UNIT I · LESSON 05

OOP & C++ Classes

OOP concepts · classes & objects · constructors · destructors · static members · this · references · friends

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

PILLARS OOP CONCEPTS

Concept	Meaning	C++ tool
Class	blueprint: data + functions	`class`, `struct`
Object	instance of a class	`Box b;`
Encapsulation	bundle data with methods, control access	access specifiers
Abstraction	show essentials, hide details	abstract class, interfaces
Inheritance	derive new class from existing	`class D : public B`
Polymorphism	one name, many forms	overloading, virtual functions
Data hiding	members not reachable from outside	`private`
Dynamic binding	call resolved at run time	`virtual`

PILLARS OOP CONCEPTS (CONTD.)

Concept	Meaning	C++ tool
Message passing	objects call each other's methods	member calls

C++ by **Bjarne Stroustrup** (1979–83, "C with Classes"). C = procedure-oriented (top-down), C++ = object-oriented (bottom-up).

COMPARE POP VS OOP

Procedure-oriented (C)	Object-oriented (C++)
program = functions	program = objects
top-down design	bottom-up design
global data shared freely	data hidden in objects
no access control	private / protected / public
hard to reuse	reuse via inheritance
no overloading	function & operator overloading

DIFFERENCES FROM C C++ BASICS

- `#include <iostream>`, `using namespace std;`
- `cout << x;` (insertion), `cin >> x;` (extraction), `endl`
- `bool`, `string`, `new` / `delete`, references, default arguments, function overloading
- `::` **scope resolution** (global variable, member definition outside class)
- Variables can be declared anywhere; `const` replaces many `#define`s
- **inline** function: body expanded at the call site (request to compiler)

```
#include <iostream>
using namespace std;
int x = 10;
int main() {
    int x = 20;
    cout << x << " " << ::x;  // 20 10
}
```

SYNTAX CLASS & OBJECT

```
class Box {
    double l, b;          // private
public:
    Box(double a, double c) { l = a; b = c; }
    double area() const;  // declared
};
double Box::area() const {  // outside
    return l * b;
}
Box r(3, 4);  cout << r.area();   // 12
```

- `class` default access = **private**; `struct` default = **public**
- Member functions defined **inside** the class are implicitly inline
- `const` member function cannot change data members
- Size of an empty class object = **1 byte**

VISIBILITY ACCESS SPECIFIERS

Specifier	Same class	Derived class	Outside
private	yes	**no**	no
protected	yes	**yes**	no
public	yes	yes	yes

INITIALISE CONSTRUCTORS

- Same name as class, **no return type**, called automatically at creation
- Can be overloaded; can have default arguments; cannot be virtual

Type	Example
Default	`Box()`
Parameterised	`Box(int l, int b)`
Copy	`Box(const Box &o)`
Move (C++11)	`Box(Box &&o)`
Conversion	single-arg `Box(int s)`

- Compiler supplies default constructor **only if none** is written
- **Copy constructor** called when: object initialised from another, passed **by value**, returned by value
- Parameter must be a **reference** (by value → infinite recursion)
- Default copy = **shallow** (member-wise); write a deep copy when the class owns heap memory
- **Initialiser list**: `Box(int a) : l(a), b(a) {}` — needed for const & reference members
- `explicit` stops implicit conversions

CLEAN-UP DESTRUCTOR

- `~Box()` — no arguments, no return, **cannot be overloaded** (only one)
- Called automatically when object goes out of scope or on `delete`
- Objects destroyed in **reverse order** of construction
- Should be **virtual** in a base class used polymorphically

```
class Arr {
    int *p;
public:
    Arr(int n)  { p = new
int[n]; }
    ~Arr()      { delete[] p; }
};
```

DYNAMIC NEW & DELETE

C	C++
malloc / free (functions)	new / delete (operators)
returns void*, needs cast	returns exact type
no constructor call	calls constructor / destructor
returns NULL on failure	throws `bad_alloc`

- `int *p = new int(5);` `delete p;`

DYNAMIC NEW & DELETE (CONTD.)

- `int *a = new int[n];` `delete[] a;` (use [] for arrays)

CLASS-LEVEL STATIC MEMBERS

- **Static data member**: one copy shared by all objects; must be defined outside: `int Box::count = 0;`
- **Static member function**: called with `Box::get()`; can access only static members; has **no this pointer**

```
class Box {
    static int count;
public:
    Box() { count++; }
    static int get() { return count; }
};
int Box::count = 0;
```

SELF THIS POINTER

- Hidden pointer to the **invoking object**, passed to every non-static member function
- `this->x = x;` resolves name clash; `return *this;` enables chaining
- Type in class X: `X * const this`

ALIAS REFERENCES

- `int &r = x;` — another name for x; must be **initialised**, cannot be reseated, no null reference

Pointer	Reference
can be NULL	always refers to an object
can change target	bound once
needs `*` to access	used like the variable
has its own address & size	same address as target

```
void swap(int &a, int &b)
{ int t = a; a = b; b = t; }
```

- Call by reference avoids copying; `const T&` for read-only large objects

ACCESS GRANT FRIENDS

- **friend function**: not a member, but can access private members; declared inside class with `friend`
- **friend class**: all its functions can access private members
- Friendship is **not inherited, not mutual, not transitive**

```
class B;
class A { int x = 5; friend int sum(A, B); };
class B { int y = 7; friend int sum(A, B); };
int sum(A a, B b)
{ return a.x + b.y; }      // 12
```

COMPILE-TIME FUNCTION OVERLOADING

- Same name, different **number/type/order** of parameters
- Return type alone **cannot** distinguish overloads
- Default arguments must be **rightmost**: `int f(int a, int b = 2, int c = 3);`
- Ambiguity: `f(int)` & `f(long)` called with `f(2.5)`

Q → A RAPID RECALL

Feature wrapping data + functions	encapsulation
Default access of class members	private
Function with class name, no return type	constructor
Number of destructors per class	exactly one
Copy constructor parameter	`const ClassName &`
Shared by all objects	static data member
Operator to define member outside class	`::` scope resolution
Pointer passed implicitly to member functions	`this`
Non-member with private access	friend function
`new` failure in C++	throws `std::bad_alloc`
Size of empty-class object	1 byte

Q → A RAPID RECALL (CONTD.)

Inline function benefit	no call overhead (code grows)
Reference must be ...	initialised at declaration
Overloading resolved at	compile time

TRAP EXAM TRAPS

- `Box b();` declares a **function**, not an object
- Constructor cannot be `virtual`; destructor can (and should in bases)
- Static function cannot be `const` or `virtual`, no `this`
- `class` members private by default, `struct` public
- Copy constructor argument must be passed by reference
- `delete` for `new[]` memory → undefined; use `delete[]`
- Friend function is called like a normal function, not with `obj.`
- Order of construction follows **declaration order** of members, not initialiser-list order