Table of Contents

1.0 Introduction

The Payroll Management System is a tailored software solution aimed at optimizing payroll operations in organizations by automating core processes related to employee compensation. This system is built with a comprehensive suite of functions and procedures designed to streamline tasks such as employee data management, payslip generation, and record maintenance, essentially minimizing the manual workload on Human Resource departments in addition to ensuring data accuracy.

The structure of the Payroll Management System is centered around an intuitive Main Menu that directs users to a variety of submenus, functions and procedures. This menu-driven design allows users to efficiently navigate through the system's primary capabilities, including managing employee records, calculating and generating payslips, and accessing historical payslip data. Key functions within the system are as follows:

1. **Main Menu**: Acting as the system's core interface, this function displays the main options available to users, including access to submenus for employee and payslip management, as well as options for generating payslips and exiting the program.
2. **Submenus**: Dedicated submenus provide focused access to specific areas within the system, namely the Employee Profile Submenu for managing employee records, and the Payslip Submenu for accessing payslip functions and procedures.
3. **Add Employee Procedure:** This procedure allows users to add new employees to the system by entering key details such as employee ID, name, department, basic salary, allowance, bonus, and overtime amount. Data validation is incorporated to maintain accuracy and consistency in employee information. Additionally, this function is also called each time a new monthly payslip is generated for an employee, ensuring that all relevant salary details are accurately recorded for each month.
4. **Update Employee Procedure**: This procedure allows users to modify existing employee information, including components such as name, department, and salary details—basic salary, allowance, bonus, and overtime—for a specified month and year. Updates can only be applied if a payslip record exists for the selected month and year. This requirement prevents users from updating information for periods that have not been recorded yet, essentially maintaining consistency with existing payroll records. When employee

information is updated, the previous record is overwritten, ensuring data remains current and free of duplication.

5. **Delete Employee Procedure**: This procedure provides a secure way to remove employee records, ensuring that all associated data including employee details and corresponding payslip records are fully deleted. It serves to maintain data integrity and to prevent any residual records.
6. **View Employee Procedure**: This procedure enables users to view a list of all employees in the system, including key information namely employee ID, name, and department. It provides a quick reference to existing records and supports easy information retrieval.
7. **Generate Payslip Procedure**: This interactive feature allows for quick computation of detailed payslip data for employees, factoring in components namely basic salary, allowances, bonuses, and deductions. It automates complex calculations to ensure accurate net salary results; however, the computed data will not be stored in the system.
8. **Search Specific Payslip Procedure**: Designed for targeted information retrieval, this procedure allows users to search for a particular employee's payslip based on a specified month and year. This is especially useful for records verification or individual employee inquiries.
9. **View All Payslips Procedure**: This procedure provides an overview of all payslips for a specific employee, organized by year and displayed in descending order. It allows users to review historical payroll data efficiently.

1.1 Report Structure

The report continues with an outline of the assumptions governing the Payroll Management System, defining the foundational rules and conditions under which the system operates. Following this, the system's design is documented through detailed pseudocode for each function, accompanied by brief explanations to clarify each component's purpose.

The report then presents a discussion of the source code, with annotated screenshots to provide an in-depth look at the programming logic behind each function. To demonstrate the system's functionality, screenshots of sample input and output are included, illustrating the expected results

of each feature. The report concludes with a summary that reflects the system's contributions to efficient payroll management and highlights areas for potential improvement.

2.0 Assumptions

The following assumptions govern the Payroll Management System:

1. Each employee has a unique identification number, which serves as the primary identifier to distinguish individual records. Employee IDs are assumed to be in string format and treated consistently as a string throughout the program.
2. The system supports a maximum of 99,999 employees, with IDs ranging from "TP00001" to "TP99999."
3. The gross salary, represented by *grossSalary* in the source code is computed as $Gross\ Salary = (\text{Basic Salary} + \text{Allowance} + \text{Bonus} + \text{Overtime}) * 0.89$. This formula applies an 11% deduction for the Employees' Provident Fund (EPF) on the sum of the employee's basic salary, allowance, bonus, and overtime. A consistent 11% EPF deduction is assumed in all salary calculations.
4. The net salary, indicated by *netSalary* in the source code, reflects the final salary after considering applicable bonuses and taxes. Specifically, a 5% bonus is applied if the gross salary is below RM2000, a 6% tax deduction applies if the gross salary exceeds RM3000, and salaries between RM2000 and RM3000 are unaffected by either bonus or tax adjustments.
5. All values for calculating gross and net salaries (i.e., basic salary, allowance, bonus, and overtime) are assumed to be accurate and up to date. These components are treated as amounts rather than rates, and the currency is assumed to be Ringgit Malaysia (RM).
6. Employee and salary records are stored in JSON files (*employees.json* and *payslips.json*). It is assumed that these files are always available and accessible for program operations.
7. The program is designed for single-user operation, with no concurrent access.
8. The system assumes no security threats, such as unauthorized access or malicious input. In a real-world application, security measures such as authentication and authorization would be necessary.
9. In this report, the terms "Function" and "Procedure" will be used to reflect a programming distinction - functions perform tasks and return a result, while procedures perform tasks without returning a result. For simplicity, both can be understood as ways to carry out specific tasks within a program, helping to streamline the overall flow and structure.

3.0 Design of the Program (Pseudocodes)

3.1 Procedure *loadData*

```
----------------------------------------PROCEDURE loadData----------------------------------------

PROCEDURE loadData()
BEGIN
    GLOBAL employees, payslips

    IF 'employees.json' file EXISTS THEN
        OPEN 'employees.json' file for reading AS f
        READ data from 'employees.json' file and LOAD into employees dictionary using JSON format
        CLOSE 'employees.json' file
    END IF

    IF 'payslips.json' file EXISTS THEN
        OPEN 'payslips.json' file for reading AS f
        READ data from 'payslips.json' file and LOAD into payslips dictionary using JSON format
        CLOSE 'payslips.json' file
    END IF
END PROCEDURE
```

The *loadData* procedure loads existing employee and payslip data from *employees.json* and *payslips.json* files into the *employees* and *payslips* dictionaries. It checks if each file exists, reads its contents, and populates the respective dictionaries, ensuring data persistence across sessions.

3.2 Procedure *saveData*

```
----------------------------------------PROCEDURE saveData----------------------------------------

PROCEDURE saveData()
BEGIN
    GLOBAL employees, payslips

    OPEN 'employees.json' file for writing AS f
    WRITE the current state of employees dictionary to 'employees.json' file using JSON format
    CLOSE 'employees.json' file

    OPEN 'payslips.json' file for writing AS f
    WRITE the current state of payslips dictionary to 'payslips.json' file using JSON format
    CLOSE 'payslips.json' file
END PROCEDURE
```

The *saveData* procedure saves the current state of the *employees* and *payslips* dictionaries to *employees.json* and *payslips.json* files, respectively. It writes the data in JSON format, ensuring that updates to employees and payslip information are preserved.

3.3 Function *promptIntergerInput*

```
-----------------------------------FUNCTION promptIntegerInput--------------------------------

FUNCTION promptIntegerInput(action)
BEGIN
    LOOP indefinitely
        TRY
            PROMPT user for input with message specified by action
            CONVERT user input to an integer and STORE in number

            IF number < 0 THEN
                PRINT "Invalid input. Please ensure that input is a non-negative number."
                CONTINUE to the next iteration of the loop
            END IF

            RETURN number as valid input
        CATCH ValueError
            PRINT "Invalid input. Please reenter a valid number."
        END TRY
    END LOOP
END FUNCTION
```

The *promptIntegerInput* function prompts the user for an integer input and ensures it is a non-negative number. It continuously loops until valid input is provided, handling errors if the input is not an integer, and prompting the user to re-enter if necessary.

3.4 Function *promptFloatInput*

```
-----------------------------------FUNCTION promptFloatInput--------------------------------

FUNCTION promptFloatInput(action)
BEGIN
    LOOP indefinitely
        TRY
            PROMPT user for input with message specified by action
            CONVERT user input to a float and STORE in number

            IF number < 0 THEN
                PRINT "Invalid input. Please ensure that input is a non-negative number."
                CONTINUE to the next iteration of the loop
            END IF

            RETURN number as valid input
        CATCH ValueError
            PRINT "Invalid input. Please reenter a valid number."
        END TRY
    END LOOP
END FUNCTION
```

The *promptFloatInput* function prompts the user for a float input, ensuring it is a non-negative number. It repeatedly requests input until a valid float is provided, handling errors if the input is not a float, and prompting the user to re-enter if necessary.

3.5 Function *promptStringsInput*

```
-----------------------------------FUNCTION promptStringsInput---------------------------------

FUNCTION promptStringsInput(action)
BEGIN
    LOOP indefinitely
        PROMPT user for input with message specified by action and STORE in string

        IF string contains characters other than alphabetic letters and spaces THEN
            PRINT "Invalid input. The string must contain only letters and spaces."
            CONTINUE to the next iteration of the loop
        END IF

        RETURN string as valid input
    END LOOP
END FUNCTION
```

The *promptStringsInput* function prompts the user for a string input, ensuring it contains only alphabetic characters and spaces. It loops until valid input is received, rejecting entries with invalid characters, and prompting the user to re-enter.

3.6 Function *promptEmpIDInput*

```
-----------------------------------FUNCTION promptEmpIDInput-----------------------------------

FUNCTION promptEmpIDInput(prompt_text = "Enter Employee ID (e.g., 'TP00001')")
BEGIN
    LOOP indefinitely
        PROMPT user for Employee ID with message specified by prompt_text
        REMOVE extra spaces, CONVERT input to uppercase, and STORE in empID

        IF empID starts with "TP" AND length of empID is 7 AND last 5 characters are digits THEN
            RETURN empID as valid input
        ELSE
            PRINT "Invalid Employee ID format. Please enter an ID in the format 'TP00001', 'TP00002', etc."
        END IF
    END LOOP
END FUNCTION
```

The *promptEmpIDInput* function prompts the user for an employee ID, enforcing a specific format (“TP” followed by 5 digits, e.g., “TP00001”). It removes extra spaces, converts input to uppercase, and loops until a valid ID is provided, displaying an error message for incorrect formats.

3.7 Procedure *updateEmployee*

```
----------------------------------------PROCEDURE updateEmployee-----------------------------------

PROCEDURE updateEmployee(empID = None, year = None, month = None)
BEGIN
    GLOBAL employees, payslips

    IF empID is not provided THEN
        PROMPT user for Employee ID using standardized input function
        STORE result in empID
    END IF

    IF month is not provided THEN
        PROMPT user for month input and capitalize it
        DEFINE valid_months as a list of month names
        WHILE month is not in valid_months DO
            PRINT "Invalid month. Please enter a valid month name."
            PROMPT user again for month input and capitalize it
        END WHILE
    END IF

    IF year is not provided THEN
        PROMPT user for year input
        WHILE year is not a 4-digit number DO
            PRINT "Invalid year. Please enter a 4-digit year."
            PROMPT user again for year input
        END WHILE
        CONVERT year to integer and STORE in year
    END IF

    IF empID, year, or month record is not found in payslips THEN
        PRINT "No payslip record found for specified month and year."
        RETURN from procedure without proceeding
    END IF
```

```
     PRINT "Existing Payslip Data for Reference"
     DISPLAY existing payslip data in a table format for user reference

     PROMPT user for updated Basic Salary Amount and STORE in basicSalary
     PROMPT user for updated Allowance Amount and STORE in allowance
     PROMPT user for updated Bonus Amount and STORE in bonus
     PROMPT user for updated Overtime Amount and STORE in overtime

     CALCULATE net salary and other salary components using calculateNetSalary function
     STORE result in salaryDetails

     DISPLAY updated details in a table format for user confirmation

     PROMPT user for confirmation to overwrite the existing payslip details
     IF confirmation is not 'yes' THEN
         PRINT "Employee update was canceled. No changes were saved."
         RETURN from procedure without saving changes
     END IF

     UPDATE payslips dictionary with new salaryDetails for empID, year, and month

     CALL saveData procedure to persist updated information to JSON files

     PRINT "Employee data has been successfully updated."
 END PROCEDURE
```

The *updateEmployee* procedure allows users to modify an employee's information for a specific month and year, including basic salary, allowance, bonus, and overtime. If the employee ID, month, or year is not provided, the procedure prompts the user to enter them, ensuring valid inputs. The procedure checks that a payslip record exists for the specified period before proceeding. It then displays the current payslip data, allows the user to update the relevant fields, and calculates the new net salary. After confirmation, the updated details overwrite the previous record, and changes are saved to the JSON files.

3.8 Procedure *addEmployee*

```
----------------------------------------PROCEDURE addEmployee----------------------------------------

PROCEDURE addEmployee()
BEGIN
    GLOBAL employees, payslips

    PROMPT user for Employee ID using standardized input function and STORE in empID

    IF empID exists in employees dictionary THEN
        PRINT "This Employee ID already exists. Adding a new payslip record for this employee."
        RETRIEVE employeeName and departmentName from existing employee record
    ELSE
        PRINT "This Employee ID does not exist yet. Adding it to the system."
        PROMPT user for Employee Name using standardized input function
        PROMPT user for Department Name using standardized input function
        STORE results in employeeName and departmentName
    END IF

    PROMPT user for month and capitalize it
    DEFINE valid_months as a list of month names
    WHILE month is not in valid_months DO
        PRINT "Invalid month. Please enter a valid month name."
        PROMPT user again for month input and capitalize it
    END WHILE

    PROMPT user for year input
    WHILE year is not a 4-digit number DO
        PRINT "Invalid year. Please enter a 4-digit year."
        PROMPT user again for year input
    END WHILE
    CONVERT year to integer
```

```
    IF empID is not in payslips dictionary THEN
        INITIALIZE empty dictionary for empID in payslips
    END IF

    IF year is not in payslips[empID] dictionary THEN
        INITIALIZE empty dictionary for year in payslips[empID]
    END IF

    IF month exists in payslips[empID][year] THEN
        PRINT "A payslip record for this month and year already exists."
        PROMPT user for confirmation to update existing record
        IF user input is 'yes' THEN
            CALL updateEmployee procedure with empID, year, and month
        ELSE
            PRINT "No changes made. Returning to the main menu."
        END IF
        RETURN from procedure
    END IF

    PROMPT user for Basic Salary Amount and STORE in basicSalary
    PROMPT user for Allowance Amount and STORE in allowance
    PROMPT user for Bonus Amount and STORE in bonus
    PROMPT user for Overtime Amount and STORE in overtime
```

```
    CALCULATE net salary and related details using calculateNetSalary function
    STORE result in salaryDetails

    PROMPT user for confirmation to save new record
    IF user input is not 'yes' THEN
        PRINT "Employee data and salary details were not saved."
        RETURN from procedure
    END IF

    IF empID is not in employees dictionary THEN
        ADD new employee record with employeeName and departmentName to employees
    END IF

    ADD salaryDetails to payslips dictionary for empID, year, and month

    CALL saveData procedure to persist data to JSON files

    PRINT "Employee data has been successfully added and salary details saved."
END PROCEDURE
```

The *addEmployee* procedure allows users to add a new employee or a new monthly payslip for an existing employee. It begins by prompting an employee ID, checks if the ID already exists, and retrieves or requests additional employee details (name, department) as needed. The procedure then prompts for the month and year, validates these inputs, and checks if a payslip already exists for the specified period. If no existing record is found, the user can enter salary components (basic salary, allowance, bonus, and overtime), and the procedure calculates the net salary. After confirmation, the new record is saved to the JSON files, ensuring persistence.

