Batch: A1 **Roll No.: 16010123012**

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:

<u>Tiered Pricing:</u>

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:

```
1. Initialize Variables-

2. Input Details-

3. Calculate units_consumed as current_reading - previous_reading

4. Calculate Bill Amount:

        Check the connection_type:

        If connection_type is "Domestic"

        Else if connection_type is "Commercial"

        Else Invalid connection type and terminate calculation.

5.Print Consumer Details and Bill Amount-
```

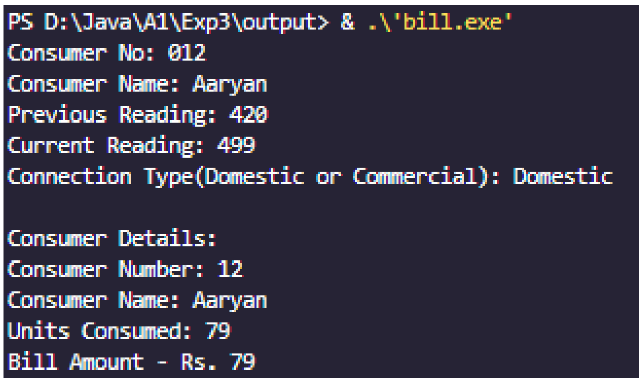
Implementation details:

```
#include<iostream>
```

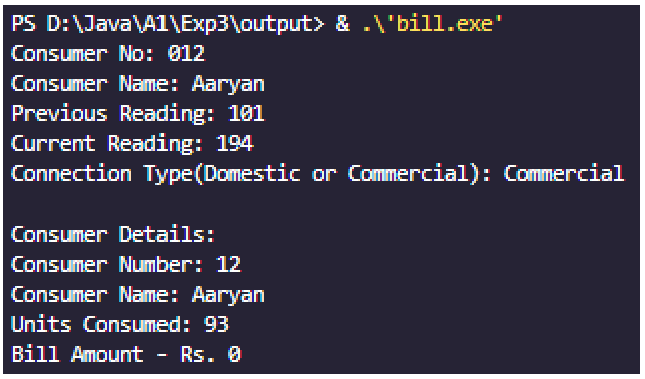
```
#include<string>

using namespace std;

class bill

{

private:

int consumer_no ;

int previous_reading ;

int current_reading ;

string consumer_name ;

string connection_type ;

public:

void get_details() {

    cout << "Consumer No: ";

    cin >> consumer_no ;

    cout << "Consumer Name: ";

    cin >> consumer_name ;

    cout << "Previous Reading: ";

    cin >> previous_reading ;

    cout << "Current Reading: ";

    cin >> current_reading ;

    cout << "Connection Type(Domestic or Commercial): ";
    cin >> connection_type ;
    }
void calculate_bill() {
    int units_consumed = current_reading - previous_reading;
    float bill_amount = 0;
    if (connection_type == "Domestic") {
        if (units_consumed <= 100) {
            bill_amount = units_consumed * 1.00 ;
        } else if (units_consumed <= 200) {
            bill_amount = 100 * 1.00 + (units_consumed - 100) * 2.50 ;
        } else if (units_consumed <= 500) {
            bill_amount = 100 * 1.00 + 100 * 2.50 + 300 * 4.00 ;
        } else if (units_consumed > 500) {
            bill_amount = 100 * 1.00 + 100 * 2.50 + 300 * 4.00 + (units_consumed - 500) *
6.00 ;
        }
    }
    else if (connection_type == "Commercial") {
        if (units_consumed <= 100) {
            bill_amount = units_consumed * 2.00 ;
        } else if (units_consumed <= 200) {
```

```
        bill_amount = 100 * 2.00 + (units_consumed - 100) * 4.50 ;
      } else if (units_consumed <= 500) {
        bill_amount = 100 * 2.00 + 100 * 4.50 + 300 * 6.00 ;
      } else if (units_consumed > 500) {
        bill_amount = 100 * 2.00 + 100 * 4.50 + 300 * 6.00 + (units_consumed - 500) *
7.00 ;
      }
    } else {
      cout << "Invalid connection type" << endl;
      return;
    }
    cout << "\nConsumer Details:" << endl;
    cout << "Consumer Number: " << consumer_no << endl;
    cout << "Consumer Name: " << consumer_name << endl;
    cout << "Units Consumed: " << units_consumed << endl;
    cout << "Bill Amount - Rs. " << bill_amount << endl;
}
};
int main() {
  bill bill;
  bill.get_details();
  bill.calculate_bill();

  return 0;

}
```

