DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

OBJECT ORIENTED PROGRAMMING CONCEPTS

CS203

LAB FILE

SUBMITTED TO:	**SUBMITTED BY:**
DR. RK YADAV	**KUNAL BANSAL**
	23/CS/227

INDEX

11.	Write a C++ program to create three objects for a class named pntr_obj with data members such as roll_no and name. Create a member function set_data() for setting data values, and print() member function to print which object has invoked it using ‘this’ pointer.		
12.	Write a C++ program to explain virtual function (polymorphism) by creating a base class c_polygon which has virtual function area(). Two classes c_rectangle and c_triangle derived from c_polygon and they have area() to calculate and return the area of rectangle and triangle respectively.		
13.	WAP to explain class template by creating a template T for a class named pair having two data members of type T which are inputted by a constructor and a member function get_max() to return the greatest of two numbers to main. Note: the value of T depends upon the data type specified during object creation.		
14.	WAP to accept values from users, find sum, product, difference, division of two numbers, (a) using 3-5 inline functions, (b) using reference variables, (c) using macros.		
15.	Write a program in C++ using static variable to get the sum of the salary of 10 employees.		
16.	Write a program using, (a) natural function as friend, (b) member function as friend, to calculate the sum of two complex numbers by using a class complex.		
17.	Write a program to find the area of different shapes with one or two arguments -type int or long or float or double or short or unsigned. Write at least 5 overloading functions for 5-6 shapes. Each function should print input, shape, output.		
18.	(A)(Hierarchical Inheritance) Write a program to calculate the salary of faculty, staff, using inheritance of class employee with constructors. (B) (Multiple Inheritance) Write a program to calculate the salary of Faculty using inheritance of class Employee and class Salary with constructors.		
19.	Write a program to find square and cube of a number using (write functions for getData, showData, showDesult) late/dynamic binding using base pointer.		
20.	Write a program to overload ! (not) operator (unary) all data members of a class.		
21.	Use Template Function to sort an array call function with 5 different combinations including pointer, float, int, etc.		

EXPERIMENT 1

OBJECTIVE : Write a C++ program to print your personal details name, surname (single char), total marks, gender (M/F), result (P/F) by taking input from the user.

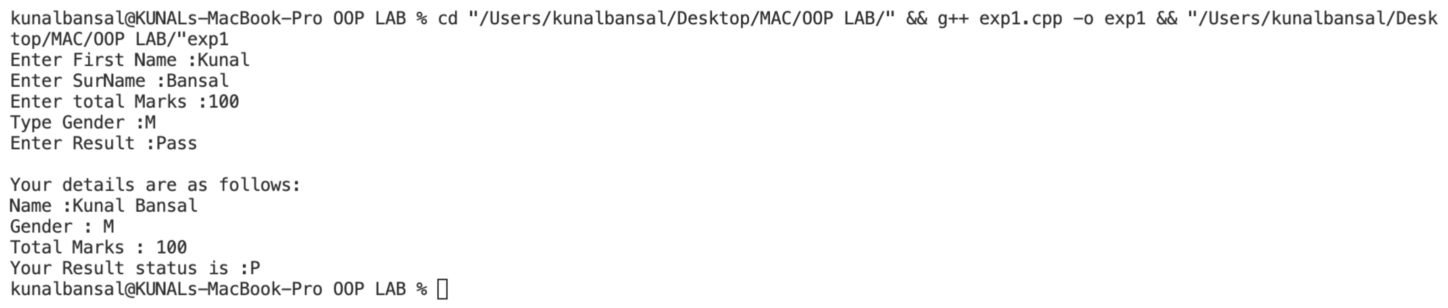
THEORY:

This C++ program is designed to collect and display a student's personal information, including their first and last name, total marks, gender, and result status. It leverages a class to encapsulate this data and provides member functions to handle input and output operations. The program showcases fundamental object-oriented programming principles, such as encapsulation and efficient data management, while offering a user-friendly interface for entering and viewing student details.

CODE :

```
#include <iostream>
using namespace std;
class details{
string firstName;
string surName;
int totalMarks;
string gender;
char result;
public:
details(){
cout<<"Enter First Name :";
cin>>firstName;
cout<<"Enter SurName :";
cin>>surName;
cout<<"Enter total Marks :";
cin>>totalMarks;
cout<<"Type Gender :";
cin>>gender;
cout<<"Enter Result :";
cin>>result;
}
void display(){
cout<<endl<<"Your details are as follows:"<<endl;
cout<<"Name :"<<firstName<<" "<<surName<<endl;
cout<<"Gender : "<<gender<<endl;
cout<<"Total Marks : "<<totalMarks<<endl;
cout<<"Your Result status is :"<<result<<endl;
}
};
int main() {
details d;
d.display();
}
```

OUTPUT:



```
kunalbansal@KUNALs-MacBook-Pro OOP LAB % cd "/Users/kunalbansal/Desktop/MAC/OOP LAB/" && g++ exp1.cpp -o exp1 && "/Users/kunalbansal/Desk
top/MAC/OOP LAB/"exp1
Enter First Name :Kunal
Enter SurName :Bansal
Enter total Marks :100
Type Gender :M
Enter Result :Pass

Your details are as follows:
Name :Kunal Bansal
Gender : M
Total Marks : 100
Your Result status is :P
kunalbansal@KUNALs-MacBook-Pro OOP LAB % 
```

Learning Outcomes:

- Understanding the concept of classes and objects in C++.
- Declaring and initializing member variables.
- Defining and implementing member functions.
- Creating and using objects of a class.
- Taking input from the user and displaying output.

Time Complexity:

- O(1) - Constant time complexity, as the operations performed inside the constructor and the `display()` function are independent of the input size. The time taken remains constant regardless of the number of characters in the names or the size of the marks.

EXPERIMENT 2

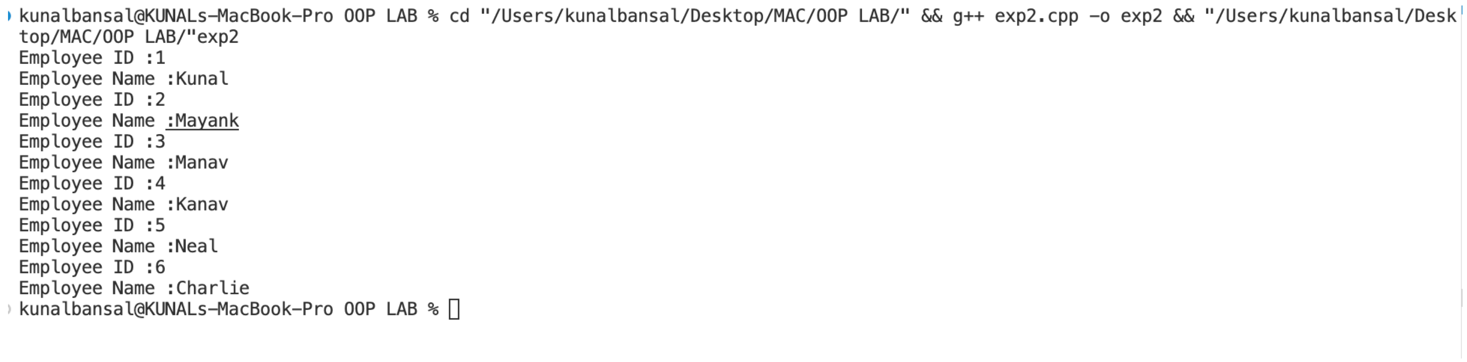
OBJECTIVE : Create a class called Employee that has Empnumber and Empname as data members and member functions getdata() to input data, display() to output data. Write a main function to create an array of Employee objects. Accept and print and accept the details of at least 6 employees.

THEORY:
This program defines an `Employee` class with data members `Empnumber` and `Empname` to encapsulate employee information. It includes member functions, `getdata()` for inputting employee details and `display()` for displaying them. In the main function, an array of `Employee` objects is created, enabling the user to input and display the details of at least six employees. This program demonstrates the use of classes and arrays in C++ for structured data management.

CODE :

```
#include<iostream>
using namespace std;
class Employee{
int EmpNumber;
string EmpName;
public:
void getdata(int n , string name){
EmpNumber=n;
EmpName=name;
}
void display(){
cout<<"Employee ID :"<<EmpNumber<<endl;
cout<<"Employee Name :"<<EmpName<<endl;
}
};
int main(){
Employee DTU[6];
DTU[0].getdata(1,"Kunal");
DTU[1].getdata(2,"Mayank");
DTU[2].getdata(3,"Manav");
DTU[3].getdata(4,"Kanav");
DTU[4].getdata(5,"Neal");
DTU[5].getdata(6,"Charlie");
for (int i = 0; i < 6; i++)
{
DTU[i].display();}}
```

OUTPUT :



```
kunalbansal@KUNALs-MacBook-Pro OOP LAB % cd "/Users/kunalbansal/Desktop/MAC/OOP LAB/" && g++ exp2.cpp -o exp2 && "/Users/kunalbansal/Desk
top/MAC/OOP LAB/"exp2
Employee ID :1
Employee Name :Kunal
Employee ID :2
Employee Name :Mayank
Employee ID :3
Employee Name :Manav
Employee ID :4
Employee Name :Kanav
Employee ID :5
Employee Name :Neal
Employee ID :6
Employee Name :Charlie
kunalbansal@KUNALs-MacBook-Pro OOP LAB %
```

Learning Outcomes:

- Understanding the concept of arrays of objects.
- Declaring and initializing an array of objects.
- Accessing and manipulating individual objects within the array.
- Iterating over the array using a loop to perform actions on each object.

Time Complexity:

- O(n) - Linear time complexity, where n is the number of employees. The `for` loop iterates through each element of the array once, performing constant time operations within the loop. Therefore, the overall time complexity is proportional to the number of employees.

EXPERIMENT 3

OBJECTIVE :Write a C++ program to swap two numbers by both call by value, and call by reference mechanism, using two functions swap_value(), and swap_reference() respectively, by getting the choice from the user, and executing the user's choice by switch case.

THEORY:

- This C++ program demonstrates two methods for swapping two numbers: **call by value** and **call by reference**. It defines two functions: `swap_value()` for swapping numbers using call by value, and `swap_reference()` for swapping them using call by reference. The program prompts the user to select a swapping method and then executes the appropriate function based on the user's choice, using a switch-case statement. This approach effectively illustrates the differences between these two parameter-passing techniques in C++.