3.9 Procedure *deleteEmployee*

```
----------------------------------------PROCEDURE deleteEmployee-----------------------------------

PROCEDURE deleteEmployee()
BEGIN
    GLOBAL employees, payslips

    PROMPT user for Employee ID using standardized input function and STORE in empID

    IF empID exists in employees dictionary THEN
        DELETE empID from employees dictionary
        PRINT "Employee data for empID has been successfully deleted from the employees dictionary."

        IF empID exists in payslips dictionary THEN
            DELETE empID from payslips dictionary
            PRINT "Employee salary records for empID have been successfully deleted from the payslips dictionary."
        ELSE
            PRINT "No salary records found for empID in the payslips dictionary."
        END IF

        CALL saveData procedure to persist deletions to JSON files
        PRINT "All changes have been successfully saved."
    ELSE
        PRINT "Employee ID does not exist!"
    END IF
END PROCEDURE
```

The *deleteEmployee* procedure removes an employee's data from both the *employees* and *payslips* dictionaries based on the provided employee ID. If the ID exists in *employees*, it deletes the record and confirms the deletion. If the ID is also found in *payslips*, it deletes the associated salary records; otherwise, it notifies the user that no salary records were found. The procedure then calls *saveData* to persist these deletions in the JSON files, ensuring that all changes are saved.

3.10 Procedure *viewEmployee*

```
---------------------------------------PROCEDURE viewEmployee-------------------------------------

PROCEDURE viewEmployee()
BEGIN
    GLOBAL employees

    IF employees dictionary is empty THEN
        PRINT "No employee records found."
        RETURN from procedure
    END IF

    CREATE PrettyTable object named employeeTable
    SET column headers of employeeTable to ["Employee ID", "Name", "Department"]

    FOR each empID, details in employees dictionary DO
        ADD a row to employeeTable with empID, details['Employee Name'], and details['Department Name']
    END FOR

    PRINT "Employee Records:"
    DISPLAY employeeTable
END PROCEDURE
```

The *viewEmployee* procedure displays a list of all employee records stored in the *employees* dictionary. If no records are found, it notifies the user and exits. Otherwise, it creates a formatted table with columns for Employee ID, Name, and Department, populates it with employee data, and then displays the table to the user.

3.11 Function *calculateNetSalary*

```
------------------------------------FUNCTION calculateNetSalary----------------------------------

FUNCTION calculateNetSalary(basicSalary, allowance, bonus, overtime)
BEGIN
    CALCULATE grossSalary AS basicSalary + allowance + bonus + overtime

    CALCULATE epfDeduction AS grossSalary * 0.11
    CALCULATE salaryAfterEPF AS grossSalary - epfDeduction

    SET additionalBonus TO 0
    SET taxDeduction TO 0

    IF salaryAfterEPF < 2000 THEN
        CALCULATE additionalBonus AS salaryAfterEPF * 0.05
    ELSE IF salaryAfterEPF > 3000 THEN
        CALCULATE taxDeduction AS salaryAfterEPF * 0.06
    END IF

    CALCULATE netSalary AS salaryAfterEPF + additionalBonus - taxDeduction

    RETURN dictionary WITH:
        'Basic Salary' = basicSalary,
        'Allowance' = allowance,
        'Bonus' = bonus,
        'Overtime' = overtime,
        'Gross Salary' = grossSalary,
        'EPF Deduction' = epfDeduction,
        'Additional Bonus' = additionalBonus,
        'Tax Deduction' = taxDeduction,
        'Net Salary' = netSalary
END FUNCTION
```

The *calculateNetSalary* function computes an employee's net salary based on their basic salary, allowance, bonus, and overtime. It first calculates the gross salary and applies an 11% EPF deduction. Depending on the remaining salary, a 5% bonus is added if it is below RM2000, or a 6% tax is deducted if it exceeds RM3000. The net salary is then derived by adjusting for these additions and deductions. The function returns a dictionary containing all salary components, including gross salary, EPF deduction, additional bonus, tax deduction, and net salary.

3.12 Procedure *generateNetSalary*

```
-----------------------------------PROCEDURE generateNetSalary---------------------------------

PROCEDURE generateNetSalary()
BEGIN
    GLOBAL employees

    SET empID TO result of promptEmpIDInput("Enter Employee ID: ")

    IF empID NOT IN employees THEN
        PRINT "Employee ID not found. Please check and try again."
        RETURN from procedure
    END IF

    SET month TO result of promptStringsInput("Enter Month (e.g., 'January', 'February'): ").capitalize()
    DEFINE valid_months AS ['January', 'February', 'March', 'April', 'May', 'June',
                            'July', 'August', 'September', 'October', 'November', 'December']
    WHILE month NOT IN valid_months DO
        PRINT "Invalid month. Please enter a valid month name."
        SET month TO result of promptStringsInput("Enter Month (e.g., 'January', 'February'): ").capitalize()
    END WHILE

    SET year_input TO input("Enter Year (e.g., 2023): ")
    WHILE year_input is NOT a 4-digit number DO
        PRINT "Invalid year. Please enter a 4-digit year."
        SET year_input TO input("Enter Year (e.g., 2023): ")
    END WHILE
    CONVERT year_input TO integer and STORE in year

    SET basicSalary TO result of promptFloatInput("Enter Basic Salary Amount: ")
    SET allowance TO result of promptFloatInput("Enter Allowance Amount: ")
    SET bonus TO result of promptFloatInput("Enter Bonus Amount: ")
    SET overtime TO result of promptFloatInput("Enter Overtime Amount: ")
```

```
    CALL calculateNetSalary(basicSalary, allowance, bonus, overtime) and STORE result in salaryDetails

    PRINT "Calculated Salary Details:"
    FOR each key, value IN salaryDetails DO
        PRINT key and formatted value
    END FOR

    PRINT "Net salary calculation complete (data not saved)."
END PROCEDURE
```

The *generateNetSalary* procedure calculates and displays an employee's net salary based on their monthly salary components. It prompts the user to enter the employee ID, month, year, basic salary, allowance, bonus, and overtime, validating each input. Once all inputs are collected, the procedure calls *calculateNetSalary* to compute the net salary. It then displays the calculated details to the user without saving the data, allowing for quick, ad-hoc salary calculations.

3.13 Procedure *searchSpecificPayslip*

```
----------------------------------PROCEDURE searchSpecificPayslip----------------------------------

PROCEDURE searchSpecificPayslip()
BEGIN
    GLOBAL payslips

    WHILE TRUE DO
        SET empID TO result of promptEmpIDInput("Enter Employee ID: ")

        IF empID NOT IN payslips THEN
            PRINT "No records found for this Employee ID."
            SET retry TO input("Would you like to try again with a different Employee ID? (yes/no): ").strip().lower()
            IF retry != 'yes' THEN
                PRINT "Returning to the main menu."
                RETURN from procedure
            ELSE
                CONTINUE to prompt for Employee ID again
            END IF
        END IF

        SET month TO result of promptStringsInput("Enter Month (e.g., 'January', 'February'): ").capitalize()
        DEFINE valid_months AS ['January', 'February', 'March', 'April', 'May', 'June',
                                'July', 'August', 'September', 'October', 'November', 'December']

        WHILE month NOT IN valid_months DO
            PRINT "Invalid month. Please enter a valid month name (e.g., 'January')."
            SET month TO result of promptStringsInput("Enter Month (e.g., 'January', 'February'): ").capitalize()
        END WHILE

         SET year_input TO input("Enter Year (e.g., 2023): ")
         WHILE year_input is NOT a 4-digit number DO
             PRINT "Invalid year. Please enter a 4-digit year."
             SET year_input TO input("Enter Year (e.g., 2023): ")
         END WHILE
         CONVERT year_input TO integer and STORE in year

         IF year NOT IN payslips[empID] OR month NOT IN payslips[empID][year] THEN
             PRINT "No payslip record found for the specified month and year for Employee ID empID."
             RETURN from procedure
         END IF

         SET payslip_details TO payslips[empID][year][month]

         CREATE PrettyTable object named payslip_table
         SET column headers of payslip_table to ["Detail", "Value"]

         FOR each key, value IN payslip_details DO
             ADD row to payslip_table with key and formatted value
         END FOR

         PRINT "Payslip for Employee ID: empID for month and year"
         DISPLAY payslip_table
         RETURN from procedure
    END WHILE
END PROCEDURE
```

The *searchSpecificPayslip* procedure allows users to search for a specific payslip by employee ID, month, and year. It first prompts for the employee ID, giving the user the option to retry if no record is found. It then requests and validates the month and year inputs. If the specified payslip record exists, it displays the details in a formatted table. If not, the procedure informs the user and exits.

3.14 Procedure *viewAllPayslips*

```
-------------------------------------PROCEDURE viewAllPayslips--------------------------------------

PROCEDURE viewAllPayslips()
BEGIN
    GLOBAL payslips

    SET empID TO result of promptEmpIDInput("Enter Employee ID: ")

    IF empID NOT IN payslips THEN
        PRINT "No payslip records found for this Employee ID."
        SET retry TO input("Would you like to try again with a different Employee ID? (yes/no): ").strip().lower()
        IF retry != 'yes' THEN
            PRINT "Returning to the main menu."
            RETURN from procedure
        ELSE
            CALL viewAllPayslips() to retry with a different Employee ID
        END IF
    END IF

    FOR each year IN sorted keys of payslips[empID] in descending order DO
        SET months TO payslips[empID][year]

        CREATE PrettyTable object named year_table
        SET column headers of year_table to ["Month", "Basic Salary", "Allowance", "Bonus", "Overtime",
                                             "Gross Salary", "EPF Deduction", "Additional Bonus", "Tax Deduction", "Net Salary"]

        FOR each month, details IN months DO
            ADD row to year_table with month and formatted values for each detail in details:
                "Basic Salary", "Allowance", "Bonus", "Overtime",
                "Gross Salary", "EPF Deduction", "Additional Bonus", "Tax Deduction", "Net Salary"
        END FOR


        PRINT "Payslip Summary for Employee ID: empID for Year: year"
        DISPLAY year_table
    END FOR
END PROCEDURE
```

The *viewAllPayslips* procedure displays all payslips for a specified employee, sorted by year in descending order. After prompting for an Employee ID, it checks for existing payslip records. If none are found, the user can retry with a different ID. For valid records, the procedure creates a table for each year, detailing monthly payslip components such as basic salary, allowances, bonuses, and deductions. Each year's data is displayed in an organized table format, allowing efficient review of the employee's payroll history.

3.15 Procedure *exit*

```
-----------------------------------------PROCEDURE exit-----------------------------------------

PROCEDURE exit()
BEGIN
    CALL saveData() to save any changes made to employees and payslips dictionaries

    PRINT "Exiting the program."

    RETURN from procedure
END PROCEDURE
```

The *exit* procedure saves any changes made to the *employees* and *payslips* dictionaries by calling the *saveData* procedure. It then displays a message indicating the program is closing and exits the program, ensuring data integrity before termination.

3.16 Procedure *empProfileSubMenu*

```
-----------------------------------PROCEDURE empProfileSubMenu-----------------------------------

PROCEDURE empProfileSubMenu()
BEGIN
    GLOBAL employees

    WHILE TRUE DO
        PRINT "\n------ Employee Profile ------"
        PRINT "1. Add Employee"
        PRINT "2. Update Employee"
        PRINT "3. Delete Employee"
        PRINT "4. View Employee List"
        PRINT "5. Return to Main Menu"

        SET option TO input("Enter your choice: ")

        IF option == "1" THEN
            CALL addEmployee()

        ELSE IF option == "2" THEN
            CALL updateEmployee()

        ELSE IF option == "3" THEN
            CALL deleteEmployee()

        ELSE IF option == "4" THEN
            CALL viewEmployee()

        ELSE IF option == "5" THEN
            BREAK from loop to return to Main Menu

        ELSE
            PRINT "Choice is invalid. Try entering choice again."

        END IF
    END WHILE
END PROCEDURE
```

The *empProfileSubMenu* procedure presents options for managing employee records, including adding, updating, deleting, and viewing employee information, as well as returning to the main menu. It operates in a loop, calling the relevant procedure based on the user's choice. If an invalid input is entered, it prompts for re-entry, ensuring smooth navigation.

3.17 Procedure *payslipSubMenu*

```
--------------------------------------PROCEDURE payslipSubMenu-----------------------------------------

PROCEDURE payslipSubMenu()
BEGIN
    WHILE TRUE DO
        PRINT "\n*** Pay Slip ***"
        PRINT "1. Search Specific Payslip"
        PRINT "2. View All Payslips"
        PRINT "3. Return to Main Menu"

        SET choice TO input("Enter your choice: ")

        IF choice == "1" THEN
            CALL searchSpecificPayslip()

        ELSE IF choice == "2" THEN
            CALL viewAllPayslips()

        ELSE IF choice == "3" THEN
            BREAK from loop to return to Main Menu

        ELSE
            PRINT "Invalid choice. Please try again."
        END IF
    END WHILE
END PROCEDURE
```

The *payslipSubMenu* procedure presents options for handling employee payslips. Users can choose to search for a specific payslip, view all payslips for a particular employee, or return to the main menu. The procedure operates within a loop, prompting the user for their choice, and calling the appropriate procedure based on the input. If an invalid option is selected, the procedure displays an error message and prompts the user to try again, ensuring correct input before proceeding.

