**ISC COMPUTER SCIENCE 2026**
**PAPER – I**

**PART I (20 Marks)**

**Question 1.**

(i) According to the Principle of Duality, the Boolean equation
Q' • 0 + P' • Q' + P' • Q = P' will be equivalent to:

(a) Q • 0 + P • Q + P • Q' = P

(b) Q' • 1 + P' • Q' + P • Q' = P'

(c) (Q' + 1) • (P' + Q') • (P' + Q) = P'

(d) (Q' + 0) • (P' + Q') • (P' + Q) = P'

**Answer: (c)**
**Reason:**
Using **Principle of Duality**: AND will be replaced by OR and v.v., 0 will be replaced by 1 and v.v., but the complement should remain unchanged.
Given: Q'•0 + P'•Q' + P'•Q = P'
Dual ⇒ (Q'+1) • (P'+Q') • (P'+Q) = P'

(ii) Consider the following statement written in class Student where school_Name is its data member.

*static final String school_Name = "Co-Ed School";*

Which of the following statements are valid for school_Name?

I. All objects of class Student share the same value of school_Name.

II. The value of school_Name cannot be changed during program execution.

III. The keywords static and final cannot be used together for a variable.

(a) Only I and II

(b) Only II and III

(c) Only I and III

(d) Only III

**Answer: (a) Only I and II**
**Reason:**
static final String school_Name = "Co-Ed School";

- static → shared by all objects ✓
- final → cannot be changed ✓
- static + final together → valid ✓

(iii) Study the given propositions and the statements marked Assertion and Reason that follow. Choose the correct option based on your analysis.

P = You practise regularly

Q = You become skilled

S1 = $P \Rightarrow Q$

S2 = $\sim P \vee Q$

**Assertion:** S1 and S2 are logically equivalent.

**Reason:** A conditional statement $P \Rightarrow Q$ can be expressed as $\sim P \vee Q$

(a) Both Assertion and Reason are true and Reason is the correct explanation for Assertion.

(b) Both Assertion and Reason are true but Reason is not the correct explanation for Assertion.

(c) Assertion is true and Reason is false.

(d) Both Assertion and Reason are false.

**Answer: (a)**

**Reason:**

$P \Rightarrow Q \equiv \sim P \vee Q$

So S1 and S2 are equivalent.

Assertion = TRUE

Reason = TRUE (correct explanation)

(iv) The Boolean equations $a + 1 = a$ and $a \cdot 0 = 0$ correspond to:

(a) Involution Law

(b) Law of Identity

(c) Distributive Law

(d) Law of Complements

**Answer: (b)**

**Reason:**

$a + 1 = 1$ and $a \cdot 0 = 0 \rightarrow$ **Identity Law**

(v) The *worst case* complexity for following code segment is:

```
for(int i=1; i<= n; i++)
{
   for(int j=1; j<= i; j++)
   {
      statement;
   }
}
```

(a) O(n+i)

(b) O(n×i)

(c) O(n)

(d) $O(n^2)$

**Answer: (d)**

**Reason:**

Nested loops:

Total iterations = $1 + 2 + 3 + \ldots + n = [n \times (n+1)] / 2 = O(n^2)$

(vi) Given below are two statements marked, Assertion and Reason. Read the two statements carefully and choose the correct option.

**Assertion:** An interface in Java contains abstract and non-abstract methods.

**Reason:** All methods in an interface must be implemented by any class that extends this interface.

(a) Both Assertion and Reason are true and Reason is the correct explanation for Assertion.

(b) Both Assertion and Reason are true but Reason is not the correct explanation for Assertion.

(c) Assertion is true and Reason is false.

(d) Both Assertion and Reason are false.

**Answer: (d)**

**Reason:**

In Java, Interfaces cannot have non-abstract **methods**, if it is an older version, i.e. before Java 8. But in newer versions, interfaces can have non-abstract methods with the default keyword. All the methods of an interface can be or may not be implemented in any class that implements the interface.

Assertion = TRUE

Reason = FALSE

(vii) The complement of the Boolean expression a • b' + a' + a' • b is:

(a) a + b' • a' • a' + b

(b) (a' + b) • a • (a + b')

(c) (a + b') • a' • (a + b')

(d) a' • b + a + a • b'

**Answer: (b)**

Given: a•b' + a' + a'•b

Complement:

= (a•b' + a' + a'•b)'

= (a•b')' • (a')' • (a'•b)'

= (a' + b) • a • (a + b')

(viii) Given below are two statements marked, Assertion and Reason. Read the two statements carefully and choose the correct option.

