Page No.
Date

22 Project Based Learning-I: Assignment-1

Name: Parinitha M. Samaga
Roll Number: SYLOD214 (Computer Engineering, Second Year)

★ AIM

→ Define a class to represent a bank account which includes the following members as:

1. Data Members:
- Name of the depositors.
- Account Number.
- Withdrawal amount.
- Balance amount in the account.

2. Member Functions:
- To assign initial values.
- To deposit an amount.
- To withdraw an amount after checking the balance.
- To display name and balance.

→ Implement the program by using features of OOP in C++.

★ OBJECTIVE

→ To apply appropriate Object-Oriented features for various applications.

★ OUTCOME

→ Apply various object-oriented features for ~~prob~~ problem solving.
→ Design solution to the real-life problem.
→ Develop lifelong learning attitude towards problem solving.

Page No.
Date

* Theory

1. Introduction to OOPs.

→ OOP is defined as a programming paradigm that relies on the concept of classes and objects.

→ Object-oriented programming aims to implement real-world entities like inheritance, polymorphism etc. in programming.

→ The characteristics of an OOP language are Class, Object, Inheritance, Polymorphism, Abstraction and Encapsulation.

→ The main aim of OOP is to bind together the data and the functions the operate on them so that no other part of the code can access this data, except that function.

2. ~~Dis~~ Class as ADT

→ Abstract Data Types (ADT) are user-defined data structures and their operations.

→ Abstract Data Type is a type for objects whose behaviour is defined by a set of values and a set of operations.

→ When we create a new class, we can reuse some or all definitions of similar objects using inheritance leading to hierarchies of objects.

→ Definitions of several classes can be inherited into one class (base class to derived class) and this is known as multiple inheritance.

→ Classes also support operation overloading and polymorphism (using virtual keyword).

```
class Fruit
{
    public:
        string
        strin
        int
};
class Mango
{
    public:
        vo
        {
        }
};
```

3. Data Mem

→ The vari
using fun
or derive
are know

→ Member
to mani

→ Data me
properti

```
class C
{
    in
    f
```

Page No.
Date

Ex.
```
class Fruit
{
    public:
        string colour;
        string taste;
        int seeds;
};
class Mango : public Fruit      //inheritance
{
    public:
        void operator << (Mango obj)   //operator overloading
        {
            cout<<" This is a mango.";
        }
};
```

3. Data Members and Member Functions.

→ The variables which are declared in any class by using fundamental data types (like int, float, etc) or derived data types (like class, structure, pointer, etc) are known as data members.

→ Member functions are the functions in a class, used to manipulate data members.

→ Data members and member functions define the properties and behaviour of objects in a class.

Ex.
```
class Cost
{
    int mrp;
    float tax;
```

```
    float totcost;
    public:
        float calctax ()
        {
            tax = mrp * 0.05;
            return tax;
        }
        void getmrp()
        {
            cout << "Enter MRP : ";
            cin >> mrp;
        }
        void calctot()
        {
            totcost = mrp + tax;
            cout << "Total cost is :" << totcost;
        }
};
```

4. Object Visibility Modes.

→ When a base class is derived by a derived class with the help of inheritance, the accessibility of base class membess by the derive class is controlled by visibility modes.

→ The visibility mode (private, public or protected) in the definition of the derived class specifies whether the features (or membes) of t base class are privately derived, publicly derive or protected derived.

→ If a base class is publicly inherited, its public members remain public and protected members remain protected in the derived class.

→ If a base class is protected inherited, its members become protected in the derived class.

→ If a base class is privately inherited, its members become private in the derived class.

Ex:

```
class Machines
{   public:
    int instock;
    string model;
    int warranty;
};
class Phones: public Machines
{
    public:
        void phonedisplay()
        {
            cout << "This is a phone.";
        }
};
class Laptops: protected Machines
{
    public:
        void laptopdisplay()
        {
            cout << "This is a laptop.";
        }
};
```

```
class Printers: private Machines
{
    public:
        void printersdisplay()
        {
            cout<<"This is a printer.";
        }
};
```

Conclusion

→ We have learned to create classes with data members and member functions.

→ We have learned to use access specifiers and how to define member functions outside class using scope resolution operator (::).

→ We have learned to create and use friend function in class and inline functions.