3.18 Procedure *mainMenu*

```
-----------------------------------------PROCEDURE mainMenu-----------------------------------------

PROCEDURE mainMenu()
BEGIN
    CALL loadData() to load data from files into employees and payslips dictionaries

    WHILE TRUE DO
        PRINT "\n------- Main Menu -------"
        PRINT "1. Employee Profile"
        PRINT "2. Salary Generator"
        PRINT "3. Payslip Profile"
        PRINT "4. Exit"

        SET option TO input("Enter your option: ")

        IF option == "1" THEN
            PRINT "Employee Profile has been chosen"
            CALL empProfileSubMenu()

        ELSE IF option == "2" THEN
            PRINT "Salary Generator has been chosen"
            CALL generateNetSalary()

        ELSE IF option == "3" THEN
            PRINT "Payslip Profile has been chosen"
            CALL payslipSubMenu()

        ELSE IF option == "4" THEN
            PRINT "Exiting program."
            CALL exit()
```

```
        ELSE
            PRINT "Choice is invalid. Try entering choice again."
        END IF
    END WHILE
END PROCEDURE
```

The *mainMenu* procedure acts as the primary navigation point for the Payroll Management System. Users can access the Employee Profile submenu, Salary Generator, Payslip Profile submenu, or Exit the program. The procedure continuously prompts for input, calling the relevant procedure based on the user's choice. If an invalid option is selected, it displays an error and re-prompts for a valid choice, ensuring smooth navigation.

4.0 Source Code and Programming Concepts

4.1 Importing Packages and Creating Empty Dictionaries

```
# Importing relevant packages

# Import PrettyTable for generating and displaying tabular outputs in the console
from prettytable import PrettyTable

# Import json for storing and saving data into JSON files, enabling data persistence
import json

# Import os for interacting with the operating system, such as checking file existence and paths
import os


# Declaring an empty dictionary called employees to store all employee data
employees = {}

# Declaring an empty dictionary called payslips to store all employee salary data
payslips = {}
```

This section sets up essential libraries and data structures for the Payroll Management System, enabling data handling, formatting, and interaction with the operating system.

Key Programming Concepts:

1. **Package Imports** (*import prettytable*, *import json*, *import os*): These import statements load external libraries required for the program. *PrettyTable* helps display tabular data neatly in the console. *json* enables data storage and retrieval in JSON format, allowing persistence between sessions. *os* provides tools for interacting with the operating system, such as verifying file existence and handling paths.

2. **Global Dictionaries** (*employees*, *payslips*): Two global dictionaries, *employees* and *payslips*, are initialized as empty dictionaries. *employees* will store individual employee details, while *payslips* will hold salary-related data for each employee. Declaring these dictionaries globally makes them accessible across all functions within the program, ensuring consistency and ease of data management.

4.2 Displaying the Main Menu

```
# Define a procedure called mainMenu to display the main menu to the user
def mainMenu():
    # Load data from files into the employees and payslips dictionaries when the program starts
    loadData()

    # Infinite loop to display the main menu continuously until the user chooses to exit
    while True:
        # Displaying the main menu options
        print("\n------- Main Menu -------")
        print("1. Employee Profile")         # Option to access employee profile functions
        print("2. Salary Generator")         # Option to generate net salary for an employee
        print("3. Payslip Profile")          # Option to view or search payslip records
        print("4. Exit")                     # Option to exit the program

        # Prompt the user to enter their option
        option = input("Enter your option: ")

        # Handle each menu option with if-elif statements to call the corresponding function
        if option == "1":
            print("Employee Profile has been chosen")
            empProfileSubMenu()              # Calls the sub menu for employee profile procedures
        elif option == "2":
            print("Salary Generator has been chosen")
            generateNetSalary()              # Calls the procedure to calculate and display net salary
        elif option == "3":
            print("Payslip Profile has been chosen")
            payslipSubMenu()                 # Calls the sub menu for payslip procedures
```

```
        elif option == "4":
            print("Exiting program.")
            exit()                           # Calls the exit procedure to save data and terminate the program
        else:
            # Informs the user if their input is invalid and prompts them to re-enter a valid choice
            print("Choice is invalid. Try entering choice again.")
```

The *mainMenu()* procedure serves as the central interface for the Payroll Management System, guiding the user through available options and facilitating data interaction.

Key Programming Concepts:

1. **Procedure Definition (*def mainMenu()*):** The *mainMenu()* procedure is defined to encapsulate the main menu operations. This organization allows the procedure to be called whenever the main interface needs to be displayed.
2. **Loading Data:** At the start of the procedure, *loadData()* is called to initialize the *employees* and *payslips* dictionaries with data from external files, ensuring the program has access to any saved records at the outset.
3. **Infinite Loop (*while True*):** This loop continuously displays the main menu until the user selects the exit option. The loop structure ensures that after each action, the main menu reappears, keeping the program flow clear and uninterrupted.

4. **Option Display and User Input Prompt:** The menu options—Employee Profile, Salary Generator, Payslip Profile, and Exit—are displayed using *print()* statements. The *input()* function prompts the user to enter a choice, which is stored in the *option* variable.
5. **Conditional Execution with *if-elif-else*:** Based on the user's input, *if-elif-else* statements determine the next step. Each condition corresponds to one of the main menu options, executing the relevant function or procedures if a match is found.
6. **Calling Sub-function or Sub-Procedures:** The procedure calls other sub-function or sub-procedures such as *empProfileSubMenu()*, *generateNetSalary()*, and *payslipSubMenu()* to handle specific tasks based on user selection. This modular structure enhances code readability and maintainability.
7. **Exiting the Program (*exit()*):** If the user selects option 4 (Exit), the *exit()* procedure is called. This procedure saves any changes made to the dictionaries and terminates the program, providing a clean exit from the system.
8. **Input Validation:** If the user enters an invalid choice, an error message is displayed, prompting the user to re-enter a valid option. This ensures that only valid inputs are processed, preventing any unexpected behaviour in the program.

4.3 Displaying the Submenu for Employee Profile

```
# Define a procedure called empProfileSubMenu to display the sub menu for Employee Profile.
def empProfileSubMenu():
    # Accessing the global variable to ensure any changes in employee data are updated
    global employees

    # Creating an infinite loop to keep displaying the sub menu until the user decides to return to the main menu
    while True:
        # Displaying the options that the user can choose from in the Employee Profile sub menu
        print("\n------ Employee Profile ------")
        print("1. Add Employee")
        print("2. Update Employee")
        print("3. Delete Employee")
        print("4. View Employee List")
        print("5. Return to Main Menu")

        # Prompt the user to enter their option
        option = input("Enter your choice: ")

        # Process the user's choice using if-elif-else statements
        if option == "1":
            addEmployee()  # Calls addEmployee procedure to add a new employee

        elif option == "2":
            updateEmployee()  # Calls updateEmployee procedure for modifying existing employee data

        elif option == "3":
            deleteEmployee()  # Calls deleteEmployee procedure to remove an employee record
```

```
        elif option == "4":
            viewEmployee()  # Calls viewEmployee procedure to display all employee records

        elif option == "5":
            # Exit the loop to return to the Main Menu
            break

        else:
            # Prompt for a valid choice if input is other than "1", "2", "3", "4", or "5"
            print("Choice is invalid. Try entering choice again.")
```

The *empProfileSubMenu()* procedure displays a submenu for managing employee records. Users can choose to add a new employee, update an existing employee, delete an employee record, view the entire employee list, or return to the main menu. This procedure operates within an infinite loop, ensuring the submenu reappears after each action until the user opts to return to the main menu.

Key Programming Concepts:

1. **Procedure Definition (*def empProfileSubMenu()*):** This line defines the *empProfileSubMenu()* procedure, encapsulating the code related to the employee profile submenu.

2. **Global Variable Access (*global employees*):** By declaring *employees* as global, the procedure ensures that any updates made to employee records within this submenu affect the main program's data.

3. **Infinite Loop (*while True*):** An infinite loop maintains the submenu display, allowing users to execute multiple operations without returning to the main menu until they choose to exit.

4. **Options Display (*print()* statements):** The available options (Add, Update, Delete, View Employee List, and Return to Main Menu) are displayed to the user, providing a clear list of choices.

5. **User Input Handling (*input()*):** The *option* variable captures the user's choice, which is then evaluated to determine the action taken.

6. **Conditional Statements (*if-elif-else*):** These statements process the user's selection. Depending on the input, the procedure calls *addEmployee()*, *updateEmployee()*, *deleteEmployee()*, or *viewEmployee()* to perform the relevant task. If the user selects an invalid option, an error message prompts them to re-enter a choice.

7. **Returning to Main Menu (*break* statement):** If the user selects the option to return to the main menu, the *break* statement exits the loop, allowing the main menu to display again.

4.4 Displaying the Submenu for Payslip Option

```
# Define a procedure called payslipSubMenu to display the sub menu for Payslip options.
def payslipSubMenu():
    # Infinite loop to keep showing the Payslip sub menu until the user decides to return to the main menu
    while True:
        # Displaying the options available in the Payslip sub menu
        print("\n*** Payslip ***")
        print("1. Search Specific Payslip")  # Option to search for a specific payslip by month and year
        print("2. View All Payslips")  # Option to view all payslips for a specific employee
        print("3. Return to Main Menu")  # Option to return to the main menu

        # Prompt the user to enter their choice
        choice = input("Enter your choice: ")

        # Processing the user's choice using if-elif-else statements
        if choice == "1":
            searchSpecificPayslip()  # Calls searchSpecificPayslip procedure to search for a particular payslip
        elif choice == "2":
            viewAllPayslips()  # Calls viewAllPayslips procedure to display all payslips for an employee
        elif choice == "3":
            break  # Exits the loop to return to the main menu
        else:
            # Prompt for a valid choice if input is other than "1", "2", or "3"
            print("Invalid choice. Please try again.")
```

The *payslipSubMenu()* procedure provides a submenu for accessing and managing payslip records. Users can select an option to search for a specific payslip by month and year, view all payslips for a specific employee, or return to the main menu. This procedure operates in an infinite loop, displaying the submenu until the user opts to return.

Key Programming Concepts:

1. **Procedure Definition (*def payslipSubMenu()*):** This line defines the *payslipSubMenu()* procedure, encapsulating operations related to the payslip submenu.
2. **Infinite Loop (*while True*):** This loop ensures the payslip submenu remains displayed after each action, allowing the user to choose multiple operations without returning to the main menu until they select the exit option.
3. **Options Display (*print()* statements):** The submenu options (Search Specific Payslip, View All Payslips, and Return to Main Menu) are clearly listed, guiding the user on available actions.
4. **User Input Handling (*input()*):** The *choice* variable captures the user's input, which is then evaluated to determine the action taken.

5. **Conditional Statements (*if-elif-else*):** The procedure processes the user's selection by calling *searchSpecificPayslip()* or *viewAllPayslips()* based on the input. If the input is invalid, an error message is shown, prompting the user to enter a valid option.
6. **Returning to Main Menu (*break* statement):** Selecting the "Return to Main Menu" option triggers the *break* statement, exiting the loop and displaying the main menu.

4.5 Procedure to Load Data from Both Dictionaries

```
# Define a procedure called loadData to load data from JSON files into the employees and payslips dictionaries
def loadData():
    # Accessing the global variables to modify them within the function
    global employees, payslips

    # Check if the employees data file exists in the current directory
    if os.path.exists('employees.json'):
        # Open the employees.json file in read mode
        with open('employees.json') as f:
            # Load data from the JSON file into the employees dictionary
            employees = json.load(f)

    # Check if the payslips data file exists in the current directory
    if os.path.exists('payslips.json'):
        # Open the payslips.json file in read mode
        with open('payslips.json') as f:
            # Load data from the JSON file into the payslips dictionary
            payslips = json.load(f)
```

The *loadData()* procedure initializes the *employees* and *payslips* dictionaries by loading data from JSON files. This ensures that previously saved records are available for program operations.

Key Programming Concepts:

1. **Procedure Definition (*def loadData()*):** This line defines the *loadData()* procedure, encapsulating the data-loading operations for the program.
2. **Global Variables *(global employees, payslips*):** The global keyword allows the procedure to modify the *employees* and *payslips* dictionaries, which are accessible across the entire program.
3. **File Existence Check (*os.path.exists()*):** Each JSON file's existence is verified to prevent errors. The procedure checks if *employees.json* and *payslips.json* exist in the directory before attempting to load data.
4. **File Handling (*with open(...) as f*):** The *with open(...) as f* syntax is used to safely open each JSON file in read mode, ensuring the files are properly closed after data is loaded.
5. **JSON Data Loading (*json.load(f)*):** The *json.load()* procedure reads data from each file and populates the *employees* and *payslips* dictionaries, making this data readily available for other functions and procedures within the program.

4.6 Procedure to Save Data into Both Dictionaries

```
# Define a procedure called saveData to save all current data in employees and payslips to JSON files
def saveData():
    # Accessing the global variables to save their updated states
    global employees, payslips

    # Open the employees JSON file in write mode
    with open('employees.json', 'w') as f:
        # Write the current state of the employees dictionary to the JSON file
        json.dump(employees, f)

    # Open the payslips JSON file in write mode
    with open('payslips.json', 'w') as f:
        # Write the current state of the payslips dictionary to the JSON file
        json.dump(payslips, f)
```

The *saveData()* procedure is responsible for saving the current state of *employees* and *payslips* dictionaries to their respective JSON files, ensuring data persistence across program sessions.

Key Programming Concepts:

1. **Procedure Definition (*def saveData()*):** This line defines the *saveData()* procedure, encapsulating the process of saving data from the program's dictionaries to JSON files.
2. **Global Variables (*global employees, payslips*):** The *global* keyword allows the procedure to access and save the current states of *employees* and *payslips* dictionaries globally, making their updated data accessible throughout the program.
3. **File Handling for Writing (*with open(...) as f*):** The procedure opens each JSON file in write mode (*'w'*) using the *with open(...) as f* syntax, which ensures that files are closed properly after data is saved.
4. **JSON Data Writing (*json.dump(...)*):** The *json.dump()* procedure writes the current content of each dictionary (*employees* and *payslips*) to their respective JSON files, preserving all updates for future program use.

4.7 Function to Validate Input

The following four functions were created to perform validations of different inputs namely integer inputs, float inputs, string, and employee ID input.