**Assertion:** The return statement enables the exit of the program control from the current method.

**Reason:** If a method's return type is void, it can still contain return 0 statement to return nothing.

(a) Both Assertion and Reason are true and Reason is the correct explanation for Assertion.

(b) Both Assertion and Reason are true but Reason is not the correct explanation for Assertion.

(c) Assertion is true and Reason is false.

(d) Both Assertion and Reason are false.

**Answer: (c)**

Assertion: return exits method. TRUE

Reason: void can return 0. FALSE

(ix) Consider the two propositions given below:

A = You use ecofriendly methods

B = Pollution is reduced

If A implies to B, then write its *Contrapositive* statement.

**Answer.**

Contrapositive of A ⇒ B is: ~B ⇒ ~A

If the pollution is not reduced, then you did not use eco-friendly methods.

(x) What is *Gray code* in Karnaugh map?

**Answer.**

Gray code: Adjacent cells differ by only **one bit** (used in K-map grouping)

**Question 2.**

(i) Convert the following *infix* notation to *prefix* form.

P / Q + ( S * F + X / R )

**Answer.**

= (P / Q) + ( (S * F) + (X / R))

= (/PQ) + (*SF + /XR)

= (/PQ) + (+*SF/XR)

**= + / P Q + * S F / X R**

(ii) A matrix G[3…7, 2…5] is stored in the memory, with each element requiring 4 bytes of storage. If the address of G[5][4] is 6000, find the **base address** when the matrix is stored **Column Major Wise**.

**Answer.**

Formula: Address = Base + w[(j-L2)×rows + (i-L1)]

Given:

- w = 4
- rows = (7-3+1) = 5
- G[5][4] address = 6000

6000 = Base + 4[(4-2)x5 + (5-3)]

= *Base* + 4[(*25*)+2]

= Base + 4 (12)

= Base + 48

Base = 6000 − 48 = **5952**

(iii) The following function **workOut( )** is a part of some class. Assume 'n' is a positive integer.

```
String workOut (int n)
{  if (n == 0)
     return "";
   int rem = n % 16;
   char cr = (rem < 10) ? (char)(rem + '0') : (char)(rem - 10 + 'A');
   return workOut (n / 16) + cr;
}
```

Answer the questions given below with the dry run / working.

(a) What will the function **workOut(220)** return?

(b) What is the function **workOut( )** performing apart from recursion?

**Answer.**
Function converts **decimal → hexadecimal**

(a) workOut(220):
220 ÷16 = 13 remainder 12 → D
13 → D
**Answer = $(DC)_{16}$**

(b) Function: **Decimal to Hexadecimal conversion**

(iv) The following function **isTech( )** is a part of some class which is used to check if a given number is a Tech number or not. There are some places in the code marked by **?1?** , **?2?** , **?3?** which may be replaced by a statement / expression so that the function works properly.

A number is **Tech** number if the count of digits is even and the square of the sum of its two equal halves is equal to the number itself.

Example: $2025 = 20 + 25 = (45)^2 = 2025$

```
boolean isTech(int n)
{
   String s = String.valueOf(n);
   int len = ?1?;
   if (len % 2 != 0)
   return ?2?;
   int first = Integer.parseInt(s.substring(0, len / 2));
   int second = Integer.parseInt(s.substring(len / 2));
   int sum = first + second;
   return ?3? == n;
}
```

What are the expressions or statements at **?1?, ?2?** and **?3?**

**Answer.**
?1? → s.length()
?2? → return false;
?3? → sum * sum

**PART II**
**SECTION A**

**Question 3.**

(i) (a) What is a *decoder*?

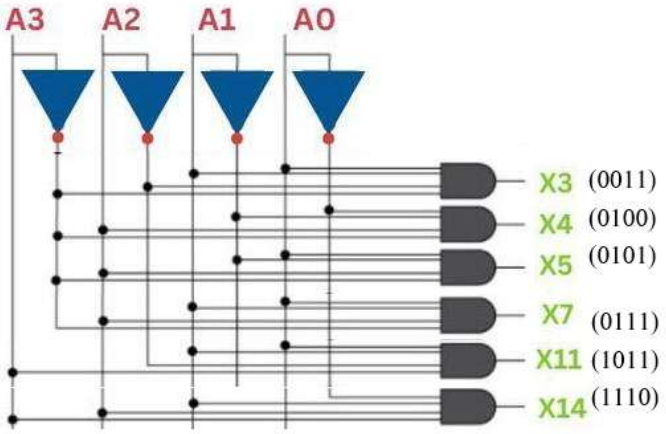
(b) Draw the logic gate diagram for decoding the binary numbers {0011, 0100, 0101, 0111, 1011, 1110} to hexa-decimal numbers.