022 Project Based Learning - I : Assignment - 2

Page No.
Date

Name: Parinitha M. Samaga
Roll Number: SYCOD214 (Computer Engineering, Second Year)

★ Aim

→ Write a program using C++ to create a student database system containing the following information:

- Name.
- Roll number.
- Class.
- Division.
- Date of Birth.
- Blood group.
- Contact, address.
- Telephone number.

→ Use class, object, inline function.

→ Use static variables and static functions to maintain count of the number of students.

→ Use constructor and destructor.

★ Objectives

→ To ~~explain~~ explore the principles of Object Oriented Programming (OOP).

→ To use to object-oriented paradigm in program design.

→ To lay a ~~func~~ foundation for advanced programming.

→ Provide programming insight using OOP constructs.

★ Outcomes

→ Analyse the strengths of object-oriented programming.

→ Apply the different object-oriented concepts to build the real time software scenarios.

→ Design object-oriented solutions for small systems involving multiple objects.

→ Develop applications using advanced object-oriented

Page No.
Date

programming concepts.

* Theory:

1. Introduction.

→ This program makes use of class, object, constr
destructor, static variable and static function.
→ The program collects student information and
displays their details at the end.

2. Constructor and its Types.

→ Constructor is a special method that is invoked
automatically at the time of object creation.
→ It is used to initialise the data members of ne
objects generally.
→ The constructor has the same name as the class
→ Constructor is invoked at the time of object
creation.
→ Constructors do not have a return type.
→ They are defined inside the public section of
the class. Definition of constructors in the priv
section would not allow creation of objects to
the class.
→ Types:

1) Default Constructor.

- A Default constructor is a constructor with no
arguments (or parameters) in the definition.
- These constructors are generally used to initia
data members with real values.
- Syntax Syntax:

```
class Class_name
{
    public:
        Class_name
        {}
};
```

2) Parameterised Constructor.

- Parameterised constructor contains parameters in the constructor definition and declaration.
- These parameters can then be used to assign values to data members.
- Syntax:

```
class Class_name
{
    public:
        Class_name(data_type variable)
        {
        }
};
```

3) Copy Constructor.

- A copy constructor is a member function that initialises an object using another object, which is already initialised, of the same class.
- It helps to copy data from one object to another.
- Syntax:

```
class Class_name
{
    public:
        Class_name (Class_name &object)
```

```
        {.....
        }
};
```

Ex.

```
class student
{
    string name;
    int classno.;
    char division;
public:
    student ()          // Default constructor
    {
        classno=5;
        division='A';
    }
    student (int cl, char div)
    {                   //Parameterised constr
        classno = cl;
        division = div;
    }
    student (student &obj) // Copy constructor
    {
        classno= obj.classno;
        division= obj.division;
    }
};
int main()
{
    student obj1, obj2(6,'B'), obj3;
    obj3 = obj2;
```

3. Destructor with syntax:

→ Destructor is an instance member function which is invoked automatically whenever an object is going to be destroyed.

→ The destructor is the only way to destroy an object and cannot be overloaded.

→ Destructor has the same name as their class name, preceded by a tilde (~) symbol.

→ ~~Destructor~~ Destructor does not require parameters and has no return value/type.

→ Syntax:

```
class Class_name
{
    public:
        ~Class_name()
        {}
};
class Number
{
    int numb;
    public:
        ~Number()      // Destructor
        {
            cout<<"Inside destructor.";
        }
};
```

Page No.
Date

4. Static Variable and Static Membes Function with Syntax:

→ When a variable is declared as static, space for it gets allocated for the lifetime of the program.

→ Static variables are initialised only once. Their default value is zero.

→ Static functions can manipulate only static variables.

→ Even if the function is called multiple times, space for the static variable is allocated only once and the value of the static variable in the previous call gets carried through the next function call.

→ Static functions can be called directly using class name.

→ Syntax:

```
class Class_name
{
    static data_type variable_name = value;
    public:
        static return_type function_name
        {
            .....
        }
};
```

Ex.

```
class Numbes
{
    static int numb = 0;
    public:
        static void numbincrement()
        { ++numb; }
};
```

Page No.
Date

5. This Pointer

→ 'this' pointer stores the address of the class instance, which is called from the member function that enables functions to access the correct object data members.

→ 'this' pointer can be accessed from every class function, including the constructor.

→ Static member functions don't have a 'this' pointer.

→ It can be used to pass current object as a parameter to another method.

→ It can be used to refer current class instance variable.