CODE :

```
# include <iostream>
using namespace std;
void swap_value(int a, int b){
int temp = a;
a = b;
b = temp;
cout<<"a inside function: "<<a<<endl;
cout<<"b inside function: "<<b<<endl;
}
void swap_reference(int &a, int &b){
int temp = a;
a = b;
b = temp;
cout<<"a inside function: "<<a<<endl;
cout<<"b inside function: "<<b<<endl;
}
int main(){
int x = -1;
int a, b;
cout<<"Enter a: "; cin>>a;
cout<<"Enter b: "; cin>>b;
while(x != 3){
cout<<"Enter choice (1 - value, 2 - reference): "; cin>>x;
switch(x){
case 1:
```

swap_value(a, b);

cout<<"a outside function: "<<a<<endl;

cout<<"b outside function: "<<b<<endl;

break;

case 2:

swap_reference(a, b);

cout<<"a outside function: "<<a<<endl;

cout<<"b outside function: "<<b<<endl;

break;

case 3:

cout<<"Program terminated"<<endl;

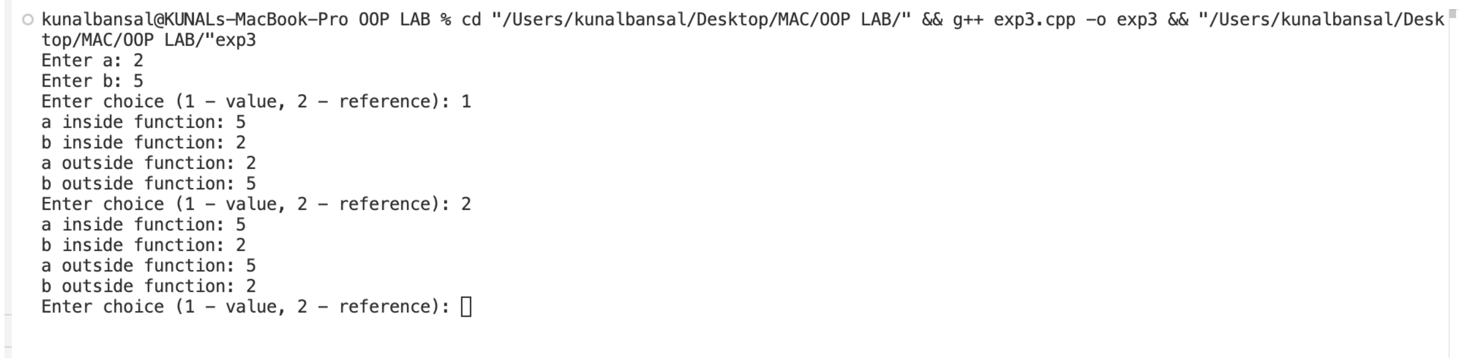
break;

default:

cout<<"Invalid choice, try again"<<endl;

break; }}}

OUTPUT:



```
kunalbansal@KUNALs-MacBook-Pro OOP LAB % cd "/Users/kunalbansal/Desktop/MAC/OOP LAB/" && g++ exp3.cpp -o exp3 && "/Users/kunalbansal/Desk
top/MAC/OOP LAB/"exp3
Enter a: 2
Enter b: 5
Enter choice (1 - value, 2 - reference): 1
a inside function: 5
b inside function: 2
a outside function: 2
b outside function: 5
Enter choice (1 - value, 2 - reference): 2
a inside function: 5
b inside function: 2
a outside function: 5
b outside function: 2
Enter choice (1 - value, 2 - reference): 
```

Learning Outcomes:

- **Call by Value:** Understanding how arguments are passed by value, creating a copy of the original variables.
- **Call by Reference:** Understanding how arguments are passed by reference, allowing the function to directly modify the original variables.
- **Swapping Values:** Implementing the swapping of two integer values using both call by value and call by reference methods.
- **Using a `while` loop and a `switch` statement:** Controlling the flow of the program based on user input.

Time Complexity:

- **O(1) for each function call:** Both `swap_value` and `swap_reference` have constant time complexity, as the number of operations remains constant regardless of the input values.
- **O(n) for the entire program:** The `while` loop iterates until the user chooses to exit, which could be a variable number of times. However, each iteration performs constant time operations, making the overall time complexity linear with respect to the number of iterations.

EXPERIMENT 4

OBJECTIVE : Write a C++ program to create a simple banking system in which the initial balance and the rate of interest are read from the keyboard and these values are initialized using the constructor. The destructor member function is defined in this program to destroy the class object created using constructor member function.

THEORY:

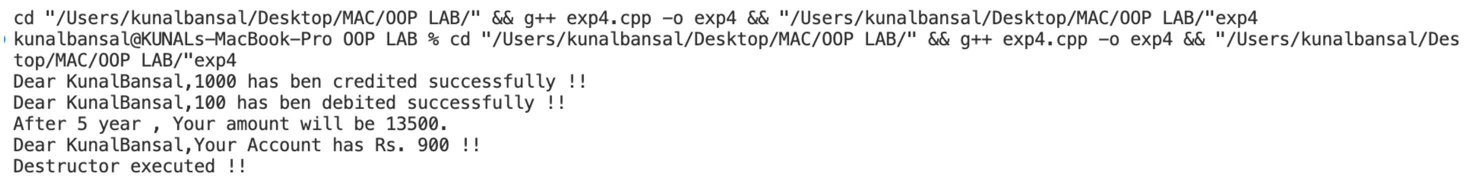
- This C++ program implements a basic banking system in which the user inputs an initial balance and an interest rate, both of which are set up through a constructor. The constructor ensures that the object's data members are initialized correctly at the time of creation. Additionally, the program includes a destructor that is automatically called when the object goes out of scope, handling cleanup and resource management. This program demonstrates the use of constructors and destructors to manage the lifecycle of class objects in C++.

CODE :

```
#include <iostream>
using namespace std;
class bank{
public:
string cname ;
int balance;
float rate;
bank(string name , int current_balance , float roi){
cname = name;
balance = current_balance;
rate=roi;
}
void credit(int amount){
balance+=amount;
cout<<"Dear "<< cname << ","<<amount<<" has ben credited successfully !!"<<endl;
}
void debit(int amount){
balance-=amount;
cout<<"Dear "<< cname << ","<<amount<<" has ben debited successfully !!"<<endl;
}
void interest_cal(int principal , int timeperiod){
float temp=((principal*timeperiod*rate)/100)+principal;
cout<<"After "<<timeperiod<<" year , Your amount will be "<<temp<<"."<<endl;
}
void display(){
```

```
cout<<"Dear "<< cname << ","<<"Your Account has Rs. " << balance << " !!"<<endl;
}
~bank(){
cout<<"Destructor executed !!"<<endl;
cout<<endl;
}
};
int main()
{ bank a1("KunalBansal" , 0 , 7);
a1.credit(1000);
a1.debit(100);
a1.interest_cal(10000 ,5 );
a1.display();
return 0;
}
```

OUTPUT:



```
cd "/Users/kunalbansal/Desktop/MAC/OOP LAB/" && g++ exp4.cpp -o exp4 && "/Users/kunalbansal/Desktop/MAC/OOP LAB/"exp4
kunalbansal@KUNALs-MacBook-Pro OOP LAB % cd "/Users/kunalbansal/Desktop/MAC/OOP LAB/" && g++ exp4.cpp -o exp4 && "/Users/kunalbansal/Des
top/MAC/OOP LAB/"exp4
Dear KunalBansal,1000 has ben credited successfully !!
Dear KunalBansal,100 has ben debited successfully !!
After 5 year , Your amount will be 13500.
Dear KunalBansal,Your Account has Rs. 900 !!
Destructor executed !!
```

Learning Outcomes:

- Object-Oriented Programming (OOP) Concepts: Classes, Objects, Constructors, Member Functions, Destructors
- Input/Output Operations: Using `cin` and `cout`
- Basic Arithmetic Operations

Time Complexity:

- Constant time complexity for each member function: O(1)
- Linear time complexity for the entire program: O(n), where n is the number of function calls.

EXPERIMENT 5

OBJECTIVE : Write a program to accept five different numbers by creating a class called friendfunc1 and friendfunc2 taking 2 and 3 arguments respectively and calculate the average of these numbers by passing object of the class to friend function.

THEORY:

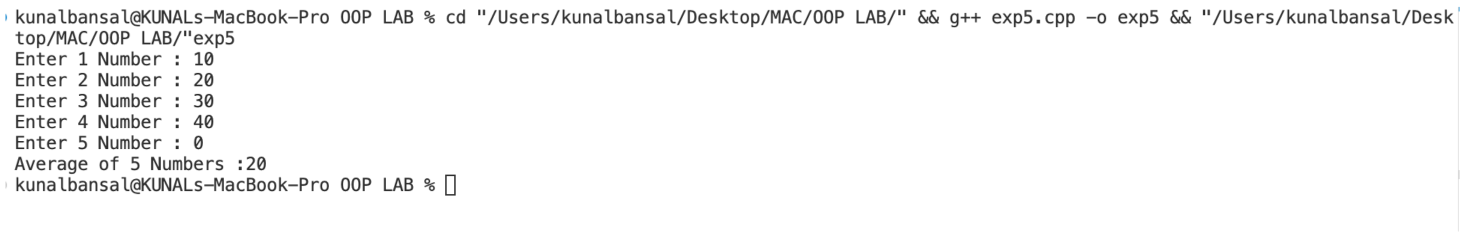
- Friend functions in C++ allow external functions to access the private and protected members of a class. In this C++ program, a single friend function is defined that can access the private members of both friendfunc1 and friendfunc2 classes. This allows the function to calculate the average of five different numbers stored within these classes, demonstrating how friend functions enable seamless interaction between multiple classes while maintaining encapsulation. By granting access to class internals, the friend function can perform necessary calculations without needing to be a member of either class, thus enhancing modularity and code clarity.

CODE:

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

class friendfunc1{
private:
int a1, a2;
public:
friendfunc1(int num1 , int num2){
a1=num1;
a2=num2;
}
friend class friendfunc2;
};
class friendfunc2{
private:
int a3,a4,a5;
public:
friendfunc2(int num3 , int num4 , int num5){
a3=num3;
a4=num4;
a5=num5;}


void display(friendfunc1 &f){
```

```
cout<<"Average of 5 Numbers :"<<(a3+a4+a5+f.a1+f.a2)/5<<endl;}};


int main() {
vector<int> a(5);
for (int i = 0; i < 5; i++)
{ cout<<"Enter "<< i+1 <<" Number : ";
cin>>a[i];
}
friendfunc1 f1(a[0],a[1]);
friendfunc2 f2(a[2],a[3],a[4]);
f2.display(f1);}
```

OUTPUT:



```
kunalbansal@KUNALs-MacBook-Pro OOP LAB % cd "/Users/kunalbansal/Desktop/MAC/OOP LAB/" && g++ exp5.cpp -o exp5 && "/Users/kunalbansal/Desk
top/MAC/OOP LAB/"exp5
Enter 1 Number : 10
Enter 2 Number : 20
Enter 3 Number : 30
Enter 4 Number : 40
Enter 5 Number : 0
Average of 5 Numbers :20
kunalbansal@KUNALs-MacBook-Pro OOP LAB % 
```

Learning Outcomes:

- **Friend Functions:** Understanding the concept of friend functions and how they can access private members of other classes.
- **Class and Objects:** Creating classes and objects to encapsulate data and behavior.
- **Input/Output Operations:** Using `cin` and `cout` for user input and output.
- **Vector Container:** Using the `vector` container to store and manipulate a dynamic array of integers.

Time Complexity:

- **O(n):** The time complexity of the `for` loop to input the numbers is linear with respect to the number of elements (n).
- **O(1):** The operations within the `display` function are constant time, as they involve simple arithmetic operations and accessing class members.

Overall Time Complexity: O(n)

EXPERIMENT 6

OBJECTIVE : Write a program to accept the student detail such as name and 3 different marks by get_data() method and display the name and average of marks using display() method. Define a friend class for calculating the average of marks using the method mark_avg().

THEORY:

- Friend classes in C++ allow one class to access the private and protected members of another class, enhancing collaboration while maintaining encapsulation. This program illustrates the use of friend classes in C++, where one class is granted access to the private members of another class. It defines a student class that collects details such as the student's name and three marks through the get_data() method. A friend class is implemented to calculate the average of the marks using the mark_avg() method, enabling seamless access to the student's private data.

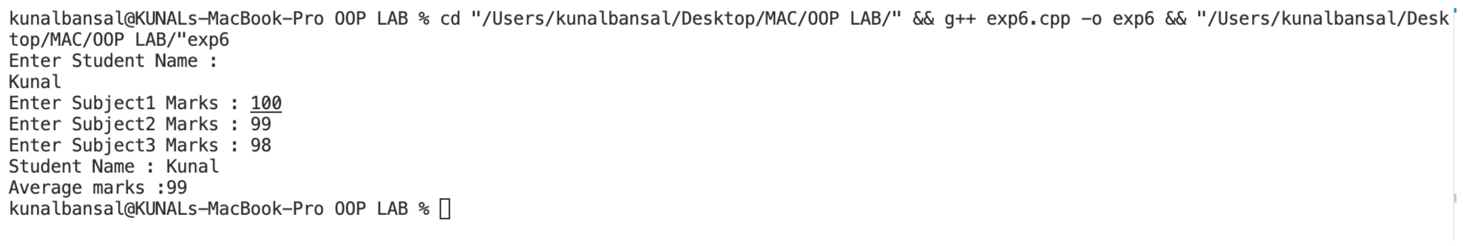
CODE:

