

12-15 days

Leave submission

Test

Home, media library

LMS

posting assignment reward work

Classmate participation feedback

Class experience

in-class
work

Directed
research

posted assignment

posting

public forum

Class TA; public student

category

Class Teacher

class students

TA

Teacher

Students

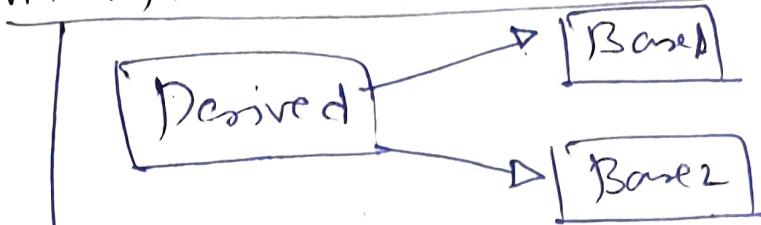
TA is a student TA is a Teacher

2nd

multiple instances

243 Class Managing Director, public Manager,
public Director;

multiple inheritance scenario



Have two or more Base class.

public and private inheritance may be mixed

Class Base1:

Class Base2:

Class derived: public Base1, public
Base2.

private inheritance: Is implemented as i.e.
is ~~of~~ Base classes if we are doing private
inheritance from it means Base classes already
implemented Derived class.

* Derived is ~~a~~ Base1, Base2;

Data members

→ Derived class inherits all Data members
of all Base classes

→ Derived class may add data members
of its own

member functions:

* Derived class inherits all member fn of
all Base classes.

(244)

* Derived class may override a member from any base class by redefining it with the same signature.

* Derived class may overload a member from any base class by redefining it with the same name but diff. signature.

Access Specification

* Derived class can not access private members of any base class.

* Derived class can access protected members of any base class.

Construction-Destruction

① A constructor of the derived class must first call all constructors of the base classes to construct the base class instances of the derived class. —
Base class constructors are called in listing order.

② the destructor of the derived class must call the destructor of the base classes to destroy the base class instances of the derived class.

(25) Data members and Object layout

* Derived LSA BaseL, Base2

Data members

- (i) Derived class inherits all Data members of all Base classes
- (ii) Derived class may add data members of its own.

Object layout

- * Derived class layout contains instances of each Base class @
- * Further Derived class layout will have data members of its own.
- * C++ does not guarantee the relative position of the Base class instances and Derived class members.

Ex: class Base1 { protected:

int i;

int data;

public:

};
class Base2 { protected:

int j;

int data

public:

};

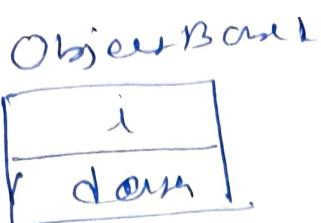
class Derived : public Base1, public
Base2 {

(246)

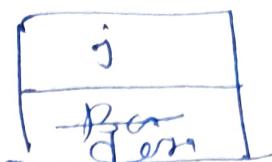
int k;

public:

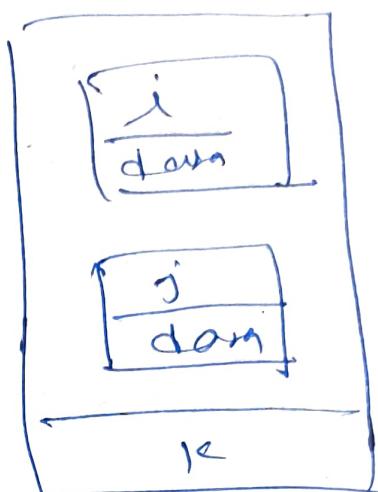
~j;



ObjBase2

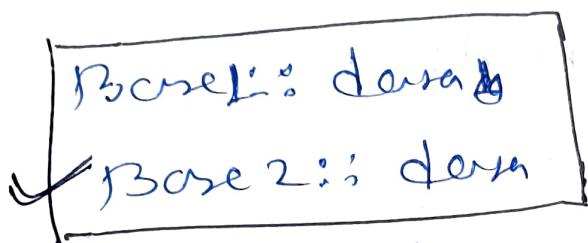


Obj Derived.