4.7.1 Validating for Integer Input

```
# Define a function called promptIntegerInput to get user input and ensure it is a non-negative integer
def promptIntegerInput(action):
    # Creating an infinite loop to continue prompting until the user enters valid input
    while True:
        try:
            # Attempt to convert user input into an integer
            number = int(input(action))

            # Check if the entered integer is non-negative
            if number < 0:
                print("Invalid input. Please ensure that input is a non-negative number.")
                continue  # Return to the beginning of the loop

            # Return the input number if it is valid
            return number

        # If input cannot be converted to an integer, handle the error and prompt again
        except ValueError:
            print("Invalid input. Please reenter a valid number.")
```

The *promptIntegerInput()* function captures and validates a user input, ensuring it is a non-negative integer. It continuously prompts until a valid input is received, enhancing user experience by guiding input accuracy.

Key Programming Concepts:

1. **Function Definition (*def promptIntegerInput(action)*):** This line defines the *promptIntegerInput()* function, which accepts an action prompt as an argument to personalize the input request message.
2. **Infinite Loop (*while True*):** The function runs an infinite loop, ensuring the user is repeatedly prompted until a valid input is provided. This loop maintains program flow by handling incorrect inputs efficiently.
3. **Exception Handling (*try-except*):** The *try* block attempts to convert the user input to an integer. If the input is invalid (e.g., a string), the *except ValueError* block captures this error and prompts the user to re-enter a valid integer, making the function resilient to input errors.

4. **Input Validation (*if number < 0*):** After successfully converting the input, this check ensures the number is non-negative. If the input is negative, a message is displayed, and the loop continues, asking for re-entry.
5. **Return Statement (*return number*):** Once a valid, non-negative integer is entered, the function returns this number, allowing it to be used elsewhere in the program.

4.7.2 Validating for Float Input

```python
# Define a function called promptFloatInput to get input, ensure that the input is a float, and validate it as non-negative
def promptFloatInput(action):
    # Creating an infinite loop to continue prompting until the user enters valid input
    while True:
        try:
            # Attempt to convert the user input into a float
            number = float(input(action))

            # Check if the entered float is non-negative
            if number < 0:
                print("Invalid input. Please ensure that input is a non-negative number")
                continue  # Return to the beginning of the loop

            # Return the input number if it is valid
            return number

        # If input cannot be converted to a float, handle the error and prompt again
        except ValueError:
            print("Invalid input. Please reenter a valid number.")
```

The *promptFloatInput()* function collects and validates a user-provided float, ensuring it is non-negative. This function continuously prompts until a valid float is entered, enhancing input accuracy and reliability.

Key Programming Concepts:

1. **Function Definition (*def promptFloatInput(action)*):** Defines the *promptFloatInput()* function, taking an action parameter to specify the prompt message for user input.
2. **Infinite Loop (*while True*):** Establishes an infinite loop to repeatedly prompt the user until a valid input is given, maintaining a continuous interaction for input correction when necessary.
3. **Exception Handling (*try-except*):** The *try* block attempts to convert the user input into a float. If the conversion fails (e.g., if a non-numeric input is entered), the *except ValueError* block catches the error and provides feedback to re-enter a valid float.
4. **Input Validation (*if number < 0*):** After successfully converting the input, the function checks if the number is non-negative. If it is negative, a message is displayed, and the loop restarts, prompting the user to enter a non-negative float.
5. **Return Statement (*return number*):** Once a valid, non-negative float is entered, the function returns this value, making it available for further use in the program.

4.7.3 Validating for Strings Input

```
# Define a function called promptStringsInput to get input and ensure that the input contains only letters and spaces
def promptStringsInput(action):
    # Creating an infinite loop to continue prompting until the user enters valid input
    while True:
        # Get the user input
        string = input(action)

        # Check if the input contains only alphabetic characters and spaces
        if not all(s.isalpha() or s.isspace() for s in string):
            print("Invalid input. The string must contain only letters and spaces.")
            continue  # Return to the beginning of the loop if invalid

        # If the input passes the check, return it
        return string
```

The *promptStringsInput()* function ensures that a user-provided string contains only alphabetic characters and spaces. This validation loop prompts the user repeatedly until they enter a valid response, maintaining data integrity for character-based inputs.

Key Programming Concepts:

1. **Function Definition (*def promptStringsInput(action)*):** This line defines the *promptStringsInput()* function, which takes an *action* parameter used as a prompt message for user input.
2. **Infinite Loop (*while True*):** An infinite loop repeatedly prompts the user until a valid string is entered, allowing continuous feedback for incorrect inputs without breaking the program flow.
3. **Input Collection (*string = input(action)*):** Captures the user's input in a variable called *string*, based on the prompt defined by *action*.
4. **Character Validation (*if not all(...)*):** This condition checks each character in *string* to ensure all are either alphabetic or spaces. If any character fails this condition, the function provides an error message and restarts the loop.
5. **Return Statement (*return string*):** Once a valid input is obtained, the function exits the loop and returns the *string*, allowing it to be used elsewhere in the program.

4.7.4 Validating for Employee ID Input

```
# Define a function called promptEmpIDInput to prompt and validate Employee ID input in a specific format
def promptEmpIDInput(prompt_text="Enter Employee ID (e.g., 'TP00001'): "):
    # Creating an infinite loop until the user enters a valid Employee ID
    while True:
        # Prompt for Employee ID input, remove any extra spaces, and convert it to uppercase for consistency
        empID = input(prompt_text).strip().upper()

        # Check if the Employee ID starts with 'TP', is exactly 7 characters, and the last 5 characters are digits
        if empID.startswith("TP") and len(empID) == 7 and empID[2:].isdigit():
            # If the Employee ID is valid, return it
            return empID
        else:
            # Print a warning if the format is incorrect
            print("Invalid Employee ID format. Please enter an ID in the format 'TP00001', 'TP00002', etc.")
```

The *promptEmpIDInput()* function ensures that a user enters a valid Employee ID in a specific format. The function continuously prompts the user until an Employee ID beginning with "TP" followed by five digits is provided.

Key Programming Concepts:

1. **Function Definition (*def promptEmpIDInput(prompt_text="Enter Employee ID (e.g., 'TP00001')")*):** Defines the *promptEmpIDInput()* function, which uses *prompt_text* as the customizable message shown to the user, guiding them on the expected input format.
2. **Infinite Loop (*while True*):** Implements an infinite loop to repeatedly prompt the user until a correctly formatted Employee ID is entered, ensuring the program does not proceed with invalid data.
3. **Employee ID Input (*empID = input(prompt_text).strip().upper()*):** This line captures the user's input, removes any extra spaces, and converts the input to uppercase for consistent validation.
4. **Format Validation *(if empID.startswith("TP") and len(empID) == 7 and empID[2:].isdigit()*):** Checks if the input starts with "TP," has a total length of 7 characters, and contains only digits after "TP." This ensures adherence to the Employee ID format "TP00001."
5. **Return Statement (*return empID*):** If the input meets the required format, the function returns the Employee ID, allowing further processing.

6. **Error Message (*print("Invalid Employee ID format. Please enter an ID in the format 'TP00001', 'TP00002', etc.")*):** If the format is incorrect, an error message prompts the user to try again, enhancing input accuracy.

4.8 Procedure to Add Details of New Employee or New Payslip for Existing Employee

```
# Define a procedure to add a new employee or add a new payslip for an existing employee
def addEmployee():
    global employees, payslips

    # Use a standardized input function to prompt for Employee ID
    empID = promptEmpIDInput("Enter Employee ID: ")

    # Check if Employee ID already exists in the `employees` dictionary
    if empID in employees:
        print("This Employee ID already exists. Adding a new payslip record for this employee.")
        # Retrieve existing employee name and department for display and use in the payslip
        employeeName = employees[empID]['Employee Name']
        departmentName = employees[empID]['Department Name']
    else:
        # Collect new employee information if the employee does not exist
        print("This Employee ID does not exist yet. Adding it to the system.")
        employeeName = promptStringsInput("Enter Employee Name: ")
        departmentName = promptStringsInput("Enter Department Name: ")

    # Prompt for month and year to associate the payslip record with a specific period
    month = promptStringsInput("Enter Month (e.g., 'January', 'February'): ").capitalize()
    valid_months = ['January', 'February', 'March', 'April', 'May', 'June',
                    'July', 'August', 'September', 'October', 'November', 'December']
    # Ensure the month is valid
    while month not in valid_months:
        print("Invalid month. Please enter a valid month name.")
        month = promptStringsInput("Enter Month (e.g., 'January', 'February'): ").capitalize()
```

```
    # Prompt for and validate the year
    year = input("Enter Year (e.g., 2023): ")
    while not year.isdigit() or len(year) != 4:
        print("Invalid year. Please enter a 4-digit year.")
    year = int(year)

    # Initialize the structure in `payslips` if this is a new employee or year
    if empID not in payslips:
        payslips[empID] = {}
    if year not in payslips[empID]:
        payslips[empID][year] = {}

    # Check if a payslip already exists for the specified month and year
    if month in payslips[empID][year]:
        print(f"A payslip record for {month} {year} already exists.")
        # Prompt the user to update the existing record if desired
        update_choice = input("Would you like to update the existing record instead? (yes/no): ").strip().lower()
        if update_choice == 'yes':
            updateEmployee(empID, year, month)  # Call `updateEmployee` with existing ID, month, and year
        else:
            print("No changes made. Returning to the main menu.")
        return

    # Prompt for salary components
    basicSalary = promptFloatInput("Enter Basic Salary Amount: ")
    allowance = promptFloatInput("Enter Allowance Amount: ")
    bonus = promptFloatInput("Enter Bonus Amount: ")
    overtime = promptFloatInput("Enter Overtime Amount: ")
```

```
# Calculate net salary and related details using a separate function
salaryDetails = calculateNetSalary(basicSalary, allowance, bonus, overtime)

# Ask for confirmation to save the new record
confirmation = input("\nDo you want to save these details? (yes/no): ").strip().lower()
if confirmation != 'yes':
    print("Employee data and salary details were not saved.")
    return  # Exit the procedure if the user chooses not to save

# Save employee information and salary details if the user confirms saving
if empID not in employees:
    employees[empID] = {
        'Employee Name': employeeName,
        'Department Name': departmentName
    }

# Save payslip information for the specified month and year
payslips[empID][year][month] = salaryDetails

# Save data to JSON files to ensure persistence
saveData()

print("Employee data has been successfully added and salary details saved.")
```

The *addEmployee()* procedure is responsible for adding new employee information or a new payslip entry for existing employees. This procedure ensures comprehensive data collection for each employee, including ID, name, department, month, year, and salary details. Here are the key programming concepts demonstrated within this procedure:

1. **Procedure Definition (*def addEmployee()*):** This line initiates the procedure, establishing *addEmployee()* as a distinct operation within the program to handle the addition of employee records.

2. **Global Variable Usage (*global employees, payslips*):** The procedure accesses the *employees* and *payslips* dictionaries, declared globally, enabling direct updates to these structures, reflecting in other parts of the program.

3. **Employee ID Input Validation (*promptEmpIDInput()*):** The procedure uses the *promptEmpIDInput()* to obtain a valid Employee ID, ensuring consistency in ID format and user input.

4. **Conditional Check for Existing Employee ID:** The procedure checks if *empID* exists in the *employees* dictionary. If it does, it retrieves existing employee details for payslip updates; otherwise, it collects new employee data.

5. **Month and Year Validation:** It uses *promptStringsInput()* to capture the month and validate it against a predefined list of valid months. The year input is also checked for the correct format, ensuring accuracy in the payroll record date.
6. **Structure Initialization in *payslips* Dictionary:** If the employee or year does not exist in the *payslips* dictionary, the procedure initializes nested dictionaries to store detailed monthly salary records.
7. **Checking for Existing Payslip Record:** The procedure verifies if a record already exists for the specified month and year. If so, it prompts the user to either update the existing record or exit without changes.
8. **Salary Input Collection and Calculation:** The procedure gathers detailed salary components (basic salary, allowance, bonus, overtime) and calculates the net salary using *calculateNetSalary()*.
9. **Confirmation and Data Saving:** After collecting all details, it prompts the user for confirmation to save. If confirmed, the procedure updates *employees* and *payslips* dictionaries and persists data to JSON files using *saveData()*, ensuring continuity and data integrity.
10. **Feedback to User:** Finally, the procedure prints a success message, indicating that the employee data and salary details have been saved successfully.

4.9 Procedure to Update the Details of Existing Employee

```
# Define a procedure to update existing employee payslip details
def updateEmployee(empID=None, year=None, month=None):
    global employees, payslips

    # Prompt for Employee ID if not provided, using standardized input
    if not empID:
        empID = promptEmpIDInput("Enter Employee ID to update: ")

    # Prompt for month if not provided, ensuring it matches valid month names
    if not month:
        month = promptStringsInput("Enter Month (e.g., 'January', 'February'): ").capitalize()
        valid_months = ['January', 'February', 'March', 'April', 'May', 'June',
                        'July', 'August', 'September', 'October', 'November', 'December']
        while month not in valid_months:
            print("Invalid month. Please enter a valid month name.")
            month = promptStringsInput("Enter Month (e.g., 'January', 'February'): ").capitalize()

    # Prompt for year if not provided, ensuring it is a valid 4-digit integer
    if not year:
        year_input = input("Enter Year (e.g., 2023): ")
        while not year_input.isdigit() or len(year_input) != 4:
            print("Invalid year. Please enter a 4-digit year.")
            year_input = input("Enter Year (e.g., 2023): ")
        year = int(year_input)
```

```
    # Check if the payslip record exists for the specified Employee ID, month, and year
    if empID not in payslips or year not in payslips[empID] or month not in payslips[empID][year]:
        print(f"No payslip record found for {month} {year}. Please add the payslip first before updating.")
        return  # Exit if no record is found

    # Display existing payslip data for reference
    print("\nExisting Payslip Data for Reference:")
    existing_table = PrettyTable()
    existing_table.field_names = ["Detail", "Value"]
    existing_data = payslips[empID][year][month]
    for key, value in existing_data.items():
        existing_table.add_row([key, f"{value:.2f}"])
    print(existing_table)

    # Prompt user to update salary components
    basicSalary = promptFloatInput("Enter updated Basic Salary Amount: ")
    allowance = promptFloatInput("Enter updated Allowance Amount: ")
    bonus = promptFloatInput("Enter updated Bonus Amount: ")
    overtime = promptFloatInput("Enter updated Overtime Amount: ")

    # Calculate net salary and other components based on input values
    salaryDetails = calculateNetSalary(basicSalary, allowance, bonus, overtime)
```

```
    # Display the updated details using PrettyTable for user confirmation
    table = PrettyTable()
    table.field_names = ["Detail", "Value"]
    table.add_row(["Employee ID", empID])
    table.add_row(["Employee Name", employees[empID]['Employee Name']])
    table.add_row(["Department Name", employees[empID]['Department Name']])
    table.add_row(["Year", year])
    table.add_row(["Month", month])
    for key, value in salaryDetails.items():
        table.add_row([key, f"{value:.2f}"])

    print("\nUpdated Employee and Salary Details:\n")
    print(table)
```

```
# Confirm if the user wants to save the updated payslip details
confirmation = input("\nDo you want to overwrite the existing payslip with these updated details? (yes/no): ").strip().lower()
if confirmation != 'yes':
    print("Employee update was canceled. No changes were saved.")
    return  # Exit without saving changes if the user chooses 'no'

# Update the payslip record with new values in `payslips` dictionary
payslips[empID][year][month] = salaryDetails

# Persist updated information to JSON files
saveData()

print("Employee data has been successfully updated.")
```

The *updateEmployee()* procedure enables updating specific payslip details for an employee, allowing modifications based on selected month and year.

