Here’s a solution guide for each question based on the image:

Assignment No. 3: Extending Classes using Inheritance

1. Describe access specifiers with examples. Difference between public and private access specifier.

• Answer:

Access specifiers in C++ determine the accessibility of class members. The main types are:

• public: Members are accessible from outside the class.

• private: Members are accessible only within the class.

• protected: Members are accessible within the class and by derived classes.

Example:

class Example {

public:

int publicVar; // Accessible anywhere

private:

int privateVar; // Accessible only within Example class

};

Difference:

• Public members can be accessed from any part of the program.

• Private members are restricted to the class where they are defined.

2. What do you mean by inline function? Write its syntax and example.

• Answer:

Inline functions are functions defined with the inline keyword. They are expanded in line when invoked, which reduces the function-call overhead.

Syntax:

inline returnType functionName(parameters) {

// function body

}

Example:

inline int square(int x) {

return x \* x;

}

3. Define derived class. Give one example.

• Answer:

A derived class is a class created from another class (base class) using inheritance. It inherits attributes and methods of the base class.

Example:

class Base {

public:

int baseVar;

};

class Derived : public Base {

public:

int derivedVar;

};

4. What is inheritance? Why is inheritance used in C++?

• Answer:

Inheritance is a feature in object-oriented programming that allows a class to inherit properties and behavior from another class. It promotes code reusability and establishes a relationship between classes.

5. Describe visibility modes with an example.

• Answer:

Visibility modes (public, protected, and private) determine how the base class members are inherited in the derived class.

Example:

class Base {

public:

int x;

protected:

int y;

private:

int z;

};

class Derived : public Base {

// x is public, y is protected, z is not accessible

};

6. Explain effects of visibility modes using a table.

• Answer:

Base Class Member Public Inheritance Protected Inheritance Private Inheritance

public Public Protected Private

protected Protected Protected Private

private Not Accessible Not Accessible Not Accessible

7. List the different types of inheritance.

• Answer:

• Single Inheritance

• Multiple Inheritance

• Multilevel Inheritance

• Hierarchical Inheritance

• Hybrid Inheritance

8. Explain various types of inheritance with examples and syntax.

• Answer:

1. Single Inheritance: One derived class from one base class.

class A {};

class B : public A {};

2. Multiple Inheritance: One derived class from multiple base classes.

class A {};

class B {};

class C : public A, public B {};

3. Multilevel Inheritance: A derived class from another derived class.

class A {};

class B : public A {};

class C : public B {};

4. Hierarchical Inheritance: Multiple derived classes from a single base class.

class A {};

class B : public A {};

class C : public A {};

5. Hybrid Inheritance: Combination of two or more types of inheritance.

9. Write a program showing use of single inheritance.

• Answer:

#include <iostream>

using namespace std;

class Base {

public:

void display() {

cout << "Base class function" << endl;

}

};

class Derived : public Base {

public:

void show() {

cout << "Derived class function" << endl;

}

};

int main() {

Derived obj;

obj.display();

obj.show();

return 0;

}

10. How to solve ambiguity occurred in multiple inheritance? Give example.

• Answer:

Ambiguity in multiple inheritance occurs when two base classes have methods with the same name. It can be resolved using the scope resolution operator.

Example:

class Base1 {

public:

void display() {

cout << "Base1 display" << endl;

}

};

class Base2 {

public:

void display() {

cout << "Base2 display" << endl;

}

};

class Derived : public Base1, public Base2 {

public:

void show() {

Base1::display(); // Resolving ambiguity

}

};

11. Explain multiple inheritance with a suitable example.

• Answer:

Multiple inheritance is when a derived class inherits from more than one base class.

Example:

class A {

public:

void funcA() {

cout << "Class A function" << endl;

}

};

class B {

public:

void funcB() {

cout << "Class B function" << endl;

}

};

class C : public A, public B {};

12. Write the program to implement inheritance as shown in figure. Assume suitable member functions, create at least one object.

• Answer:

#include <iostream>

using namespace std;

class Employee {

protected:

int Emp\_id;

string name;

public:

void setEmployee(int id, string n) {

Emp\_id = id;

name = n;

}

void showEmployee() {

cout << "Employee ID: " << Emp\_id << ", Name: " << name << endl;

}

};

class Worker : public Employee {

protected:

int Daily\_wages;

public:

void setWages(int wages) {

Daily\_wages = wages;

}

void showWorker() {

showEmployee();

cout << "Daily Wages: " << Daily\_wages << endl;

}

};

int main() {

Worker w;

w.setEmployee(1, "John");

w.setWages(500);

w.showWorker();

return 0;

}

Question 13

#include <iostream>

using namespace std;

class Student {

protected:

int roll\_no;

string name;

public:

void getStudentData() {

cout << "Enter roll number: ";

cin >> roll\_no;

cout << "Enter name: ";

cin >> name;