(c) State *any one* application of a *multiplexer*.

**Answer.**

(a) Decoder: A circuit that converts binary input into one of many outputs.

(b)




(c) Application: (Any one)

- Data selection in communication systems
- Data Routing: In CPUs, to choose between multiple data sources.
- Communication Channels: Combine multiple signals into one line to save bandwidth.
- Control Systems: Select sensor input based on control logic.

(ii) The Chain rule states that [(a=>b) • (b=>c)] => (a=>c). Prove this rule using Boolean laws.

**Answer.**

**Chain Rule** $(a \Rightarrow b)(b \Rightarrow c)$

$= (\sim a + b)(\sim b + c)$

$= (\sim a \sim b + \sim a\ c + b \sim b + bc)$

$= (\sim a\ c + bc)$

$= (a \Rightarrow c)$ **Proved**

(iii) Given that P = 0, Q = 1, R = 0, S = 0, write its:

(a) Maxterm

(b) Minterm

**Answer.**

Given: P=0, Q=1, R=0, S=0

Maxterm = (P + Q' + R + S)

Minterm = (P'. Q. R'. S')

**Question 4.**

(i) According to the ancient laws of the Valley of Peace, a candidate can become the Dragon Warrior only if they satisfy any one of the following conditions:

- The candidate belongs to the Panda Clan and has been trained for more than 5 years under the Grand Master

**OR**

- The candidate possesses the Secret Chi Power but does not belong to the Panda Clan

**OR**

- The candidate is recommended by the Grand Master, but neither belongs to the Panda Clan nor has been trained for more than 5 years

The inputs are:

| INPUTS | |
|---|---|
| **C** | Belongs to the Panda Clan |
| **T** | Trained for more than 5 years |
| **P** | Possesses Secret Chi Power |
| **R** | Recommended by the Grand Master |

(In all the above cases, 1 indicates YES and 0 indicates NO)

**Output: X** - Denotes eligibility to become the Dragon Warrior
(1 = eligible and 0 = not eligible)

Draw the truth table for the inputs and outputs given above. Write the **SOP** expression for **X(C, T, P, R)**.

For the given conditions, we get –

1. C•T
2. C'•P
3. C'•T'•R

**Truth table**

| C | T | P | R | X | Minterm |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | **0** | |
| 0 | 0 | 0 | 1 | **1** | C’T’P’R |
| 0 | 0 | 1 | 0 | **1** | C’T’PR’ |
| 0 | 0 | 1 | 1 | **1** | C’T’PR |
| 0 | 1 | 0 | 0 | **0** | |
| 0 | 1 | 0 | 1 | **0** | |
| 0 | 1 | 1 | 0 | **1** | C’TPR’ |
| 0 | 1 | 1 | 1 | **1** | C’TPR |
| 1 | 0 | 0 | 0 | **0** | |
| 1 | 0 | 0 | 1 | **0** | |
| 1 | 0 | 1 | 0 | **0** | |
| 1 | 0 | 1 | 1 | **0** | |
| 1 | 1 | 0 | 0 | **1** | CTP’R’ |
| 1 | 1 | 0 | 1 | **1** | CTP’R |
| 1 | 1 | 1 | 0 | **1** | CTPR’ |
| 1 | 1 | 1 | 1 | **1** | CTPR |

**SOP expression:**

**X(C,T,P,R) =** C’T’P’R + C’T’PR’ + C’T’PR + C’TPR’ + C’TPR + CTP’R’ + CTP’R + CTPR’ + CTPR
= ∑(1,2,3,6,7,12,13,14,15)

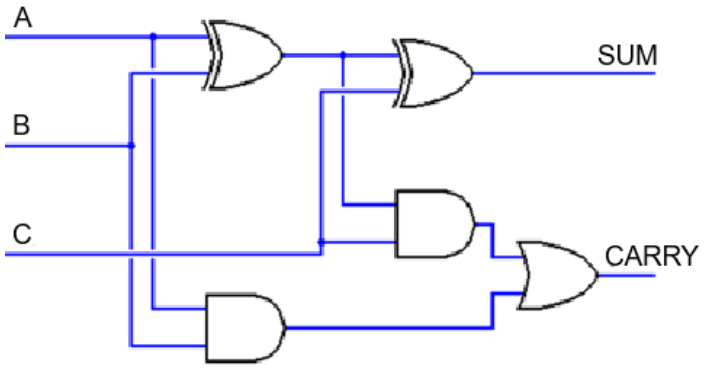
(ii) Construct the logic gate diagram for a *Full Adder* using two *Half Adders*.