Key Programming Concepts:

1. **Procedure Definition (*def updateEmployee()*):** The *updateEmployee()* procedure is defined to handle the updating of employee payslip details.
2. **Global Variables (global employees, payslips):** Declares *employees* and *payslips* as global variables, ensuring modifications are applied throughout the system.
3. **Employee ID Prompt:** Checks if *empID* is provided. If not, it prompts the user to input an ID using the *promptEmpIDInput()* function.
4. **Month and Year Validation:** Ensures the month is valid and capitalized and checks the year input for a 4-digit integer, helping locate the correct payslip entry in the *payslips* dictionary.
5. **Existence Check for Payslip Data:** Verifies if the specified *empID*, month, and year exist in *payslips*. If not, it notifies the user to add the payslip first and exits the procedure.
6. **Displaying Current Payslip Data:** Uses *PrettyTable* to display the existing payslip data, allowing the user to review before making changes.
7. **Prompting for Salary Components:** Requests updated values for *basicSalary*, *allowance*, *bonus*, and *overtime*, ensuring non-negative float values.
8. **Net Salary Calculation:** Calls *calculateNetSalary()* with the new inputs to calculate the updated salary details.

9. **Displaying Updated Details for Confirmation:** Displays the newly calculated details using *PrettyTable*, giving the user a chance to review changes before saving.
10. **Confirmation and Saving Updates:** Prompts the user for confirmation to overwrite the existing payslip data. If confirmed, updates the record in *payslips[empID][year][month]*.
11. **Data Persistence (saveData()):** Calls *saveData()* to save the updates in JSON files, ensuring data persistence.
12. **Completion Message:** Prints a message indicating that the update was successfully saved.

4.10 Procedure to View the Details of All Existing Employees

```
# Define a procedure called viewEmployee to display all employee records from the employees dictionary
def viewEmployee():
    # Accessing the global employees dictionary to get the data
    global employees

    # Check if there are any employee records in the dictionary
    if not employees:
        print("No employee records found.")  # Inform the user if the dictionary is empty
        return  # Exit the procedure if no records are found

    # Create a PrettyTable object for organized tabular display
    employeeTable = PrettyTable()
    # Set column headers for the table
    employeeTable.field_names = ["Employee ID", "Name", "Department"]

    # Loop through each employee record in the employees dictionary
    for empID, details in employees.items():
        # Add a row to the table for each employee with their ID, name, and department
        employeeTable.add_row([empID, details['Employee Name'], details['Department Name']])

    # Display the complete table of employee records
    print("\nEmployee Records:")
    print(employeeTable)
```

The *viewEmployee()* procedure is designed to display all employee records in a structured tabular format.

Key Programming Concepts:

1. **Procedure Definition (*def viewEmployee()*):** Defines the *viewEmployee()* procedure to encapsulate the logic for viewing employee details.
2. **Global Variable Access (global employees):** Declares *employees* as a global variable to access and display the data stored in it.
3. **Checking for Empty Records:** Uses an *if not employees* condition to check if the *employees* dictionary is empty. If empty, it notifies the user and exits the procedure to avoid displaying an empty table.
4. **Creating PrettyTable for Tabular Display:** Initializes a *PrettyTable* object named *employeeTable* to format employee data for easy reading.
5. **Setting Column Headers:** Sets the headers of *employeeTable* to "Employee ID," "Name," and "Department" to organize the data presentation.
6. **Looping Through Employee Records:** Uses a *for* loop to iterate through each entry in the *employees* dictionary, retrieving the employee ID, name, and department details.

7. **Adding Rows to Table:** Adds each employee's data (ID, name, department) as a new row in *employeeTable* for structured presentation.
8. **Displaying the Table:** Prints a header "Employee Records:" followed by the *employeeTable*, displaying all employee data in a readable format.

4.11 Procedure to Delete Employee details and Salary Details

```
# Define a procedure called deleteEmployee to remove an employee's details from both employees and payslips dictionaries,
# and subsequently from the respective JSON files.
def deleteEmployee():
    # Accessing the global variables
    global employees, payslips

    # Use a standardized input function to prompt for Employee ID
    empID = promptEmpIDInput("Enter Employee ID to delete: ")

    # Check if the Employee ID exists in the employees dictionary
    if empID in employees:
        # Delete the employee's record from the employees dictionary
        del employees[empID]
        print(f"Employee data for {empID} has been successfully deleted from the employees dictionary.")

        # Check if the employee has any salary records in the payslips dictionary
        if empID in payslips:
            # Delete the employee's salary records from the payslips dictionary
            del payslips[empID]
            print(f"Employee salary records for {empID} have been successfully deleted from the payslips dictionary.")
        else:
            # Inform the user if no salary records are found for the employee in payslips
            print(f"No salary records found for {empID} in the payslips dictionary.")

        # Save changes to ensure the deletions are reflected in the JSON files
        saveData()
        print("All changes have been successfully saved.")
```

```
    else:
        # Inform the user if the Employee ID does not exist in the employees dictionary
        print("Employee ID does not exist!")
```

The *deleteEmployee()* procedure manages the removal of an employee's data from both the employees and payslips dictionaries and updates the JSON files accordingly.

Key Programming Concepts:

1. **Procedure Definition (*def deleteEmployee()*):** This procedure is designed to delete an employee's records from both the employees and payslips dictionaries based on a provided Employee ID.
2. **Global Variables Access:** The procedure accesses the *employees* and *payslips* dictionaries globally to ensure any modifications are reflected program wide.
3. **Employee ID Input:** It prompts for an Employee ID using a standardized input function, ensuring the ID format is consistent.
4. **Employee Existence Check:** A conditional checks if the provided Employee ID exists in the *employees* dictionary.

- **If Exists in *employees*:** Deletes the employee's record from the *employees* dictionary and confirms the deletion.
- **If Exists in *payslips*:** Further checks if the employee has salary records in the *payslips* dictionary. If found, deletes these records and confirms; if not, notifies the user that no salary records exist.

5. **JSON File Update (*saveData()*):** Calls *saveData()* to ensure deletions are saved to JSON files for data persistence.
6. **Non-Existence Notification:** If the Employee ID is not found in *employees*, the procedure notifies the user that the ID does not exist.

4.12 Function to Define the Formula for Calculating the Net Salary

```
# Define a function to calculate the net salary, applying EPF deduction, tax, and bonus adjustments based on predefined criteria
def calculateNetSalary(basicSalary, allowance, bonus, overtime):
    # Calculate the gross salary as the sum of all components
    grossSalary = basicSalary + allowance + bonus + overtime

    # Calculate EPF (Employees Provident Fund) deduction at 11%
    epfDeduction = grossSalary * 0.11
    # Calculate the salary after EPF deduction
    salaryAfterEPF = grossSalary - epfDeduction

    # Initialize variables for additional adjustments
    additionalBonus = 0  # Additional bonus, if applicable
    taxDeduction = 0     # Tax deduction, if applicable

    # Apply additional adjustments based on salaryAfterEPF criteria
    if salaryAfterEPF < 2000:
        # Add 5% bonus for salary less than 2000
        additionalBonus = salaryAfterEPF * 0.05
    elif salaryAfterEPF > 3000:
        # Deduct 6% tax for salary greater than 3000
        taxDeduction = salaryAfterEPF * 0.06

    # Calculate net salary after applying additional bonus and tax deduction
    netSalary = salaryAfterEPF + additionalBonus - taxDeduction
```

```
    # Return a dictionary containing all calculated salary components
    return {
        'Basic Salary': basicSalary,
        'Allowance': allowance,
        'Bonus': bonus,
        'Overtime': overtime,
        'Gross Salary': grossSalary,
        'EPF Deduction': epfDeduction,
        'Additional Bonus': additionalBonus,
        'Tax Deduction': taxDeduction,
        'Net Salary': netSalary
    }
```

The *calculateNetSalary* function computes an employee's net salary by applying EPF deductions, additional bonuses, and tax adjustments.

1. **Function Definition (*def calculateNetSalary*)**: Defines *calculateNetSalary*, which takes *basicSalary*, *allowance*, *bonus*, and *overtime* as input parameters to calculate the total net salary.

2. **Gross Salary Calculation:** The function first calculates *grossSalary* by summing up *basicSalary*, *allowance*, *bonus*, and *overtime*.

3. **EPF Deduction Calculation:** It calculates an 11% deduction on *grossSalary*, assigned to *epfDeduction*, then computes *salaryAfterEPF* by subtracting *epfDeduction* from *grossSalary*.

4. **Additional Adjustments (Bonus and Tax):** Initializes *additionalBonus* and *taxDeduction* to 0, then applies a 5% bonus if *salaryAfterEPF* is below 2000 or a 6% tax if it exceeds 3000.
5. **Net Salary Calculation:** Computes *netSalary* by adding *additionalBonus* and subtracting *taxDeduction* from *salaryAfterEPF*.
6. **Return Statement:** Returns a dictionary of all calculated salary components, including Basic Salary, Allowance, Bonus, Overtime, Gross Salary, EPF Deduction, Additional Bonus, Tax Deduction, and Net Salary.

4.13 Procedure to Generate the Salary of Employee

```
# Define a procedure to interactively calculate and display an employee's net salary based on inputted salary components
def generateNetSalary():
    global employees  # Accessing the global 'employees' dictionary

    # Prompt user to enter the employee ID
    empID = promptEmpIDInput("Enter Employee ID: ")

    # Check if Employee ID exists in the system
    if empID not in employees:
        print("Employee ID not found. Please check and try again.")
        return

    # Prompt for the specific month and year
    month = promptStringsInput("Enter Month (e.g., 'January', 'February'): ").capitalize()
    valid_months = ['January', 'February', 'March', 'April', 'May', 'June',
                    'July', 'August', 'September', 'October', 'November', 'December']
    while month not in valid_months:
        print("Invalid month. Please enter a valid month name.")
        month = promptStringsInput("Enter Month (e.g., 'January', 'February'): ").capitalize()

    # Prompt for the year and ensure it is a valid 4-digit number
    year_input = input("Enter Year (e.g., 2023): ")
    while not year_input.isdigit() or len(year_input) != 4:
        print("Invalid year. Please enter a 4-digit year.")
    year = int(year_input)  # Convert validated year input to integer
```

```
    # Prompt for salary components
    basicSalary = promptFloatInput("Enter Basic Salary Amount: ")
    allowance = promptFloatInput("Enter Allowance Amount: ")
    bonus = promptFloatInput("Enter Bonus Amount: ")
    overtime = promptFloatInput("Enter Overtime Amount: ")

    # Calculate the net salary and other components using calculateNetSalary
    salaryDetails = calculateNetSalary(basicSalary, allowance, bonus, overtime)

    # Display the calculated salary details
    print("\nCalculated Salary Details:")
    for key, value in salaryDetails.items():
        print(f"{key}: {value:.2f}")  # Output each component in formatted form

    print("Net salary calculation complete (data not saved).")  # Indicate that data is not saved
```

The *generateNetSalary* procedure prompts the user to input employee salary details and calculates the net salary based on the entered components.

1. **Procedure Definition (*def generateNetSalary()*):** This line defines the *generateNetSalary* procedure, establishing it as a distinct section of code dedicated to calculating and displaying net salary.

2. **Global Variables (*global employees*):** This allows the procedure to access and utilize the *employees* dictionary, where employee data is stored.

3. **Employee ID Prompt and Validation:** The procedure uses *promptEmpIDInput()* to request an Employee ID and checks if the ID exists in the *employees* dictionary. If not, it informs the user and exits the procedure, ensuring valid input.

4. **Month and Year Input Validation:** The procedure prompts the user for the month using *promptStringsInput()* and verifies it against a predefined list of valid month names. For the year, it ensures the input is a 4-digit integer before converting it to integer type to maintain data consistency.

5. **Salary Component Prompts:** The procedure gathers inputs for *basicSalary*, *allowance*, *bonus*, and *overtime* using *promptFloatInput()*, ensuring each is a valid floating-point number.

6. **Salary Calculation:** *calculateNetSalary*() is called with the salary components to compute the net salary, considering EPF deduction and any additional bonus or tax.

7. **Formatted Salary Display:** The procedure iterates over the items in *salaryDetails*, displaying each key-value pair in a formatted style for clarity.

8. **Completion Message:** It notifies the user that the calculation is complete, but data is not saved, emphasizing that this procedure is only for interactive calculation without data persistence.