```
#include <iostream>
using namespace std;
class student_details{
private:
string name;
int a1, a2 , a3;
public:
student_details( string s, int marks1,int marks2 , int marks3){
name = s;
a1=marks1;
a2=marks2;
a3=marks3;
}
friend class marks_avg;
};
class marks_avg{
public:
void display(student_details s1){
cout<<"Student Name : "<<s1.name<<endl;
cout<<"Average marks :"<<(s1.a1+s1.a2+s1.a3)/3<<endl;
}
};
int main() {
cout<<"Enter Student Name : "<<endl;
string name ;
```

```
getline(cin,name);
vector<int> a(3);
for (int i = 0; i < 3; i++)
{ cout<<"Enter Subject"<< i+1 <<" Marks : ";
cin>>a[i];
}
student_details s1(name , a[0] , a[1] , a[2]);
marks_avg evaluator;
evaluator.display(s1);
}
#include <iostream>
using namespace std;
class student_details{
private:
string name;
int a1, a2 , a3;
public:
student_details( string s, int marks1,int marks2 , int marks3){
name = s;
a1=marks1;
a2=marks2;
a3=marks3;
}
friend class marks_avg;
};
class marks_avg{
public:
void display(student_details s1){
cout<<"Student Name : "<<s1.name<<endl;
cout<<"Average marks :"<<(s1.a1+s1.a2+s1.a3)/3<<endl;
}
};
int main() {
cout<<"Enter Student Name : "<<endl;
string name ;
getline(cin,name);
vector<int> a(3);
```

```
for (int i = 0; i < 3; i++)
{ cout<<"Enter Subject"<< i+1 <<" Marks : ";
cin>>a[i];
}
student_details s1(name , a[0] , a[1] , a[2]);
marks_avg evaluator;
evaluator.display(s1);
}
```

OUTPUT:



```
kunalbansal@KUNALs-MacBook-Pro OOP LAB % cd "/Users/kunalbansal/Desktop/MAC/OOP LAB/" && g++ exp6.cpp -o exp6 && "/Users/kunalbansal/Desk
top/MAC/OOP LAB/"exp6
Enter Student Name :
Kunal
Enter Subject1 Marks : 100
Enter Subject2 Marks : 99
Enter Subject3 Marks : 98
Student Name : Kunal
Average marks :99
kunalbansal@KUNALs-MacBook-Pro OOP LAB %
```

Learning Outcomes:

- Friend Classes: Understanding the concept of friend classes and their usage.
- Class and Objects: Creating classes and objects.
- Input/Output: Using `cin` and `cout`.
- Vector Container: Using vectors for dynamic arrays.

Time Complexity:

- O(1): Constant time complexity.

EXPERIMENT 7

OBJECTIVE : Write a C++ program to perform different arithmetic operations such as addition, subtraction, division, modulus, and multiplication using inline functions.

THEORY:
Inline functions in C++ are a type of function defined with the keyword inline, which suggests to the compiler that the function code should be inserted directly into the calling code instead of being called through the usual function call mechanism. This can enhance performance by reducing function call overhead, especially for small, frequently called functions. In the context of this program, inline functions are used to perform various arithmetic operations—addition, subtraction, multiplication, division, and modulus—allowing for efficient execution of these operations while keeping the code organized and readable.

CODE:

```
#include <iostream>
using namespace std;
inline void add(int a, int b) {
cout<<a + b<< endl;
}
inline void subtract(int a, int b) {
cout<<a - b<<endl;
}
inline void multiply(int a, int b) {
cout<<a * b<<endl;
}
inline void divide(double a, double b) {
if(b != 0) {
cout<<(double)a / b<<endl;
} else {
cout<<"Error: Division by zero!"<<endl;
}
}
inline void mod(int a, int b) {
cout<<a % b<<endl;
}
int main() {
int num1, num2;
cout<<"Enter two numbers: ";
cin>>num1>>num2;
cout<<"Addition: "; add(num1, num2);
cout<<"Subtraction: "; subtract(num1, num2);
```

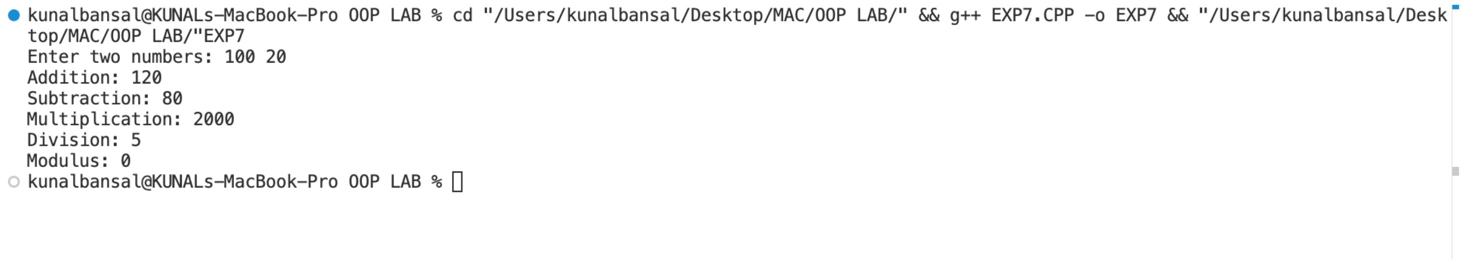
cout<<"Multiplication: "; multiply(num1, num2);

cout<<"Division: "; divide(num1, num2);

cout<<"Modulus: "; mod(num1, num2);

}

OUTPUT:



```
● kunalbansal@KUNALs-MacBook-Pro OOP LAB % cd "/Users/kunalbansal/Desktop/MAC/OOP LAB/" && g++ EXP7.CPP -o EXP7 && "/Users/kunalbansal/Desk
  top/MAC/OOP LAB/"EXP7
  Enter two numbers: 100 20
  Addition: 120
  Subtraction: 80
  Multiplication: 2000
  Division: 5
  Modulus: 0
○ kunalbansal@KUNALs-MacBook-Pro OOP LAB % 
```

Learning Outcomes:

- **Inline Functions:** Understanding the concept of inline functions and their use to optimize function calls.
- **Arithmetic Operations:** Performing basic arithmetic operations (addition, subtraction, multiplication, division, and modulus).
- **Error Handling:** Implementing basic error handling for division by zero.
- **Input/Output Operations:** Using `cin` and `cout` for user input and output.

Time Complexity:

- **O(1):** All the inline functions perform constant time operations.

EXPERIMENT 8

OBJECTIVE : Write a C++ program to return absolute value of variable types integer and float using function overloading.

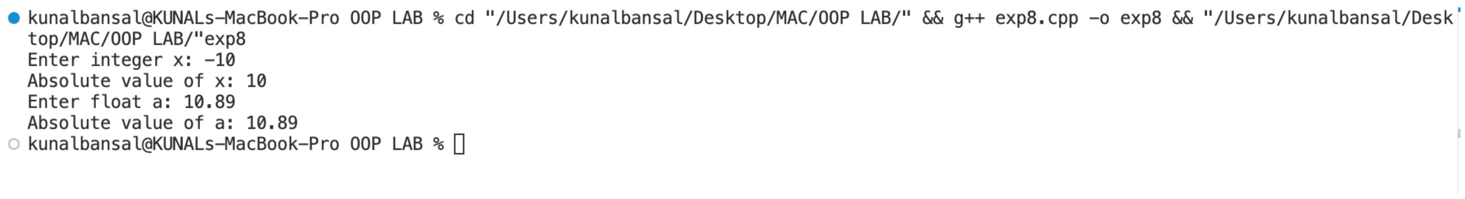
THEORY:

- Function overloading in C++ allows multiple functions to have the same name but differ in the number or type of their parameters. This feature enables programmers to define a set of operations that can handle different data types seamlessly, improving code readability and organization. This C++ code demonstrates function overloading by implementing two versions of the absolute() function to return the absolute value for both integer and float variable types. By allowing multiple functions to share the same name but differ in their parameter types, the code enhances readability and provides a consistent interface for users.

CODE:

```
# include <iostream>
using namespace std;
int absolute(int x){
if(x >= 0) return x;
else return -x;
}
float absolute(float x){
if(x >= 0.0) return x;
else return -x;
}
int main(){
int x;
cout<<"Enter integer x: "; cin>>x;
cout<<"Absolute value of x: "<<absolute(x)<<endl;
float a;
cout<<"Enter float a: "; cin>>a;
cout<<"Absolute value of a: "<<absolute(a)<<endl;
}
```

OUTPUT:



```
kunalbansal@KUNALs-MacBook-Pro OOP LAB % cd "/Users/kunalbansal/Desktop/MAC/OOP LAB/" && g++ exp8.cpp -o exp8 && "/Users/kunalbansal/Desk
top/MAC/OOP LAB/"exp8
Enter integer x: -10
Absolute value of x: 10
Enter float a: 10.89
Absolute value of a: 10.89
kunalbansal@KUNALs-MacBook-Pro OOP LAB % 
```

Learning Outcomes:

- **Function Overloading:** Understanding how to define multiple functions with the same name but different parameter types.
- **Conditional Statements:** Using `if` and `else` to determine the sign of a number.
- **Absolute Value:** Calculating the absolute value of a number.
- **Input/Output Operations:** Using `cin` and `cout` for user input and output.

Time Complexity:

- O(1): Constant time complexity, as the operations within the `absolute` functions are independent of the input size.

EXPERIMENT 9

OBJECTIVE :WAP to perform string operations using operator overloading in C++: (i) = String Copy, (ii) == Equality, (iii) + Concatenation.

THEORY:

- Operator overloading in C++ allows developers to redefine the behavior of operators for user-defined types, enabling intuitive interactions with objects. In this program, operator overloading is utilized to implement string operations, including the assignment operator (=) for string copying, the equality operator (==) for comparing two strings, and the addition operator (+) for string concatenation. This approach enhances code readability and usability, allowing string objects to be manipulated in a manner similar to built-in data types.

CODE:

```
#include<iostream>

using namespace std;

class String{

string s;

public:

String(string temp){

s=temp;

}

bool operator==(String s1){

return s1.s==s;

}

String operator+(String s1){

String temp=s+" "+s1.s;

return temp;

}

String operator=(String s1){

return s=s1.s;

}

void display(){

cout<<s<<endl;
```

}

};

int main(){

// == operator

String s1("Kunal");

String s2("Bansal");

bool ans = s1==s2;

cout<<ans<<endl;

// + operator

String s3=s1+s2;

s3.display();

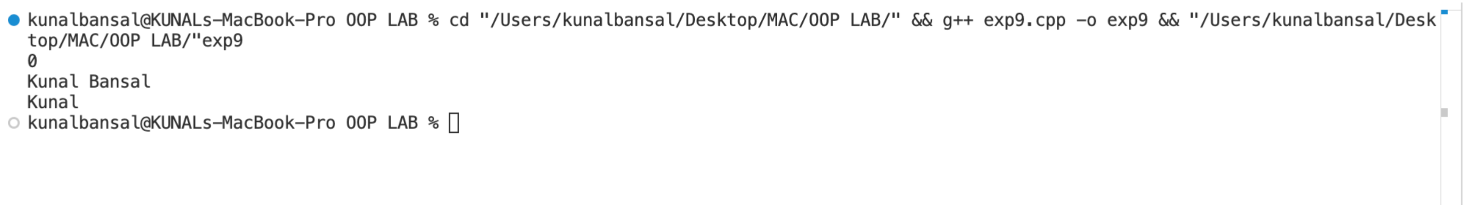
// = operater

String s4=s1;

s4.display();

}

OUTPUT :