}

void displayStudentData() {

cout << "Roll No: " << roll\_no << endl;

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

}

};

class Engg\_stud : public Student {

private:

string subject;

public:

void getEnggData() {

getStudentData();

cout << "Enter subject: ";

cin >> subject;

}

void displayEnggData() {

displayStudentData();

cout << "Subject: " << subject << endl;

}

};

class Diploma\_stud : public Student {

private:

string result;

public:

void getDiplomaData() {

getStudentData();

cout << "Enter result: ";

cin >> result;

}

void displayDiplomaData() {

displayStudentData();

cout << "Result: " << result << endl;

}

};

int main() {

Engg\_stud enggStudent;

Diploma\_stud diplomaStudent;

cout << "Enter details for Engineering Student:" << endl;

enggStudent.getEnggData();

cout << "\nEngineering Student Details:" << endl;

enggStudent.displayEnggData();

cout << "\nEnter details for Diploma Student:" << endl;

diplomaStudent.getDiplomaData();

cout << "\nDiploma Student Details:" << endl;

diplomaStudent.displayDiplomaData();

return 0;

}

Question 14

#include <iostream>

using namespace std;

class Customer {

protected:

string name;

string phone;

public:

void getCustomerData() {

cout << "Enter name: ";

cin >> name;

cout << "Enter phone number: ";

cin >> phone;

}

void displayCustomerData() {

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

cout << "Phone: " << phone << endl;

}

};

class Depositor : public Customer {

protected:

int acc\_no;

float balance;

public:

void getDepositorData() {

getCustomerData();

cout << "Enter account number: ";

cin >> acc\_no;

cout << "Enter balance: ";

cin >> balance;

}

void displayDepositorData() {

displayCustomerData();

cout << "Account No: " << acc\_no << endl;

cout << "Balance: " << balance << endl;

}

};

class Borrower : public Depositor {

private:

float loan;

float loan\_amount;

public:

void getBorrowerData() {

getDepositorData();

cout << "Enter loan: ";

cin >> loan;

cout << "Enter loan amount: ";

cin >> loan\_amount;

}

void displayBorrowerData() {

displayDepositorData();

cout << "Loan: " << loan << endl;

cout << "Loan Amount: " << loan\_amount << endl;

}

};

int main() {

Borrower borrower;

cout << "Enter Borrower details:" << endl;

borrower.getBorrowerData();

cout << "\nBorrower Details:" << endl;

borrower.displayBorrowerData();

return 0;

}

Question 15

#include <iostream>

using namespace std;

class Staff {

protected:

int code;

public:

void getStaffData() {

cout << "Enter staff code: ";

cin >> code;

}

void displayStaffData() {

cout << "Staff Code: " << code << endl;

}

};

class Teacher : public Staff {

private:

string subject;

public:

void getTeacherData() {

getStaffData();

cout << "Enter subject: ";

cin >> subject;

}

void displayTeacherData() {

displayStaffData();

cout << "Subject: " << subject << endl;

}

};

class Officer : public Staff {

private:

string grade;

public:

void getOfficerData() {

getStaffData();

cout << "Enter grade: ";

cin >> grade;

}

void displayOfficerData() {

displayStaffData();

cout << "Grade: " << grade << endl;

}

};

int main() {

Teacher teacher;

Officer officer;

cout << "Enter details for Teacher:" << endl;

teacher.getTeacherData();

cout << "\nTeacher Details:" << endl;

teacher.displayTeacherData();

cout << "\nEnter details for Officer:" << endl;

officer.getOfficerData();

cout << "\nOfficer Details:" << endl;

officer.displayOfficerData();

return 0;

}

Question 16

#include <iostream>

using namespace std;

class Employee {

protected:

int emp\_no;

string emp\_name;

public:

void getEmployeeData() {

cout << "Enter employee number: ";

cin >> emp\_no;

cout << "Enter employee name: ";

cin >> emp\_name;

}

void displayEmployeeData() {

cout << "Employee No: " << emp\_no << endl;

cout << "Employee Name: " << emp\_name << endl;

}

};

class Fitness : public Employee {

private:

float height;

float weight;

public:

void getFitnessData() {

getEmployeeData();

cout << "Enter height: ";

cin >> height;

cout << "Enter weight: ";

cin >> weight;

}

void displayFitnessData() {

displayEmployeeData();

cout << "Height: " << height << endl;

cout << "Weight: " << weight << endl;

}

};

int main() {

Fitness fitness;

cout << "Enter Fitness details for employee:" << endl;

fitness.getFitnessData();

cout << "\nFitness Details:" << endl;

fitness.displayFitnessData();

return 0;

