run length encoding is 15B. In 15B, the first byte, 15, is the run count and contains the number of repetitions. The second byte, B, is the run value and contains the actual repeated value in the run.
- Every encoded string is a repetition of a string of digits followed by a single character. The string of digits is the decimal representation of a positive integer.

Run length decoding is a process of interpretation and translation of coded information into a comprehensible form.

To generate the decoded string, need to convert this sequence of digits into its integer equivalent and then write the character that many times. do this for each character.

**Approach:**

1. Start with an empty string for the code, and start traversing the string.
2. Pick the first character and append it to the code.

3. Now count the number of subsequent occurances of the currently chosen character and append it to the code.
4. When you encounter a different character, repeat the above steps.
5. In a similar manner, traverse the entire string and generate the code.

**Given an input string, write a function that returns the Run Length Encoded string for the input string. Assume the string to be encoded consists of letters of the alphabet, with no digits.**

**Program:**

```
import java.util.Scanner;

public class RunLengthEncodingProg1 {

        public static void main(String[] args) {

                Scanner sc = new Scanner(System.in);
                System.out.print("Enter a string for run-length encoding : ");
                String str = sc.nextLine();

                String encodestr = encoding(str);

                System.out.println("The run-length encoding is " + encodestr);
        }

        public static String encoding(String str) {

                //Create object of mutable empty string by using StringBuilder()
                StringBuilder sb = new StringBuilder();

                // Count occurrences of current character
                int count = 1;

                //Iterate till length of the string
                for (int i = 1; i <= str.length(); ++i) {

                        // Compares current  and the previous character
                        // if current character is a new character, write
                        // the count of previous character.
                        if (i==str.length() || str.charAt(i) != str.charAt(i - 1)) {
```

```
                    sb.append(count);
                    sb.append(str.charAt(i - 1));

                    //set count equal to 1, for new character
                    count = 1;
                }

                // Compares current and the previous character and counts
                // the number of times the previous character is repeated
                else { // str.charAt(i) == str.charAt(i - 1).

                    ++count ;
                }
            }

            return sb.toString();
        }
}
```

**Output:**

```
Enter a string for run-length encoding : aaaabcccaa
The run-length encoding is 4a1b3c2a
```

**Time and Space Complexity:**

- The time complexity is *0(n),* where *n* is the length of the string.

**Given an input Run Length Encoded string, consisting of digits and lowercase alphabet characters write a function that returns the decoded string. The decoded string is guaranteed not to have numbers in it.**

**Program:**

```
import java.util.Scanner;

public class RunLengthDecoding {

        public static void main(String[] args) {

                Scanner sc = new Scanner(System.in);
                System.out.print("Enter a string for run-length decoding : ");
                String str = sc.nextLine();

                String decodestr = decoding(str);
```

```
            System.out.println("The run-length decoded string is " + decodestr);
        }

        public static String decoding(String s) {

            int count = 0;

            StringBuilder result = new StringBuilder () ;

            for (int i = 0; i < s.length(); i++) {

                char c = s.charAt(i);

                if (Character . isDigit(c)) {

                    count = count * 10 + c - '0' ;

                } else { // c is a letter of alphabet.


                    while (count > 0){ // Appends count copies of c to result.

                        result.append(c);

                        count -- ;
                    }
                }
            }

            return result.toString();
        }
}
```

**Output:**

```
Enter a string for run-length decoding : 3e4f2e
The run-length decoded string is eeeffffee
```

**Time and Space Complexity:**

- The time complexity is *0(n)*, where *n* is the length of the string.

## COMPUTE ALL VALID IP ADDRESSES

- Internet Protocol (**IPv4 or IP)** addresses are canonically represented in dot-decimal notation, which consists of four decimal numbers, each ranging from 0 to 255, separated by periods and cannot have leading zeros.

