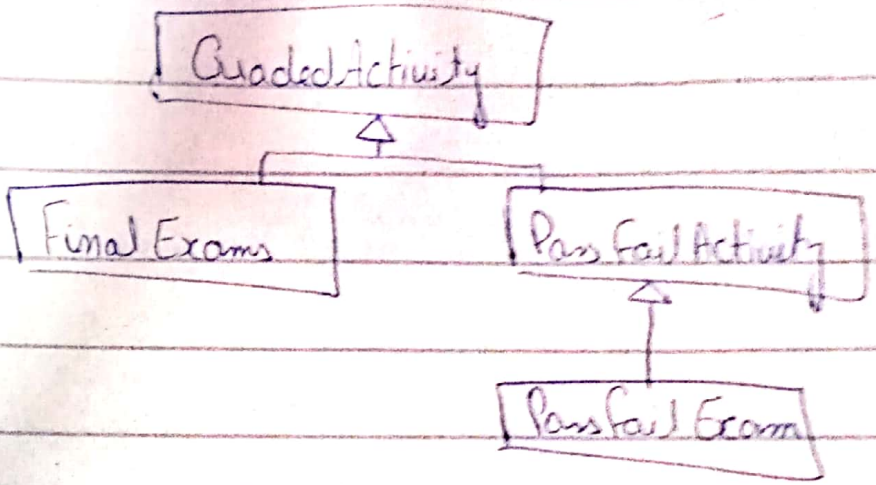
Topics :

15.6

- Polymorphism.
- ↳ Require Pointer or Refrence
- Base Class Pointer
- Base* Array with Base + Derived obj elements
- "is a" Relationship doesn't work Reverse
- Redefining vs OverRiding
- Virtual Destructor for Polymorphic ad Pointer obj
- override ad final Keyword

23

- Software designers mostly use Family Tree like
- In Hierarchies,
  General Classes are on top of the tree
  Specialized Classes are on Bottom of the tree



- Polymorphism:
  It allows an object "refrence variable" or "pointer"
  ↳ to call objects of different types
  ↳ to call correct member function
  ↳ Depending upon the type of object being refrenced.

Exaple

void displayGrade (const GradedActivity &activity)

→ GradedActivity test1(88.0)
~~Final~~ FinalExam test2(100,25)

displayGrade (test1); // Base object
displayGrade (test2); // Derived object

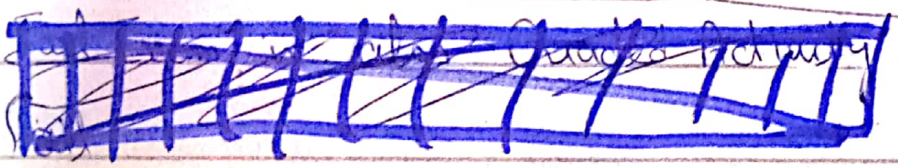
work for

Why it can call both object.
when it data type is Base Class?

As there is a "Relation" b/w BC and DC

Thus

(DC is also BC) or | DC $\xrightarrow{is\ also}$ BC



→ Final Exam is not only Final Exam object, it is (Final Exam) is also Grade Activity object

→ Final Exam is a Graded Activity

Because of this Relationship we can also pass the object of Final Exam in display Grade function.

Polymorphism Requires Reference or P

Polymorphism behaviour is not possible by value

↳ So that's why they [redacted] Requires Refrence or Pointer.

So that's why when we call displayGrade( ) we pass an object (BC, DC) by Refrence.

So In Case of Virtual function It will be call according to its [redacted] class level.

Base Class Pointer:

Pointer of BC, may be assigned the address of a derived Class object.

```
GradedActivity *exam = new PassFailExa-(100,25,75);
↳
cout << exam -> getScore() << endl;
cout << exam -> getLetterGrade() << endl;
```

A way for Base Array of having Different Derived Class Object :-

```
GradedActivity *tests[NUM_TESTS] =
    { new GradedActivity(88.0),
      new PassFailExam(100,25,70.0),
      new GradeActivity(67.0),
      new PassFailExam(50,12,60.0)
    };
// to Display Grade
for (int count=0; count< NUM_TEST; count++) {
    cout << "Test " << (count+1) << ":\n";
    displayGrade(test[count]);
    cout << endl;
}
```

Base Class Pointers and Refrence know only About Base Class Mentees.