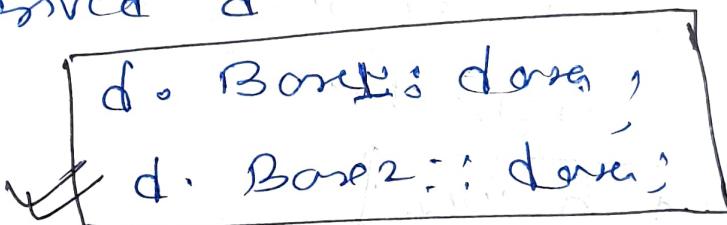


✓

- * Both Base class
have no data members
with some name so
refer them as



Derived d



multiple inheritance member fn overriding

and overloads

- * derived class does not inherit friend
member fr of any Base class
- * derived class does not inherit the static
member fr of any Base class
- * derived class does not inherit the friend
fr of any Base class.

Ex-1

(24)

Class Base & protected :

int i;

int data;

public: Base1(int a, int b) : i(a),
data(b);

void f(int) { cout << "Base::f(" << i << "

void g()

{ cout << "Base::g()" << endl; }

}

Class Base2 & protected :

int j;

int dealer;

public: Base2(int a, int b) : j(a),
dealer(b);

void h(int)

{ cout << "Base2::h(" << i << ")" << endl; }

}

Class Derived : public Base1, public Base2

{ int k;

public: Derived(int a, int y, int z,
int v, int w);

void f(int) // override.

{ cout << "Derived::f(" << i << ")" << endl; }

}

(24)

void h(string) // Overload.

{ cout << " Derived :: h(string)" << endl;

}

void e(char)

{ cout << " Derived :: e(char)" <<
endl; }

}

Derived d(1, 2, 3, 4, 5)

c.h(5) // Derived :: func

c.ges → // Base :: ges

c.h("ppd"); // Derived :: h(string)

c.e('a') // Derived :: e(char)

~~multiple inheritance~~: (can be used only for non polymorphic types)
member fn using for Name resolution.

Ambiguous calls

class Base1

{ int i, data;

public:

Base1(int a, int b): i(a), data(b);
{ }

void f(int)

{ cout << "Base1:: f(int)" << endl;

}

void ges

{ cout << "Base1:: ges" << endl;

}

g;

Class Base2

(249)

f. ϕ int j;
int data;

public: ~~Base2()~~
Base2(int a, int b): j(a), data
(b) {}

void f(int)

{ cout << "Base2:: f(int)" << endl;

}

void g(int)

{ cout << "Base2:: g(int)" <<
endl;

}

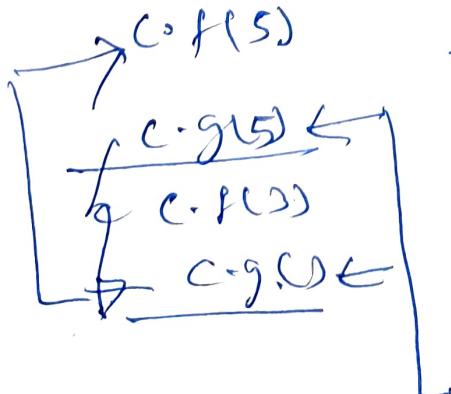
}

class Derived: public Base1, public
Base2

{ public: Derived(int x, int y, int z,
int v, int w)

}

Derived c(1, 2, 3, 4, 5)



Question which copy of
f will be included in Deri-
ved class

Here overload
ambiguous resoln doesn't
work for you

(250) ~~Note~~: Overload resolution does not work
between diff namespaces. It works b/w same
name space.

How to resolve the previous issue

We have to restrict single copy of f or
g from Base1 or Base2 in Derived class
like:

class Derived: public Base1, public
Base2

f.
public: Derived (int x, int y, int
z, int v, int w);

using Base1::f // Inherit f from
Base1
using Base2::g // Inherit g from
Base2

g.
Derived d (1, 2, 3, 4, 5)

c. f(5); // Base1::f(m)

c. g(5); // Base2::g(m)

c. Base1::f(3); // Base1::f(m)

c. Base2::g() // Base2::g()

* If two base class has fn with
the same name ~~it respects~~ irrespective of
signature then the derived class must
use using to access those fn.
key

Constructor - destructor

(25)

- * Derived class inherits all constructors and destructors of Base classes (but in a diff scenario's)
- * Derived class cannot override or overload a constructor or a destructor of any Base class.

