**Batch: C3 Roll No.: 16010123217**

**Experiment / assignment / tutorial No. 3**

**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:
  + Consumer number.
  + Consumer name.
  + Previous meter reading.
  + Current meter reading.
  + Connection type (Domestic or Commercial).

 **Calculate Units Consumed**:

* Subtract the previous reading from the current reading to calculate the total units consumed.

 **Determine Connection Type**:

* Check if the connection type is "Domestic" or "Commercial". Proceed based on the connection type.

 **Bill Calculation Based on Connection Type**:

* **For Domestic Connection**:
  + If units consumed ≤ 100:
    - Charge 1.0 per unit.
  + If units consumed > 100 and ≤ 200:
    - Charge 1.0 per unit for the first 100 units.
    - Charge 2.5 per unit for the next units up to 200.
  + If units consumed > 200 and ≤ 500:
    - Charge 1.0 per unit for the first 100 units.
    - Charge 2.5 per unit for the next 100 units (101 to 200).
    - Charge 4.0 per unit for the next units up to 500.
  + If units consumed > 500:
    - Charge 1.0 per unit for the first 100 units.
    - Charge 2.5 per unit for the next 100 units (101 to 200).
    - Charge 4.0 per unit for the next 300 units (201 to 500).
    - Charge 6.0 per unit for the remaining units.
* **For Commercial Connection**:
  + If units consumed ≤ 100:
    - Charge 2.0 per unit.
  + If units consumed > 100 and ≤ 200:
    - Charge 2.0 per unit for the first 100 units.
    - Charge 4.5 per unit for the next units up to 200.
  + If units consumed > 200 and ≤ 500:
    - Charge 2.0 per unit for the first 100 units.
    - Charge 4.5 per unit for the next 100 units (101 to 200).
    - Charge 6.0 per unit for the next units up to 500.
  + If units consumed > 500:
    - Charge 2.0 per unit for the first 100 units.
    - Charge 4.5 per unit for the next 100 units (101 to 200).
    - Charge 6.0 per unit for the next 300 units (201 to 500).
    - Charge 7.0 per unit for the remaining units.

 **Display Bill Breakdown**:

* Display the breakdown of charges, showing:
  + Total units consumed.
  + Charge per unit for each tier.
  + Total amount for each tier.

 **Display Consumer Bill**:

* Print the consumer's details including:
  + Consumer number.
  + Consumer name.
  + Previous reading.
  + Current reading.
  + Units consumed.
  + Connection type.
  + Total bill amount.

 **End**.

**Implementation details:**

#include <bits/stdc++.h>

using namespace std;

class Consumer

{

    int consumer\_no;

    string consumer\_name;

    int previous\_reading;

    int current\_reading;

    string connection\_type;

    double total;

    public:

    Consumer()

    {

        cout << "Enter consumer number: "; cin >> consumer\_no;

        cout << "\nEnter consumer name: "; cin >> consumer\_name;

        cout << "\nEnter previous reading: "; cin >> previous\_reading;

        cout << "\nEnter current reading: "; cin >> current\_reading;

        cout << "\nEnter connection type: (Domestic/Commercial)\n"; cin >> connection\_type;

        cout << endl;

    }

    void calculate\_bill()

    {

        int units = current\_reading - previous\_reading;

        total = 0;

        cout << "Bill Breakdown:\n";

        cout << "-------------------------------------------\n";

        cout << "Units consumed: " << units << " units\n";

        if (connection\_type == "Domestic" || connection\_type == "domestic" || connection\_type == "DOMESTIC")

        {

            cout << "Connection type: Domestic\n";

            if (units <= 100)

            {

                total = units \* 1.0;

                cout << "First 100 units @ Rs. 1.0/unit: " << total << "\n";

            }

            else if (units <= 200)

            {

                total = 100 \* 1.0 + (units - 100) \* 2.5;

                cout << "First 100 units @ Rs. 1.0/unit: 100.0\n";

                cout << "Next " << units - 100 << " units @ Rs. 2.5/unit: " << (units - 100) \* 2.5 << "\n";

            }

            else if (units <= 500)

            {

                total = 100 \* 1.0 + 100 \* 2.5 + (units - 200) \* 4.0;

                cout << "First 100 units @ Rs. 1.0/unit: 100.0\n";

                cout << "Next 100 units @ Rs. 2.5/unit: 250.0\n";

                cout << "Next " << units - 200 << " units @ Rs. 4.0/unit: " << (units - 200) \* 4.0 << "\n";

            }

            else

            {

                total = 100 \* 1.0 + 100 \* 2.5 + 300 \* 4.0 + (units - 500) \* 6.0;

                cout << "First 100 units @ Rs. 1.0/unit: 100.0\n";

                cout << "Next 100 units @ Rs. 2.5/unit: 250.0\n";

                cout << "Next 300 units @ Rs. 4.0/unit: 1200.0\n";

                cout << "Next " << units - 500 << " units @ Rs. 6.0/unit: " << (units - 500) \* 6.0 << "\n";

            }

        }

        else if (connection\_type == "Commercial" || connection\_type == "commercial" || connection\_type == "COMMERCIAL")

        {

            cout << "Connection type: Commercial\n";

            if (units <= 100)

            {

                total = units \* 2.0;

                cout << "First 100 units @ Rs. 2.0/unit: " << total << "\n";

