KG REDDY College of Engineering & Technology
AN AUTONOMOUS INSTITUTION

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

ADVANCED DATA STRUCTURE LAB MANUAL

Subject Code: KGR2458111

Regulation: KGR24

Academic Year: 2024-25

I M. TECH I SEMESTER

COMPUTER SCIENCE AND ENGINEERING

KG REDDY COLLEGE OF ENGINEERING AND TECHNOLOGY

Affiliated To JNTUH, Chilkur,(V), Moinabad(M) R. R Dist, TS-501504

KG REDDY College of Engineering & Technology
AN AUTONOMOUS INSTITUTION

VISION AND MISSION OF THE INSTITUTION

VISION:

To become an institution which is internationally recognized for its holistic approach to engineering, innovative teaching and learning culture, research and entrepreneurial ecosystem, and sustainable social impact in the community.

MISSION:

• To offer undergraduate and post-graduate programs which are supported through industry relevant curriculum and innovative teaching and learning processes that would help students succeed in their professional careers.
• To provide faculty and students with an ecosystem that fosters innovation, research, entrepreneurship, and international exposure through strategic partnerships with government organizations and collaboration with industries.
• To provide holistic learning environment to students which will contribute to their personal and professional growth and enable them to become leaders in their respective fields.
• To contribute to the development of the region by using our technological expertise to work with nearby communities and support them in their social and economic development.

\

KG REDDY College of Engineering & Technology
AN AUTONOMOUS INSTITUTION

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VISION:

To be recognized as a department of excellence by stimulating a learning environment in which students and faculty will thrive and grow to achieve their professional, institutional and societal goals.

MISSION:

• To provide high quality technical education to students that will enable life-long learning and build expertise in advanced technologies in Computer Science and Engineering.
• To promote research and development by providing opportunities to solve complex engineering problems in collaboration with industry and government agencies.
• To encourage professional development of students that will inculcate ethical values and leadership skills through entrepreneurship while working with the community to address societal issues.

PROGRAM EDUCATIONAL OBJECTIVES

PEO 1: Graduates will provide solutions to difficult and challenging issues in their profession by applying computer science and engineering theory and principles.
PEO 2: Graduates have successful careers in computer science and engineering fields or will be able to successfully pursue advanced degrees.
PEO 3: Graduates will communicate effectively, work collaboratively and exhibit high levels of professionalism, moral and ethical responsibility.
PEO 4: Graduates will develop the ability to understand and analyze engineering issues in a broader perspective with ethical responsibility towards sustainable development.

KG REDDY College of Engineering & Technology
AN AUTONOMOUS INSTITUTION

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PROGRAM OUTCOMES

• **PO I: Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
• **PO II: Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
• **PO III: Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
• **PO IV: Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
• **PO V: Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
• **PO VI: The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
• **PO VII: Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of and need for sustainable development.
• **PO VIII: Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
• **PO IX: Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
• **PO X: Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

• **PO XI: Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
• **PO XII: Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

KG REDDY College of Engineering & Technology
AN AUTONOMOUS INSTITUTION

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PROGRAM SPECIFIC OUTCOMES

PSO1: The Computer Science and Engineering graduates are able to analyze, design, develop, test and apply management principles, mathematical foundations in the development of intelligent systems with computational solutions, make them to expert in designing the secure application and hardware prototype
PSO2: The graduating student will be analyze the contemporary research issues in different areas of computer science & engineering and explore research gaps, analyze and carry out research in the specialized/emerging areas.
PSO3: Develop their skills to solve problems in the broad area of programming concepts and appraise environmental and social issues with ethics and manage different projects in multi-disciplinary field to conducive in cultivating skills for successful career, entrepreneurship and higher studies.

KG REDDY College of Engineering & Technology
AN AUTONOMOUS INSTITUTION

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ADVANCED DATA STRUCTURES LAB

M.Tech CSE

Prerequisites:

1. A course on Computer Programming & Data Structures

Course Objectives:

1. Introduces the basic concepts of Abstract Data Types.
2. Reviews basic data structures such as stacks and queues.
3. Introduces a variety of data structures such as hash tables, search trees, tries, heaps, graphs, and B-trees.
4. Introduces sorting and pattern matching algorithms.
5. Able to identify and use a suitable data structure and algorithm to solve a real world problem

Course Outcomes:

1. Ability to select the data structures that efficiently model the information in a problem.
2. Ability to assess efficiency trade-offs among different data structure implementations or Combinations.
3. Implement and know the application of algorithms for sorting and pattern matching.
4. Apply Non-linear data structures: Trees and Graphs.
5. Design programs using a variety of data structures, including hash tables, binary and general tries, heaps and B-trees.