```
● kunalbansal@KUNALs-MacBook-Pro OOP LAB % cd "/Users/kunalbansal/Desktop/MAC/OOP LAB/" && g++ exp9.cpp -o exp9 && "/Users/kunalbansal/Desk
  top/MAC/OOP LAB/"exp9
  0
  Kunal Bansal
  Kunal
○ kunalbansal@KUNALs-MacBook-Pro OOP LAB % 
```

Learning Outcomes:

- **Operator Overloading:** Understanding how to overload operators to customize their behavior for user-defined classes.
- **String Manipulation:** Concatenating and comparing strings.
- **Class and Objects:** Creating classes and objects to represent strings.

Time Complexity:

- The time complexity of the overloaded operators depends on the length of the strings involved.
  - `operator==`: O(n), where n is the length of the shorter string.
  - `operator+`: O(n), where n is the total length of the concatenated string.
  - `operator=`: O(n), where n is the length of the string being copied.

EXPERIMENT 10

OBJECTIVE : Consider a class network as given below. The class master derives information from both account and admin classes which in turn derive information from the class person. Define all the four classes and write a program to create, update and display the information contained in master objects. Also demonstrate the use of different access specifiers by means of member variables and member functions.

THEORY:

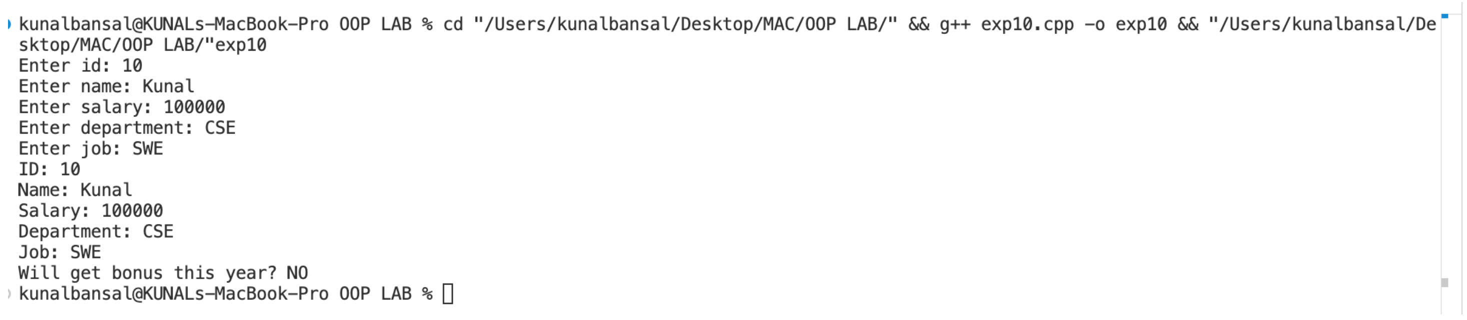
- Inheritance: A fundamental object-oriented programming concept that allows a class (derived class) to inherit attributes and behaviours (methods) from another class (base class), promoting code reuse and establishing a hierarchical relationship. Access Specifiers: Public: Members declared as public are accessible from any part of the program, including outside the class. Protected: Members declared as protected are accessible within the class itself and by derived classes, but not from outside the class hierarchy. Private: Members declared as private are accessible only within the class itself, restricting access from derived classes and outside code.

CODE:

```
# include <iostream>
using namespace std;
class person{
protected:
int id; string name;
public:
void get_per(){
cout<<"Enter id: "; cin>>id;
cout<<"Enter name: "; cin>>name;
}
};
class account : public virtual person{
protected:
int salary;
public:
void get_ac(){
cout<<"Enter salary: "; cin>>salary;
}
};
class admin : public virtual person{
protected:
string dept; string job;
public:
void get_adm(){
```

```
cout<<"Enter department: "; cin>>dept;
cout<<"Enter job: "; cin>>job;
}
};
class master: public virtual account, public virtual admin{
private:
bool bonus;
public:
master(){
id = 0; name = "NA"; salary = 0; dept = "NA"; job = "NA";
}
void check_bonus(){
if(salary <= 5000000 && dept == "Engineering") bonus = true;
else bonus = false;
}
void display(){
cout<<"ID: "<<id<<endl;
cout<<"Name: "<<name<<endl;
cout<<"Salary: "<<salary<<endl;
cout<<"Department: "<<dept<<endl;
cout<<"Job: "<<job<<endl;
cout<<"Will get bonus this year? "<<(bonus ? "YES" : "NO")<<endl;
}
};
int main(){
master m1;
m1.get_per();
m1.get_ac();
m1.get_adm();
m1.check_bonus();
m1.display();
}
```

OUTPUT:



```
kunalbansal@KUNALs-MacBook-Pro OOP LAB % cd "/Users/kunalbansal/Desktop/MAC/OOP LAB/" && g++ exp10.cpp -o exp10 && "/Users/kunalbansal/De
sktop/MAC/OOP LAB/"exp10
Enter id: 10
Enter name: Kunal
Enter salary: 100000
Enter department: CSE
Enter job: SWE
ID: 10
Name: Kunal
Salary: 100000
Department: CSE
Job: SWE
Will get bonus this year? NO
kunalbansal@KUNALs-MacBook-Pro OOP LAB %
```

Learning Outcomes:

- **Inheritance: Understanding single, multiple, and hierarchical inheritance.**
- **Virtual Inheritance: Resolving the ambiguity that arises in multiple inheritance scenarios.**
- **Access Specifiers: Using `public`, `protected`, and `private` access specifiers.**
- **Function Overriding: Redefining functions in derived classes to provide specific implementations.**
- **Conditional Statements: Using `if` and `else` to make decisions.**

Time Complexity:

- **The time complexity of the `main` function is O(1), as it involves a fixed number of operations, regardless of the input size.**

EXPERIMENT 11

OBJECTIVE :Write a C++ program to create three objects for a class named pntr_obj with data members such as roll_no and name. Create a member function set_data() for setting data values, and print() member function to print which object has invoked it using 'this' pointer.

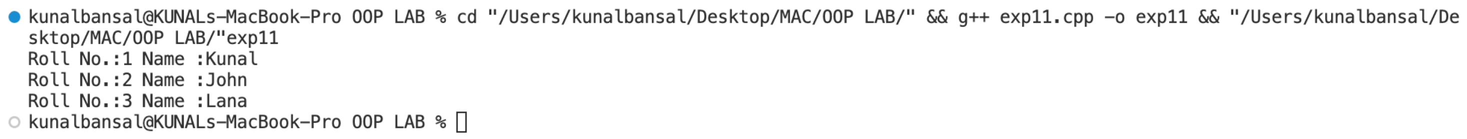
THEORY:

- This C++ program defines a class named pntr_obj with data members roll_no and name. It includes a member function set_data() to initialize these values and a print() member function that utilizes the this pointer to identify the specific object invoking it. The use of the this pointer allows the program to access the calling object's members, demonstrating the concept of object identity and encapsulation in class design.

CODE:

```
#include<iostream>
using namespace std;
class pntr_obj{
int rollNo;
string Name;
public :
void setData(int rollNo,string Name){
this->rollNo = rollNo;
this->Name= Name;
}
void display(){
cout<<"Roll No.:"<<this->rollNo<<" "<<"Name :"<<this->Name<<endl; }
};
int main(){
pntr_obj obj1,obj2,obj3;
obj1.setData(1,"Kunal");
obj2.setData(2,"John");
obj3.setData(3,"Lana");
obj1.display();
obj2.display();
obj3.display();
}
```

OUTPUT:



```
● kunalbansal@KUNALs-MacBook-Pro OOP LAB % cd "/Users/kunalbansal/Desktop/MAC/OOP LAB/" && g++ exp11.cpp -o exp11 && "/Users/kunalbansal/De
  sktop/MAC/OOP LAB/"exp11
  Roll No.:1 Name :Kunal
  Roll No.:2 Name :John
  Roll No.:3 Name :Lana
○ kunalbansal@KUNALs-MacBook-Pro OOP LAB % ▯
```

Learning Outcomes:

- **Classes and Objects:** Understanding the concept of classes and objects in C++.
- **Member Functions:** Defining and using member functions to manipulate object data.
- **Object Initialization and Data Assignment:** Initializing objects and assigning values to their member variables.
- **`this` Pointer:** Using the `this` pointer to refer to the current object.

Time Complexity:

- The time complexity of the `main` function is O(1), as it involves a fixed number of operations, regardless of the input size.

EXPERIMENT 12

OBJECTIVE : Write a C++ program to explain virtual function (polymorphism) by creating a base class c_polygon which has virtual function area(). Two classes c_rectangle and c_triangle derived from c_polygon and they have area() to calculate and return the area of rectangle and triangle respectively

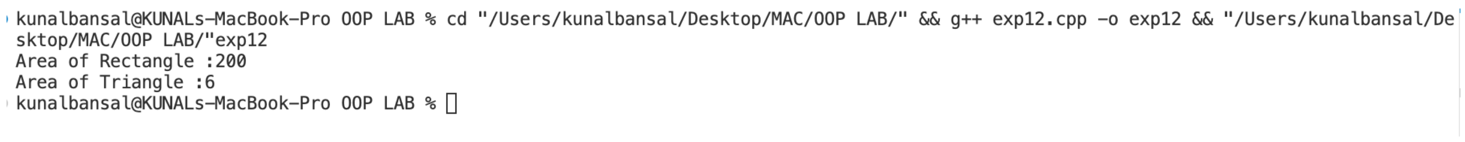
THEORY:

- Virtual functions are member functions in a base class that can be overridden in derived classes, enabling polymorphism in C++. Polymorphism allows for invoking derived class methods through base class pointers or references, ensuring the correct method is called at runtime. This mechanism facilitates dynamic binding, promoting code flexibility and reusability. The base class c_polygon defines a virtual function area(), which is overridden in the derived classes c_rectangle and c_triangle. Each derived class provides its specific implementation of the area() function to calculate the area of a rectangle and a triangle, respectively.

CODE:

```
#include<iostream>
using namespace std;
class c_polygon{
protected:
float area;
};
class c_rectangle : public c_polygon{
int length;
int breadth;
public:
c_rectangle(int length , int breadth){
this->length = length;
this->breadth = breadth;
area = length*breadth;
}
void display(){
cout<<"Area of Rectangle :"<<area<<endl;
}
};
class c_triangle : public c_polygon{
int a;
int b;
int c;
public:
c_triangle(int a , int b , int c){
this->a = a;
this->b = b;
this->c = c;
float s = (a+b+c)/2;
area = sqrtl(s*(s - a)*(s - b)*(s - c));
}
void display(){
cout<<"Area of Triangle :"<<area<<endl;
}
};
int main(){
c_rectangle a(10,20);
a.display();
c_triangle b(3,4,5);
b.display();
}
```

OUTPUT:



```
kunalbansal@KUNALs-MacBook-Pro OOP LAB % cd "/Users/kunalbansal/Desktop/MAC/OOP LAB/" && g++ exp12.cpp -o exp12 && "/Users/kunalbansal/De
sktop/MAC/OOP LAB/"exp12
Area of Rectangle :200
Area of Triangle :6
kunalbansal@KUNALs-MacBook-Pro OOP LAB % 
```

Learning Outcomes:

- **Inheritance:** Understanding the concept of inheritance, specifically single inheritance.
- **Protected Access Specifier:** Using the `protected` access specifier to share members between base and derived classes.
- **Area Calculation:** Implementing formulas to calculate the area of rectangles and triangles.
- **Object-Oriented Programming:** Applying OOP principles to create classes and objects.

Time Complexity:

- The time complexity of the `main` function is O(1), as it involves a fixed number of operations, regardless of the input size.

EXPERIMENT 13

OBJECTIVE : WAP to explain class template by creating a template T for a class named pair having two data members of type T which are inputted by a constructor and a member function get_max() to return the greatest of two numbers to main. Note: the value of T depends upon the data type specified during object creation.

THEORY:

- Templates in C++ allow the creation of generic classes and functions that can operate with any data type. This C++ program demonstrates the use of class templates by defining a template class pair that can handle two data members of a generic type T. The class constructor initializes these data members, and the member function get_max() returns the maximum of the two values. This approach allows for flexibility, as the data type of T is determined at the time of object creation, enabling the same code to work with different data types such as int, float, or double.

CODE:

```
# include <iostream>
using namespace std;
template <class T>
class Pair{
T first;
T second;
public:
Pair(){
cout<<"Enter first: "; cin>>first;
cout<<"Enter second: "; cin>>second;
}
T get_first(){
return first;
}
T get_second(){
return second;
}
T get_max(){
return max(first, second);
}
};
int main(){
cout<<"Enter integer pair: ";
Pair<int> p1;
cout<<"Integer pair entered: ";
```

cout<<"First value: "<<p1.get_first()<<endl;

cout<<"Second value: "<<p1.get_second()<<endl;

cout<<"Maximum value: "<<p1.get_max()<<endl;

cout<<"Enter double pair: ";

Pair<double> p2;

cout<<"Double pair entered: ";

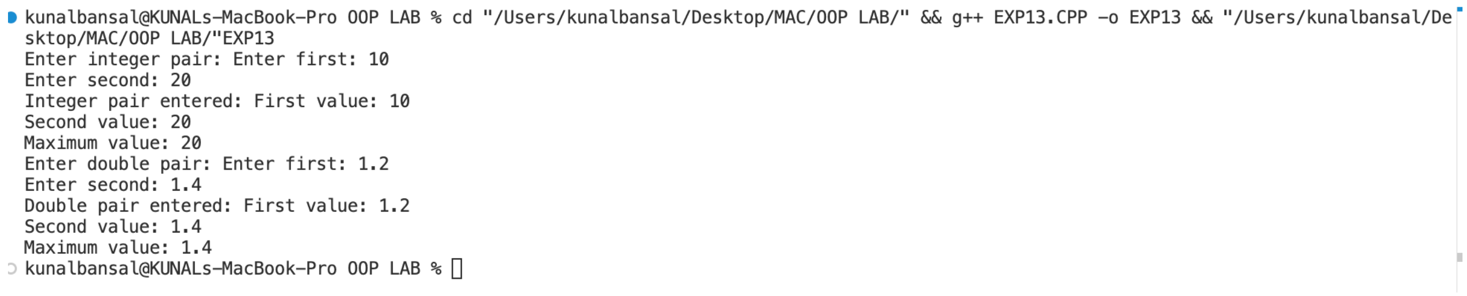
cout<<"First value: "<<p2.get_first()<<endl;

cout<<"Second value: "<<p2.get_second()<<endl;

cout<<"Maximum value: "<<p2.get_max()<<endl;

}

OUTPUT:



```
kunalbansal@KUNALs-MacBook-Pro OOP LAB % cd "/Users/kunalbansal/Desktop/MAC/OOP LAB/" && g++ EXP13.CPP -o EXP13 && "/Users/kunalbansal/De
sktop/MAC/OOP LAB/"EXP13
Enter integer pair: Enter first: 10
Enter second: 20
Integer pair entered: First value: 10
Second value: 20
Maximum value: 20
Enter double pair: Enter first: 1.2
Enter second: 1.4
Double pair entered: First value: 1.2
Second value: 1.4
Maximum value: 1.4
kunalbansal@KUNALs-MacBook-Pro OOP LAB %
```

Learning Outcomes:

- **Templates: Understanding the concept of templates and their use in creating generic classes.**
- **Class and Objects: Creating classes and objects to encapsulate data and behavior.**
- **Input/Output Operations: Using `cin` and `cout` for user input and output.**
- **Generic Programming: Writing code that can work with different data types.**

Time Complexity:

- **The time complexity of the `main` function is O(1), as it involves a fixed number of operations, regardless of the input size.**

EXPERIMENT 14

OBJECTIVE :WAP to accept values from users, find sum, product, difference, division of two numbers, (a) using 3-5 inline functions, (b) using reference variables, (c) using macros.

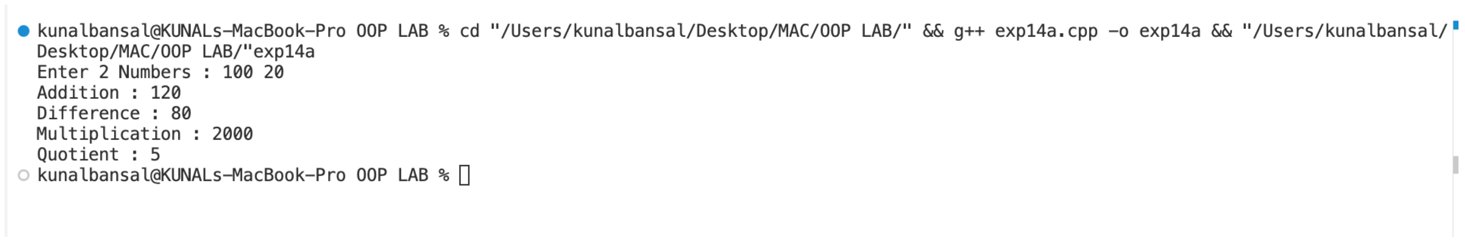
THEORY:

- This C++ program demonstrates three different methods for performing arithmetic operations (sum, product, difference, division) on two numbers input by the user. (a) Inline functions are used to define simple operations that can be executed quickly without the overhead of a regular function call. (b) Reference variables allow direct manipulation of arguments passed to functions, enabling changes to the original variables without copying them. (c) Macros are preprocessor directives that define reusable code snippets, which can improve code readability but may introduce complexity in debugging due to lack of type safety

CODE:

```
// Using Inline Functions
#include<iostream>
using namespace std;
inline int sum(int a , int b){
return a+b;
}
inline int diff(int a , int b){
return a-b;
}
inline int mul(int a , int b){
return a*b;
}
inline int divide(int a , int b){
return a/b;
}
int main(){
int a, b;
cout<<"Enter 2 Numbers : ";
cin>>a>>b;
cout<<"Addition : "<<sum(a,b)<<endl;
cout<<"Difference : "<<diff(a,b)<<endl;
cout<<"Multiplication : "<<mul(a,b)<<endl;
cout<<"Quotient : "<<divide(a,b)<<endl;
}
```

Output :



```
● kunalbansal@KUNALs-MacBook-Pro OOP LAB % cd "/Users/kunalbansal/Desktop/MAC/OOP LAB/" && g++ exp14a.cpp -o exp14a && "/Users/kunalbansal/
  Desktop/MAC/OOP LAB/"exp14a
  Enter 2 Numbers : 100 20
  Addition : 120
  Difference : 80
  Multiplication : 2000
  Quotient : 5
○ kunalbansal@KUNALs-MacBook-Pro OOP LAB % 
```

Learning Outcomes:

- **Inline Functions:** Understanding the concept of inline functions and their use to optimize function calls.
- **Arithmetic Operations:** Performing basic arithmetic operations (addition, subtraction, multiplication, and division).
- **Input/Output Operations:** Using `cin` and `cout` for user input and output.

Time Complexity:

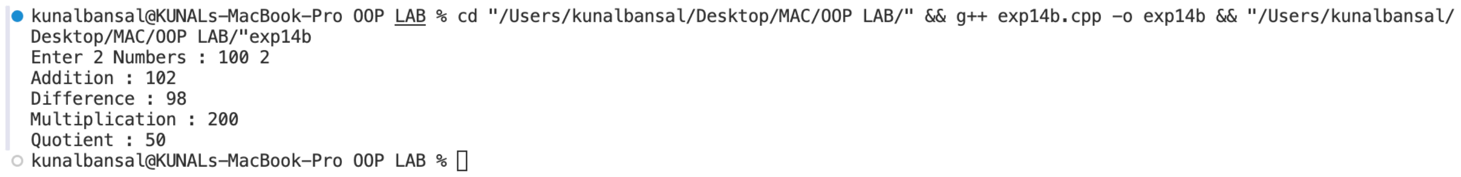
- **O(1):** All the inline functions perform constant time operations.

//Using Reference Variables

Code:

```
#include<iostream>
using namespace std;
int sum(int& a , int& b){
return a+b;
}
int diff(int& a , int& b){
return a-b;
}
int mul(int& a , int& b){
return a*b;
}
int divide(int& a , int& b){
return a/b;
}
int main(){
int a, b;
cout<<"Enter 2 Numbers : ";
cin>>a>>b;
cout<<"Addition : "<<sum(a,b)<<endl;
cout<<"Difference : "<<diff(a,b)<<endl;
cout<<"Multiplication : "<<mul(a,b)<<endl;
cout<<"Quotient : "<<divide(a,b)<<endl;
}
```

Output :



```
kunalbansal@KUNALs-MacBook-Pro OOP LAB % cd "/Users/kunalbansal/Desktop/MAC/OOP LAB/" && g++ exp14b.cpp -o exp14b && "/Users/kunalbansal/
Desktop/MAC/OOP LAB/"exp14b
Enter 2 Numbers : 100 2
Addition : 102
Difference : 98
Multiplication : 200
Quotient : 50
kunalbansal@KUNALs-MacBook-Pro OOP LAB %
```

Learning Outcomes:

- **Pass by Reference:** Understanding the concept of pass by reference and its use to modify variables within a function.
- **Arithmetic Operations:** Performing basic arithmetic operations (addition, subtraction, multiplication, and division).
- **Input/Output Operations:** Using `cin` and `cout` for user input and output.

Time Complexity:

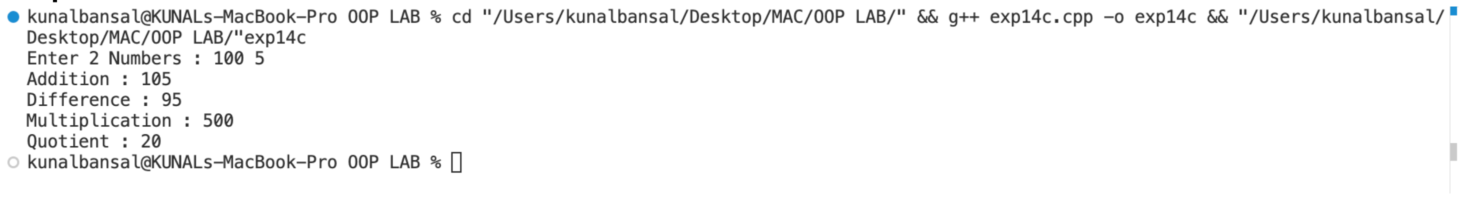
- **O(1):** All the functions perform constant time operations.