A base class pointer can store address of any class that derives from the base class.

But there are some limitations with the Base Pointe

(Base Class Pointer)
It can only call the functions written in it (Base Class)

If there is some member function in DC, BC pointer cannot call it, to call that function make a Virtual function in Base Class ad overridden in Derived Class.

Thus

The "is-a" Relationship doesnot Wolks Reverse:

HoneyBee is an Insect

Insect is an HoneyBee

Why?

Cause not all insects are HoneyBee

likewise not all insects obj are HoneyBee Obj

Thus not all BaseClass obj are DerivedClass Obj

But All Derived Class obj are Base Class Obj

As All HoneyBees are Insects.

Let's come to Point:

You cannot assign address of

Base Class obj ⟶ Derived Class Pointer.

But with the help of Static cast we can do this

```
GradeActivity * gaPointer = new GradeActivity (88.0);
// Error this will not work
FinalExam * fePointer = gaPointer;
```

But we with the help of Static Cast we can do this

```
FinalExam *fePointer = static_cast<FinalExam *> (gaPointer);
// This will work with limitations.
```

What error will cause by Static_cast

```
FePointer -> getScore() << endl;
FePointer -> getLetterGrade() << endl;
FePointer -> getPointsEach() << endl;
```

This will get Error at runtime.
Cause this object doesnot have gets Point Each (that it is PassFailExam class in GradeActivity.

Redefining vs OverRiding:

When a Class Redefine a Virtual Function it is called OverRidden

OverRidden are dynamically bound.
Redefined are statically bound.

Virtual Destructor:

A good Programming Practice,
Any Class that has a Virtual member Function, should also have virtual Destructor.

What if the class doesn't require destructor then?

It should have a virtual destructor with no Statement.

As it have no Statement, All the derived class will have automatically the virtual destructor,

It will be overridden when a Derived class have ~~const~~ defination of its Destructor.

For Polymorphism:

make sure Base Class Must have Virtual Destructors.

As if we assign address of DC to BC, when BC destroys DC destructor doesn't execute, This can cause error of some functional issues or memory leakage.

Thus Just add virtual with Base Destructor So ~~both~~ DC destructor can be executed and BC destructor (due to delete obj;)

Before Changing:

Code:

```
Base * BasePointer = new DerivedObj
delete * BasePointer
```

Result:

Base Class Constructor called.
Derived Class Constructor called.
Base Class Destructor called

Changing:

```
virtual ~Base() {
    .....
}
```

After Changing:

Base Class Constructor called
Derived Class Constructor Called
Derived Class Destructor Called
Base Class Destructor Called.

C++ 11's override and Final Key words.

Due to change of Parameters but same Name even with the use of Virtual Function, It will not be overridden.

to Prevent this we write overridden at the end.

Sol:

B.C virtual void Function A (int arg) const

D.C virtual void Function A (long arg) const override

Override Keyword is added in heading.

Preventing a Member Function From Being Overridden:

To make a virtual function to not further overridden ~~we~~ we use down in class hierarchy. we use Final keyword.

```
Virtual void message() const final;
```

→ If the derived class attempts to override a final member functi., the compiler generate an error.

Topics:

15-6

- Polymorphism.
- ↳ Require Pointer or Refrence
- Base Class Pointer
- Base* Array with Base + Derived obj elements
- "is a" Relationship doesn't work Reverse
- Redefining vs OverRiding
- Virtual Destructor for Polymorphism and Pointer obj
- override and final Keyword

Polymorphism & Virtual Member Function.

- Virtual Member Function.
  - Function in Base Class
  - That expects to be redefine
    - ↳ in Derived Class.
  - This functions are defined with **virtual** :

~~virtual~~

```
virtual void Y() { ... }
```

- Supports Dynamic Binding
  - ↳ Functions bounds at run time to function that they call
- Without Virtual keyword, C++ uses Static Binding (Compile time).