List of Programs

1. Write a program to perform the following operations:
a) Insert an element into a binary search tree.
b) Delete an element from a binary search tree.
c) Search for a key element in a binary search tree.

2. Write a program for implementing the following sorting methods:
a) Merge sort b) Heap sort c) Quick sort

3. Write a program to perform the following operations:
a) Insert an element into a B- tree.
b) Delete an element from a B- tree.
c) Search for a key element in a B- tree.

4. Write a program to perform the following operations:
a) Insert an element into a Min-Max heap
b) Delete an element from a Min-Max heap
c) Search for a key element in a Min-Max heap

5. Write a program to perform the following operations:
a) Insert an element into a Lefiist tree
b) Delete an element from a Leftist tree
c) Search for a key element in a Leftist tree

6. Write a program to perform the following operations:
a) Insert an element into a binomial heap

b) Delete an element from a binomial heap.
c) Search for a key element in a binomial heap

7. Write a program to perform the following operations:
a) Insert an element into a AVL tree.
b) Delete an element from a AVL search tree.
c) Search for a key element in a AVL search tree.

8. Write a program to perform the following operations:
a) Insert an element into a Red-Black tree.
b) Delete an element from a Red-Black tree.
c) Search for a key element in a Red-Black tree.

9. Write a program to implement all the functions of a dictionary using hashing.

10. Write a program for implementing Knuth-Morris-Pratt pattern matching algorithm.

11. Write a program for implementing Brute Force pattern matching algorithm.

12. Write a program for implementing Boyer pattern matching algorithm.

TEXT BOOKS:

1. Fundamentals of Data structures in C, E. Horowitz, S. Sahni and Susan Anderson Freed, 2nd Edition, Universities Press
2. Data Structures Using C – A.S. Tanenbaum, Y. Langsam, and M.J. Augenstein, PHI/Pearson education.
3. Introduction to Data Structures in C, Ashok Kamthane, 1st Edition, Pearson.

REFERENCES:

1. The C Programming Language, B.W. Kernighan, Dennis M. Ritchie, PHI/Pearson Education
2. C Programming with problem solving, J.A. Jones & K. Harrow, Dreamtech Press
3. Data structures: A Pseudocode Approach with C, R.F. Gilberg And B.A. Forouzan, 2nd Edition,
Cengage Learning

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
DATABASE PROGRAMMING WITH PL/SQL LAB

1. WRITE A pl/sql program using for loop to insert ten rows into a database table?

Program:

```
DECLARE
   -- Declare a variable to loop through the rows
   v_employee_id NUMBER;
   v_first_name  VARCHAR2(50);
   v_last_name   VARCHAR2(50);
BEGIN
   -- Loop through 10 iterations to insert 10 rows
   FOR i IN 1..10 LOOP
      -- Assign values to the variables (you can modify the values as needed)
      v_employee_id := i; -- Assign a unique ID for each iteration
      v_first_name := 'FirstName' || i; -- Concatenate "FirstName" with the loop counter
      v_last_name := 'LastName' || i; -- Concatenate "LastName" with the loop counter

      -- Insert the values into the employees table
      INSERT INTO employees (employee_id, first_name, last_name)
      VALUES (v_employee_id, v_first_name, v_last_name);
   END LOOP;

   -- Commit the transaction
   COMMIT;
END;
/
```

Example Output:

After running the script, your `employees` table will have the following rows:

employee_id	first_name	last_name
1	FirstName1	LastName1
2	FirstName2	LastName2
3	FirstName3	LastName3
4	FirstName4	LastName4
5	FirstName5	LastName5
6	FirstName6	LastName6
7	FirstName7	LastName7
8	FirstName8	LastName8
9	FirstName9	LastName9
10	FirstName10	LastName10

2. Given the table employee (empno,name,salary,designation,deptid),write a cursor to select the five highest paid employees from the table