4.14 Procedure to Search for an Employee's Payslip

```
# Define a procedure called searchSpecificPayslip to search for a particular employee's payslip based on month and year
def searchSpecificPayslip():
    global payslips  # Accessing the global 'payslips' dictionary

    # Loop to allow re-entry if Employee ID is not found
    while True:
        # Prompt for Employee ID
        empID = promptEmpIDInput("Enter Employee ID: ")

        # Check if Employee ID exists in the payslips dictionary
        if empID not in payslips:
            print("No records found for this Employee ID.")
            retry = input("Would you like to try again with a different Employee ID? (yes/no): ").strip().lower()
            if retry != 'yes':
                print("Returning to the main menu.")
                return  # Exit if user chooses not to retry
            else:
                continue  # Prompt for Employee ID again if user chooses to retry

        # If Employee ID exists, proceed to prompt for month and year
        month = promptStringsInput("Enter Month (e.g., 'January', 'February'): ").capitalize()
        valid_months = ['January', 'February', 'March', 'April', 'May', 'June',
                        'July', 'August', 'September', 'October', 'November', 'December']
```

```
        # Validate the entered month
        while month not in valid_months:
            print("Invalid month. Please enter a valid month name (e.g., 'January').")
            month = promptStringsInput("Enter Month (e.g., 'January', 'February'): ").capitalize()

        # Prompt for the year, ensuring it's a valid 4-digit year
        year_input = input("Enter Year (e.g., 2023): ")
        while not year_input.isdigit() or len(year_input) != 4:
            print("Invalid year. Please enter a 4-digit year.")
            year_input = input("Enter Year (e.g., 2023): ")
        year = int(year_input)  # Convert validated year to integer

        # Check if payslip exists for the specified month and year
        if year not in payslips[empID] or month not in payslips[empID][year]:
            print(f"No payslip record found for {month} {year} for Employee ID {empID}.")
            return  # Exit if no record is found

        # Retrieve the payslip details
        payslip_details = payslips[empID][year][month]

        # Display the payslip details using PrettyTable
        payslip_table = PrettyTable()
        payslip_table.field_names = ["Detail", "Value"]

        for key, value in payslip_details.items():
            payslip_table.add_row([key, f"{value:.2f}"])
```

```
        # Print the payslip details table
        print(f"\nPayslip for Employee ID: {empID} for {month} {year}")
        print(payslip_table)
        return  # End function after displaying the payslip
```

The *searchSpecificPayslip* procedure enables users to locate and view a specific employee's payslip based on month and year, ensuring valid input for accurate retrieval.

Key Programming Concepts:

1. **Procedure Definition**: *searchSpecificPayslip()* is designed to locate and display an employee's payslip for a specified month and year, ensuring the user inputs correct identifiers.
2. **Global Dictionary Access**: The procedure accesses the global *payslips* dictionary, allowing it to retrieve payslip data for the requested employee.
3. **Employee ID Input and Validation**: The procedure prompts for an *Employee ID* using *promptEmpIDInput()*. If the ID does not exist in the *payslips* dictionary, the user is given the option to retry or exit to the main menu.
4. **Month Input and Validation**: After confirming the *Employee ID*, the procedure prompts for the month, which is validated against a predefined list of valid month names (*valid_months*). If an invalid month is entered, the user is prompted to re-enter until it matches one in *valid_months*.
5. **Year Input and Validation**: The procedure prompts for a 4-digit *year*, checks its format, and converts it to an integer to ensure consistency. If the year input is invalid, the user is prompted to re-enter.
6. **Payslip Existence Check**: With a valid *Employee ID*, month, and year, the procedure checks if a payslip exists in the *payslips* dictionary for the specified period. If not found, it informs the user and exits.
7. **Displaying Payslip Details**: When a matching record is located, it retrieves and displays the payslip details in a tabular format using *PrettyTable*.

4.15 Procedure to View All Payslips of an Employee

```
# Define a procedure called viewAllPayslips to view all payslips for a specific employee in descending year order
def viewAllPayslips():
    global payslips  # Accessing the global 'payslips' dictionary

    # Prompt for Employee ID using standardized input
    empID = promptEmpIDInput("Enter Employee ID: ")

    # Check if Employee ID exists in the payslips dictionary
    if empID not in payslips:
        print("No payslip records found for this Employee ID.")
        # Allow user to retry if no records are found
        retry = input("Would you like to try again with a different Employee ID? (yes/no): ").strip().lower()
        if retry != 'yes':
            print("Returning to the main menu.")
            return  # Exit the procedure if user does not wish to retry
        else:
            return viewAllPayslips()  # Retry for a different Employee ID

    # Sort the years in descending order to display most recent payslips first
    for year in sorted(payslips[empID].keys(), reverse=True):
        months = payslips[empID][year]

        # Create a PrettyTable for displaying payslip details for the current year
        year_table = PrettyTable()
        year_table.field_names = ["Month", "Basic Salary", "Allowance", "Bonus", "Overtime", "Gross Salary",
                                  "EPF Deduction", "Additional Bonus", "Tax Deduction", "Net Salary"]
```

```
        # Populate table with each month's details in the current year
        for month, details in months.items():
            year_table.add_row([
                month,
                f"{details.get('Basic Salary', 0.00):.2f}",
                f"{details.get('Allowance', 0.00):.2f}",
                f"{details.get('Bonus', 0.00):.2f}",
                f"{details.get('Overtime', 0.00):.2f}",
                f"{details.get('Gross Salary', 0.00):.2f}",
                f"{details.get('EPF Deduction', 0.00):.2f}",
                f"{details.get('Additional Bonus', 0.00):.2f}",
                f"{details.get('Tax Deduction', 0.00):.2f}",
                f"{details.get('Net Salary', 0.00):.2f}"
            ])

        # Display the formatted table for each year
        print(f"\nPayslip Summary for Employee ID: {empID} for Year: {year}")
        print(year_table)
```

The *viewAllPayslips* procedure provides an interface to display all payslips for a specific employee in descending year order, offering a structured overview of their payment history.

Key Programming Concepts:

1. **Employee ID Prompt**: The procedure starts by prompting the user for an Employee ID using *promptEmpIDInput*() to standardize input.
2. **Employee ID Validation**: After receiving the Employee ID, the procedure checks if records are available. If no records are found, it allows the user to either re-enter an ID or return to the main menu.

3. **Year Sorting**: The procedure sorts available years in descending order to ensure the most recent payslips appear first.
4. **Table Creation**: Using a *PrettyTable* object, the procedure sets up a table with column headers for key salary components like Basic Salary, Allowance, and Net Salary.
5. **Monthly Data Population**: Each month's payslip data is added to the table with formatted values, enhancing readability.
6. **Displaying Payslip Summary**: The procedure prints a summary table for each year, showing the employee's monthly payslip details for quick reference.

4.16 Procedure to Exit the Program

```
# Define a procedure called exit to exit the program
def exit():
    # Call saveData to save any changes made to employees and payslips dictionaries
    saveData()

    # Inform the user that the program is closing
    print("Exiting the program.")

    # End the function, signaling the end of the program flow
    return
```

The *exit* procedure is designed to close the program, ensuring that any updates made to employee or payslip records are saved before termination.

Key Programming Concepts:

1. **Data Saving**: The procedure calls *saveData()* to ensure any changes to the *employees* and *payslips* dictionaries are stored, preserving updates.
2. **User Notification**: It informs the user that the program is closing by printing an exit message.
3. **Program Termination**: The procedure ends with a *return* statement, signalling the termination of program flow.

5.0 Screenshots of Sample Input and Output

This section provides a detailed demonstration of each core function and procedure within the program, using various input scenarios to validate functionality. Screenshots of valid input cases illustrate expected outputs and flow, while scenarios with invalid inputs demonstrate error handling and user prompts.

The employee records in this system are limited in number purely for illustration purposes. All employee details are entirely fictional and intended to demonstrate the system's functionality. Consequently, some records may display anomalies, such as entries for January 2021, followed by January and February 2023, and then January 2024. These inconsistencies exist solely to showcase the functionality without requiring the input of every monthly record for multiple years, which would be extensive and impractical for demonstration.

5.1 Main Menu

```
------- Main Menu -------
1. Employee Profile
2. Salary Generator
3. Payslip Profile
4. Exit
Enter your option: 1
Employee Profile has been chosen

------ Employee Profile ------
1. Add Employee
2. Update Employee
3. Delete Employee
4. View Employee List
5. Return to Main Menu
Enter your choice:
```

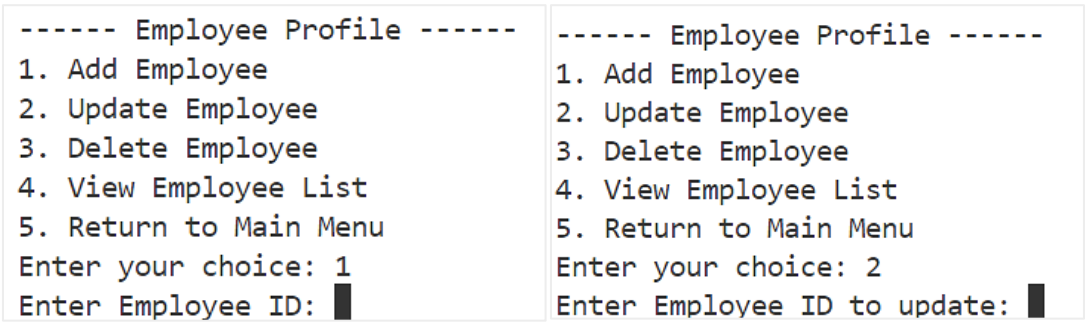
Scenario A: In this scenario, the user enters a valid input, selecting option "1" from the Main Menu to access the Employee Profile section. The system responds by displaying the Employee Profile submenu, confirming that the selected option was processed successfully.

```
------- Main Menu -------
1. Employee Profile
2. Salary Generator
3. Payslip Profile
4. Exit
Enter your option: One
Choice is invalid. Try entering choice again.

------- Main Menu -------
1. Employee Profile
2. Salary Generator
3. Payslip Profile
4. Exit
Enter your option:
```

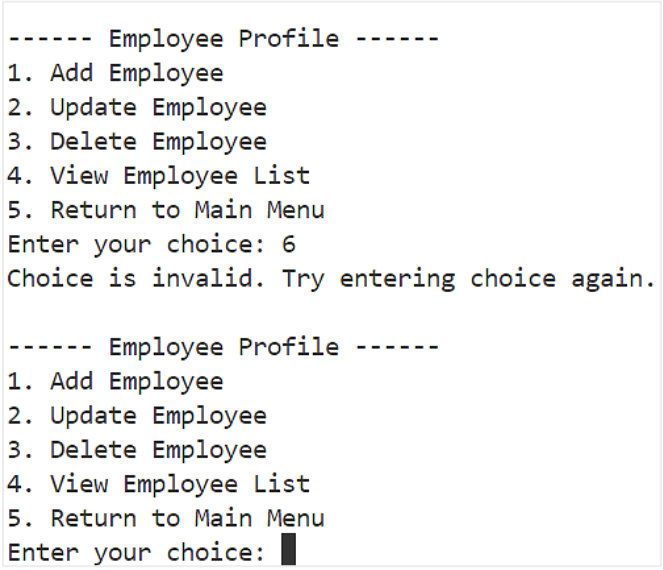
```
------- Main Menu -------
1. Employee Profile
2. Salary Generator
3. Payslip Profile
4. Exit
Enter your option: 5
Choice is invalid. Try entering choice again.

------- Main Menu -------
1. Employee Profile
2. Salary Generator
3. Payslip Profile
4. Exit
Enter your option:
```

Scenario B: Here, the user provides invalid inputs— by typing "One" instead of a number, and by entering "5," which is outside the range of valid options. The system detects these errors, displays an appropriate error message, and prompts the user to re-enter a valid choice, ensuring the program's robustness in handling incorrect inputs.

5.2 Employee Profile Submenu

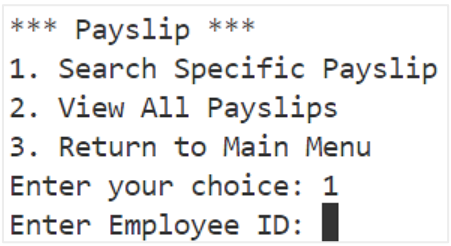


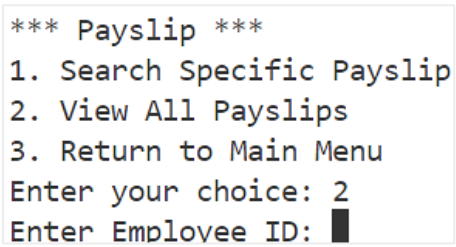
Scenario A: The screenshots demonstrate successful operation of the Employee Profile submenu. Selecting option "1" prompts for the addition of a new employee by asking for an Employee ID, while selecting option "2" initiates the update process for an existing employee, prompting for the Employee ID to update. Both valid selections confirm that the system navigates to the intended function within the Employee Profile submenu when a correct option is entered.



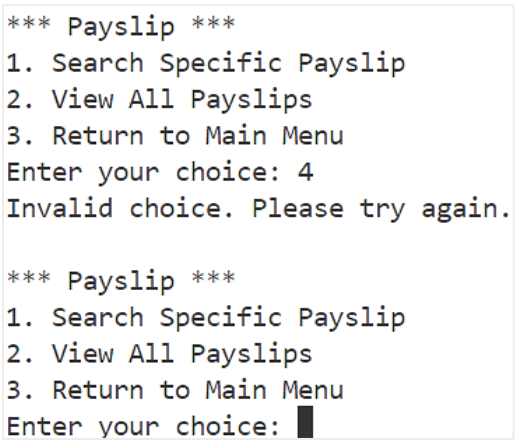
Scenario B: In this scenario, when an invalid input is entered, whether numeric ("6") or text-based ("six"), the system displays an error message. This demonstrates the system's robust input validation, ensuring that only the exact menu options (numeric values 1–5) are accepted. After showing this message, the system automatically redirects the user back to the Employee Profile submenu, allowing another attempt to select a valid option.

5.3 Payslip Profile Submenu





Scenario A: In this scenario, the user navigates to the Payslip Profile submenu and selects valid options to proceed with either "Search Specific Payslip" by entering option "1" or "View All Payslips" by entering option "2." In each case, the system prompts for an Employee ID, confirming that the user's selection is correctly recognized and that the Payslip Profile submenu is operating as expected.



Scenario B: In this scenario, the user attempts to enter an invalid option "4" within the Payslip Profile submenu. The system responds by displaying an error message and redirects the user back to the Payslip Profile submenu, prompting them to re-enter a valid choice.