- Examples of valid IP addresses are 0.1.2.201 and 192.168.1.1

- Examples of invalid IP addresses are 0.011.255.245 and 192.168.1.312

- The IP address 0.011.255.245 is invalid, due to leading 0 present in the second decimal number, i.e., 011.

- The IP address 192.168.1.312 is invalid, because the fourth decimal number, i.e., 312 is beyond the range 0 to 255.

**Let the given decimal string. A decimal string is a string consisting of digits between 0 and 9. Write a program that determines where to add periods to a decimal string so that the resulting string is a valid IP address. There may be more than one valid IP address corresponding to a string, hence return all possible valid IP addresses that can be obtained.**

**Example:**

- Let the input is "19216811"

- The output returns 9 different IP addresses as an array

  - [92.168.1.11, 19.2.168.11, 19.21.68.11, 19.216.8.11, 19.216.81.1, 192.68.1.11, 192.16.8.11, 192.16.81.1, 192.168.1.1]

**Approach:**

- There are three periods in a valid **IP** address, hence enumerate all possible placements of these periods, and check whether all four corresponding substrings are between 0 and 255. It is possible to reduce the number of placements by spacing the periods 1 to 3 characters apart. It is also potential to lead by stopping as soon as a substring is invalid.

- **Example**:

  - Let the string is "19216811", one could put the first period after "1", "19" and "192". If the first part is "1", after placing the second period, the second part might be "9", "92" and "921". Amongst these, "921" is illegal, so do not go on with it.

**Program:**

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class ComputeAllValidIPAddressesProg1 {

        public static void main(String[] args) {

                Scanner sc = new Scanner(System.in);
                System.out.print("Enter a decimal string : ");
                String str = sc.nextLine();

                List<String> result = getValidIpAddress(str);

                System.out.println("All possible valid IP addresses :");
                for(String ip : result)

                        System.out.println(ip);

         }

// This method calculate different combinations and
// uses isValidPart to valid IP address part or substring
 public static List<String> getValidIpAddress(String s) {

        List<String> result = new ArrayList();

        for (int i = 1; i < 4 && i < s.length(); ++i) {

             final String first = s.substring(0, i) ;

             if (isValidPart(first)) {

                for (int j = 1; i + j < s.length() && j < 4; ++ j ) {

                     final String second = s.substring(i, i + j);

                     if (isValidPart(second)) {

                          for(int k = 1; i + j + k < s.length() && k < 4 ; ++k) {

                               final String third = s.substring(i + j, i + j + k) ;

                               final String fourth = s.substring(i + j + k) ;

                               if ( isValidPart(third) && isValidPart( fourth) ) {

                                          result . add(first + '.' + second + '.' + third + '.' + fourth);
```

```
                                    } //end if
                                } //end for
                            } //end if
                        } //end for
                    } //end if
                } //end for

                return result;
        }

                //Check if this string forms valid IP address part.
                private static boolean isValidPart (String s) {


                        // if string length is greater than 3,
                        // then that string is not a part of valid IP address.
                        if (s.length() > 3) {

                                return false;
                        }

                        // If the string start with "0" and length greater than 1,
                        // then that string is not a part of valid IP address.
                        // "00", "000" , "01", etc. are not valid, but "0" is valid.
                        if (s.startsWith("0") && s.length() > 1) {

                                return false;
                        }

                        //Parses the string argument as a signed decimal integer.
                        int val = Integer.parseInt(s) ;

                        // return value if it's ranging from 0 to 255
                        return val <= 255 && val >= 0;
                }
}
```

**Output:**

```
Enter a decimal string : 19216811
All possible valid IP addresses :
1.92.168.11
19.2.168.11
19.21.68.11
19.216.8.11
```

```
19.216.81.1
192.1.68.11
192.16.8.11
192.16.81.1
192.168.1.1
```

**Time and Space Complexity:**

- The total number of IP addresses is a constant (232), implying an (9(1) time complexity for the above algorithm.