→ It can be used to declare indexers.

Ex.
```
class Number
{
    int integer;
    float rational;
    public:
        Number (int integer, float rational)
        {
            this->integer = integer;
            this->rational = rational;
        }
};
```

6. Inline Code

→ For small, commonly-used functions, the overhead of the function call is often a lot more than the time needed to actually execute the function's code. This overhead occurs for small functions because execution

time of small function is less than the switching time.

→ Inline functions are used to reduce to function call overhead.

→ When the inline function is called, whole code of the inline function gets inserted or substituted at the point of inline function call.

→ This substitution is performed by the compiler at compile time.

→ Inline function is a request to the compiler.

→ Syntax:

```
inline return_type function_name(parameters)
{
.....
}
```

Ex.
```
inline void display(int numb, char alphabet)
{
    cout<<numb<<" "<<alphabet<<endl;
}
```

★ Conclusion

→ We have learned to create and use, class, objects, static data members and static functions inside class.

→ We have learned to use constructors, destructor and inline function.

Project Based Learning - I : Assignment 3

Page No.
Date

Name: Parinitha M. Samaga
Roll Number: SYCOD214 (Computer Engineering, Second Year)

Aim

→ Consider we want to store the information of different vehicles. Create a class named Vehicle with two data members named mileage and price. Creates its two subclasses:

- Car with data members to store ownership cost, warranty (by years), seating capacity and fuel type (diesel or petrol).
- Bike with data members to store the number of cylinders, number of gears, cooling type (air, liquid or oil), wheel type (alloys or spokes) and fuel tank size (in inches).

→ Make another two subclasses Audi and Ford of Car, each having a data member to store the model type.

→ Next, make two subclasses Bajaj and TVS, each having a data member to store the make-type.

→ Now, store and print the information of an Audi and a Ford car (i.e. model type, ownership cost, warranty, seating capacity, fuel type, mileage and price).

Objective

→ To develop critical thinking and problem-solving ability by exploring and proposing solutions to real life application.

→ To apply appropriate Object-Oriented features for various applications.

Date

★ Outcomes

→ Apply various object-oriented features for Problem solving.
→ Design solution to the real-life problem.
→ Develop lifelong learning attitude towards problem solving.

★ Theory:-

1. Introduction.

→ This program is based on inheritance of classes.
→ It makes use of hierarchical inheritance.

2. Inheritance.

→ The capability of a class to derive properties and characteristics from another class is called Inheritance.
→ Inheritance is a feature or a process in which new classes are created from the existing classes.
→ The new class created is called "derived class" or "child class" and the existing class is known as the "base class" ~~or~~ or "parent class".
→ The derived class inherits all the properties of the base class without changing the properties of the bass class, and may add new features to its own.

3 Derived and Base Class

→ A base class in an existing class from which the other classes are determined and properties are

inherited. It is also known as parent class or super class.

→ A derived class is a class that is constructed from a base class or an existing class. It has a tendency to acquire all the methods and properties of a base class. It is also known as child class or subclass.

Ex.
```
class Fruit                    // Base class
{   public:
    string name;
    string colour;
    int numberofseeds;
};
class Mango : public Fruit     // Derived class
{
    string taste;
};
```

4. Private, Protected and Public Inheritance.

→ Private members of the base class are inaccessible to the derived class.

→ Private inheritance makes the public and protected members of the base class private in the derived class.

→ Protected inheritance makes the public and protected members of the base class protected in the derived class.

→ Public inheritance makes public members of the base class public in the derived class and the protected members of the base class remain protected in the

Date

derived class:

Ex.
```
class Fruit
{
    public:
        string name;
        string colour;
        int seeds;
};
class Mango: private Fruit
{
    public:
        string taste;
};
class Grapes: protected Fruit
{
    public:
        int numberofgrapes;
};
class Apple: public Fruit
{
    public:
        string variety_type;
};
```

5. Types of Inheritance.

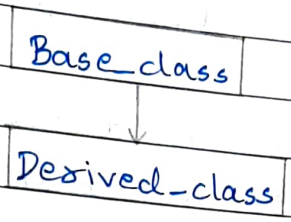
1) Single Inheritance

→ In single inheritance, a single class inherits the properties of a base class.

→ All the data members of the base class are

accessed by the derived class according to the visibility modes that is specified during the inheritance.