**Full Adder**

Sum = A ⊕ B ⊕ C

Carry = A⊕B.C + AB

(using 2 half adders)




(iii) Draw a truth table to verify if the following proposition is a Tautology, a Contradiction or a Contingency.

$$( A \wedge \sim B ) \Rightarrow ( \sim A \vee B )$$

**Truth Table**

| A | B | A' | B' | A.B' | A'+B | A.B'⇒A'.B |
|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 |

**It is a contingency**

**Question 5.**

(i) (a) Reduce the Boolean function **F(A,B,C,D) = π (0,1,2,3,7,8,9,10,11,12,13,15)** by using 4-variable Karnaugh map, showing the various groups (i.e., octal, quads and pairs).

(b) Draw the logic gate diagram for the reduced expression. Assume that the variables and their complements are available as inputs.

**Given, F(A,B,C,D) = ∏(0,1,2,3,7,8,9,10,11,12,13,15)**

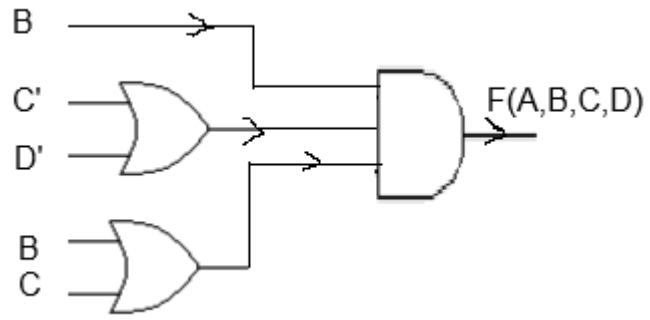
**K-MAP**

| AB\CD | C+D | C+D' | C'+D' | C'+D |
|---|---|---|---|---|
| A+B | 0 (0) | 0 (1) | 0 (3) | 0 (2) |
| A+B' | 1 (4) | 1 (5) | 0 (7) | 1 (6) |
| A+B | 0 (12) | 0 (13) | 0 (15) | 1 (14) |
| A'+B | 0 (8) | 0 (9) | 0 (11) | 0 (10) |

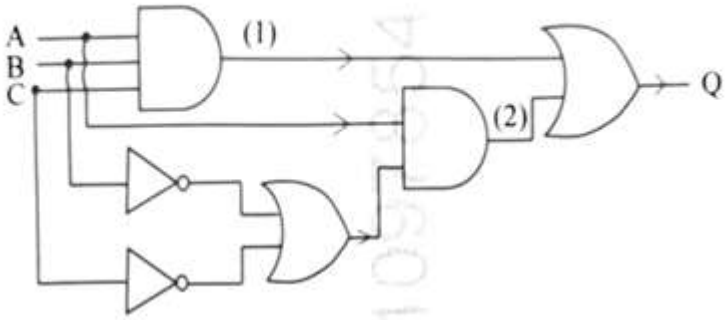
**Octet (m0,m1,m3,m2,m8,m9,m11,m10) = B**
**Quad 1 (m3,m7,m15,m11) = C'+D'**
**Quad 2 (m12,m13,m8,m9) = B+C**
**Final K-map simplified: F = B . (C'+D') . (B+C)**

**(b)**

**Logic diagram of reduced expression:**




(ii) From the logic gate diagram given below, derive the Boolean expression for (1), (2) and Q. Reduce the derived expression.

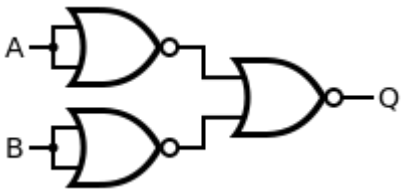



**(1) A.B.C**
**(2) A . (B'+C')**
**(3) A.B.C + A.(B'+C')**

**F(A,B,C) = A.B.C + A.(B'+C')**
**= A.B.C + A.B' +A.C'**
**= A. (B.C + B' + C')**
**= A. (B'+C+C') ….** [B'+BC = (B'+B).(B'+C) by applying Distributive law]
= **A. (B'+1) ….** [C+C' = 1 by applying Complement's law]
**= A**

(iii) Draw the logic gate diagram for 2-input AND gate using NOR gates only. Show the expression at each step.