Ex:

Class Base1 & protected : int i, ~~int deta~~,

public: Base1(int a, int b) :

i(a), data(b) { cout << "Base1:" <

Base1("i<< endl"); }

~Base1()

{ cout << "Base1:" <> ~Base1() << endl;

}

};

Class Base2 & protected : int j, data;

public:

Base2(int a=0, int b=0) : j(a),

data(b)

{ cout << "Base2:" <> Base2() << Base2("j<< endl");

}

~Base2()

{ cout << "Base2:" <> ~Base2() <<

endl;

}

};

(25)

class Derived: public Base1, public Base2

 int k;

public:

 Derived(int x, int y, int z);

 Base1(x, y), k(z)

 cout << "Derived:: Derived() \n";
 endl;

y

// Base1 ex Base1::Base1
// explicit, Base2::Base2
// default

~Derived()

 cout << "Derived :: ~Derived() \n";
 endl;

y

Derived

derived d(5, 3, 2);

Obj: Base1::Base1()

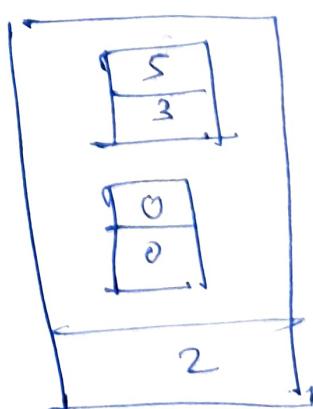
Base2::Base2()

Derived::Derived()

Derived :: ~Derived()

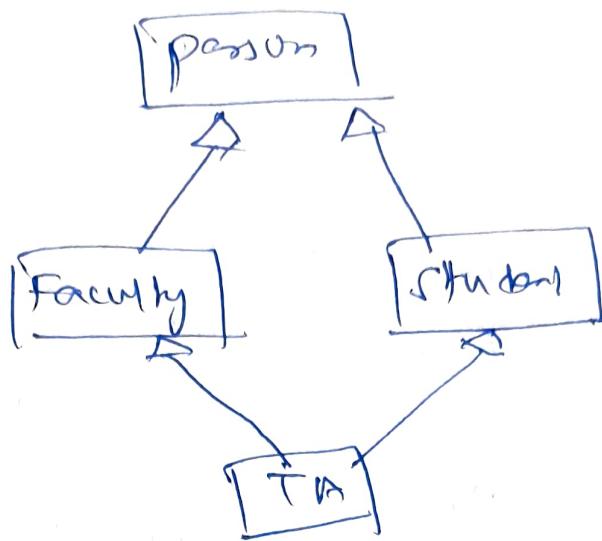
Base2 :: ~Base2()

Base1 :: ~Base1()



multiple inheritance - diamond problem

(253)



class person;

class student : public person;

class faculty : public person;

class TA : public student, public faculty;

Ex:

class Person

{ public: Person(int); }

class "Person": Person(magic)

end;

}

};

class Faculty : public Person

{ public: Faculty(int); : Person(); }

class "Faculty": Person(); <<

end;

}

};

class Student : public Person

{}

(254) public: Student(int); Person()

{ course "Student"; Student(m); }
endl;

}

};

class TA: public Faculty, public
Student {

public: TA(int); Student(a),
Faculty(m);

course "TA"; TA(m); } } endl;

}

};

TA ta(30);

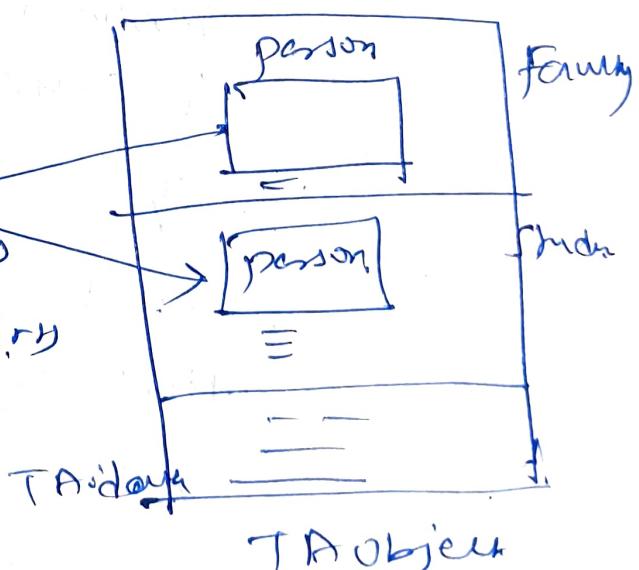
obj: Person(); Person(m)