```
//Using Macros

#include<iostream>
using namespace std;
#define sum(a,b) (a+b)
#define diff(a,b) (a-b)
#define mul(a,b) (a*b)
#define divide(a,b) (a/b)
int main(){
int a, b;
cout<<"Enter 2 Numbers : ";
cin>>a>>b;
cout<<"Addition : "<<sum(a,b)<<endl;
cout<<"Difference : "<<diff(a,b)<<endl;
cout<<"Multiplication : "<<mul(a,b)<<endl;
```

cout<<"Quotient : "<<divide(a,b)<<endl;

}

Output:



```
kunalbansal@KUNALs-MacBook-Pro OOP LAB % cd "/Users/kunalbansal/Desktop/MAC/OOP LAB/" && g++ exp14c.cpp -o exp14c && "/Users/kunalbansal/Desktop/MAC/OOP LAB/"exp14c
Enter 2 Numbers : 100 5
Addition : 105
Difference : 95
Multiplication : 500
Quotient : 20
kunalbansal@KUNALs-MacBook-Pro OOP LAB % 
```

Learning Outcomes:

- **Macros:** Understanding the concept of macros and their use in defining reusable code snippets.
- **Preprocessor Directives:** Using the preprocessor to replace macros with their corresponding expressions before compilation.
- **Arithmetic Operations:** Performing basic arithmetic operations (addition, subtraction, multiplication, and division).
- **Input/Output Operations:** Using `cin` and `cout` for user input and output.

Time Complexity:

- **O(1):** The macro expansions are simple arithmetic operations, resulting in constant time complexity.

EXPERIMENT 15

OBJECTIVE :Write a program in C++ using static variables to get the sum of the salary of 10 employees.

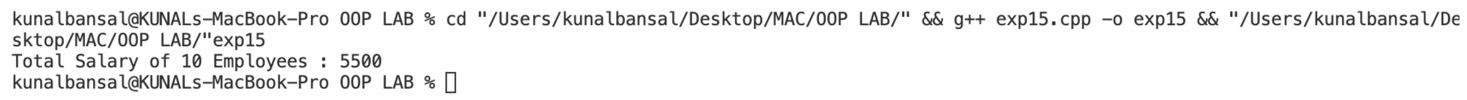
THEORY:

- In C++, a static data member is a class member that retains its value between function calls and is shared across all instances of the class. It is initialized only once and exists for the lifetime of the program. A static member function can access static variables but cannot access non-static members directly since it does not operate on a specific object instance. This program uses a static variable to accumulate the salaries of 10 employees, demonstrating how static members can maintain state and perform calculations that apply to all instances of a class.

CODE:

```
#include<iostream>
using namespace std;
class Employee{
int id;
int salary;
static int total_salary;
public:
Employee(int id , int salary){
this->id = id;
this->salary = salary;
total_salary+=salary;
}
static void display_totalSalary(){
cout<<"Total Salary of 10 Employees : "<<total_salary<<endl;
}
};
int Employee::total_salary;
int main(){
Employee a1(1,100);
Employee a2(2,1000);
Employee a3(3,200);
Employee a4(4,300);
Employee a5(5,400);
Employee a6(6,500);
Employee a7(7,600);
Employee a8(8,700);
Employee a9(9,800);
Employee a10(10,900);
a1.display_totalSalary();
}
```

Output:



```
kunalbansal@KUNALs-MacBook-Pro OOP LAB % cd "/Users/kunalbansal/Desktop/MAC/OOP LAB/" && g++ exp15.cpp -o exp15 && "/Users/kunalbansal/De
sktop/MAC/OOP LAB/"exp15
Total Salary of 10 Employees : 5500
kunalbansal@KUNALs-MacBook-Pro OOP LAB % 
```

Learning Outcomes:

- **Static Members:** Understanding the concept of static members, specifically static variables.
- **Class and Objects:** Creating classes and objects to represent employees.
- **Constructor:** Using constructors to initialize object members.
- **Static Member Functions:** Accessing and modifying static variables using static member functions.

Time Complexity:

- **O(1):** The `display_totalSalary` function has constant time complexity as it directly accesses and prints the static `total_salary` variable.
- **O(n):** The constructor is called 10 times, each with constant time complexity. So, the overall time complexity for object creation is O(n), where n is the number of employees.

EXPERIMENT 16

OBJECTIVE :Write a program using, (a) natural function as friend, (b) member function as friend, to calculate the sum of two complex numbers by using a class complex.

THEORY:

- This C++ program demonstrates the use of friend functions and friend classes to calculate the sum of two complex numbers. In the first approach, a natural function is defined as a friend of the Complex class, allowing it to access the private data members directly to perform the addition. In the second approach, a member function of another class is declared as a friend of the Complex class, enabling it to access the complex number's private members.

CODE:

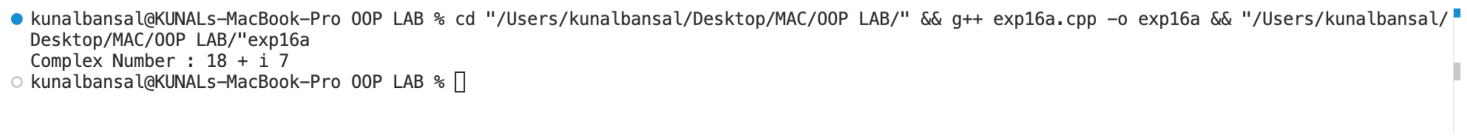
```
//Using Natural Function as friend
#include<iostream>
using namespace std;
class Complex{
int real;
int imaginary;
public:
Complex(){}
Complex(int a , int b){
real = a;
imaginary = b;
}
friend Complex sum(Complex,Complex);
void display(){
cout<<"Complex Number : "<<real<<" + i "<<imaginary<<endl;
}
};
Complex sum(Complex c1 , Complex c2){
Complex c3;
c3.real = c1.real + c2.real;
c3.imaginary = c1.imaginary + c2.imaginary;
return c3;
}
int main(){
Complex c1(10,2);
Complex c2(8,5);
```

Complex c3 = sum(c1,c2);

c3.display();

}

Output:



```
● kunalbansal@KUNALs-MacBook-Pro OOP LAB % cd "/Users/kunalbansal/Desktop/MAC/OOP LAB/" && g++ exp16a.cpp -o exp16a && "/Users/kunalbansal/
  Desktop/MAC/OOP LAB/"exp16a
  Complex Number : 18 + i 7
○ kunalbansal@KUNALs-MacBook-Pro OOP LAB % ▯
```

Learning Outcomes:

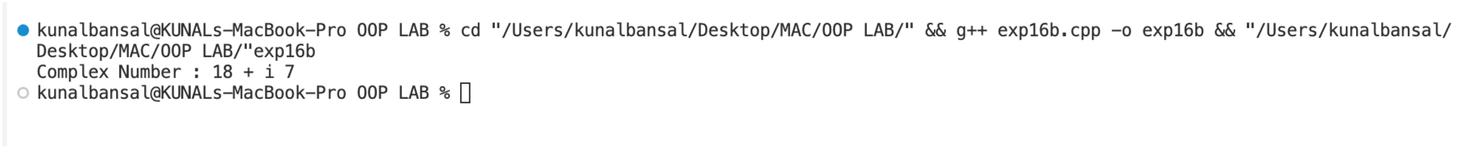
- **Friend Functions:** Understanding the concept of friend functions and how they can access private members of a class.
- **Class and Objects:** Creating classes and objects to represent complex numbers.
- **Operator Overloading:** Simulating operator overloading using friend functions.
- **Input/Output Operations:** Using `cin` and `cout` for input and output.

Time Complexity:

- The time complexity of the `sum` function is O(1), as it performs a fixed number of operations, regardless of the input size.
- The `main` function also has O(1) time complexity.

```
// Using Member function as Friend
#include<iostream>
using namespace std;
class Complex;
class Sum_Calc{
public:
Complex sum(Complex c1, Complex c2);
};
class Complex{
int real;
int imaginary;
public:
Complex(){}
Complex(int a , int b){
real = a;
imaginary = b;
}
friend Complex Sum_Calc::sum(Complex,Complex);
void display(){
cout<<"Complex Number : "<<real<<" + i "<<imaginary<<endl;
}
};
Complex Sum_Calc::sum(Complex c1,Complex c2){
Complex temp;
temp.real = c1.real + c2.real;
temp.imaginary = c1.imaginary + c2.imaginary;
return temp;
}
int main(){
Complex c1(10,2);
Complex c2(8,5);
Sum_Calc s;
Complex c3 = s.sum(c1,c2);
c3.display();
}
```

Output:



```
● kunalbansal@KUNALs-MacBook-Pro OOP LAB % cd "/Users/kunalbansal/Desktop/MAC/OOP LAB/" && g++ exp16b.cpp -o exp16b && "/Users/kunalbansal/
  Desktop/MAC/OOP LAB/"exp16b
  Complex Number : 18 + i 7
○ kunalbansal@KUNALs-MacBook-Pro OOP LAB % 
```

Learning Outcomes:

- **Friend Functions:** Understanding the concept of friend functions and how they can access private members of a class.
- **Class and Objects:** Creating classes and objects to represent complex numbers.
- **Operator Overloading:** Simulating operator overloading using friend functions.
- **Input/Output Operations:** Using `cin` and `cout` for input and output.

Time Complexity:

- The time complexity of the `sum` function is O(1), as it performs a fixed number of operations, regardless of the input size.
- The `main` function also has O(1) time complexity.

EXPERIMENT 17

OBJECTIVE :Write a program to find the area of different shapes with one or two arguments -type int or long or float or double or short or unsigned. Write at least 5 overloading functions for 5- 6 shapes. Each function should print input, shape, output.

THEORY:

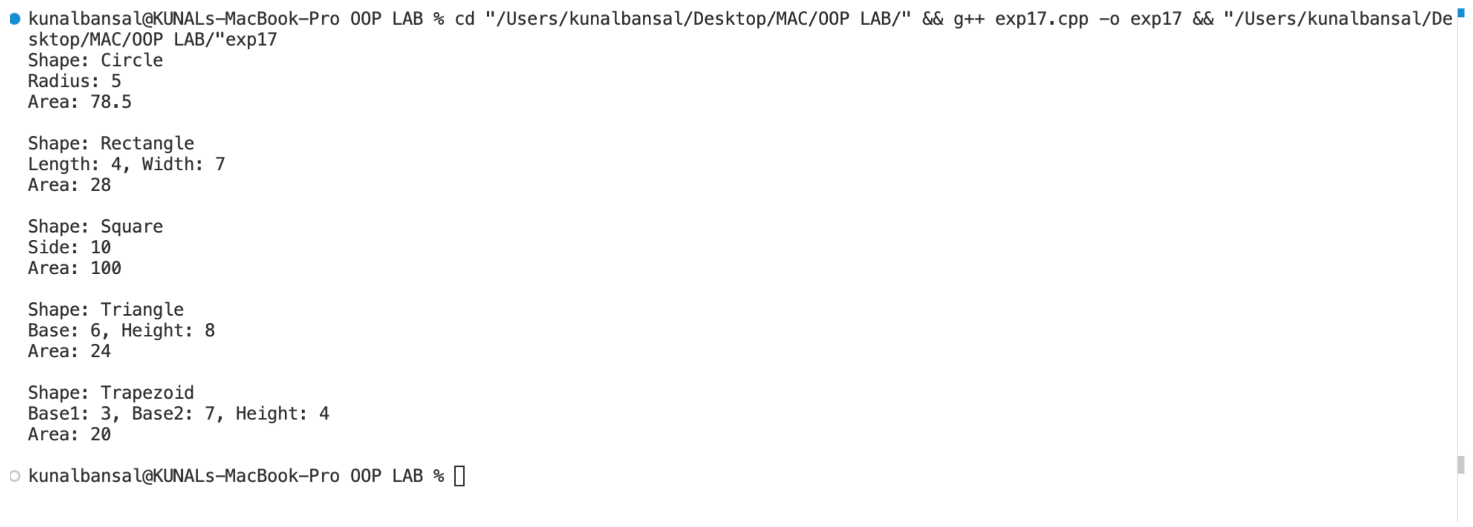
- Operator overloading in C++ allows developers to redefine the behavior of operators for user-defined types, making them behave like built-in types. This feature enables the creation of intuitive interfaces for classes, such as shapes in this program. By overloading functions for calculating the area of various shapes (like circles, rectangles, and triangles), we can provide multiple implementations that accept different argument types (e.g., int, float, double). This allows the same function name to be used for different shapes and data types, enhancing code readability and maintainability.

CODE:

```
#include <iostream>
using namespace std;
# define pi 3.14
//circle
void findArea(float radius) {
cout << "Shape: Circle" << endl;
cout << "Radius: " << radius << endl;
float area = pi * radius * radius;
cout << "Area: " << area << endl << endl;
}
//rectangle
void findArea(int length, int width) {
cout << "Shape: Rectangle" << endl;
cout << "Length: " << length << ", Width: " << width << endl;
int area = length * width;
cout << "Area: " << area << endl << endl;
}
// square
void findArea(long side) {
cout << "Shape: Square" << endl;
cout << "Side: " << side << endl;
long area = side * side;
cout << "Area: " << area << endl << endl;
}
// triangle
```

```
void findArea(unsigned int base, unsigned int height) {
cout << "Shape: Triangle" << endl;
cout << "Base: " << base << ", Height: " << height << endl;
double area = (base * height)/2;
cout << "Area: " << area << endl << endl;
}
// rapezoid
void findArea(short base1, short base2, short height) {
cout << "Shape: Trapezoid" << endl;
cout << "Base1: " << base1 << ", Base2: " << base2 << ", Height: " << height << endl;
float area = ((base1 + base2) * height)/2;
cout << "Area: " << area << endl << endl;
}
int main() {
findArea(5.0f);
findArea(4, 7);
findArea(10L);
findArea(6U, 8U);
findArea(3, 7, 4);
}
```

OUTPUT :



```
kunalbansal@KUNALs-MacBook-Pro OOP LAB % cd "/Users/kunalbansal/Desktop/MAC/OOP LAB/" && g++ exp17.cpp -o exp17 && "/Users/kunalbansal/De
sktop/MAC/OOP LAB/"exp17
Shape: Circle
Radius: 5
Area: 78.5

Shape: Rectangle
Length: 4, Width: 7
Area: 28

Shape: Square
Side: 10
Area: 100

Shape: Triangle
Base: 6, Height: 8
Area: 24

Shape: Trapezoid
Base1: 3, Base2: 7, Height: 4
Area: 20

kunalbansal@KUNALs-MacBook-Pro OOP LAB %
```

Learning Outcomes:

- **Function Overloading: Understanding the concept of function overloading, where functions with the same name but different parameter lists can be defined.**
- **Data Types: Using different data types (float, int, long, unsigned int, short) to represent different types of shapes and their dimensions.**
- **Geometric Formulas: Applying formulas to calculate the area of various shapes.**
- **Input and Output: Using `cout` to display the calculated areas.**

Time Complexity:

- **Each function has a constant time complexity of O(1), as the calculations involved are simple arithmetic operations.**

EXPERIMENT 18

OBJECTIVE : (A)(Hierarchical Inheritance) Write a program to calculate the salary of faculty, staff, using inheritance of class employees with constructors. (B) (Multiple Inheritance) Write a program to calculate the salary of Faculty using inheritance of class Employee and class Salary with constructors.

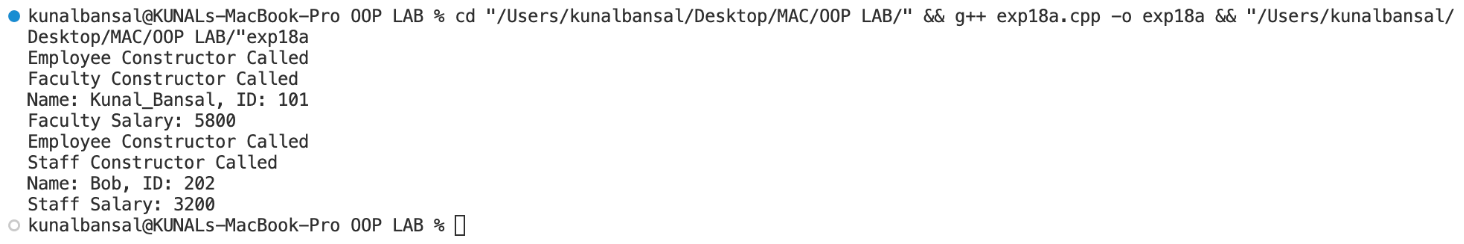
THEORY:

- Hierarchical Inheritance involves a single base class being inherited by multiple derived classes. This structure allows for a clear organizational hierarchy and promotes code reuse, as the base class can provide common attributes and methods that all derived classes can utilize. Multiple Inheritance occurs when a derived class inherits from more than one base class. This allows the derived class to access features from multiple parent classes, enhancing flexibility and functionality.

CODE:

```
// HIERARCHIAL INHERITANCE
#include <iostream>
using namespace std;
class Employee {
protected:
string name;
int id;
public:
Employee(string empName, int empID) : name(empName), id(empID) {
cout << "Employee Constructor Called" << endl;
}
void displayEmployee() {
cout << "Name: " << name << ", ID: " << id << endl;
}
};
class Faculty : public Employee {
private:
float basicSalary, bonus;
public:
Faculty(string facName, int facID, float basic, float bon)
: Employee(facName, facID), basicSalary(basic), bonus(bon) {
cout << "Faculty Constructor Called" << endl;
}
void calculateSalary() {
float salary = basicSalary + bonus;
displayEmployee();
cout << "Faculty Salary: " << salary << endl;
}
};
class Staff : public Employee {
private:
float hourlyWage;
int hoursWorked;
public:
Staff(string staffName, int staffID, float wage, int hours)
: Employee(staffName, staffID), hourlyWage(wage), hoursWorked(hours) {
cout << "Staff Constructor Called" << endl;
}
void calculateSalary() {
float salary = hourlyWage * hoursWorked;
displayEmployee();
cout << "Staff Salary: " << salary << endl;
}
};
int main() {
Faculty facultyMember("Kunal_Bansal", 101, 5000, 800);
facultyMember.calculateSalary();
Staff staffMember("Bob", 202, 20, 160);
staffMember.calculateSalary();
}
```

OUTPUT:



```
● kunalbansal@KUNALs-MacBook-Pro OOP LAB % cd "/Users/kunalbansal/Desktop/MAC/OOP LAB/" && g++ exp18a.cpp -o exp18a && "/Users/kunalbansal/
  Desktop/MAC/OOP LAB/"exp18a
  Employee Constructor Called
  Faculty Constructor Called
  Name: Kunal_Bansal, ID: 101
  Faculty Salary: 5800
  Employee Constructor Called
  Staff Constructor Called
  Name: Bob, ID: 202
  Staff Salary: 3200
○ kunalbansal@KUNALs-MacBook-Pro OOP LAB % 
```

Learning Outcomes:

- **Inheritance: Understanding the concept of inheritance, specifically single inheritance.**
- **Constructors: Using constructors to initialize object members.**
- **Access Specifiers: Using `public`, `protected`, and `private` access specifiers.**
- **Function Overriding: Redefining functions in derived classes to provide specific**

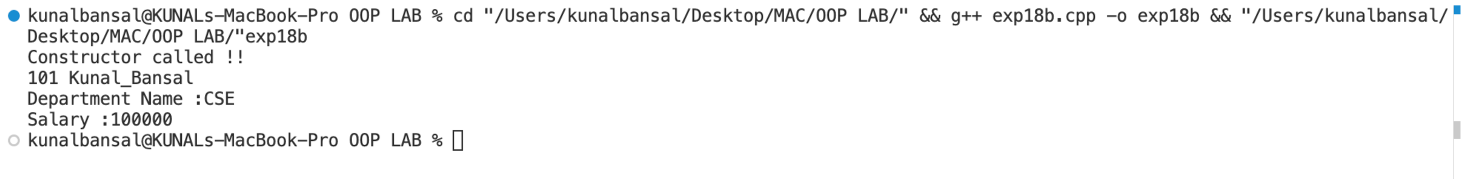
implementations.
- Object-Oriented Programming: Applying OOP principles to create a hierarchy of classes.

Time Complexity:

- The time complexity of the `main` function is O(1), as it involves a fixed number of operations, regardless of the input size.

```
//MULTIPLE INHERITANCE
#include<iostream>
using namespace std;
class Staff{
protected:
int id;
string Name;
public:
Staff(int id , string Name){
this->id = id;
this->Name = Name;
}
};
class Salary{
protected:
float Salary_staff;
public:
Salary(float amt){
this->Salary_staff = amt;
}
} ;
class Faculty : public Staff , public Salary {
string Department;
public:
Faculty(int id , string name , float salary , string Dep) : Staff(id , name) , Salary(salary) {
this->Department = Dep;
cout<<"Constructor called !!"<<endl;
}
void display(){
cout<<id<<" "<<Name<<endl;
cout<<"Department Name :"<<Department<<endl;
cout<<"Salary :"<<Salary_staff<<endl;
}
};
int main(){
Faculty Kunal(101,"Kunal_Bansal" , 100000 , "CSE");
Kunal.display();
}
```

OUTPUT:



```
● kunalbansal@KUNALs-MacBook-Pro OOP LAB % cd "/Users/kunalbansal/Desktop/MAC/OOP LAB/" && g++ exp18b.cpp -o exp18b && "/Users/kunalbansal/
  Desktop/MAC/OOP LAB/"exp18b
  Constructor called !!
  101 Kunal_Bansal
  Department Name :CSE
  Salary :100000
○ kunalbansal@KUNALs-MacBook-Pro OOP LAB % 
```

Learning Outcomes:

- **Multiple Inheritance: Understanding the concept of multiple inheritance, where a class inherits from two or more base classes.**
- **Constructor Chaining: Using the initializer list to call base class constructors.**
- **Access Specifiers: Using `public`, `protected`, and `private` access specifiers to control member accessibility.**
- **Object-Oriented Programming: Applying OOP principles to create a hierarchy of classes.**

Time Complexity:

- **The time complexity of the `main` function is O(1), as it involves a fixed number of operations, regardless of the input size.**

EXPERIMENT 19

OBJECTIVE : Write a program to find square and cube of a number using (write functions for getData, showData, showResult) late/dynamic binding using base pointer.

THEORY:

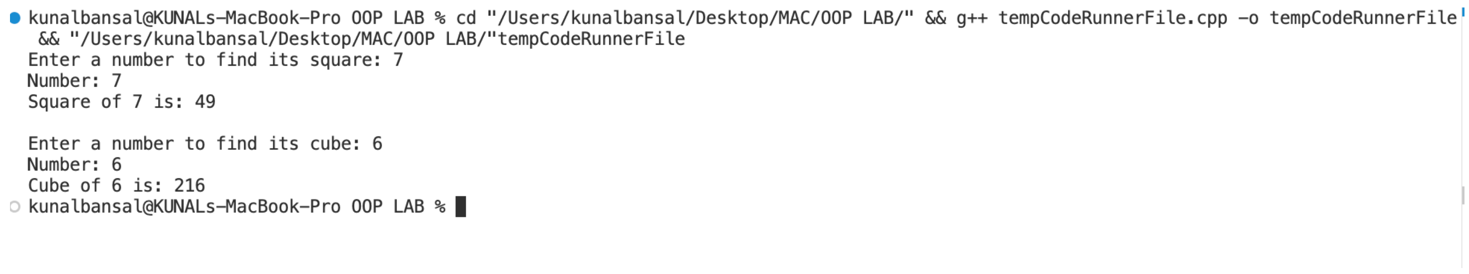
- Dynamic binding in C++ refers to the mechanism of resolving function calls at runtime based on the type of the object being pointed to, rather than the type of the pointer itself. This enables polymorphism, allowing derived classes to provide specific implementations of functions declared in a base class. By using virtual functions, a base class pointer can invoke the appropriate derived class function, facilitating flexible and extensible code design. A pure abstract class in C++ is a class that contains at least one pure virtual function, making it impossible to instantiate and primarily serving as an interface for derived classes.

CODE:

