Batch:	**D3**	**Roll No.:**	**16010123294**

Experiment / assignment / tutorial No. 03

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

TITLE : Implementing a billing application using OOP concepts using C++

AIM: Develop a C++ application that generates an Electricity Bill using a Consumer class.

Expected OUTCOME of Experiment:

CO1:Apply the features of object oriented programming languages. (C++ and Java)
CO2:Explore arrays, vectors, classes and objects in C++ and Java

---

Books/ Journals/ Websites referred:

1. E. Balagurusamy, “Programming with Java”, McGraw-Hill.
2. E. Balagurusamy, “Object Oriented Programming with C++”, McGraw-Hill.

Pre Lab/ Prior Concepts:

Class Definition:
The Consumer class should encapsulate the following information:

- consumer_no (integer): Unique identification number for the consumer.
- consumer_name (string): Name of the consumer.
- previous_reading (integer): Meter reading from the previous month.
- current_reading (integer): Meter reading from the current month.
- connection_type (string): Type of electricity connection (domestic or commercial).
- calculate_bill (member function): This function should calculate the electricity bill amount based on the connection_type and the number of units consumed (current reading - previous reading). The function should utilize a tiered pricing structure as

specified below:

Tiered Pricing:

Domestic Connection:
First 100 units: Rs. 1 per unit
101-200 units: Rs. 2.50 per unit
201-500 units: Rs. 4 per unit
Above 501 units: Rs. 6 per unit

Commercial Connection:
First 100 units: Rs. 2 per unit
101-200 units: Rs. 4.50 per unit
201-500 units: Rs. 6 per unit
Above 501 units: Rs. 7 per unit

Additional Considerations:

- The application should prompt the user to enter the details for a consumer (consumer number, name, previous reading, current reading, and connection type).
- The calculate_bill function should implement logic to determine the applicable unit charges based on the connection type and the number of units consumed within each tier.
- The application should display a clear breakdown of the bill, including the consumer details, number of units consumed, charge per unit for each tier, and the total bill amount.

Algorithm:

Start

Input Consumer Details:

- Prompt the user to enter the consumer number.
- Prompt the user to enter the consumer name.
- Prompt the user to enter the previous meter reading.
- Prompt the user to enter the current meter reading.
- Prompt the user to enter the connection type (Domestic or Commercial).

Calculate Units Consumed:

- Subtract the previous meter reading from the current meter reading to get the units consumed.

Calculate Bill Amount:

- **If the connection type is "Domestic":**

If units consumed are less than or equal to 100, the rate is 1.0 per unit.

If units are between 101 and 200, the first 100 units are billed at 1.0 per unit, and the remaining units at 2.5 per unit.

If units are between 201 and 500, the first 100 units are billed at 1.0 per unit, the next 100 units at 2.5 per unit, and the remaining units at 4.0 per unit.

If units exceed 500, the first 100 units are billed at 1.0 per unit, the next 100 units at 2.5 per unit, the next 300 units at 4.0 per unit, and the remaining units at 6.0 per unit.

- **If the connection type is "Commercial":**

If units consumed are less than or equal to 100, the rate is 2.0 per unit.

If units are between 101 and 200, the first 100 units are billed at 2.0 per unit, and the remaining units at 4.5 per unit.

If units are between 201 and 500, the first 100 units are billed at 2.0 per unit, the next 100 units at 4.5 per unit, and the remaining units at 6.0 per unit.

If units exceed 500, the first 100 units are billed at 2.0 per unit, the next 100 units at 4.5 per unit, the next 300 units at 6.0 per unit, and the remaining units at 7.0 per unit.

Display Bill Details:

- Print the consumer name.
- Print the consumer number.
- Print the units consumed.
- Print the total bill amount.