→



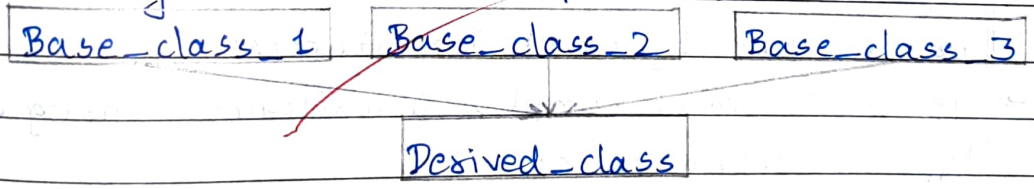
2) Multiple Inheritance

→ The inheritance in which a class can inherit or derive the characteristics of multiple classes, or a derived class can have over one base class, is known as multiple inheritance.

→ It specifies access specifiers separately for all the base classes at the time of inheritance.

→ The derived class can derive the joint features of all these classes and the data members of all the base classes are accessed by the derived or child class according to the access specifiers.

→

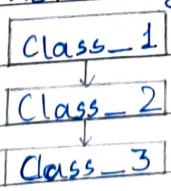


3) Multilevel Inheritance

→ The inheritance in which a class can be derived from another derived class is known as Multilevel inheritance.

→ The data members of each respective base class are accessed by their respective derived classes according to their specified visibility modes.

→



Page No.
Date

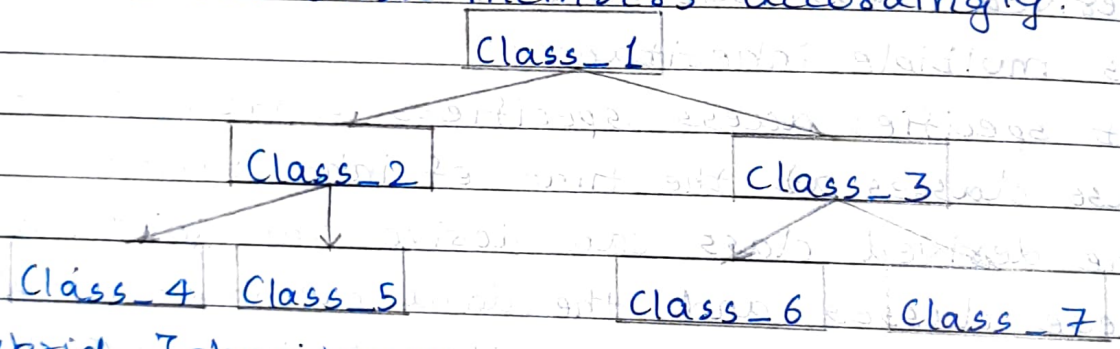
4) Hierarchical Inheritance

→ The inheritance in which a single base class inherits multiple derived classes is known as the hierarchical inheritance.

→ This inheritance has a tree-like structure as every class acts as a base class ~~for~~ for one or more child classes.

→ The visibility mode for each derived class is specified separately during the inheritance and it accesses the data members accordingly.

→

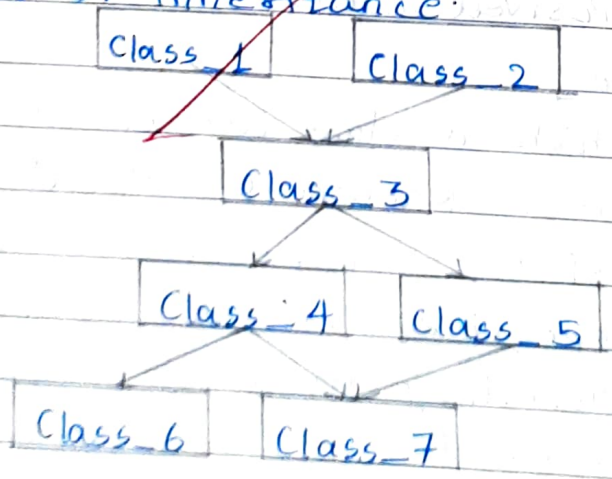


5) Hybrid Inheritance

→ Hybrid inheritance is the combination of two or over two types of inheritances.

→ It is the most complex inheritance among all the types of inheritance.

→



★ Conclusion

→ We have learned to use Hierarchical inheritance on classes.

→ We have learned to use friend functions inside classes.