5.4 Add Employee Procedure

```
------ Employee Profile ------
1. Add Employee
2. Update Employee
3. Delete Employee
4. View Employee List
5. Return to Main Menu
Enter your choice: 1
Enter Employee ID: TP00001
This Employee ID does not exist yet. Adding it to the system.
Enter Employee Name: Chloe Ting
Enter Department Name: Finance
Enter Month (e.g., 'January', 'February'): January
Enter Year (e.g., 2023): 2023
Enter Basic Salary Amount: 3000
Enter Allowance Amount: 300
Enter Bonus Amount: 1000
Enter Overtime Amount: 500

Do you want to save these details? (yes/no): yes
Employee data has been successfully added and salary details saved.
```

Scenario A: Adding New Employee for the First Time with Salary Details Upon Employment

In this scenario, the user selects the "Add Employee" option from the Employee Profile submenu. The system prompts for various details, including Employee ID, Name, Department, Month, Year, Basic Salary, Allowance, Bonus, and Overtime Amount. Upon entering the details, the system confirms that the Employee ID is new and does not exist in the current records. After inputting all required data, the user is asked to confirm the entry. Once confirmed with "yes," the system saves the details and displays a success message, indicating that the employee's information and salary data have been successfully added to the system.

```
Employee data and salary details were not saved.

------ Employee Profile ------
1. Add Employee
2. Update Employee
3. Delete Employee
4. View Employee List
5. Return to Main Menu
Enter your choice: 1
Enter Employee ID: TP00001
This Employee ID already exists. Adding a new payslip record for this employee.
Enter Month (e.g., 'January', 'February'): February
Enter Year (e.g., 2023): 2023
Enter Basic Salary Amount: 0
Enter Allowance Amount: 0
Enter Bonus Amount: 0
Enter Overtime Amount: 0

Do you want to save these details? (yes/no): no
Employee data and salary details were not saved.
```

Scenario B: Cancelling the Addition of Payslip Record

In this scenario, an attempt is made to add a new payslip record for an existing employee (Employee ID: TP00001). The system prompts the user to enter details such as Month, Year, Basic Salary, Allowance, Bonus, and Overtime Amount. If there is an error in the entered details or the user decides not to proceed, they have the option to decline saving by responding "no" when asked for confirmation. As a result, a message confirms that the employee data and salary details were not saved, allowing the user to reattempt with correct information if needed.

```
------ Employee Profile ------
1. Add Employee
2. Update Employee
3. Delete Employee
4. View Employee List
5. Return to Main Menu
Enter your choice: 1
Enter Employee ID: TP00001
This Employee ID already exists. Adding a new payslip record for this employee.
Enter Month (e.g., 'January', 'February'): February
Enter Year (e.g., 2023): 2023
Enter Basic Salary Amount: 3000
Enter Allowance Amount: 300
Enter Bonus Amount: 1000
Enter Overtime Amount: 0

Do you want to save these details? (yes/no): yes
Employee data has been successfully added and salary details saved.
```

Scenario C: Adding Additional Monthly Details for Existing Employee

In this scenario, additional payslip details for 2023 are being added for the employee with ID TP00001. This process is consistent with Scenario A, where January details for the same year were initially recorded. The system recognizes the existing employee ID and facilitates the addition of a new month's payslip data, allowing for continuity in salary record-keeping within the year 2023. Upon confirmation with "yes," the new details are saved.

```
------ Employee Profile ------
1. Add Employee
2. Update Employee
3. Delete Employee
4. View Employee List
5. Return to Main Menu
Enter your choice: 1
Enter Employee ID: 000
Invalid Employee ID format. Please enter an ID in the format 'TP00001', 'TP00002', etc.
Enter Employee ID: TP0001
Invalid Employee ID format. Please enter an ID in the format 'TP00001', 'TP00002', etc.
Enter Employee ID: TPP
Invalid Employee ID format. Please enter an ID in the format 'TP00001', 'TP00002', etc.
Enter Employee ID: TP00004
This Employee ID does not exist yet. Adding it to the system.
Enter Employee Name: Kimberly Lim
Enter Department Name: Information Technology
Enter Month (e.g., 'January', 'February'): Setember
Invalid month. Please enter a valid month name.
Enter Month (e.g., 'January', 'February'): September
Enter Year (e.g., 2023): 2024
Enter Basic Salary Amount: 6000
Enter Allowance Amount: 1000
Enter Bonus Amount: 1000
Enter Overtime Amount: 0

Do you want to save these details? (yes/no): yes
Employee data has been successfully added and salary details saved.
```

Scenario D: Handling Invalid Employee ID and Month Inputs

In this scenario, the add employee function demonstrates robust input validation for Employee ID and Month fields. When entering an Employee ID, multiple incorrect formats ("000," "TP001," "TPP") trigger validation messages indicating the required format ("TP00001," "TP00002," etc.). Only upon entering a correctly formatted ID, "TP00004," does the system accept the input.

Similarly, when prompted to enter the month, an invalid entry ("Setember") is flagged by the system, prompting the user to re-enter a valid month name. After correcting the entry to "September," the system proceeds. This scenario highlights the function's capability to ensure data accuracy through rigorous format validation, guiding the user until correct input is provided.

5.5 Update Employee Procedure

```
------ Employee Profile ------
1. Add Employee
2. Update Employee
3. Delete Employee
4. View Employee List
5. Return to Main Menu
Enter your choice: 2
Enter Employee ID to update: TP00001
Enter Month (e.g., 'January', 'February'): January
Enter Year (e.g., 2023): 2023

Existing Payslip Data for Reference:
+------------------+---------+
|      Detail      |  Value  |
+------------------+---------+
|   Basic Salary   | 3000.00 |
|    Allowance     |  300.00 |
|      Bonus       | 1000.00 |
|     Overtime     |  500.00 |
|   Gross Salary   | 4800.00 |
|  EPF Deduction   |  528.00 |
| Additional Bonus |   0.00  |
|  Tax Deduction   |  256.32 |
|    Net Salary    | 4015.68 |
+------------------+---------+
Enter updated Basic Salary Amount: 3000
Enter updated Allowance Amount: 300
Enter updated Bonus Amount: 500
Enter updated Overtime Amount: 0
```

```
Updated Employee and Salary Details:

+------------------+------------+
|      Detail      |   Value    |
+------------------+------------+
|   Employee ID    |  TP00001   |
|  Employee Name   | Chloe Ting |
| Department Name  |  Finance   |
|       Year       |    2023    |
|      Month       |  January   |
|   Basic Salary   |  3000.00   |
|    Allowance     |   300.00   |
|      Bonus       |   500.00   |
|     Overtime     |    0.00    |
|   Gross Salary   |  3800.00   |
|  EPF Deduction   |   418.00   |
| Additional Bonus |    0.00    |
|  Tax Deduction   |   202.92   |
|    Net Salary    |  3179.08   |
+------------------+------------+

Do you want to overwrite the existing payslip with these updated details? (yes/no): yes
Employee data has been successfully updated.
```

Scenario A: Updating Existing Employee Payslip Details for January 2023

In this scenario, the user selects option 2 from the Employee Profile submenu to update an existing payslip for Employee ID TP00001 for the month of January in the year 2023. Upon entering the necessary updates (such as adjustments to salary components), the system displays the original payslip details for reference, allowing the user to confirm the existing values. The updated payslip details are shown in a formatted table, and the user is prompted to confirm if the new details should overwrite the previous records. By selecting "yes," the user finalizes the update, and the system successfully saves the new information, displaying a confirmation message to indicate that the employee data has been updated.

```
------ Employee Profile ------
1. Add Employee
2. Update Employee
3. Delete Employee
4. View Employee List
5. Return to Main Menu
Enter your choice: 2
Enter Employee ID to update: TP00001
Enter Month (e.g., 'January', 'February'): February
Enter Year (e.g., 2023): 2023

Existing Payslip Data for Reference:
+------------------+---------+
|      Detail      |  Value  |
+------------------+---------+
|   Basic Salary   | 3000.00 |
|    Allowance     |  300.00 |
|      Bonus       | 1000.00 |
|     Overtime     |   0.00  |
|   Gross Salary   | 4300.00 |
|  EPF Deduction   |  473.00 |
| Additional Bonus |   0.00  |
|  Tax Deduction   |  229.62 |
|    Net Salary    | 3597.38 |
+------------------+---------+
Enter updated Basic Salary Amount: 3000
Enter updated Allowance Amount: 300
Enter updated Bonus Amount: 1000
Enter updated Overtime Amount: 50
```

```
Updated Employee and Salary Details:

+------------------+------------+
|      Detail      |   Value    |
+------------------+------------+
|   Employee ID    |  TP00001   |
|  Employee Name   | Chloe Ting |
| Department Name  |  Finance   |
|       Year       |    2023    |
|      Month       |  February  |
|   Basic Salary   |  3000.00   |
|    Allowance     |   300.00   |
|      Bonus       |  1000.00   |
|     Overtime     |   50.00    |
|   Gross Salary   |  4350.00   |
|  EPF Deduction   |   478.50   |
| Additional Bonus |    0.00    |
|  Tax Deduction   |   232.29   |
|    Net Salary    |  3639.21   |
+------------------+------------+

Do you want to overwrite the existing payslip with these updated details? (yes/no): no
Employee update was canceled. No changes were saved.
```

Scenario B: Payslip Update Cancelled for Existing Employee in 2023

In this scenario, the system demonstrates that if the user decides not to save the updated payslip details, no changes are made to the original record. When asked, "Do you want to overwrite the

existing payslip with these updated details?" the user enters "no," resulting in the cancellation of the update process. The message "Employee update was cancelled. No changes were saved." confirms that the system preserves the existing details without applying any modifications.

```
------ Employee Profile ------
1. Add Employee
2. Update Employee
3. Delete Employee
4. View Employee List
5. Return to Main Menu
Enter your choice: 1
Enter Employee ID: TP00001
This Employee ID already exists. Adding a new payslip record for this employee.
Enter Month (e.g., 'January', 'February'): January
Enter Year (e.g., 2023): 2023
A payslip record for January 2023 already exists.
Would you like to update the existing record instead? (yes/no): yes

Existing Payslip Data for Reference:
+------------------+---------+
|      Detail      |  Value  |
+------------------+---------+
|   Basic Salary   | 3000.00 |
|    Allowance     |  300.00 |
|      Bonus       |  500.00 |
|     Overtime     |   0.00  |
|   Gross Salary   | 3800.00 |
|  EPF Deduction   |  418.00 |
| Additional Bonus |   0.00  |
|  Tax Deduction   |  202.92 |
|    Net Salary    | 3179.08 |
+------------------+---------+
Enter updated Basic Salary Amount: 3000
Enter updated Allowance Amount: 300
Enter updated Bonus Amount: 500
Enter updated Overtime Amount: 1500
```

```
Updated Employee and Salary Details:

+------------------+------------+
|      Detail      |   Value    |
+------------------+------------+
|   Employee ID    |  TP00001   |
|  Employee Name   | Chloe Ting |
| Department Name  |  Finance   |
|       Year       |    2023    |
|      Month       |  January   |
|   Basic Salary   |  3000.00   |
|    Allowance     |   300.00   |
|      Bonus       |   500.00   |
|     Overtime     |  1500.00   |
|   Gross Salary   |  5300.00   |
|  EPF Deduction   |   583.00   |
| Additional Bonus |    0.00    |
|  Tax Deduction   |   283.02   |
|    Net Salary    |  4433.98   |
+------------------+------------+

Do you want to overwrite the existing payslip with these updated details? (yes/no): yes
Employee data has been successfully updated.
```

Scenario C: Alternative Method for Updating an Existing Payslip Record Using the Add Function

In this scenario, an attempt is made to add a new payslip record for employee "TP00001" for January 2023, even though a record for this period already exists. The system prompts to confirm

whether the user wants to update the existing record. Upon selecting "yes," updated salary components, including an increase in overtime to 1500, are entered. The system displays the revised payslip details for confirmation, and, after final approval, the changes are saved successfully, confirming that the existing January 2023 record has been updated with the new values.

5.6 View All Employee Procedure

```
------ Employee Profile ------
1. Add Employee
2. Update Employee
3. Delete Employee
4. View Employee List
5. Return to Main Menu
Enter your choice: 4

Employee Records:
+-------------+----------------+------------------------+
| Employee ID |      Name      |       Department       |
+-------------+----------------+------------------------+
|   TP00002   |  Francis Then  |       Marketing        |
|   TP00003   |   Noel Chan    | Information Technology |
|   TP00004   |  Kimberly Lim  | Information Technology |
|   TP00005   | Elizabeth Wong |     Human Resource     |
|   TP00001   |   Chloe Ting   |        Finance         |
+-------------+----------------+------------------------+
```

Scenario A: Viewing All Employee Records

In this scenario, the user selects the option to view the list of all employee records within the system. Upon choosing option 4, "View Employee List," the system displays a formatted table with details for each employee, including their ID, name, and department. This output verifies that the function successfully retrieves and organizes employee information, allowing the user to review the current employee records comprehensively.

5.7 Delete Employee Procedure

```
------ Employee Profile ------
1. Add Employee
2. Update Employee
3. Delete Employee
4. View Employee List
5. Return to Main Menu
Enter your choice: 3
Enter Employee ID to delete: TP00005
Employee data for TP00005 has been successfully deleted from the employees dictionary.
Employee salary records for TP00005 have been successfully deleted from the payslips dictionary.
All changes have been successfully saved.
```

Scenario A: Deleting an Employee Record

In this scenario, the user selects the option to delete an employee by choosing "Delete Employee" from the Employee Profile menu. Upon entering a valid Employee ID (e.g., TP00005), the system confirms that the employee's data has been successfully removed from both the employees and payslips dictionaries. This deletion is followed by a message indicating that all changes have been saved.

5.8. Generate Salary Procedure

```
------- Main Menu -------
1. Employee Profile
2. Salary Generator
3. Payslip Profile
4. Exit
Enter your option: 2
Salary Generator has been chosen
Enter Employee ID: TP00005
Enter Month (e.g., 'January', 'February'): March
Enter Year (e.g., 2023): 2027
Enter Basic Salary Amount: 10000
Enter Allowance Amount: 2000
Enter Bonus Amount: 30000
Enter Overtime Amount: 4000

Calculated Salary Details:
Basic Salary: 10000.00
Allowance: 2000.00
Bonus: 30000.00
Overtime: 4000.00
Gross Salary: 46000.00
EPF Deduction: 5060.00
Additional Bonus: 0.00
Tax Deduction: 2456.40
Net Salary: 38483.60
Net salary calculation complete (data not saved).
```

Scenario A: Generating Salary Details for an Employee

In this scenario, the Salary Generator function is used to compute the salary details for an employee with ID "TP00001" for January 2023. The user enters the basic salary, allowance, bonus, and overtime amounts, after which the system calculates the gross salary, EPF deduction, and applicable tax deductions based on predefined criteria.