Program:

```
DECLARE
   -- Declare a record type to hold employee data
   TYPE emp_record IS RECORD (
      empno       employee.empno%TYPE,
      name        employee.name%TYPE,
      salary      employee.salary%TYPE,
      designation employee.designation%TYPE,
      deptid      employee.deptid%TYPE
   );

   -- Declare a cursor to select the five highest paid employees
   CURSOR highest_paid_employees IS
      SELECT empno, name, salary, designation, deptid
      FROM employee
      ORDER BY salary DESC
      FETCH FIRST 5 ROWS ONLY;

   -- Declare a variable to hold each record fetched by the cursor
   emp_data emp_record;
BEGIN
   -- Open the cursor and loop through the results
   OPEN highest_paid_employees;

   LOOP
      -- Fetch a record from the cursor
      FETCH highest_paid_employees INTO emp_data;

      -- Exit the loop if no more rows are fetched
      EXIT WHEN highest_paid_employees%NOTFOUND;

      -- Output the employee data (this could be replaced with any processing logic)
      DBMS_OUTPUT.PUT_LINE('Empno: ' || emp_data.empno ||
                           ', Name: ' || emp_data.name ||
                           ', Salary: ' || emp_data.salary ||
                           ', Designation: ' || emp_data.designation ||
                           ', Dept ID: ' || emp_data.deptid);
   END LOOP;

   -- Close the cursor
   CLOSE highest_paid_employees;
END;
/
```

Output:

```
Empno: 1001, Name: John Doe, Salary: 90000, Designation: Manager, Dept ID: 10
Empno: 1002, Name: Jane Smith, Salary: 85000, Designation: Senior Developer, Dept ID: 20
Empno: 1003, Name: Mary Johnson, Salary: 80000, Designation: Developer, Dept ID: 10
Empno: 1004, Name: Robert Brown, Salary: 75000, Designation: Developer, Dept ID: 30
Empno: 1005, Name: David Wilson, Salary: 72000, Designation: Senior Developer, Dept ID: 20
```

3. Illustrate how you can embed PL/SQL in a high -level host blanguage such as c/java and demonstrate how a banking debit transaction might be done

Program:

```
import java.sql.*;

public class BankingDebitTransaction {
    public static void main(String[] args) {
        // Database credentials
        String url = "jdbc:oracle:thin:@localhost:1521:xe"; // Oracle DB URL
        String username = "yourUsername";
        String password = "yourPassword";

        // Account details
        int accountNumber = 12345;
        double debitAmount = 500.00;

        // PL/SQL block to perform the debit transaction
        String plsqlBlock =
            "BEGIN " +
            "  DECLARE " +
            "    v_balance NUMBER; " +
            "  BEGIN " +
            "    -- Fetch current balance of the account " +
            "    SELECT balance INTO v_balance FROM accounts WHERE account_number = :accountNumber; " +
            "    -- Check if sufficient balance exists " +
            "    IF v_balance >= :debitAmount THEN " +
            "      -- Perform the debit transaction " +
            "      UPDATE accounts SET balance = balance - :debitAmount WHERE account_number = :accountNumber;
" +
            "      -- Insert transaction into the history table " +
            "      INSERT INTO transactions (account_number, transaction_type, amount, transaction_date) " +
            "      VALUES (:accountNumber, 'DEBIT', :debitAmount, SYSDATE); " +
            "    ELSE " +
            "      RAISE_APPLICATION_ERROR(-20001, 'Insufficient funds'); " +
            "    END IF; " +
            "  END; " +
            "END;";

        // Establish database connection and execute the PL/SQL block
        try (Connection conn = DriverManager.getConnection(url, username, password)) {
            // Create a callable statement for the PL/SQL block
            CallableStatement stmt = conn.prepareCall(plsqlBlock);

            // Set the input parameters for the PL/SQL block
            stmt.setInt(1, accountNumber);       // Account number
            stmt.setDouble(2, debitAmount);      // Debit amount
            stmt.setInt(3, accountNumber);       // Account number (for balance check)
            stmt.setDouble(4, debitAmount);      // Debit amount (for update)
            stmt.setInt(5, accountNumber);       // Account number (for transaction history)

            // Execute the PL/SQL block
            stmt.execute();
```

```
            System.out.println("Transaction successful: Debited " + debitAmount + " from account " + accountNumber);
        } catch (SQLException e) {
            e.printStackTrace();
            System.err.println("Error occurred: " + e.getMessage());
        }
    }
}
```

