Digital Design Verification

Inheritance & Virtual Functions in C++

Assignment # 01

Release: 1.0

Date: 05-March-2025

NUST Chip Design Centre (NCDC), Islamabad, Pakistan

Revision History

Revision Number	Revision Date	Revision By	Nature of Revision	Approved By
1.0	05/03/2025	Junaid Khan	Manual	Name / Designation

TABLE OF CONTENTS

Contents

Objective

The objective of this assignment is to familiarize students with the concept of inheritance and runtime polymorphism in C++ using virtual functions.

Tools & Language

- GNU G++ compiler
- C++

Introduction

Inheritance in C++

Inheritance is a form of software reuse: we create a new class that absorbs the data and behaviors of an existing class and enhances them with new capabilities (new members, redefined members).

The terms superclass and subclass are commonly used to describe the new and old class, but C++ uses the terms derived class and base class. Although a derived class possesses all of the members of its base class private members of the base class cannot be accessed directly in the methods or friend functions of the derived class; if we need to access these we must do so by calls to methods or friend functions of the base class.

C++ provides three kinds of inheritance: public, protected and private. When using public inheritance the public and protected members of the base class are regarded as public and protected members of the derived class. When using protected inheritance the public and protected members of the base class are regarded as protected members of the derived class and when using private inheritance all members of the base class are regarded as private members of the derived class.

If a class A is a subclass of B, which is in turn a subclass of C then C is said to be an indirect base class of A whereas B is a direct base class of A. In C++ (unlike Java) multiple inheritance is allowed –a class may have more than one direct base class.

Virtual Function

C++ normally uses static binding, the choice of which parent of child class method to invoke is made by the compiler according to the type of the object variable, not the type of the object. To get dynamic binding in C++, we have to declare a member function in a base class to be a virtual function. This is done by preceding its name with the keyword virtual. Example declaration:

virtual float area() const { return 0.0; }

The function in the derived class should not be declared as virtual unless we expect to further extend this class with other subclasses that will need different versions of the function.

Task # 01:

A commission employee earns only commission on the sales that he makes, where as a base-plus-commission employee also earns a basic flat rate salary in addition to his commission on sales. Write separate classes for commission and base-plus-commission employees. Extend base-plus-commission from commission.

The classes should have the below mentioned attributes.

For commission employee, specify firstName, lastName, socialSecurityNumber, grossSales and commissionRate member variables. Also provide an implementation of the following functions:

- setFirstName
- getFirstName
- setLastName
- getLastName
- setSocialSecurityNumber
- getSocialSecurityNumber
- setGrossSales
- getGrossSales
- setCommissionRate
- getCommissionRate

- earnings

Print the commission employee earnings on the console.

For base-plus-commission set a member variable baseSalary. Implement the methods:

- setBaseSalary
- getBaseSalary
- earnings

The earnings method in the derived class overrides it's base class implementation. Use base salary and earnings from commission employee to calculate base plus commission employee earnings. Print the output on console.

Task #02:

In the base class of task1, make the earnings function as virtual. Create a commission employee reference variable and assign base plus commission employee object to it. Now print the earnings using the reference variable. Which class method is called?