UNIT I · LESSON 06

Polymorphism, Inheritance & Exceptions

operator overloading · inheritance · virtual functions · abstract classes · templates · exceptions · file streams

20
questions in Unit I
C · Python · C++ / OOP

TWO KINDS POLYMORPHISM

Compile-time (static, early)	Run-time (dynamic, late)
function overloading	virtual functions
operator overloading	overriding through base pointer/reference
templates	uses **vtable** / vptr
resolved by compiler	resolved at execution
faster	flexible, small overhead

COMPARE OVERLOADING VS OVERRIDING

Overloading	Overriding
same scope	base & derived class
different parameter list	same signature
compile time	run time (with virtual)
return type may differ	return type same (or covariant)

OPERATOR@ OPERATOR OVERLOADING

```
class Cx {
    double r, i;
public:
    Cx(double a=0, double b=0)
        : r(a), i(b) {}
    Cx operator+(const Cx &o) const {
        return Cx(r + o.r, i + o.i);
    }
    friend ostream& operator<<(ostream &s,
                               const Cx &c);
};
```

- Member binary operator takes **1** argument; friend binary takes **2**
- Unary member takes **0** args; postfix `++` uses dummy `int`: `operator++(int)`
- **Cannot overload**: `::` `.` `.*` `?:` `sizeof` `typeid`
- Must be members: `=` `()` `[]` `->`

OPERATOR@ OPERATOR OVERLOADING (CONTD.)

- `<<` and `>>` for streams are written as **friend** functions
- Cannot change precedence, associativity or number of operands; cannot invent new operators

REUSE INHERITANCE

Type	Shape
Single	A → B
Multiple	A, B → C
Multilevel	A → B → C
Hierarchical	A → B, A → C
Hybrid	mix (e.g. diamond)

```
class Shape { protected: int w, h; };
class Rect : public Shape {
public: int area() { return w * h; }
};
```

- Base class constructor runs **first**, destructor runs **last**
- Constructors, destructors, friends and `=` are **not inherited**
- Pass arguments: `D(int a, int b) : B(a), y(b) {}`

INHERITED ACCESS VISIBILITY MODE

Base member →	public inherit.	protected inherit.	private inherit.
public	public	protected	private
protected	protected	protected	private
private	not accessible	not accessible	not accessible

Default mode: `class D : B` → **private**; `struct D : B` → public.

VIRTUAL BASE DIAMOND PROBLEM

```
class A { public: int x; };
class B : virtual public A {};
class C : virtual public A {};
class D : public B, public C {};  // one A
```

- Without `virtual`, D holds **two copies** of A → ambiguous `d.x`
- Virtual base is constructed by the **most derived** class

LATE BINDING VIRTUAL FUNCTIONS

```
class Base {
public:
    virtual void show() { cout << "Base"; }
    virtual ~Base() {}
};
class Der : public Base {
public:
    void show() override { cout << "Der"; }
};
Base *p = new Der;  p->show();  // Der
```

- Without `virtual` → `Base` (static binding by pointer type)
- Each class with virtual functions has a **vtable**; each object a hidden **vptr**
- Overriding needs the **same signature**; `override` (C++11) checks it
- Virtual call through an **object** (not pointer/reference) is resolved statically
- **Virtual destructor** ensures the derived destructor runs on `delete basePtr`
- `final` stops further overriding / inheriting

PURE VIRTUAL ABSTRACT CLASS

```
class Shape {
public:
    virtual double area() = 0; // pure
};
```

- Class with at least one **pure virtual** function = **abstract**; **cannot be instantiated**
- Pointers/references to it are allowed
- Derived class must override all pure virtuals, else it is abstract too
- Interface = class with only pure virtual functions

GENERIC TEMPLATES

```
template <typename T>
T maxOf(T a, T b) { return a > b ? a : b; }
template <class T>
class Stack { T a[50]; int top = -1;
public: void push(T x) { a[++top] = x; } };
maxOf(3, 7);  maxOf<double>(2.5, 1);
```

- **Function template** & **class template**; resolved at compile time
- **STL** = containers (vector, list, deque, set, map, stack, queue), iterators, algorithms (sort, find, count)

TRY · THROW · CATCH EXCEPTION HANDLING

```
try {
    if (d == 0)
        throw runtime_error("div by 0");
    cout << n / d;
}
catch (const runtime_error &e)
    { cout << e.what(); }
catch (...) { cout << "?"; }  // catch-all
```

- `throw` can throw **any type** (int, string, object)
- First **matching** catch runs → put derived classes before base classes
- `catch(...)` must be **last**
- Uncaught exception → `terminate()`
- **Stack unwinding**: destructors of local objects run while leaving scopes
- Rethrow with `throw;` inside a catch
- Standard: `exception → logic_error, runtime_error, bad_alloc, out_of_range`

FSTREAM FILE STREAMS

Class	Purpose
ifstream	input (read)
ofstream	output (write)
fstream	both

FSTREAM FILE STREAMS (CONTD.)

Mode flag	Meaning
ios::in / ios::out	read / write
ios::app	append at end
ios::ate	open & go to end
ios::trunc	discard contents

FSTREAM FILE STREAMS (CONTD.)

Mode flag	Meaning
ios::binary	binary mode

```
ofstream f("out.txt");
f << "SIPCOT " << 566 << endl;
f.close();
ifstream g("out.txt");
string w; while (g >> w) cout <<
w;
```

- `seekg/tellg` (get pointer, input), `seekp/tellp` (put pointer, output)
- `read()/write()` for binary objects; `eof()`, `fail()`, `good()`

TRAP EXAM TRAPS

- Base pointer → derived object, non-virtual call → **base** version
- Object slicing: assigning derived object to base **object** drops the derived part
- Abstract class object: `Shape s;` → error
- Calling a virtual function inside a constructor uses the class being constructed
- `sizeof` of a class with virtual function includes the **vptr** (4/8 B)
- Overloading `?:` or `::` → error
- A `catch(Base&)` placed before `catch(Derived&)` hides the second
- Multiple inheritance without virtual base → two copies of the grandparent

Q → A RAPID RECALL

Run-time polymorphism needs	virtual function + base pointer/reference
Class with a pure virtual function	abstract class
Syntax of pure virtual	`virtual void f() = 0;`
Table holding virtual function addresses	vtable
Solves diamond problem	virtual base class
Operators that cannot be overloaded	`:: . .* ?: sizeof`
Order of constructor calls	base → member objects → derived
Order of destructor calls	derived → members → base
Generic programming in C++	templates
Block catching every exception	`catch(...)`
Stream class for reading files	`ifstream`
Mode to append to a file	`ios::app`
Get-pointer functions	`seekg`, `tellg`
Default inheritance mode for `class`	private