OUTPUT:

Transaction successful: Debited 500.00 from account 12345

C Example (Using OCI - Oracle Call Interface)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <oci.h>

#define USERNAME "yourUsername"
#define PASSWORD "yourPassword"
#define DBNAME   "localhost:1521/xe"

// Function to handle Oracle errors
void checkError(OCIError *errhp) {
    text errbuf[512];
    ub4 errcode;
    OCIErrorGet((dvoid *)errhp, (ub4)1, (text *)NULL, &errcode, errbuf, (ub4)sizeof(errbuf), OCI_HTYPE_ERROR);
    printf("Oracle error: %s\n", errbuf);
}

// Main function to perform debit transaction
int main() {
    OCIEnv *envhp;
    OCIError *errhp;
    OCISvcCtx *svchp;
    OCIStmt *stmt;
    OCIBind *bind1 = NULL, *bind2 = NULL, *bind3 = NULL, *bind4 = NULL, *bind5 = NULL;
    sword status;
    char *sql =
        "BEGIN "
        "  DECLARE "
        "    v_balance NUMBER; "
        "  BEGIN "
        "    SELECT balance INTO v_balance FROM accounts WHERE account_number = :accountNumber; "
        "    IF v_balance >= :debitAmount THEN "
        "      UPDATE accounts SET balance = balance - :debitAmount WHERE account_number = :accountNumber; "
        "      INSERT INTO transactions (account_number, transaction_type, amount, transaction_date) "
        "      VALUES (:accountNumber, 'DEBIT', :debitAmount, SYSDATE); "
        "    ELSE "
        "      RAISE_APPLICATION_ERROR(-20001, 'Insufficient funds'); "
        "    END IF; "
        "  END; "
        "END;";

    int accountNumber = 12345;
```

```
    double debitAmount = 500.00;

    // Initialize OCI environment and error handle
    OCIEnvInit(&envhp, OCI_DEFAULT, 0, NULL);
    OCIHandleAlloc(envhp, (dvoid **)&errhp, OCI_HTYPE_ERROR, 0, NULL);
    OCIHandleAlloc(envhp, (dvoid **)&svchp, OCI_HTYPE_SVCCTX, 0, NULL);

    // Connect to Oracle database
    status = OCIConnect(svchp, errhp, (text *)USERNAME, (text *)PASSWORD, (text *)DBNAME, 0,
OCI_DEFAULT);
    if (status != OCI_SUCCESS) {
        checkError(errhp);
        exit(1);
    }

    // Prepare and execute the SQL statement
    OCIStmtPrepare(stmt, errhp, (text *)sql, (ub4)strlen(sql), OCI_NTV_SYNTAX, OCI_DEFAULT);

    // Bind input parameters
    OCIBindByPos(stmt, &bind1, errhp, 1, (dvoid *)&accountNumber, (sword)sizeof(accountNumber), SQLT_INT,
NULL, NULL, NULL, 0, NULL, OCI_DEFAULT);
    OCIBindByPos(stmt, &bind2, errhp, 2, (dvoid *)&debitAmount, (sword)sizeof(debitAmount), SQLT_FLT, NULL,
NULL, NULL, 0, NULL, OCI_DEFAULT);
    OCIBindByPos(stmt, &bind3, errhp, 3, (dvoid *)&accountNumber, (sword)sizeof(accountNumber), SQLT_INT,
NULL, NULL, NULL, 0, NULL, OCI_DEFAULT);
    OCIBindByPos(stmt, &bind4, errhp, 4, (dvoid *)&debitAmount, (sword)sizeof(debitAmount), SQLT_FLT, NULL,
NULL, NULL, 0, NULL, OCI_DEFAULT);
    OCIBindByPos(stmt, &bind5, errhp, 5, (dvoid *)&accountNumber, (sword)sizeof(accountNumber), SQLT_INT,
NULL, NULL, NULL, 0, NULL, OCI_DEFAULT);

    // Execute the statement
    status = OCIStmtExecute(svchp, stmt, errhp, 0, 0, NULL, NULL, OCI_DEFAULT);
    if (status != OCI_SUCCESS) {
        checkError(errhp);
        exit(1);
    }

    printf("Transaction successful: Debited %.2f from account %d\n", debitAmount, accountNumber);