End

Implementation details:

```
#include<iostream>
#include<string>
using namespace std;

class Bill {
   int cons_Id;
   string cons_name;
   int prev_read;
   int curr_read;
   string type;

   int cal_unit() {
      return curr_read - prev_read;
   }

   double cal_amount(int units, const string& type) {
      double amount = 0;
      if (type == "Domestic") {
         if (units <= 100) {
            amount = units * 1.0;
         } else if (units <= 200) {
            amount = 100 * 1.0 + (units - 100) * 2.5;
```

```
        } else if (units <= 500) {
            amount = 100 * 1.0 + 100 * 2.5 + (units - 200) * 4.0;
        } else {
            amount = 100 * 1.0 + 100 * 2.5 + 300 * 4.0 + (units - 500) * 6.0;
        }
    } else if (type == "Commercial") {
        if (units <= 100) {
            amount = units * 2.0;
        } else if (units <= 200) {
            amount = 100 * 2.0 + (units - 100) * 4.5;
        } else if (units <= 500) {
            amount = 100 * 2.0 + 100 * 4.5 + (units - 200) * 6.0;
        } else {
            amount = 100 * 2.0 + 100 * 4.5 + 300 * 6.0 + (units - 500) * 7.0;
        }
    }
    return amount;
}

public:
    void get_input() {
        cout << "Enter Consumer Number: ";
        cin >> cons_Id;
```

```
        cout << "Enter name: ";
        cin.ignore(); // Ignore leftover newline character from previous input
        getline(cin, cons_name);
        cout << "Enter Previous Reading: ";
        cin >> prev_read;
        cout << "Enter Current Reading: ";
        cin >> curr_read;
        cout << "Enter connection type (Domestic/Commercial): ";
        cin >> type;
    }

    void display() {
        int units = cal_unit();
        double amount = cal_amount(units, type);
        cout << "Consumer Name: " << cons_name << endl;
        cout << "Consumer Id: " << cons_Id << endl;
        cout << "Units Consumed: " << units << endl;
        cout << "Bill Amount: " << amount << endl;
    }
};

int main() {
    Bill bill;
```

```
    bill.get_input();
    bill.display();
    return 0;
}
```

Output:

```
Enter Consumer Number: 1
Enter name: Saish
Enter Previous Reading: 750
Enter Current Reading: 1690
Enter connection type (Domestic/Commercial): Domestic
Consumer Name: Saish
Consumer Id: 1
Units Consumed: 940
Bill Amount: 4190
```

Conclusion:

Applied concepts of C++ and used its application for Bill Generation .

Date: _____ **Signature of faculty in-charge**

Post Lab Descriptive Questions:

Q.1 Explain the concept of constructors and destructors in C++.

Purpose: Constructors are used to initialize objects of a class. They set up the initial state of an object when it is created.

Characteristics:

Same Name as the Class: A constructor has the same name as the class and does not have a return type, not even void.

Automatic Invocation: Constructors are automatically called when an object of the class is created.

Overloading: You can overload constructors to provide different ways of initializing objects.

Default Constructor: If no constructor is defined, C++ provides a default constructor that initializes members with default values.

Parameterized Constructor: Allows passing arguments to initialize the object with specific values.

Purpose: Destructors are used to perform cleanup tasks when an object is destroyed, such as releasing resources (e.g., memory, file handles).

Characteristics:

Same Name as the Class with a Tilde (~): A destructor has the same name as the class but is preceded by a tilde (~).

No Return Type and No Parameters: Destructors do not have a return type and cannot take parameters.

Automatic Invocation: A destructor is automatically called when an object goes out of scope or is explicitly deleted.

Single Destructor: A class can have only one destructor, and it cannot be overloaded.

Q.2 Write the output of following program with suitable explanation

```
#include<iostream>
```

```
using namespace std;

class Test
{
  static int i;
  int j;
};

int Test::i;

int main()
{
    cout << sizeof(Test);
    return 0;
}
```

Output:

4

Since int i is declared Static it does not contribute to the class memory thus the output is 4 bytes.

Q.3 Explain all the applications of the scope resolution operator in C++.

Accessing Global Variables/Functions: Used to access a global variable or function when a local variable or function with the same name exists.

Defining Class Members Outside the Class: Used to define member functions or static variables outside the class definition.

Accessing Static Members of a Class: Used to access static variables or static functions of a class.

Accessing Nested Classes/Enums: Used to access classes, structs, or enums that are nested within another class or namespace.

Resolving Ambiguity in Multiple Inheritance: Used to specify which base class's member to access in cases of multiple inheritance.