Project Based Learning - I: Assignment 4

Page No.
Date

Name: Parinitha M. Samaga
Roll number: 5YCOD214 (Computer Engineering, Second Year)

★ Aim

→ Implement a class Complex which represents the Complex Number data type. Implement the following operations:
- A constructor (including a default constructor which creates the complex number 0+0i)
- Overloaded operator + to add two complex numbers.
- Overloaded operator * to multiply two complex numbers.
- Overloaded << and >> to print and read Complex Numbers. To do this, you will need to decide what you want your input and output format to look like.

★ Objective

→ To understand and implement operator overloading.

★ Outcome

→ Students will be able to use concepts of OOP to implement operator overloading.

★ Theory:

1. Operator Overloading

→ Operator overloading is an important concept in C++.
→ It is a type of polymorphism in which an operator is overloaded to give user defined meaning to it.
→ Overloaded operator is used to perform operation on user-defined data type.
→ For example, '+' operator can be overloaded to perform

addition on various data types, like for Integer, String (concatenation), etc.

→ Almost any operator can be overloaded in C++.

→ However, there are a few operators which cannot be overloaded.

→ Operators that are not overloaded are follows:

- Scope operator (::)
- Sizeof
- Member selector (.)
- Member pointer selector (*)
- Ternary operator (?:)

2. Operator Overloading Syntax:

→
```
Class_name operator + (Class_name c1)
{ ..... ;
  return result;
}
int main()
{
   Class_name o1, o2, result;
   ..... ;
   result = o1 + o2;
   ..... ;
}
```

3. Implementing Operator Overloading.

→ Operator overloading can be done by implementing a function which can be:

- Member Function.

- Non-member Function.
- Friend Function.

→ Operator overloading function can be a member function if the Left operand is an Object of that class, but if the Left operand is different, the Operator overloading function must be a non-member function.

→ Operator overloading function can be made friend function if it needs access to the private and protected members of class.

4. Restrictions on Operator Overloading:

→ Following are some restrictions to be kept in mind while implementing operator overloading:

- Precedence and Associativity of an operator cannot be changed.
- Arity (numbers of Operands) cannot be changed. Unary operator remains unary, binary remains binary, etc.
- No new operators can be created, only existing operators can be overloaded.
- Cannot redefine the meaning of a procedure. You cannot change how integers are added.

5. Constructor

→ Constructors are the special type of member function that initialises the object automatically when it is created.

→ Compiler identifies that the given member function is a constructor by its name and return type.

→ Constructor has same name as that of class and it does not have any return type.

6. Use of Constructor in C++.

→ Suppose you are working on 100's of objects and the default value of a data member is 0.

→ Initialising all objects manually will be very tedious.

→ Instead, you can define a constructor which initialises that data member to 0.

→ Then all you have to do is define object and constructor will initialise object automatically.

→ These types of situation arises frequently while handling array of objects.

→ Also, if you want to execute some codes immediately after object is created, you can place that code inside the body of constructor

7. Constructor Overloading.

→ Constructor can be overloaded in similar way as function overloading.

→ Overloaded constructors have same name (name of the class) but different number of argument passed.

→ Depending upon the number and type of argument passed, specific constructor is called.

→ Since, constructor are called when object is created.

→ Argument to the constructor also should be passed

while creating object.

→ Here is the ~~modification of ab~~

★ Conclusion:

→ Thus, we have successfully implemented a class Complex which represents the Complex Number data type.

→ We have learned to overload operators and use them.

2022 Project Based Learning - I : Assignment 5

Page No.
Date

Name: Pasinitha M. Samaga
Roll number: SYCOD214 (Computer Engineering, Second Year)

* Aim

→ Create a base class called 'SHAPE' having two data members of type double, member function get_data() to initialise base class data members, pure virtual member function display_area() to compute and display the area of the geometrical object.

→ Derive two specific classes 'TRIANGLE' and 'RECTANGLE' from the base class.

→ Using these three classes, design a program that will accept the dimension of a ~~trig~~ triangle / rectangle interactively and display the area.

→ Implement using C++.

* Objective

→ To develop critical thinking and problem-solving ability by exploring and proposing solutions to real life applications.

→ To apply appropriate Object-Oriented features for various applications.

* Outcomes

→ Identify the technical aspects of the chosen project with a comprehensive and systematic approach.

→ Apply various object-oriented features for problem solving.

→ Design solution to real-life problem.