Logic diagram: OR gate using NOR gates.




Working:

$Q = \overline{\overline{A} + \overline{B}}$

$= \overline{\overline{A}} . \overline{\overline{B}}$ (by DeMorgan's Law)

= A.B (by Involution Rule)

## SECTION B (JAVA OOPS)

**Question 6.**

A class **TimeOp** has been defined to add any two accepted time periods.

Example: Time A = 6 hours 35 minutes 40 seconds
Time B = 7 hours 45 minutes 30 seconds
Time A + Time B = 14 hours 21minutes 10 seconds
(where 60 minutes = 1 hour and 60 seconds = 1 minute)

The details of the members of the class are given below:

| | | |
|---|---|---|
| **Class name** | : | **TimeOp** |
| **Data member/instance variable:** | | |
| arr[ ] | : | integer array to hold three elements (hours, minutes and seconds) |
| **Methods/Member functions:** | | |
| TimeOp( ) | : | default constructor |
| void readTime( ) | : | to accept the elements of the array |
| TimeOp addTime(TimeOp tt) | : | to add the time of the parameterised object *tt* and the current object, to store it in a local object and return it |
| void dispTime( ) | : | to display the array elements in hours:minutes:seconds format |

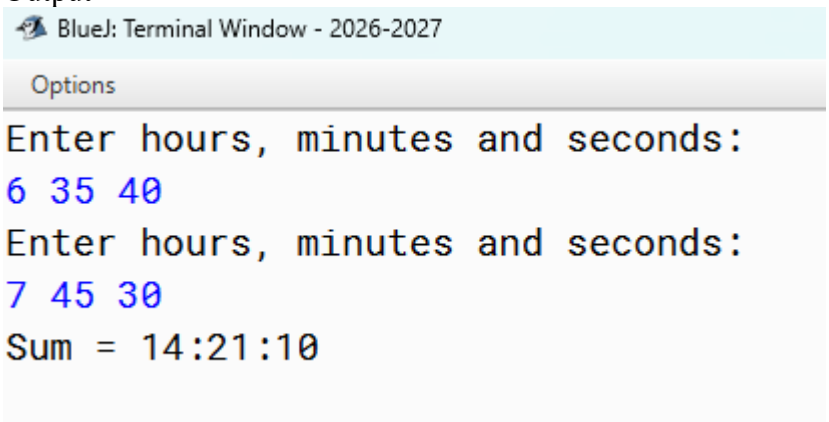
Specify the class **TimeOp** giving the details of the **constructor( ), void readTime( ), TimeOp addTime(TimeOp)** and **void dispTime( ).** Define the **main( )** function to create objects and call the functions accordingly to enable the task.

**Solution.**

```
import java.util.Scanner;
class TimeOp
{
  int arr[]; // [hours, minutes, seconds]

  // Constructor
  TimeOp()
  {
    arr = new int[3];
  }

  // Input method
  void readTime()
  {
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter hours, minutes and seconds:");
    arr[0] = sc.nextInt();
    arr[1] = sc.nextInt();
    arr[2] = sc.nextInt();
  }

  // Add two time objects
  TimeOp addTime(TimeOp t)
  {
    TimeOp temp = new TimeOp();
```

```
        temp.arr[2] = this.arr[2] + t.arr[2];
        temp.arr[1] = this.arr[1] + t.arr[1];
        temp.arr[0] = this.arr[0] + t.arr[0];

        // Adjust seconds
        temp.arr[1] += temp.arr[2] / 60;
        temp.arr[2] %= 60;

        // Adjust minutes
        temp.arr[0] += temp.arr[1] / 60;
        temp.arr[1] %= 60;

        return temp;
    }

    // Display
    void dispTime()
    {
        System.out.println(arr[0] + ":" + arr[1] + ":" + arr[2]);
    }

    public static void main(String args[])
    {
        TimeOp t1 = new TimeOp();
        TimeOp t2 = new TimeOp();
        t1.readTime();
        t2.readTime();
        TimeOp result = t1.addTime(t2);
        System.out.print("Sum = ");
        result.dispTime();
    }
}
```

Output



**Question 7.**

(i) A class **Trimorphic** has been defined to accept a positive integer from the user and display if it is a Trimorphic number or not.