Faculty(); Faculty(m)

Person(); Person(m)

Student(); Student(m)

TA(); TA(m)



Two instances of base class Object

~~multiple instances of the root class (or diamond class) into the derived class Object. This is known as Diamond problem.~~

How to resolve this

(25)

↳ Using virtual inheritance

class Person {

public: Person(int n)

{ cout << "Person:: Person(" <<

endl;

}
}; Default constructor for virtual
Person // instance

{ cout << "Person:: Person(" <<

endl;

}

};
↳ virtual Base class

class Faculty : virtual public Person

public: Faculty(int n) : Person(n)

{ cout << "Faculty:: Faculty(" <<

endl;

}

class Student : virtual public Person

public: Student(int n) : Person(n)

{ cout << "Student:: Student(" <<

endl;

(256) class TA: public Faculty, public Student
 i. public TA (int): student,
 Faculty (n) & course "TA"; TAC
 (int) "Lend";
 ?
 ?;
 TA for (30)

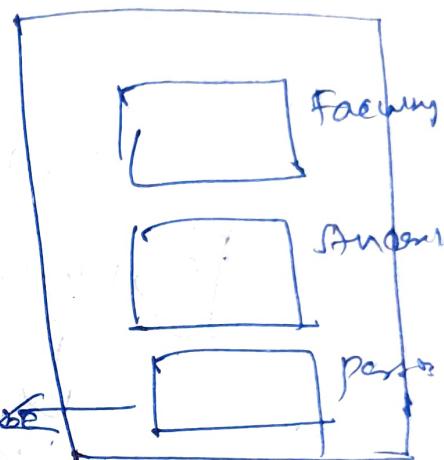
* one common person
 instance for both
 Fac and Stud

Q1: Person:: Person

Faculty:: Faculty (n)

Student:: Student (n)

TA:: TA (n)



this will be
 constructed through TA
 default constructor of Base class

~~if inheritance is virtual then it knows that
 some other class might be constructed
 so derived class' base could be multiple
 Base class so Faculty will not construct
 its Base class.~~

~~Virtual Base Class: they will not
 directly construct Base class part of
 Base class if it will be common in
 future derived class.~~

(25)

What if we want to pass the parameter to constructor.

Then we have to explicitly call the constructor or use constructor class or diamond class or diamond LTM with parameter like.

classes TA : public Faculty, public
student &

public:

TA(int n); student(n), Faculty(n)

Person(n) & now << "TA:: facing"
<cond>

↳ giving parameterized constructor
of person

TA t1(30)

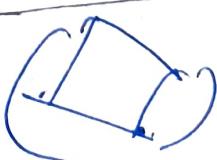
obj: person:: person(n)

Faculty:: Faculty(n)

student:: student(n)

TA:: TA(n)

Ambiguity in multiple inheritance with
diamond structure



(25) class Person
L. public: Person (int n)
L. {
L. L. cout << "Person:: Person(" n) " <<
L. L. endl;
L. }
L. Person()
L. {
L. L. cout << "Person:: Person() " <<
L. L. endl;
L. }
L. virtual ~Person();
L. virtual void teach();
Y;
class Faculty : virtual public Person
L. public: Faculty (int n) : Person(n)
L. {
L. L. cout << "Faculty:: Faculty(" n) " <<
L. L. endl;
L. }
L. virtual void teach();
Y;
class Student : ~~public~~ virtual public
Person
L. public: Student (int n) : Person(n)
L. {
L. L. cout << "Student:: Student(" n) " <<
L. L. endl;
L. }
Y;

class TA: public Faculty, public student
 {
 public:
 TA(m2): student(m), Faculty(m)

{ cout << "TA :: TA(m)" << endl;
 }
}

 }
 // TA fails // Mo answered. since
 // in the absence of TA:: teach,
 // which of teach should be inherited.
 // This cannot be resolved by using
 // like nonpolymorphic type. It can be
 // resolved by overriding teach(b) teach(s)
 // in TA like

class TA: public Faculty, public student

{ public:

 TA(m2): student(m), Faculty(m)

{ cout << "TA :: TA(m)" << endl;

}

 virtual void teach()

};

- Now TA ta(30) // works

* exercise from slides

(260)

Exception Handling

What are exceptions?