Exaple (15-9)

```
void displayGrade ( const GradeActivity &activity)
{
  cout << setprecision(1) << fixed;
  cout << "The activity's numeric score is"
       << activity.getScore() << endl;
  cout << "The activity's letter grade is"
       << activity.getLetterGrade() << endl;
}
```

This function have Parameter GradeActivity reference variable

↳ So it can refrence any object that is derived from Grade Activity

That's mean we can pass any object of
- Grade Activity
- Final Exam
- Pass Fail Exam
- or any other object that is derived from Grad Activity.

⇒ But a Static Bounding Problem occur.

```
#include "PassFailActivity.h"
void displayGrade(const GradeActivity &);
int main()
{
  // Create a PassFailActivity object.
  // Minimum Passing Score is 70
  PassFailActivity test(70);

  // Set the Score to 72
  test.setScore(72);

  // Display the object's grade data.
  // The letter grade should be 'P'
  // What will be displayed?
  displayGrade(test);
  return 0;
}
void displayGrade(const GradeActivity &activity)
{
  // Code written before.
}
```

Output:

The activities num grade is 72.0

" " is C

⇒ As you can see from the example output the getLetterGrade member function returned 'C' instead of 'P'. This is because the

⇒ Grade Activity class's getLetterGrade functio was executed ~~instead of~~

instead of

⇒ Pass Fail Activity class's version of the functio.

Static Binding:

As it display 'C' instead of 'P'

- because the call to getLetterGrade is statically bound (at compile time) with Graded Activity class's version of the function.

To Solve this Problem:

It can be solve by making the function virtual in Base class.

~~So that~~

Virtual Functions:

- A virtual function is dynamically bound to calls at runtime

- At runtime, C++
  - C++ determines the type of object making the call
  - and Binds the function to the appropriate version of the function

- Making a function Virtual
  Place Virtual Key word before return type in Base Class Declaration

```
virtual char getLetterGrade() const;
```

- The compiler will not bind the function to call
  Instead
  The program will bind them at runtime

```
Class Graded Activity {
    protected:
        double Score;
    public:
        GradedScore() { Score = 0.0; }
        GradedScore(double s) { Score = s; }
        void setScore(double s) { score = s; }
        double getScore() const { return score; }
virtual char getLetterGrade() const;
}
```

- Now, this function is now virtual.
- The function also becomes virtual in all derived classes automatically!

When we compile the Program

" " is 72.0

is P

Note:

This type of behavior is known as polymorphism. The term polymorphism means the ability to take many forms.

True Meaning of Polymorphism.

Demonstrate Polymorphism.

In this we pass object of Graded Activity and Pass Fail Exam class to the Display Grade function

```
void displayGrade(const GradeActivity &);
int main()
{
  GradeActivity test1(88.0);
  // Total Question 100, Missed 25, Passing Score 70
  PassFailExams test2(100, 25, 70.0);

  cout << "Test 1:\n";
  displayGrade(test1);
  cout << "Test 2:\n";
  displayGrade(test2);
  return 0;
}
```

Output:

Test 1:

The activity ..... is 88.0

" " " is B

Test 2:

" " " is 75.0

" " " is P

Polymorphism requires < References / Pointers.

- Polymorphism behavior is only possible when an object is referenced by a reference variable or a pointer as Demonstrated in the DisplayGrade function.

Base Class Pointers:

- Can define a pointer to a base Class object
- Can assign it the address of a derived class object

Address کا D.C's object کو B.C's Pointer میں۔
کر سکتے ہیں - تب ہی Polymorphism start ہوگا

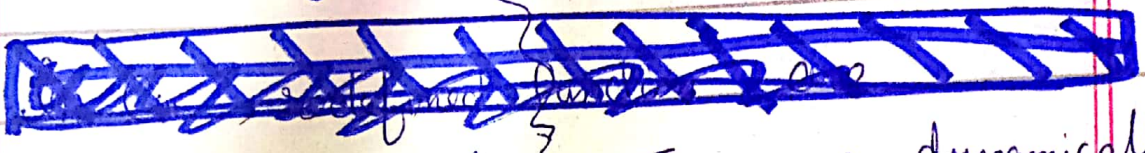
```
GradedActivity *exam = new PassFailActivity (100, 25, 200);

cout << exam -> getScore() << endl;
cout << exam -> getLetterGrade() << endl;
```