}

Question 17

#include <iostream>

using namespace std;

class Employee {

protected:

string name;

int id;

public:

Employee() {

cout << "Enter Employee Name: ";

cin >> name;

cout << "Enter Employee ID: ";

cin >> id;

}

void putData() {

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

cout << "ID: " << id << endl;

}

};

class Salary : private Employee {

private:

float basic\_pay, hra, da, cla;

public:

Salary() : Employee() {

cout << "Enter Basic Pay: ";

cin >> basic\_pay;

cout << "Enter HRA: ";

cin >> hra;

cout << "Enter DA: ";

cin >> da;

cout << "Enter CLA: ";

cin >> cla;

}

void calculateSalary() {

float totalSalary = basic\_pay + hra + da + cla;

putData();

cout << "Total Salary: " << totalSalary << endl;

}

};

int main() {

Salary salary;

cout << "\nEmployee Salary Details:" << endl;

salary.calculateSalary();

return 0;

}

13. Write a program to implement inheritance as shown in figure (for Student and related classes).

• Answer:

#include <iostream>

using namespace std;

class Student {

protected:

int roll\_no;

string name;

public:

void setStudent(int roll, string n) {

roll\_no = roll;

name = n;

}

void showStudent() {

cout << "Roll No: " << roll\_no << ", Name: " << name << endl;

}

};

// Assuming the inheritance structure is expanded

class Exam : public Student {

protected:

int marks;

public:

void setMarks(int m) {

marks = m;

}

void showExam() {

showStudent();

cout << "Marks: " << marks << endl;

}

};

int main() {

Exam e;

e.setStudent(101, "Alice");

e.setMarks(85);

e.showExam();

return 0;

}

1. What is a pointer? Describe pointer operator and address operator.

• A pointer is a variable that stores the memory address of another variable. The & (address-of) operator is used to get the address of a variable, and the \* (dereference) operator is used to access the value at the address stored in the pointer.

int x = 10;

int \*p = &x; // p stores the address of x

cout << \*p; // Outputs the value of x, which is 10

2. Explain the concept of pointer to objects with example.

• A pointer to an object stores the address of an object instance, allowing access to the object’s members through the pointer.

class Sample {

public:

void display() { cout << "Hello, Object Pointer!"; }

};

Sample \*objPtr = new Sample();

objPtr->display(); // Output: Hello, Object Pointer!

3. Explain the concept of pointer to array of objects with example.

• A pointer to an array of objects allows you to manage multiple objects through a single pointer.

class Sample {

public:

void show() { cout << "Object Array"; }

};

Sample \*arr = new Sample[3];

arr[0].show(); // Output: Object Array

4. Explain pointer arithmetic.

• Pointer arithmetic allows moving the pointer to different memory locations. For example, adding 1 to a pointer moves it to the next memory location based on the data type.

int arr[] = {10, 20, 30};

int \*p = arr;

cout << \*p; // Output: 10

p++;

cout << \*p; // Output: 20

5. Describe the use of this pointer with a suitable example.

• The this pointer in C++ is a pointer that points to the object calling the member function.

class Sample {

int x;

public:

Sample(int x) { this->x = x; }

void display() { cout << "Value of x: " << x; }

};

6. Define polymorphism. Give example. State its types.

• Polymorphism allows different functions to be called with the same name. Types:

• Compile-time polymorphism (function and operator overloading)

• Run-time polymorphism (achieved with inheritance and virtual functions)

7. What is runtime polymorphism? Explain with suitable example.

• Runtime polymorphism is achieved with virtual functions and allows functions to behave differently in derived classes.

class Animal {

public:

virtual void sound() { cout << "Animal Sound"; }

};

class Dog : public Animal {

public:

void sound() override { cout << "Bark"; }

};

Animal \*a = new Dog();

a->sound(); // Output: Bark

8. What is static polymorphism? Explain with suitable example.

• Static polymorphism is resolved at compile-time, achieved through function overloading and operator overloading.

class Print {

public:

void display(int i) { cout << "Integer: " << i; }

void display(double d) { cout << "Double: " << d; }

};

9. What is function overloading? Give one example.

• Function overloading is defining multiple functions with the same name but different parameters.

void print(int i) { cout << i; }

void print(double d) { cout << d; }

10. Difference between function overloading and overriding.

• Overloading is defining multiple functions with the same name but different signatures.

• Overriding is redefining a base class function in a derived class using virtual functions.

11. Write any eight rules of operator overloading.

• Only existing operators can be overloaded.

• Overloading doesn’t change operator precedence.

• Overloaded operators must have at least one user-defined type as an operand.

• Some operators like ::, ?:, sizeof cannot be overloaded.

• The assignment operator must be overloaded as a non-static member.

• Cannot overload operators to change number of operands.

• Overloading doesn’t change the operator’s basic meaning.

