**LAB 7:TO ILLUSTRATE THE CONCEPT OF UNARY AND BINARY OPERATOR IN C++.**

**OBJECTIVE:**

1.Understand the concept of operator overloading in C++.

2.Implement unary operator overloading for a user-defined class.

3.Implement binary operator overloading for a user-defined class.

4.Demonstrate how to use overloaded operators to perform custom operations on objects of the user-defined class.

**THEORY:**

In this lab session, we will explore the concept of operator overloading in C++. The primary focus will be on understanding both unary and binary operator overloading and gaining practical experience through illustrative examples.

**Operator Overloading Overview:** Operator overloading is a feature in C++ that allows us to redefine the behavior of built-in operators for user-defined classes. This means we can create custom behaviors for operators such as +, -, \*, /, ++, --, and more when they are used with objects of our own classes.

**Unary Operator Overloading:** Unary operators work on a single operand. Examples include increment (++), decrement (--), and negation (-). The purpose of unary operator overloading is to customize how these operators behave when applied to objects of user-defined classes. For instance, we can redefine the behavior of the increment operator (++) to increment a custom counter class by a specific value.

**Binary Operator Overloading:** Binary operators, on the other hand, operate on two operands. Examples include addition (+), subtraction (-), multiplication (\*), division (/), and equality (==). Binary operator overloading enables us to define the behavior of these operators when used with objects of our custom classes. For example, we can implement custom addition for complex numbers by overloading the + operator.

**PROGRAM 1:WAP TO OVERLOAD THE ++ OPERATOR.**

**//SOURCE CODE:**

#include<iostream>

using namespace std;

class Person{

private:

string name;

int age;

public:

Person(string n , int a){

name=n;

age=a;

}

void display(){

cout<<"name:"<<name<<endl;

cout<<"age:"<<age<<endl;

}

void operator++(){

age++;

}

};

int main(){

Person p("Samikshya" , 19);

p.display();

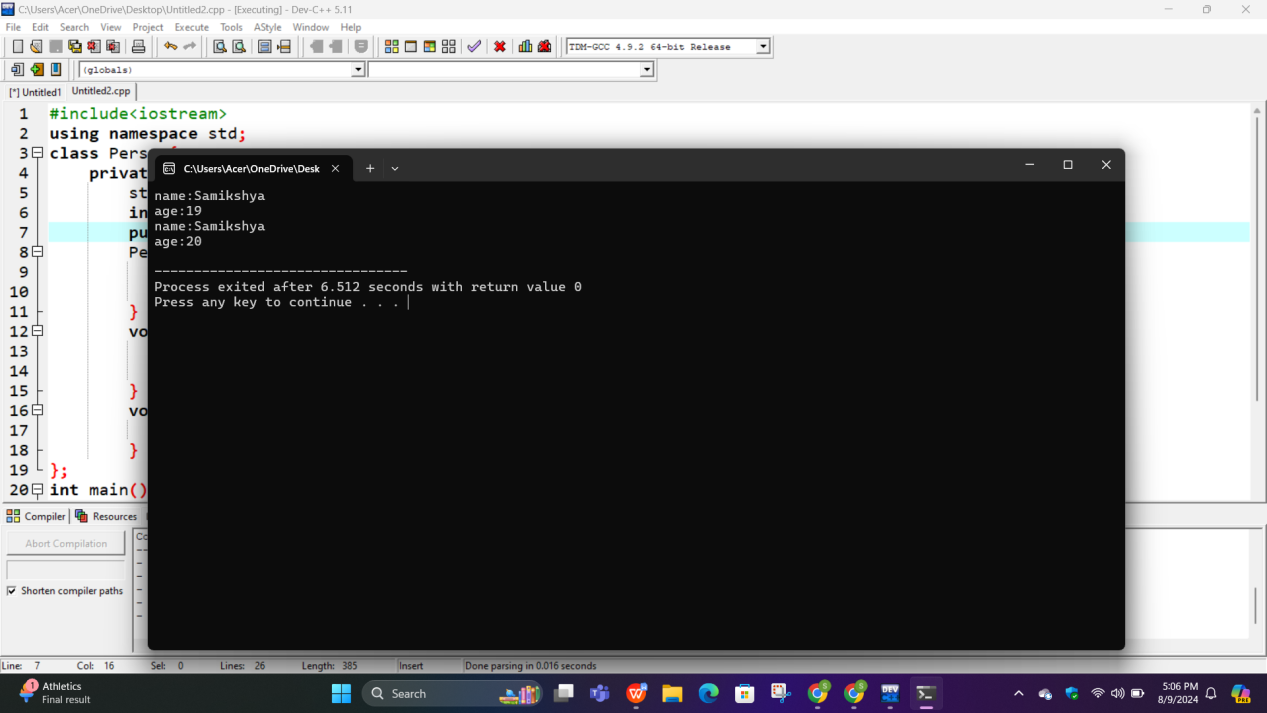
++p;

p.display();

return 0;