(Imp) • Base Class Pointers only knows about members of the base Class

→ So you cannot use a base Class pointer to call a derived Class function

- Redefined function in derived class will be ignored unless base class declares the function ~~base~~ virtual

Redefining vs Overiding:



They are statically bound

They are dynamically bound

non-virtual function is redefined

virtual function is overridden

Virtual Destructor:

- Make the Destructor of class virtual
  - ↳ If the class could ever become a base class

- Other wise, the compiler will perform Static binding on the destructor if the class ever is derived from.

→ Example in 15-14 o

15.7 Abstract Base Classes and Pure Virtual Function

Pure Virtual Function:

A virtual function that must be overridden in a derived class that has objects.

virtual void Y() = 0; = 0 indicates pure virtual function

Abstract Base Class

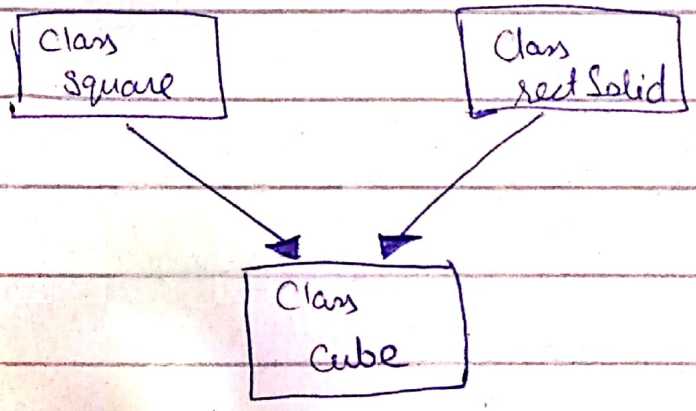
- That have at least one Pure Virtual Function
- Can't have object
- Made for Derive classes
  - whose object have to be made
- Its Pointer can be made

15.8 Multiple Inheritence

⇒ A Derive class can have more than one base class.

- Each class can have its own access specification (public, protected, private) in derived class's definition

- class cube : public square, public rectSolid;



- To Pass Argument, we can Pass to both: base classes constructor

~~class cube~~

cube :: cube (int side) : square (side),
rectSolid (side, side, side);

→ This thing can be only done in Constructor.

- Base class constructor
  - They are called in order given in Class Declaration
  - not in order used in class constructor.

- **Problem:**
  what if base classes have member variables / function with the same name?

Solution:

- Derived classes redefines the multiply-defined function
- Derived class invokes member function in a particular base class using scope resolution operator ::

↳ Compile errors occur if derived class uses base class function without one of these solutions

Note:

(1) If the base class has no default constructor, then the derived class must have a constructor that calls one of the base class constructors.

(2) If the base class access specification is left out of a declaration,
↳ The default Access Specification is Private

For Example:

→ Class Test : Grade

Similar too → Class Test : private Grade

(3) The BC constructor called before the DC constructor
The DC destructors called ~~after~~ before the BC destructor,
In Inheritence.

(4) Passing Arguments to Base Class Constructor:

Q ① If BC constructor takes arguments?

② what if there are more than one constructor in BC

Ans: Let the DC constructor pass arguments to the base class constructor.

e.g. Cube() : Rectangle()
↑ DC constructor ↑ BC constructor.

This is write only in defination of a Constructor not in Parameter of the Constructor

↓

- ClassName :: ClassName (ParametersList) : BaseClassName (Argument List)

-> Cube (double w, double len, double h) : Rectagle (w, h)

↳ orders of Arguments doesn't mater.

- When Rectagle executed the Cube will execute.

- **What value we can Pass on Base constructor:**
  - Derived Class Constructor parameter
  - Literal values
  - Global values that are accessible to the file containing the derived class constructor defination
  - Expression making any of these values.

(in Polymorphism)

- Sometimes it is useful to overload a base class function with a function of same name in the derived function.

- BaseClassName :: FunctionName (Argument List)

  » we can used this in Derived Class to call Base Class function that is overridden in Derived Class.