• Certain operators should be overloaded as member functions.

12. Can the base class pointer be used to call virtual function? Justify your answer.

• Yes, a base class pointer can call virtual functions.

If a virtual function is overridden in a derived class, the derived class function will be called due to runtime polymorphism.

13. Write any six rules of virtual function.

• Virtual functions must be in the base class.

• Virtual functions cannot be static.

• Virtual functions can be a friend of another class.

• A virtual function must be accessed through a pointer or reference.

• If a virtual function is defined in a base class, it should also be implemented in derived classes.

• Virtual functions are slower due to v-table lookup.

14. Write a program to overload + operator to concatenate two strings.

class String {

string str;

public:

String(string s) : str(s) {}

String operator+(String const &s) {

return String(str + s.str);

}

void display() { cout << str; }

};

15. Write a program to reverse a string by overloading ~ operator.

class String {

string str;

public:

String(string s) : str(s) {}

String operator~() {

reverse(str.begin(), str.end());

return String(str);

}

};

16. Write a program to overload - operator to negate the value of a variable.

class Number {

int value;

public:

Number(int v) : value(v) {}

int operator-() {

return -value;

}

};

17-19. Function overloading for area and volume calculations.

- Use function overloading for each calculation by defining multiple calculate methods in a class.

Assignment No. 5: Chapter 5 - File Handling

1. Write a C++ program to write “Welcome to poly” in a file.

ofstream outFile("welcome.txt");

outFile << "Welcome to poly";

outFile.close();

2. Write a C++ program to copy data from abc.txt to xyz.txt.

ifstream inFile("abc.txt");

ofstream outFile("xyz.txt");

string line;

while (getline(inFile, line)) {

outFile << line << endl;

}

3. Develop C++ program to open and read content of file.

ifstream inFile("file.txt");

string content;

while (getline(inFile, content)) {

cout << content << endl;

}

4. Develop C++ program to check Detection of end of file.

ifstream file("file.txt");

while (!file.eof()) {

string line;

getline(file, line);

cout << line << endl;

}

5. Enlist the different modes of file handling.

• ios::in, ios::out, ios::app, ios::ate, ios::trunc, ios::binary

Feel free to reach out if you need further details on any specific answer or additional explanations!

Assignment No. 4: Chapter 4 - Pointer and Polymorphism

18. Write a program using function overloading to calculate area of a rectangle and a circle.

#include <iostream>

using namespace std;

class Shape {

public:

// Function to calculate area of rectangle

double area(double length, double width) {

return length \* width;

}

// Function to calculate area of circle

double area(double radius) {

return 3.14159 \* radius \* radius;

}

};

int main() {

Shape shape;

cout << "Area of Rectangle (5 x 4): " << shape.area(5, 4) << endl; // Output: 20

cout << "Area of Circle (radius 3): " << shape.area(3) << endl; // Output: ~28.27

return 0;

}

19. Write a program using function overloading to calculate the volume of a rectangle, cube, and cylinder.

#include <iostream>

using namespace std;

class Volume {

public:

// Function to calculate volume of rectangle (length \* width \* height)

double calculate(double length, double width, double height) {

return length \* width \* height;

}

// Function to calculate volume of cube (side^3)

double calculate(double side) {

return side \* side \* side;

}

// Function to calculate volume of cylinder (π \* radius^2 \* height)

double calculate(double radius, double height) {

return 3.14159 \* radius \* radius \* height;

}

};

int main() {

Volume volume;

cout << "Volume of Rectangle (5 x 4 x 3): " << volume.calculate(5, 4, 3) << endl; // Output: 60

cout << "Volume of Cube (side 3): " << volume.calculate(3) << endl; // Output: 27

cout << "Volume of Cylinder (radius 3, height 5): " << volume.calculate(3, 5) << endl; // Output: ~141.37

return 0;

}

20. Write a program to swap two integer numbers and two float numbers using function overloading.

#include <iostream>

using namespace std;

class Swapper {

public:

// Function to swap two integers

void swap(int &a, int &b) {

int temp = a;

a = b;

b = temp;

}

// Function to swap two floats

void swap(float &a, float &b) {

float temp = a;

a = b;

b = temp;

}

};

int main() {

Swapper swapper;

int int1 = 5, int2 = 10;

cout << "Before swap: int1 = " << int1 << ", int2 = " << int2 << endl;

swapper.swap(int1, int2);

cout << "After swap: int1 = " << int1 << ", int2 = " << int2 << endl;

float float1 = 3.5, float2 = 7.2;

cout << "Before swap: float1 = " << float1 << ", float2 = " << float2 << endl;

swapper.swap(float1, float2);

cout << "After swap: float1 = " << float1 << ", float2 = " << float2 << endl;

return 0;

}