            }

            else if (units <= 200)

            {

                total = 100 \* 2.0 + (units - 100) \* 4.5;

                cout << "First 100 units @ Rs. 2.0/unit: 200.0\n";

                cout << "Next " << units - 100 << " units @ Rs. 4.5/unit: " << (units - 100) \* 4.5 << "\n";

            }

            else if (units <= 500)

            {

                total = 100 \* 2.0 + 100 \* 4.5 + (units - 200) \* 6.0;

                cout << "First 100 units @ Rs. 2.0/unit: 200.0\n";

                cout << "Next 100 units @ Rs. 4.5/unit: 450.0\n";

                cout << "Next " << units - 200 << " units @ Rs. 6.0/unit: " << (units - 200) \* 6.0 << "\n";

            }

            else

            {

                total = 100 \* 2.0 + 100 \* 4.5 + 300 \* 6.0 + (units - 500) \* 7.0;

                cout << "First 100 units @ Rs. 2.0/unit: 200.0\n";

                cout << "Next 100 units @ Rs. 4.5/unit: 450.0\n";

                cout << "Next 300 units @ Rs. 6.0/unit: 1800.0\n";

                cout << "Next " << units - 500 << " units @ Rs. 7.0/unit: " << (units - 500) \* 7.0 << "\n";

            }

        }

        cout << "-------------------------------------------\n";

    }

    void show\_bill()

    {

        cout << "-------------------------------------------\n";

        cout << "TOTAL BILL : \n";

        cout << "-------------------------------------------\n";

        cout << "Consumer no: " << consumer\_no << "\n";

        cout << "Consumer name: " << consumer\_name << "\n";

        cout << "Previous reading: " << previous\_reading << "\n";

        cout << "Current reading: " << current\_reading << "\n";

        cout << "Units consumed: " << current\_reading - previous\_reading << "\n";

        cout << "Connection type: " << connection\_type << "\n";

        cout << "Total amount = Rs. " << total << "\n";

        cout << "-------------------------------------------\n";

    }

};

int main()

{

    Consumer c1;

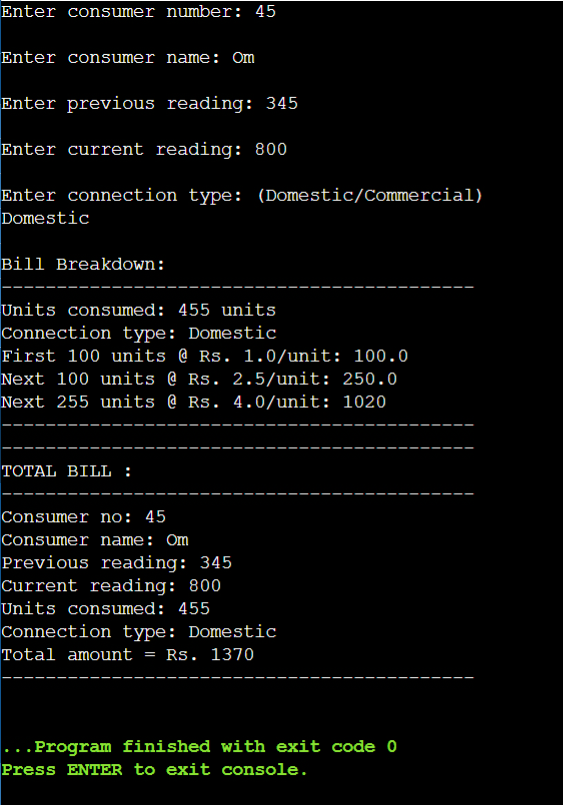
    c1.calculate\_bill();

    c1.show\_bill();

    return 0;

}

**Output:**



**Conclusion:**

**We learnt how to create a class in C++ by generating electricity bill for the consumer.**

**Date: 05/09/24 Signature of faculty in-charge**

**Post Lab Descriptive Questions:**

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

Ans.

* Constructor and Destructor are the special member functions of the class which are created by the C++ compiler or can be defined by the user.
* Constructor is called by the compiler whenever the object of the class is created, it allocates the memory to the object and initializes class data members.
* A destructor is called by the compiler when the object is destroyed and its main function is to deallocate the memory of the object.
* Constructors have the same as of class while destructors have the same name of the class with the prefix a tilde (~) operator.
* Both Constructor and destructor can be defined as public, private, or protected. But it is better to declare the constructor as public.
* The constructor can have parameters but the destructor doesn’t receive any parameters.

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

The program prints 4 because only the non-static variable j (which is an integer) is counted in the size of the class. The static variable i is stored separately and doesn't affect the size of individual objects.

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

 **Accessing Global Variables**: Used to access a global variable when there is a local variable with the same name.

 **Defining Functions Outside a Class**: Used to define a class function outside the class.

 **Accessing Static Class Members**: Used to access static variables of a class, even if a local variable has the same name.

 **Resolving Ambiguity in Multiple Inheritance**: Used to differentiate between variables in base classes when there is a name conflict in multiple inheritance.

 **Referring to a Namespace**: Used to refer to classes or variables in a specific namespace to avoid name conflicts.

 **Nested Classes**: Used to refer to a nested class or its members from the outer class.

 **Accessing Base Class Methods in Derived Class**: Used to access methods of the base class in the derived class when both have methods with the same name.