→ Develop lifelong learning attitude towards problem solving.

Page No.
Date

★ Theory:

1. Introduction

→ This program uses classes, objects and inheritance.

→ It also makes use of pure virtual function and function ~~overring~~ overriding.

2. ~~What is~~ Polymorphism and its types

→ Polymorphism simply means ~~thew~~ more than one form.

→ Polymorphism means the same entity (function or operator) behaves differently in different scenarios.

→ Types:

1) Compile-time Polymorphism

→ This type of polymorphism is achieved by function overloading or operator overloading.

• Function Overloading

- When these are multiple functions with the same name but different parameters, then the functions are said to be overloaded. This is known as Function Overloading.

- Functions can be overloaded by changing the number of arguments or/and changing the type of arguments.

Ex.
```
int addition (int a, int b)
{
    return (a+b);
}
float addition (float a, float b)
{  return (a+b); }
```

• Operator Overloading

- In operator overloading, the operator is overloaded to provide special meaning to the user-defined data type.
- Operator overloading redefines most of the operators available in C++ and is used to perform operations on user-defined data types.

Ex:
```
Class Complex
{
    int real;
    int imaginary;
    public:
        Complex operator + (Complex c1)
        {
            Complex c2;
            c2.real = real + c1.real;
            c2.imaginary = imaginary + c1.imaginary;
            return c2;
        }
};
```

2) Runtime Polymorphism

→ It is also known as late binding and dynamic polymorphism.

→ It is achieved by Function overriding.

→ The function call is resolved at runtime in runtime polymorphism.

• Function Overriding

- Function overriding occurs when a derived class has a definition for one of the member functions of the base class.

- That base function is said to be overridden.

Ex.
```
class Fruit
{
    public:
    virtual void display()
        {
            cout<<"This is a fruit.";
        }
};
class Mango: public Fruit
{
    public:
        void display()
        {
            cout<<"This is a mango.";
        }
};
```

- Virtual Function

- A virtual function is a member function that is declared in the base class using the keyword 'virtual' and is redefined / overridden in the derived class.
- Virtual functions are dynamic in nature.

Ex.
```
class Fruit
{
    public:
        virtual void display ()=0;
};
class Mango: public Fruit
```

Page No.
Date

```
{
    public:
        void display()
        { cout<<"This is a mango."; }
};
```

3. Function Overriding with Syntax:

→ Function overriding is a concept by which we can define a function of the same name and the same function signature (parameters and their data types) in both the base class and derived class with a different function definition.

→ It redefines a function of the base class inside the derived class, which overrides the base class function.

→ Function overriding is an implementation of the runtime polymorphism.

→ So, it overrides the function ~~of~~ at run-time of the program.

→ Syntax:

```
class Base_class
{
  ..........
  public:
    return_type function_name()
    {}
};
class Derived_class: public Base_class
{
  ..........
  public:
    return_type function_name()  //overridden function
```

Page No.
Date

```
{
.....
}
};
```

→ As function overriding redefines the function of the base class in the derived class, the return-type, function name and function parameters should be the same to achieve function overriding.

→ Function overriding can be useful when a child class requires its own version of a functionality.

4. ~~Explain~~ Virtual Functions.

→ A virtual function is a member function which is declared within a base class and is redefined (overridden) by a derived class.

→ Virtual functions ensure that the correct function is called for an object, regardless of the type of reference (or pointer) used for function call.

→ Functions are declared with a 'virtual' keyword in base class.

→ Resolutions of the function call is done at runtime

5. Role of Pure Virtual Functions

→ A pure virtual function makes it so that the base class can not be instantiated, and the derive classes are forced to define these functions before they can be instantiated.

→ This helps ensure that the derived classes do not forget to redefine functions that the base class was expecting them to.

Page No.
Date

Importance of Abstract Class:

→ Abstract classes are used to express broad concepts from which more concrete classes can be derived.

→ An abstract class type object cannot be created.

→ At least one pure virtual member feature must be declared while creating an abstract class.

→ The pure specifier (=0) syntax is used to declare a virtual function.

→ The abstract class's descendants must define the pure virtual function. Otherwise, the subclass would become an abstract class on its own.

→ An abstract class is mostly used to provide a base for subclasses to extend and implement the abstract methods and override or use the implemented methods in the abstract class.

Conclusion

→ We have learned how to create pure virtual functions.

→ We have learned how to override functions in derived classes.