```
#include <iostream>
using namespace std;
class Number {
protected:
int num;
public:
virtual void getData() = 0;
virtual void showData() = 0;
virtual void showResult() = 0;
};
class Square : public Number {
public:
void getData(){
cout << "Enter a number to find its square: ";
cin >> num;
}
void showData(){
cout << "Number: " << num << endl;
}
void showResult(){
cout << "Square of " << num << " is: " << num * num << endl;
}
};
class Cube : public Number {
public:
```

```
void getData(){
cout << "Enter a number to find its cube: ";
cin >> num;
}
void showData(){
cout << "Number: " << num << endl;
}
void showResult(){
cout << "Cube of " << num << " is: " << num * num * num << endl;
}
};
int main() {
Number *ptr;
Square sq;
ptr = &sq;
ptr->getData();
ptr->showData();
ptr->showResult();
cout<<endl;
Cube cb;
ptr = &cb;
ptr->getData();
ptr->showData();
ptr->showResult();
}
```

OUTPUT:



```
kunalbansal@KUNALs-MacBook-Pro OOP LAB % cd "/Users/kunalbansal/Desktop/MAC/OOP LAB/" && g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile
 && "/Users/kunalbansal/Desktop/MAC/OOP LAB/"tempCodeRunnerFile
Enter a number to find its square: 7
Number: 7
Square of 7 is: 49

Enter a number to find its cube: 6
Number: 6
Cube of 6 is: 216
kunalbansal@KUNALs-MacBook-Pro OOP LAB % 
```

Learning Outcomes:

- **Abstract Classes: Understanding the concept of abstract classes and their use in creating base classes with pure virtual functions.**
- **Polymorphism: Demonstrating polymorphism through a base class pointer pointing to derived**

class objects.
- **Virtual Functions: Using virtual functions to achieve dynamic binding and polymorphism.**
- **Inheritance: Understanding the concept of inheritance and how derived classes inherit from base classes.**

Time Complexity:

- **The time complexity of the `main` function is O(1) for each object, as the operations involved are simple input, output, and arithmetic calculations.**

EXPERIMENT 20

OBJECTIVE : Write a program to overload ! (not) operator (unary) all data members of a class.

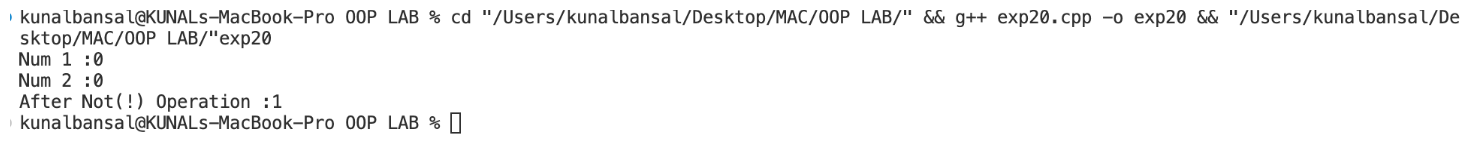
THEORY:

- Operator overloading in C++ allows developers to define custom behaviors for operators when they are applied to user-defined types (classes). By overloading operators, we can make objects of a class work intuitively with operators such as +, -, and even logical operators. This enhances code readability and usability, enabling operators to interact with class instances as if they were built-in types. In this program, the logical NOT operator ! is overloaded to determine if all data members of an object are zero.

CODE:

```
#include<iostream>
using namespace std;
class Num{
int a;
int b;
public:
Num(int a , int b){
this->a=a;
this->b=b;
}
bool operator!(){
return !(a||b);
}
void display(){
cout<<"Num 1 :"<<a<<endl;
cout<<"Num 2 :"<<b<<endl;
}
};
int main(){
Num a(0,0);
bool output = !a;
a.display();
cout<<"After Not(!) Operation :"<<output<<endl;
}
```

OUTPUT:



```
kunalbansal@KUNALs-MacBook-Pro OOP LAB % cd "/Users/kunalbansal/Desktop/MAC/OOP LAB/" && g++ exp20.cpp -o exp20 && "/Users/kunalbansal/De
sktop/MAC/OOP LAB/"exp20
Num 1 :0
Num 2 :0
After Not(!) Operation :1
kunalbansal@KUNALs-MacBook-Pro OOP LAB %
```

Learning Outcomes:

- **Operator Overloading: Understanding the concept of operator overloading to customize the behavior of operators for user-defined classes.**
- **Logical NOT Operator Overloading: Overloading the `!` operator to perform logical negation on objects of the `Num` class.**
- **Class and Objects: Creating classes and objects to represent numbers.**
- **Input/Output Operations: Using `cin` and `cout` for input and output.**

Time Complexity:

- **The time complexity of the `!` operator overload is O(1), as it performs a simple logical negation.**
- **The `main` function also has O(1) time complexity.**

EXPERIMENT 21

OBJECTIVE :Use Template Function to sort an array call function with 5 different combinations including pointer, float, int, etc.

THEORY:

- Template functions in C++ enable a single function to operate across multiple data types, enhancing code reusability and flexibility. By defining a template, the same function can be adapted to handle various types, such as `int`, `float`, `double`, or even pointers. This is particularly useful for generic tasks, like sorting. When a template function is invoked with a specific data type, the compiler automatically generates the correct version of the function for that type. This approach minimizes code duplication and simplifies maintenance.

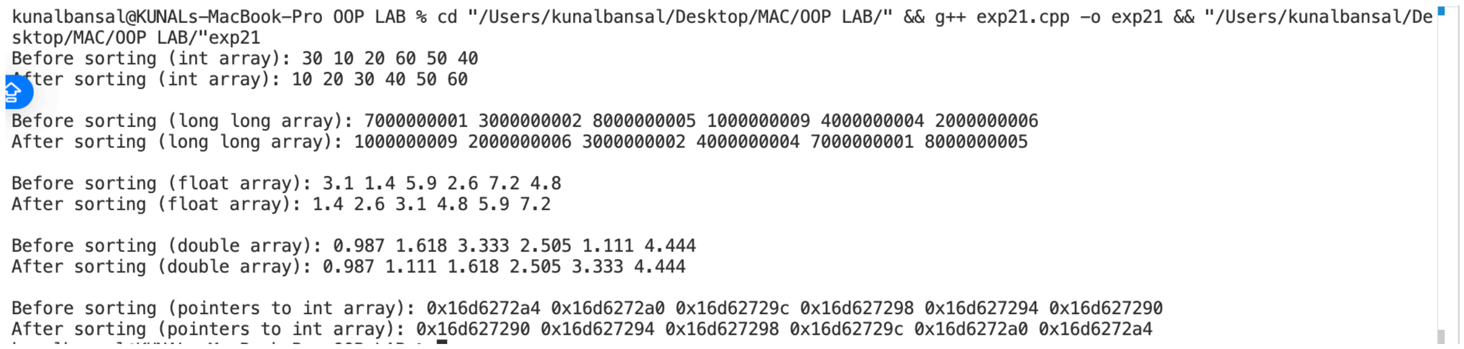
CODE:

```
#include <iostream>

using namespace std;

// Template function to sort an array

template <typename T>

void sortArray(T arr[], int size) {

for (int i = 0; i < size - 1; i++) {

for (int j = 0; j < size - 1 - i; j++) {

if (arr[j] > arr[j + 1]) {

T temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

// Template function to print an array

template <typename T>

void displayArray(T arr[], int size) {

for (int i = 0; i < size; i++) {
```

```
cout << arr[i] << " ";

}

cout << endl;

}

int main() {

const int arrSize = 6;

int intArr[arrSize] = {30, 10, 20, 60, 50, 40};

cout << "Before sorting (int array): ";

displayArray(intArr, arrSize);

sortArray(intArr, arrSize);

cout << "After sorting (int array): ";

displayArray(intArr, arrSize);

long long llArr[arrSize] = {7000000001LL, 3000000002LL, 8000000005LL, 1000000009LL, 4000000004LL,
2000000006LL};

cout << "\nBefore sorting (long long array): ";

displayArray(llArr, arrSize);

sortArray(llArr, arrSize);

cout << "After sorting (long long array): ";

displayArray(llArr, arrSize);

float floatArr[arrSize] = {3.1f, 1.4f, 5.9f, 2.6f, 7.2f, 4.8f};

cout << "\nBefore sorting (float array): ";

displayArray(floatArr, arrSize);

sortArray(floatArr, arrSize);

cout << "After sorting (float array): ";

displayArray(floatArr, arrSize);

double doubleArr[arrSize] = {0.987, 1.618, 3.333, 2.505, 1.111, 4.444};

cout << "\nBefore sorting (double array): ";

displayArray(doubleArr, arrSize);

sortArray(doubleArr, arrSize);
```

```
cout << "After sorting (double array): ";

displayArray(doubleArr, arrSize);

int x = 25, y = 45, z = 5, w = 55, u = 35, v = 15;

int* ptrArr[arrSize] = {&x, &y, &z, &w, &u, &v};

cout << "\nBefore sorting (pointers to int array): ";

displayArray(ptrArr, arrSize);

sortArray(ptrArr, arrSize);

cout << "After sorting (pointers to int array): ";

displayArray(ptrArr, arrSize);

return 0;

}
```

OUTPUT:



```
kunalbansal@KUNALs-MacBook-Pro OOP LAB % cd "/Users/kunalbansal/Desktop/MAC/OOP LAB/" && g++ exp21.cpp -o exp21 && "/Users/kunalbansal/De
sktop/MAC/OOP LAB/"exp21
Before sorting (int array): 30 10 20 60 50 40
After sorting (int array): 10 20 30 40 50 60

Before sorting (long long array): 7000000001 3000000002 8000000005 1000000009 4000000004 2000000006
After sorting (long long array): 1000000009 2000000006 3000000002 4000000004 7000000001 8000000005

Before sorting (float array): 3.1 1.4 5.9 2.6 7.2 4.8
After sorting (float array): 1.4 2.6 3.1 4.8 5.9 7.2

Before sorting (double array): 0.987 1.618 3.333 2.505 1.111 4.444
After sorting (double array): 0.987 1.111 1.618 2.505 3.333 4.444

Before sorting (pointers to int array): 0x16d6272a4 0x16d6272a0 0x16d62729c 0x16d627298 0x16d627294 0x16d627290
After sorting (pointers to int array): 0x16d627290 0x16d627294 0x16d627298 0x16d62729c 0x16d6272a0 0x16d6272a4
```

Learning Outcomes:

- **Template Functions: Understanding the concept of template functions and their use in creating generic functions that can work with different data types.**
- **Sorting Algorithms: Implementing a simple bubble sort algorithm to sort arrays.**
- **Pointer Arithmetic: Using pointers to access and manipulate array elements.**
- **Generic Programming: Writing code that can work with different data types without modification.**

Time Complexity:

- **The time complexity of the `sortArray` function is O(n^2) in the worst case, where n is the size of the array. This is due to the nested loops used in the bubble sort algorithm.**