Conditions that arise.

- * Infrequently and unexpectedly
- * generally being a program error
- * Require a considered programmatic response
- * Runtime Anomalies - Yes, but not necessary

leading to

- * crashing the program
- * may put the entire system down.
- * Defensive technique
 - * crashing exception versus Tangled Design and Code

Exception causes

- * unexpected system state
 - * Exhaustion of Resources
 - * Low free store mem.
 - * Low disk space.
 - * putting to fun space.
- * external events
 - * control + C
 - * socket event
- * logical errors

- * pop from empty stack. (26)
- * Resuse var like man & ready write.

* Runtime error

- * Arithmetic overflow/underflow
- * Out of Range

* Undefined opn

- * Division by zero

Exception handling

- * exception handling is a mechanism that separates the detection and handling of circumstances Exception Flow from Normal Flow
- * current state saved in a predefined box
- * execution switched to a pre-defined handler

Types of exception

- * Asynchronous exception
 - * Exception that come unexpectedly
 - * Example - an interrupt in a program
 - * Take control away from the executing thread context to a context that is diff from the that which caused the exception

Synchronous exception (262)

- * planned exception
- * Handled in an organized manner
- * The most common type of synchronous exception is implemented as a known

Exception stages

int f()

```

    {
        int error; // low mem
        if(error) /* Stage 1: error occurred */
            return -1; // Stage 2: generate
        /* exception object */ → Report
    }

```

int main(void) { → Prey.

```

    int(f()=0); // Stage 3: detect
    /* exception
    */
    } // Stage 4: Handle → Handle
    /* exception */ → Handle exception

```

/* Stage 5: recover / Recover.

}

Stage 1: Error Detection

- * Synchronous (S/W) logical errors
- * Asynchronous (H/W) Interrupt (S/W Interrupt)

Stage 2:

Create object and Raise exception.

⑥

- * An exception Object can be of any complete type

- * An int to a full blown C++ class object

③ Decorate exception

- * polling slow test.

- * Notification control (stack)
Adjustments.

④ Handle exception

- * Ignore; hope someone else handles it
that is do not catch

- * Ack; but allow others to handle
it afterwards, that is catch, handle
and rethrow

- * Own; take complete ownership
that is, catch and handle.

⑤ Recover from exception

- * Continue execution: if handled
inside the pgm.

- * Abort execution: if handled
outside the pgm

(264)

support for exception in C

- * C does not support anything keeping error handling in mind but we can use some of C features to handle errors in more structured way

* Language feature

- * return value and parameters
- * Locals

* Standard lib support

- * Global variable
- * Abnormal termination
- * Conditional "
- * Non local goto
- * signals

Global Variable mechanism

- * use a designated global error variable
- * set it ~~to~~ on error
- * poll/ check it for detection

* Standard lib C99 mechanism

{errno} {errno}

~~Ex~~ #include <errno.h>

(265)

main()

double n, y, result;

errno = 0

result = pow(n, y);

if (errno == EDOM)

 printf("domain error: %f/%f\n",
 n, y);

else if (errno == ERANGE)

 printf("range error in
 result (%f)\n", result);

else

 printf("%f to %f = %f\n",
 n, y, result);

return 0;

}

Exception Handling in C++Exceptions

- (i) separate error handling code from ordinary code
- (ii) Lang. mechanism rather than of the library
- (iii) compiler for tracking automatic variables
- (iv) schemes for destruction of dynamic mem
- (v) less overhead for exceptions
- (vi) exception propagation from the deepest of levels
- (vii) various exceptions handle by single Handler

try - throw - catch

```
class userExcp{ public: exceptions(); }
```

void g()

L: A

userExcp ex("From g()"); — create

report ex. throw ex; — (It will know then the control goes to catch clause)

return