The calculated breakdown is displayed, showing each salary component, including the net salary. It is noted that this calculation is for reference only, as the data is not saved, as indicated by the message "Net salary calculation complete (data not saved)."

```
------- Main Menu -------
1. Employee Profile
2. Salary Generator
3. Payslip Profile
4. Exit
Enter your option: 2
Salary Generator has been chosen
Enter Employee ID: TP00008
Employee ID not found. Please check and try again.
```

Scenario B: Handling an Invalid Employee ID in Salary Generation

In this scenario, the Salary Generator function is tested with an invalid Employee ID input ("TP00008"). Since this ID does not exist in the system, an error message—"Employee ID not found. Please check and try again."—is displayed. This scenario confirms that the function effectively validates employee IDs and prompts the user to re-enter a valid ID if the input does not match any existing records, preventing any calculation errors due to non-existent employee data.

5.9. Search Specific Payslip Procedure

```
------- Main Menu -------
1. Employee Profile
2. Salary Generator
3. Payslip Profile
4. Exit
Enter your option: 3
Payslip Profile has been chosen

*** Payslip ***
1. Search Specific Payslip
2. View All Payslips
3. Return to Main Menu
Enter your choice: 1
Enter Employee ID: TP00001
Enter Month (e.g., 'January', 'February'): January
Enter Year (e.g., 2023): 2023

Payslip for Employee ID: TP00001 for January 2023
+------------------+---------+
|      Detail      |  Value  |
+------------------+---------+
|   Basic Salary   | 3000.00 |
|    Allowance     |  300.00 |
|      Bonus       |  500.00 |
|     Overtime     | 1500.00 |
|   Gross Salary   | 5300.00 |
|  EPF Deduction   |  583.00 |
| Additional Bonus |   0.00  |
|  Tax Deduction   |  283.02 |
|    Net Salary    | 4433.98 |
+------------------+---------+
```

Scenario A: Searching for a Specific Payslip

In this scenario, the system demonstrates the ability to retrieve detailed payslip information for a specified employee (Employee ID: TP00001) for a particular month and year, as selected by the user. By selecting the "Search Specific Payslip" option, the user can enter the employee ID along with the desired month and year to view detailed payslip data, including basic salary, allowances, deductions, and net salary. This feature facilitates quick access to an individual’s specific payroll details for review or verification.

```
*** Payslip ***
1. Search Specific Payslip
2. View All Payslips
3. Return to Main Menu
Enter your choice: 1
Enter Employee ID: TP00008
No records found for this Employee ID.
Would you like to try again with a different Employee ID? (yes/no): yes
Enter Employee ID: TP00009
No records found for this Employee ID.
Would you like to try again with a different Employee ID? (yes/no): no
Returning to the main menu.

*** Payslip ***
1. Search Specific Payslip
2. View All Payslips
3. Return to Main Menu
Enter your choice:
```

Scenario B: Handling Non-Existent Employee ID in Payslip Search

In this scenario, the user attempts to retrieve payslip details for employee IDs (e.g., TP00008 and TP00009) that do not exist in the system. When an unregistered employee ID is entered, the system displays a message indicating that no records were found and prompts the user to try again with a different ID. The user can choose to re-enter an ID or return to the main menu. This scenario highlights the system's error-handling capability, ensuring that only valid employee records are accessed while providing a clear, user-friendly option for retrying or exiting.

5.10 View All Payslips Procedure

```
*** Payslip ***
1. Search Specific Payslip
2. View All Payslips
3. Return to Main Menu
Enter your choice: 2
Enter Employee ID: TP00001

Payslip Summary for Employee ID: TP00001 for Year: 2024
+---------+--------------+-----------+---------+----------+--------------+--------------+-----------------+--------------+------------+
|  Month  | Basic Salary | Allowance |  Bonus  | Overtime | Gross Salary | EPF Deduction | Additional Bonus | Tax Deduction | Net Salary |
+---------+--------------+-----------+---------+----------+--------------+--------------+-----------------+--------------+------------+
| January |   3500.00    |   350.00  | 1000.00 |   0.00   |   4850.00    |     533.50    |       0.00      |     258.99    |  4057.51   |
+---------+--------------+-----------+---------+----------+--------------+--------------+-----------------+--------------+------------+

Payslip Summary for Employee ID: TP00001 for Year: 2023
+----------+--------------+-----------+---------+----------+--------------+--------------+-----------------+--------------+------------+
|  Month   | Basic Salary | Allowance |  Bonus  | Overtime | Gross Salary | EPF Deduction | Additional Bonus | Tax Deduction | Net Salary |
+----------+--------------+-----------+---------+----------+--------------+--------------+-----------------+--------------+------------+
| January  |   3000.00    |   300.00  |  500.00 | 1500.00  |   5300.00    |     583.00    |       0.00      |     283.02    |  4433.98   |
| February |   3000.00    |   300.00  | 1000.00 |   0.00   |   4300.00    |     473.00    |       0.00      |     229.62    |  3597.38   |
+----------+--------------+-----------+---------+----------+--------------+--------------+-----------------+--------------+------------+

Payslip Summary for Employee ID: TP00001 for Year: 2022
+---------+--------------+-----------+--------+----------+--------------+--------------+-----------------+--------------+------------+
|  Month  | Basic Salary | Allowance | Bonus  | Overtime | Gross Salary | EPF Deduction | Additional Bonus | Tax Deduction | Net Salary |
+---------+--------------+-----------+--------+----------+--------------+--------------+-----------------+--------------+------------+
| January |   2800.00    |   100.00  | 200.00 |   0.00   |   3100.00    |     341.00    |       0.00      |     0.00     |  2759.00   |
+---------+--------------+-----------+--------+----------+--------------+--------------+-----------------+--------------+------------+
```

Scenario A: Viewing All Payslips for an Employee

This scenario demonstrates the functionality of viewing all payslips associated with a specific employee ID, organized by year. Upon selecting the "View All Payslips" option, the user is prompted to enter an employee ID. The system then retrieves and displays the payslip summaries for each year, including the year 2022, 2023, and 2024 for Employee ID TP00001. Each summary includes detailed information such as basic salary, allowance, bonus, overtime, gross salary, deductions, and net salary. This feature allows users to easily review an employee's entire payslip history across multiple years.

```
*** Payslip ***
1. Search Specific Payslip
2. View All Payslips
3. Return to Main Menu
Enter your choice: 2
Enter Employee ID: TP00002

Payslip Summary for Employee ID: TP00002 for Year: 2024
+-------+-------------+-----------+-------+----------+--------------+---------------+-----------------+---------------+------------+
| Month | Basic Salary | Allowance | Bonus | Overtime | Gross Salary | EPF Deduction | Additional Bonus | Tax Deduction | Net Salary |
+-------+-------------+-----------+-------+----------+--------------+---------------+-----------------+---------------+------------+
| March |   1000.00    |   200.00  | 0.00  |   0.00   |   1200.00    |     132.00    |      53.40      |     0.00      |  1121.40   |
+-------+-------------+-----------+-------+----------+--------------+---------------+-----------------+---------------+------------+
```

```
*** Payslip ***
1. Search Specific Payslip
2. View All Payslips
3. Return to Main Menu
Enter your choice: 2
Enter Employee ID: TP00001

Payslip Summary for Employee ID: TP00001 for Year: 2024
+---------+-------------+-----------+---------+----------+--------------+---------------+-----------------+---------------+------------+
|  Month  | Basic Salary | Allowance |  Bonus  | Overtime | Gross Salary | EPF Deduction | Additional Bonus | Tax Deduction | Net Salary |
+---------+-------------+-----------+---------+----------+--------------+---------------+-----------------+---------------+------------+
| January |   3500.00    |   350.00  | 1000.00 |   0.00   |   4850.00    |     533.50    |      0.00       |     258.99    |  4057.51   |
+---------+-------------+-----------+---------+----------+--------------+---------------+-----------------+---------------+------------+

Payslip Summary for Employee ID: TP00001 for Year: 2023
+----------+-------------+-----------+--------+----------+--------------+---------------+-----------------+---------------+------------
|  Month   | Basic Salary | Allowance |  Bonus | Overtime | Gross Salary | EPF Deduction | Additional Bonus | Tax Deduction | Net Salary
+----------+-------------+-----------+--------+----------+--------------+---------------+-----------------+---------------+------------
| January  |   3000.00    |   300.00  | 500.00 | 1500.00  |   5300.00    |     583.00    |      0.00       |     283.02    |  4433.98
| February |   3000.00    |   300.00  | 1000.00|   0.00   |   4300.00    |     473.00    |      0.00       |     229.62    |  3597.38
+----------+-------------+-----------+--------+----------+--------------+---------------+-----------------+---------------+------------

Payslip Summary for Employee ID: TP00001 for Year: 2022
+---------+-------------+-----------+-------+----------+--------------+---------------+-----------------+---------------+------------+
|  Month  | Basic Salary | Allowance | Bonus | Overtime | Gross Salary | EPF Deduction | Additional Bonus | Tax Deduction | Net Salary |
+---------+-------------+-----------+-------+----------+--------------+---------------+-----------------+---------------+------------+
| January |   2800.00    |   100.00  | 200.00|   0.00   |   3100.00    |     341.00    |      0.00       |     0.00      |  2759.00   |
+---------+-------------+-----------+-------+----------+--------------+---------------+-----------------+---------------+------------+
```

Scenario B: Comprehensive Validation of Net Salary Conditions

The presented screenshots validate that the payroll system correctly applies bonus and tax adjustments based on gross salary thresholds. In the first screenshot, the employee's gross salary triggers a 5% additional bonus as it falls below RM2000, in alignment with the system's condition for low gross salaries. In the second screenshot, tax deductions of 6% are accurately applied for 2024 and 2023 records where the gross salary exceeds RM3000, while the 2022 records remain unaffected by bonus or tax adjustments, meeting the condition for salaries between RM2000 and RM3000. This demonstrates the system's effective execution of all predefined conditions for bonus and tax application.

6.0 Conclusion

In conclusion, this payroll system successfully implements key functions and procedures, including managing employee profiles, generating and updating salaries, and viewing payslip records. These functions and procedures allow efficient handling of employee and salary data, supporting payroll tasks such as adding, updating, and viewing records. Currently, the system only allows for the deletion of payslip data by removing the entire employee record, which includes both the employee's profile and all associated payslip records. For potential improvements, implementing a *deletePayslip* procedure would allow users to remove individual payslip entries without affecting the complete employee record. This would be particularly beneficial when correcting isolated payslip errors, as it would save considerable effort, especially for long-term employees with extensive payslip histories. Additionally, the *viewEmployee* procedure could be refined by arranging employees in ascending order based on Employee ID, facilitating quicker and more logical access to employee information.

Details on the current system's limitations are included in the appendix - issues related to *deletePayslip* can be found in *Appendix A*, and suggestions for *viewEmployee* are outlined in *Appendix B*. While simple in design, this program establishes a strong foundation for a more advanced and comprehensive Payroll Management System, laying the groundwork for future development to support the Human Resources team's evolving needs.

Appendices

Appendix A: Proposed Delete Payslip Function for Error Correction

```
*** Payslip ***
1. Search Specific Payslip
2. View All Payslips
3. Return to Main Menu
Enter your choice: 2
Enter Employee ID: TP00007

Payslip Summary for Employee ID: TP00007 for Year: 3030
+---------+--------------+-----------+-------+----------+--------------+---------------+-----------------+---------------+------------+
|  Month  | Basic Salary | Allowance | Bonus | Overtime | Gross Salary | EPF Deduction | Additional Bonus | Tax Deduction | Net Salary |
+---------+--------------+-----------+-------+----------+--------------+---------------+-----------------+---------------+------------+
| January |     0.00     |    5.00   |  0.00 |   0.00   |     5.00     |      0.55     |       0.22      |      0.00     |    4.67    |
+---------+--------------+-----------+-------+----------+--------------+---------------+-----------------+---------------+------------+

Payslip Summary for Employee ID: TP00007 for Year: 2023
+---------+--------------+-----------+-------+----------+--------------+---------------+-----------------+---------------+------------+
|  Month  | Basic Salary | Allowance | Bonus | Overtime | Gross Salary | EPF Deduction | Additional Bonus | Tax Deduction | Net Salary |
+---------+--------------+-----------+-------+----------+--------------+---------------+-----------------+---------------+------------+
| January |     8.00     |    8.00   |  8.00 |   8.00   |    32.00     |      3.52     |       1.42      |      0.00     |    29.90   |
+---------+--------------+-----------+-------+----------+--------------+---------------+-----------------+---------------+------------+
```

The payslip entry for the year 3030, with a basic salary of 0 and an allowance of 5, is clearly an error. Implementing a *deletePayslip* function would allow for targeted deletion of this single incorrect payslip without requiring the deletion of the entire employee record, which would otherwise remove all associated salary data. This enhancement would streamline corrections and prevent the need to re-enter complete employee and salary details.

Appendix B: Suggested Improvement for Employee List Ordering

```
------ Employee Profile ------
1. Add Employee
2. Update Employee
3. Delete Employee
4. View Employee List
5. Return to Main Menu
Enter your choice: 4

Employee Records:
+-------------+----------------+------------------------+
| Employee ID |      Name      |       Department       |
+-------------+----------------+------------------------+
|   TP00002   |  Francis Then  |       Marketing        |
|   TP00003   |   Noel Chan    | Information Technology |
|   TP00004   |  Kimberly Lim  | Information Technology |
|   TP00005   | Elizabeth Wong |     Human Resource     |
|   TP00001   |   Chloe Ting   |        Finance         |
+-------------+----------------+------------------------+
```

In the screenshot provided, the employee list shows "TP00001" as the last entry, even though it should appear at the top to maintain sequential order based on Employee ID. Arranging employees in ascending order by Employee ID would improve readability, allowing users to locate specific employees more efficiently. Implementing this ordering in the *viewEmployee* function would enhance logical flow and user experience, especially as the employee database grows.

Appendix C: Structure of Employee Data Storage in *employees.json* File

This screenshot shows the *employees.json* file, which stores all employee data within the system. The data in this file is organized by Employee ID, with each entry containing relevant details like the employee's name and department. It is worth noting that the details in this file may not be fully consistent with those shown in the scenarios in *Section 5.0*, as the purpose of displaying this file is simply to illustrate the data structure and storage method rather than the exact content.

Appendix D: Structure of Payslip Data Storage in *payslips.json* File

This screenshot shows the *payslips.json* file, which stores all the payslip data within the system. Each employee's records are organized by Employee ID, followed by the year and month, and include details such as basic salary, allowances, bonuses, and other components. Similar to *Appendix C*, data in this file may not fully align with the information in the screenshots in *Section 5.0*. The purpose of this display is to illustrate the file structure and format for storing payslip information, rather than to focus on specific content within the file.