[A number is said to be Trimorphic if the cube of the number ends with the number itself.]

Example 1: $24^3 = 13824$ ends with 24

Example 2: $5^3 = 125$ ends with 5

The details of the members of the class are given below:

| | | |
|---|---|---|
| **Class name** | : | **Trimorphic** |
| **Data members/instance variables:** | | |
| n | : | to store the number |
| cube | : | to store the cube of the number |
| **Methods/Member functions:** | | |
| Trimorphic( ) | : | constructor to initialise the data members with legal initial values |
| void accept( ) | : | to accept a number |
| boolean check(int num, long c) | : | to compare *num* with the ending digits of *c* using **recursive technique** |
| void result( ) | : | to check whether the given number is a trimorphic number by invoking the function *check( )* and to display an appropriate message |

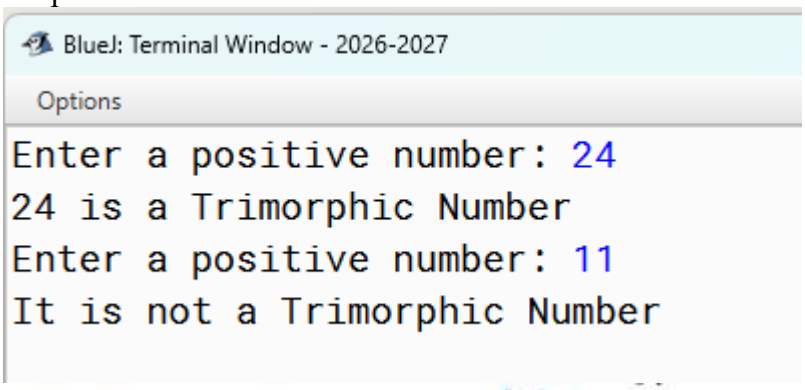
Specify the class **Trimorphic** giving the details of the **constructor( ), void accept( ), boolean check( )** and **void result( )**. Define the **main( )** function to create an object and call the functions accordingly to enable the task.

**Solution.**

```
import java.util.Scanner;
class Trimorphic
{
   int n;
   long cube;

   Trimorphic()
   {
      n = 0;
      cube = 0;
   }

   void accept()
   {
      Scanner sc = new Scanner(System.in);
      System.out.print("Enter number: ");
      n = Math.abs(sc.nextInt());
      cube = (long) (n * n * n);
   }
```

```
    boolean check(int num, long c)
    {
        if(num == 0)
            return true;
        if(num % 10 != c % 10)
            return false;
        return check(num / 10, c / 10);
    }

    void result()
    {
        if(check(n, cube))
            System.out.println(n + "is a Trimorphic Number");
        else
            System.out.println("It is not a Trimorphic Number");
    }

    public static void main(String args[])
    {
        Trimorphic obj = new Trimorphic();
        obj.accept();
        obj.result();
    }
}
```

Output




(ii) State *any two* differences between *iteration* and *recursion*.

**Difference:**

| **Iteration** | **Recursion** |
|---|---|
| It uses loops (for/while/do while) | It uses functions/methods |
| It is faster than recursion | It is slower than iteration |
| It takes less memory | It takes more memory |

**Question 8.**

Design a class **PendulumS** to perform an operation on a word containing alphabets in upper case only. Rearrange the word by putting the lowest ASCII value character at the centre and the second lowest ASCII value character to its right and the third to its left and so on.

Example 1 : Input : COMPUTER

Output : TPMCEORU

Example 2 : Input : SCIENCE

Output : SIECCEN

The details of the members of the class are given below:

| | | |
|---|---|---|
| **Class name** | : | **PendulumS** |
| **Data members/instance variables:** | | |
| wrd | : | to store the original word |
| newwrd | : | to store the rearranged word |
| **Methods/Member functions:** | | |
| PendulumS(String k) | : | parameterised constructor to initialise *wrd* = *k* and *newwrd* = "" |
| int minCharIndex(String str) | : | to find the index of the minimum ASCII value character in *str* and return it |
| void arrange( ) | : | to rearrange the characters of *wrd* as per the given instructions and store it in *newwrd* by invoking *minCharIndex( )* |
| void display( ) | : | to display the original word and the rearranged word |

Specify the class **PendulumS** giving the details of the **constructor( )**, **int minCharIndex(String)**, **void arrange( )** and **void display( )**. Define the **main( )** function to create an object and call the functions accordingly to enable the task.