}

**OUTPUT:**

****

**PROGRAM 2:WAP TO OVERLOAD THE < OPERATOR TO COMPARE TWO PERSON BASED ON THEIR AGE.**

**//SOURCE CODE:**

#include <iostream>

using namespace std;

class Person {

private:

int age;

public:

Person(int a) : age(a) {}

Int operator<(Person& a){

return age < a.age;

}

};

int main() {

Person p1(95);

Person p2(30);

if (p1 < p2) {

cout << "Person 1 is younger than Person 2." <<endl;

} else {

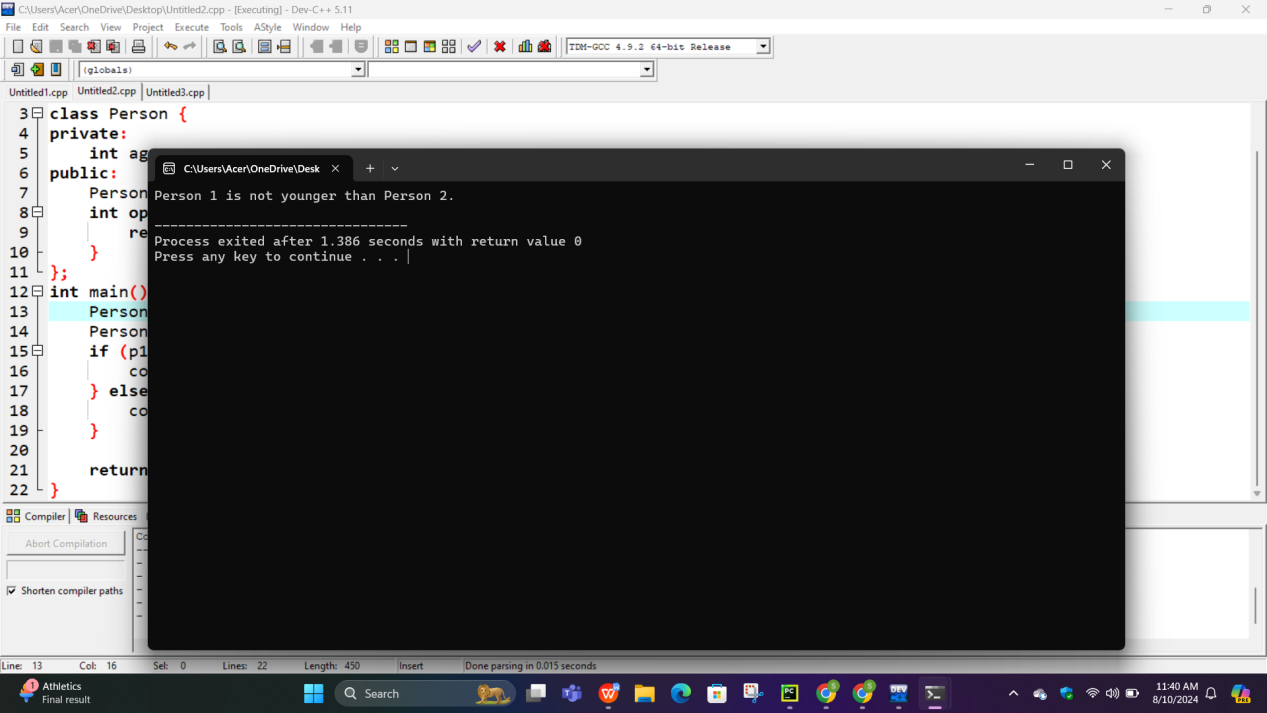
cout << "Person 1 is not younger than Person 2." <<endl;

}

return 0;

}

**OUTPUT:**

****

**CONCLUSION:**In this lab report, we explored the core concepts of unary and binary operator overloading in C++. Unary Operator Overloading was demonstrated with a age increment example, In summary, operator overloading is a valuable feature that enhances code expressiveness and flexibility in C++, allowing developers to create more user-friendly and intuitive custom classes.