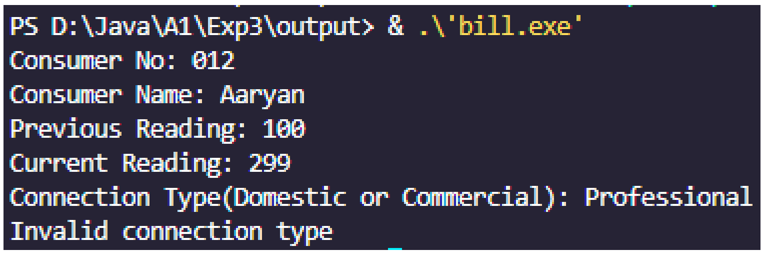
Output:



```
PS D:\Java\A1\Exp3\output> & .\'bill.exe'
Consumer No: 012
Consumer Name: Aaryan
Previous Reading: 420
Current Reading: 499
Connection Type(Domestic or Commercial): Domestic

Consumer Details:
Consumer Number: 12
Consumer Name: Aaryan
Units Consumed: 79
Bill Amount - Rs. 79
```



```
PS D:\Java\A1\Exp3\output> & .\'bill.exe'
Consumer No: 012
Consumer Name: Aaryan
Previous Reading: 101
Current Reading: 194
Connection Type(Domestic or Commercial): Commercial

Consumer Details:
Consumer Number: 12
Consumer Name: Aaryan
Units Consumed: 93
Bill Amount - Rs. 0
```



```
PS D:\Java\A1\Exp3\output> & .\'bill.exe'
Consumer No: 012
Consumer Name: Aaryan
Previous Reading: 100
Current Reading: 299
Connection Type(Domestic or Commercial): Professional
Invalid connection type
```

Conclusion:
We have successfully completed this experiment and learnt about C++and how OOP concepts are implemented in it and found out the electric bill of the consumer.

Date: 13/08/24 **Signature of faculty in-charge**

Post Lab Descriptive Questions:

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

A constructor is a special function that is automatically called when an object of a class is created. It is used to initialize the object, often by assigning initial values to the object's member variables or allocating resources. A constructor has the same name as the class. Constructors do not have a return type, not even void.

A destructor is a special member function that is automatically called when an object is destroyed. It is used to perform cleanup tasks, such as releasing resources that were allocated during the object's lifetime. A destructor has the same name as the class, preceded by a tilde (~). Destructors do not have a return type, not even void.

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

Explanation: This output indicates that the size of an object of the class Test is 4 bytes, which is the size of the non-static member variable j. This is because the class Test only contains one non-static member variable j of type int, and the static member i does not contribute to the size of the individual object.

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

Applications of scope resolution operator in C++ are:

- Accessing Global Variables: When a local variable and a global variable share the same name, the scope resolution operator can be used to access the global variable.
- Defining a Class Member Function outside the Class: The scope resolution operator is used to define a member function of a class outside its class definition.
- Accessing Static Members of a Class: The scope resolution operator is used to access static member variables and functions of a class without creating an object.
- Nested Classes: When a class is nested within another class, the scope resolution operator is used to define the member functions of the nested class outside of both classes.
- Enumerations: In C++, scoped enumerations (enum classes) can use the scope resolution operator to access enumeration values.
- Overriding Base Class Members: When a derived class overrides a base class member, the scope resolution operator can be used to access the base class version of the member.
- Accessing Base Class Constructors: The scope resolution operator is used to call base class constructors in an initializer list of a derived class constructor.