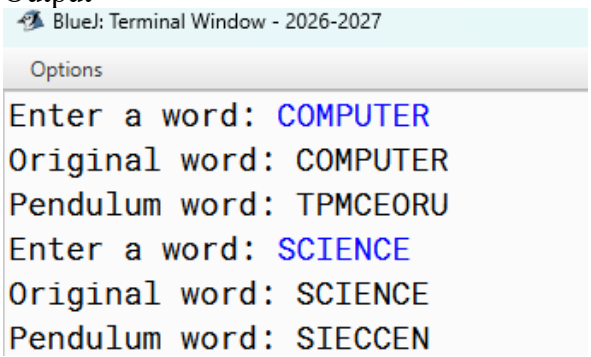
**Solution.**

```
import java.util.*;
class PendulumS
{
   // Data Members
   String wrd;
   String newwrd;

   // Parameterised Constructor
   PendulumS(String k)
   {
      wrd = k;
      newwrd = "";
   }
```

```
// Method to find index of minimum ASCII character
int minCharIndex(String str)
{
   int minIndex = 0;
   for (int i = 1; i < str.length(); i++)
   {
      if (str.charAt(i) < str.charAt(minIndex))
         minIndex = i;
   }
   return minIndex;
}

// Method to arrange characters in pendulum form
void arrange()
{
   String temp = wrd.toUpperCase(); // working string
   int n = temp.length();
   char result[] = new char[n];

   // Find middle position
   int mid;
   if (n % 2 == 0)
      mid = (n / 2) - 1;
   else
      mid = n / 2;
   int left = mid - 1;
   int right = mid + 1;
   boolean placeRight = true;

   for (int i = 0; i < n; i++)
   {
      // Find smallest character
      int index = minCharIndex(temp);
      char ch = temp.charAt(index);

      // Remove that character from temp
      temp = temp.substring(0, index) + temp.substring(index + 1);

      // Place in result
      if (i == 0)
         result[mid] = ch;
      else
      {
         if (placeRight)
         {
            if (right < n)
            {
               result[right] = ch;
               right++;
            }
         }
         else
         {
```

```
                if (left >= 0)
                {
                    result[left] = ch;
                    left--;
                }
            }
            placeRight = !placeRight;
        }
    }
    // Convert result array to string
    newwrd = "";
    for (int i = 0; i < n; i++)
    {
        newwrd = newwrd + result[i];
    }
}

// Display method
void display()
{
    System.out.println("Original word: " + wrd);
    System.out.println("Pendulum word: " + newwrd);
}

public static void main(String args[])
{
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter a word: ");
    String input = sc.nextLine();
    PendulumS obj = new PendulumS(input);
    obj.arrange();
    obj.display();
}
}
```

Output




**SECTION C**

**Question 9.**

In any internet browser, a user can visit new webpages and go back to previously visited webpages. Each new webpage URL is stored in browser's memory such that when the user clicks 'Back' button, the previous webpage gets displayed.

The details of the members of the class are given below:

**Class name** : **Browser**

**Data members/instance variables:**

| | | |
|---|---|---|
| pages[ ] | : | an array to hold the URLs of visited webpages |
| max | : | to store the maximum capacity of the array |
| top | : | to point to the index of the last visited webpages |

**Methods/Member functions:**

| | | |
|---|---|---|
| Browser(int cap) | : | constructor to assign *max* = *cap* and *top* = *–1* |
| void visit(String url) | : | to add URL of a new webpage if possible, else display the message "Browser history full" |
| String back( ) | : | to remove and return the last visited webpage URL, if present, else to return the message "No previous browser history" |

(i) Specify the class **Browser** giving details of the functions **void visit(String)** and **String back( )**. *Assume that the other functions have been defined.*

(ii) Name the entity described above and state its principle.