    // Cleanup and disconnect
    OCIStmtRelease(stmt, errhp, NULL, 0, OCI_DEFAULT);
    OCIHandleFree(svchp, OCI_HTYPE_SVCCTX);
    OCIHandleFree(errhp, OCI_HTYPE_ERROR);
    OCIHandleFree(envhp, OCI_HTYPE_ENV);

    return 0;
}
```

OUTPUT:

Transaction successful: Debited 500.00 from account 12345

4. Given an inter i write a PL/SQL procedure to insert the tuple(i,'xxx') into a given relation

Table Definition:

```
CREATE TABLE my_table (col1 NUMBER, col2 VARCHAR2(50));
```

PL/SQL Procedure:

```
CREATE OR REPLACE PROCEDURE insert_tuple(i IN NUMBER) AS
BEGIN
   -- Insert the tuple (i, 'xxx') into the table
   INSERT INTO my_table (col1, col2)
   VALUES (i, 'xxx');

   -- Commit the transaction (optional depending on your transaction management)
   COMMIT;

   -- Optionally, output a message (for debugging or confirmation)
   DBMS_OUTPUT.PUT_LINE('Inserted tuple (' || i || ', "xxx") into my_table.');
END;
/

BEGIN
   -- Call the procedure with i = 1
   insert_tuple(1);

   -- Call the procedure with i = 2
   insert_tuple(2);

   -- Call the procedure with i = 3
   insert_tuple(3);
END;
/
```

Output:

```
Inserted tuple (1, 'xxx') into my_table.
Inserted tuple (2, 'xxx') into my_table.
Inserted tuple (3, 'xxx') into my_table.
```

OUTPUT

col1	col2
1	xxx
2	xxx
3	xxx

5. Write a plsql program to demonstrate eceptions

Table

```
CREATE TABLE employees (
   emp_id NUMBER PRIMARY KEY,
   emp_name VARCHAR2(50)
);
```

PL/SQL Program to Demonstrate Exceptions:

```
DECLARE
   -- Define a variable to hold the employee ID
   v_emp_id NUMBER := 1;
   v_emp_name VARCHAR2(50);

   -- Declare a user-defined exception
   e_emp_not_found EXCEPTION;

BEGIN
   -- Attempt to select an employee with the given emp_id (which may not exist)
   BEGIN
      -- Trying to fetch employee name
      SELECT emp_name INTO v_emp_name FROM employees WHERE emp_id = v_emp_id;
   EXCEPTION
      -- If no data is found for the given emp_id
      WHEN NO_DATA_FOUND THEN
         DBMS_OUTPUT.PUT_LINE('Error: Employee with ID ' || v_emp_id || ' not found.');
         -- Raise a user-defined exception
         RAISE e_emp_not_found;
   END;

   -- Division by zero operation to trigger ZERO_DIVIDE exception
   BEGIN
      DECLARE
         v_result NUMBER;
      BEGIN
         v_result := 10 / 0;  -- This will raise a ZERO_DIVIDE exception
      EXCEPTION
         WHEN ZERO_DIVIDE THEN
            DBMS_OUTPUT.PUT_LINE('Error: Division by zero.');
      END;
   END;

   -- Raise a user-defined exception explicitly
   BEGIN
      -- Raise the user-defined exception
      RAISE e_emp_not_found;
   EXCEPTION
      WHEN e_emp_not_found THEN
         DBMS_OUTPUT.PUT_LINE('User-defined exception: Employee not found.');
   END;

   -- Simulating another operation (successful one)
   BEGIN
      -- Insert operation (inserting a new employee)
```

```
        INSERT INTO employees (emp_id, emp_name) VALUES (v_emp_id, 'John Doe');
        DBMS_OUTPUT.PUT_LINE('Employee inserted successfully: ID = ' || v_emp_id || ', Name = John Doe');
    EXCEPTION
        WHEN DUP_VAL_ON_INDEX THEN
            DBMS_OUTPUT.PUT_LINE('Error: Employee with ID ' || v_emp_id || ' already exists.');
    END;
        -- Commit the transaction
    COMMIT;
END;
/
```

Expected Output:

```
Error: Employee with ID 1 not found.
User-defined exception: Employee not found.
Error: Division by zero.
Employee inserted successfully: ID = 1, Name = John Doe
Error: Employee with ID 1 already exists.
```

6. Write a pl/sql program to demonstrate cursors

Table:

```
CREATE TABLE employees (
    emp_id NUMBER PRIMARY KEY,
    emp_name VARCHAR2(50),
    salary NUMBER
);

DECLARE
    -- Declare a cursor to fetch employees' details (emp_id, emp_name, salary)
    CURSOR emp_cursor IS
        SELECT emp_id, emp_name, salary FROM employees;

    -- Declare variables to hold the fetched data
    v_emp_id employees.emp_id%TYPE;
    v_emp_name employees.emp_name%TYPE;
    v_salary employees.salary%TYPE;

BEGIN
    -- Open the cursor to fetch the data
    OPEN emp_cursor;

    -- Loop through each row fetched by the cursor
    LOOP
        -- Fetch the next row into variables
        FETCH emp_cursor INTO v_emp_id, v_emp_name, v_salary;

        -- Exit the loop if no more rows are available
        EXIT WHEN emp_cursor%NOTFOUND;
        -- Display the fetched data
        DBMS_OUTPUT.PUT_LINE('Employee ID: ' || v_emp_id || ', Name: ' || v_emp_name || ', Salary: ' || v_salary);
    END LOOP;
    -- Close the cursor when done
    CLOSE emp_cursor;
END;
```

Expected Output:

Assuming the `employees` table contains the following data:

emp_id	emp_name	salary
1	John Doe	50000
2	Jane Smith	60000
3	Jim Brown	55000

7. Write a pl/sql program to demonstrate Functions

Table:

```
CREATE TABLE employees (
   emp_id NUMBER PRIMARY KEY,
   emp_name VARCHAR2(50),
   basic_salary NUMBER,
   bonus NUMBER
);
```

Program:

```
-- Create a function to calculate the monthly salary of an employee
CREATE OR REPLACE FUNCTION calculate_monthly_salary(
   p_emp_id IN NUMBER -- Input parameter: Employee ID
) RETURN NUMBER IS
   v_basic_salary employees.basic_salary%TYPE; -- Variable to store basic salary
   v_bonus employees.bonus%TYPE; -- Variable to store bonus
   v_monthly_salary NUMBER; -- Variable to store the calculated monthly salary
BEGIN
   -- Retrieve basic salary and bonus for the employee
   SELECT basic_salary, bonus
   INTO v_basic_salary, v_bonus
   FROM employees
   WHERE emp_id = p_emp_id;

   -- Calculate the monthly salary (basic salary + annual bonus / 12)
   v_monthly_salary := v_basic_salary + (v_bonus / 12);

   -- Return the calculated monthly salary
   RETURN v_monthly_salary;
EXCEPTION
   -- Handle the case where no employee is found
   WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('Employee with ID ' || p_emp_id || ' not found.');
      RETURN NULL;
   -- Handle any other unexpected errors
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
      RETURN NULL;
END calculate_monthly_salary;
/
```

Output:

Monthly Salary for Employee 1: 60000

8. Write a pl/sql program to demonstrate packages

Create the Package

```
CREATE OR REPLACE PACKAGE BODY employee_pkg AS
   -- Implementation of add_employee procedure
   PROCEDURE add_employee(p_emp_id IN NUMBER, p_emp_name IN VARCHAR2, p_salary IN NUMBER) IS
   BEGIN
      INSERT INTO employees (emp_id, emp_name, salary)
      VALUES (p_emp_id, p_emp_name, p_salary);
      COMMIT;
   EXCEPTION
      WHEN DUP_VAL_ON_INDEX THEN
         DBMS_OUTPUT.PUT_LINE('Error: Employee with ID ' || p_emp_id || ' already exists.');
      WHEN OTHERS THEN
         DBMS_OUTPUT.PUT_LINE('An error occurred while adding the employee: ' || SQLERRM);
   END add_employee;

   -- Implementation of get_salary function
   FUNCTION get_salary(p_emp_id IN NUMBER) RETURN NUMBER IS
      v_salary NUMBER;
   BEGIN
      SELECT salary INTO v_salary
      FROM employees
      WHERE emp_id = p_emp_id;