**Solution.**

```
class Browser
{
   String pages[];
   int max, top;

   Browser(int cap)
   {
      max = cap;
      pages = new String[max];
      top = -1;
   }

   void visit(String url)
   {
      if(top == max - 1)
         System.out.println("Browser history full");
      else
      {
         top++;
         pages[top] = url;
      }
   }
```

```
  String back()
  {
    if(top == -1)
      return "No previous browser history";
    else
    {
      String temp = pages[top];
      top--;
      return temp;
    }
  }
}
```

(ii) Entity: **Stack**
Principle: **LIFO (Last In First Out)**

**Question 10.**

**Customer** to calculate the total bill for a customer as per the following criteria:

| Room type | Additional Amount |
|---|---|
| Executive | 10% of room rent |
| Suite | 20% of room rent |

The details of the members of both the classes are given below:

**Class name** : **Hotel**

**Data members/instance variables:**

hname : to store hotel name

roomrent : to store the rent per day

**Methods/Member functions:**

Hotel(...) : parameterised constructor to assign values to its data members

void show( ) : to display hotel details

**Class name** : **Customer**

**Data members/instance variables:**

cname : to store customer name

days : to store the number of days of stay

type : to store the room type

surcharge : to store the additional amount

amt : to store the total amount

**Methods/Member functions:**

Customer(...) : parameterised constructor to assign values to data members of both the classes

void compute( ) : to calculate the surcharge based on the room type as given above. Also, to calculate the total amount as:
(room rent + surcharge) × days

void show( ) : to display hotel and customer details

*Assume that the superclass **Hotel** has been defined*. Using the **concept of Inheritance**, specify the class **Customer**, giving details of **constructor(...)**, **void compute( )** and **void show( )**.

**The super class, main function and algorithm need NOT be written.**

**Solution.**

```
class Customer extends Hotel
{
   String cname, type;
   int days;
   double surcharge, amt;

   Customer(String hname, double rent, String cname, int days, String type)
   {
      super(hname, rent);
      this.cname = cname;
      this.days = days;
      this.type = type;
   }

   void compute()
   {
      if(type.equals("Executive"))
         surcharge = 0.10 * roomrent;
      else if(type.equals("Suite"))
         surcharge = 0.20 * roomrent;
      amt = (roomrent + surcharge) * days;
   }

   void show()
   {
      super.show();
      System.out.println("Customer: " + cname);
      System.out.println("Days: " + days);
      System.out.println("Type: " + type);
      System.out.println("Total Amount: " + amt);
   }
}
```

**Question 11.**

(i) A linked list is formed from the objects of the class **Word**. The structure of the class Word is given below:

```
class Word
{
   String value;
   Word next;
}
```

Write an *Algorithm* **OR** a *Method* to count and display the number of nodes whose *value* starts with a consonant.

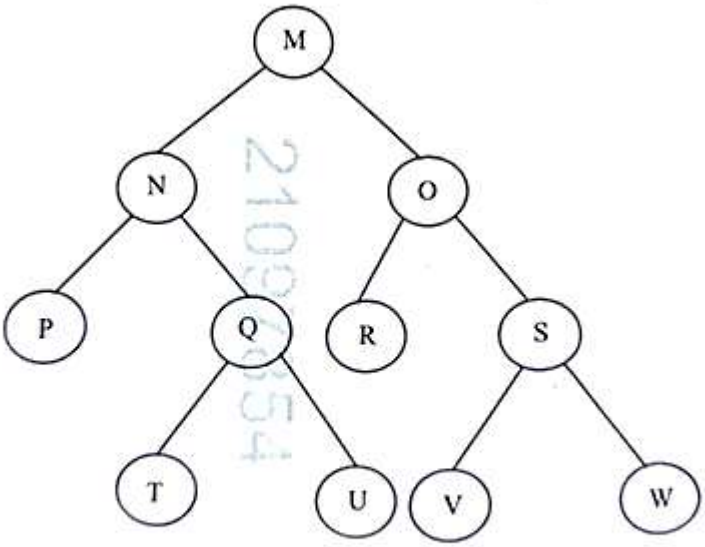
The method declaration is as follows:

**void countConsonant(Word first)**

**Answer.**

```
void countConsonant(Word first)
{
   int count = 0;
   while(first != null)
   {
      char ch = Character.toLowerCase(first.value.charAt(0));
      if("aeiou".indexOf(ch) == -1)
         count++;
      first = first.next;
   }
   System.out.println("Consonant nodes = " + count);
}
```

(ii) Answer the following questions based on the diagram of a Binary Tree given below:




(a) Write the *pre-order* traversal of the above tree.

(b) State the level of Node Q, when the root is at level 0.

(c) State the size of the left subtree and the right subtree.

**Answer.**

(a) Preorder: M N P Q T U O R S V W

(b) Level of Q = 2

(c) Left subtree size = 5
Right subtree size = 5

---