      RETURN v_salary;
   EXCEPTION
      WHEN NO_DATA_FOUND THEN
         DBMS_OUTPUT.PUT_LINE('Error: Employee with ID ' || p_emp_id || ' not found.');
         RETURN NULL;
      WHEN OTHERS THEN
         DBMS_OUTPUT.PUT_LINE('An error occurred while fetching salary: ' || SQLERRM);
         RETURN NULL;
   END get_salary;

   -- Implementation of update_salary procedure
   PROCEDURE update_salary(p_emp_id IN NUMBER, p_new_salary IN NUMBER) IS
   BEGIN
      UPDATE employees
      SET salary = p_new_salary
      WHERE emp_id = p_emp_id;

      COMMIT;
   EXCEPTION
      WHEN NO_DATA_FOUND THEN
         DBMS_OUTPUT.PUT_LINE('Error: Employee with ID ' || p_emp_id || ' not found.');
      WHEN OTHERS THEN
         DBMS_OUTPUT.PUT_LINE('An error occurred while updating salary: ' || SQLERRM);
   END update_salary;

END employee_pkg;
/
```

Or

```
DECLARE
   v_emp_id NUMBER := 1;
   v_emp_name VARCHAR2(50) := 'John Doe';
   v_salary NUMBER := 50000;
   v_new_salary NUMBER := 60000;
   v_current_salary NUMBER;
BEGIN
   -- Add a new employee
   employee_pkg.add_employee(v_emp_id, v_emp_name, v_salary);

   -- Get the current salary of the employee
   v_current_salary := employee_pkg.get_salary(v_emp_id);
   DBMS_OUTPUT.PUT_LINE('Current Salary for Employee ' || v_emp_id || ': ' || v_current_salary);

   -- Update the salary of the employee
   employee_pkg.update_salary(v_emp_id, v_new_salary);

   -- Get the updated salary of the employee
   v_current_salary := employee_pkg.get_salary(v_emp_id);
   DBMS_OUTPUT.PUT_LINE('Updated Salary for Employee ' || v_emp_id || ': ' || v_current_salary);

END;
/
```

Output:

```
Current Salary for Employee 1: 50000
Updated Salary for Employee 1: 60000
```

9. Write a PL/SQL QUERIES TO CREATE PROCEDURES

```
DECLARE
   v_emp_id NUMBER := 1;
   v_emp_name VARCHAR2(50) := 'John Doe';
   v_salary NUMBER := 50000;
   v_new_salary NUMBER := 60000;
BEGIN
   -- Call the add_employee procedure to add a new employee
   add_employee(v_emp_id, v_emp_name, v_salary);

   -- Call the update_salary procedure to update the salary of the employee
   update_salary(v_emp_id, v_new_salary);

   -- Try to add the same employee again to see error handling
   add_employee(v_emp_id, 'Jane Smith', 70000);

   -- Try to update the salary of a non-existing employee
   update_salary(999, 80000); -- Employee with ID 999 does not exist
END;
/
```

Output:

```
Employee added successfully!
Salary updated successfully for Employee 1
Error: Employee with ID 1 already exists.
Error: No employee found with ID 999
```

10. Write a pl/sql program to demonstrate triggers

```
CREATE OR REPLACE TRIGGER trg_salary_update
BEFORE UPDATE OF salary ON employees
FOR EACH ROW
DECLARE
   v_emp_id employees.emp_id%TYPE;
   v_old_salary employees.salary%TYPE;
   v_new_salary employees.salary%TYPE;
BEGIN
   -- Log the old and new salary in the audit_salary_changes table before updating
   v_emp_id := :OLD.emp_id;  -- Use OLD to refer to the original value before update
   v_old_salary := :OLD.salary;
   v_new_salary := :NEW.salary;  -- Use NEW to refer to the new value after update

   INSERT INTO audit_salary_changes (audit_id, emp_id, old_salary, new_salary, change_date)
   VALUES (audit_salary_changes_seq.NEXTVAL, v_emp_id, v_old_salary, v_new_salary, SYSDATE);

   DBMS_OUTPUT.PUT_LINE('Salary update detected for Employee ID ' || v_emp_id ||
                        '. Old Salary: ' || v_old_salary || ', New Salary: ' || v_new_salary);
END trg_salary_update;
/

SELECT * FROM audit_salary_changes;
```

Output:

audit_id	emp_id	old_salary	new_salary	change_date
1	1	50000	55000	2025-01-16 10:00:00