PNS SCHOOL OF ENGINEERING & TECHNOLOGY

Nishamani Vihar, Marshaghai, Kendrapara

LECTURE NOTES ON

(CSEPC 201) Programming

with C++

DEPARTMENT OF COMPUTER SCIENCE

3rd SEMESTER

PREPARED BY

MISS. JYOTSNAMAYEE BISWAL

Unit 1

Unit Title: Introduction to OOP & C++ Basics

1. Introduction to Object-Oriented Programming (OOP)

Object-Oriented Programming (OOP) is a programming paradigm based on the concept of “objects” that contain data and functions.

Core Principles of OOP:

Principle	Meaning
Encapsulation	Wrapping data and methods into a single unit (class)
Polymorphism	One function or operator behaves differently based on context
Inheritance	A class (child) can inherit properties from another class (parent)
Abstraction	Hiding unnecessary details and showing only the essentials

2. User-Defined Types: Structures & Unions

Structure (struct):

```
struct Student {
   int roll;
   char name[20];
   float marks;
};
```

Union:

```
union Data {
   int i;
   float f;
   char str[20];
};
```

PNS, Marshaghai ,kendrapara

3. Getting Started with C++

```
#include <iostream>
using namespace std;

int main() {
   cout << "Welcome to C++!";
   return 0;
}
```

4. Data Types & Variables

Type	Example
int	int age = 20;
float	float pi = 3.14;
char	char ch = 'A';
string	string name = "Beb";

5. Strings in C++

```
#include <iostream>
#include <string>
using namespace std;

int main() {
   string name = "Darlo";
   cout << "Hello, " << name;
   return 0;
}
```

6. Functions & Default Parameters

```
int add(int a, int b = 5) {
   return a + b;
}
```

PNS, Marshaghai ,kendrapara

7. Recursion

```
int factorial(int n) {
   if(n == 1) return 1;
   return n * factorial(n - 1);
}
```

8. Namespaces

```
namespace demo {
   int x = 100;
}
```

9. Operators in C++

Arithmetic: +, -, *, /
Relational: ==, !=, >, <
Logical: &&, ||, !

10. Flow Control Statements

if, else, else if
switch
for, while, do-while

```
if (a > b) {
   cout << "A is greater";
}
```

11. Arrays

```
int marks[5] = {80, 90, 85, 70, 60};
```

12. Pointers

```
int x = 10;
int *p = &x;
cout << *p; // outputs 10
```

Short Questions

1. What is polymorphism?

PNS, Marshaghai ,kendrapara

2. Define a structure in C++.

3. What is the difference between struct and union?

4. Write a simple function with default parameters.

5. Explain recursion with an example.

Unit 2

Unit Title: Abstraction Mechanism & Inheritance

1. Abstraction Mechanism

Classes are user-defined data types that contain data members and member functions.

Access Specifiers:
- private: accessible only within the class
- public: accessible from outside the class

Constructor: Special function with the same name as class, used to initialize objects.

```
class Demo {
public:
   Demo() {
      cout << "Constructor called";
   }
};
```

Destructor: Special function with `~` before class name, used to clean up.

```
class Demo {
public:
   ~Demo() {
      cout << "Destructor called";
   }
};
```

Member Data: Variables inside a class.
Member Functions: Functions inside a class.

Inline Function: Small functions defined inside class that compiler replaces at call site.

```
class Test {
public:
   inline void show() {
      cout << "Inline function";
   }
};
```

Friend Function: A function not in class but allowed to access private members.

```
class A {
   int x = 10;
   friend void show(A);
};
void show(A obj) {
   cout << obj.x;
}
```

Static Members: Shared by all objects of class.

```
class Count {
public:
   static int objCount;
};
int Count::objCount = 0;
```

Reference Variables: Alternate names for existing variables.

```
int a = 10;
int &b = a;
```

2. Inheritance

Inheritance is a mechanism where a class (derived) inherits from another class (base).

Class Hierarchy: Structure showing base and derived classes.

Derived Class: Class that inherits properties from another class.

Single Inheritance: One base and one derived class.

```
class Base {};
class Derived : public Base {};
```

Multiple Inheritance: One class inherits from more than one base class.

```
class A {};
class B {};
class C : public A, public B {};
```

Multilevel Inheritance: Class derived from derived class.

```
class A {};
class B : public A {};
class C : public B {};
```

Hybrid Inheritance: Combination of multiple and multilevel inheritance.

Virtual Base Class: Used to avoid ambiguity in hybrid inheritance.

```
class A {};
class B : virtual public A {};
class C : virtual public A {};
class D : public B, public C {};
```

Constructor & Destructor Execution Order: Base class constructor runs before derived. Destructor runs in reverse.

Base Initialization: Derived class constructor can initialize base class.

```
class Base {
public:
   Base(int x) {}
};
class Derived : public Base {
public:
   Derived(int y) : Base(y) {}
```

PNS, Marshaghai ,kendrapara

};

Short Questions

1. What is abstraction in C++?

2. Explain the role of a constructor and destructor.

3. Write an example of a friend function.

4. Define static member with an example.

5. What is hybrid inheritance?

6. What is the purpose of a virtual base class?

7. Explain the order of constructor execution in inheritance.

Unit 3

Unit Title: Polymorphism

1. Polymorphism and Binding

Polymorphism means 'many forms'. In C++, it allows the same function name or operator to behave differently in different contexts.

Binding is the process of linking a function call with its definition.
- **Static Binding**: Done at compile-time.
- **Dynamic Binding**: Done at run-time using virtual functions.

2. Static Polymorphism: Function Overloading

Function Overloading: Multiple functions with same name but different parameters.

```
class Demo {
public:
   void show() {
      cout << "No argument";
   }
   void show(int x) {
      cout << "Integer: " << x;
   }
};
```

3. Ambiguity in Function Overloading

Ambiguity occurs when the compiler cannot determine which overloaded function to call.

```
// Ambiguous function call
void func(int x);
void func(float x);

func(5.0); // Ambiguous: could match both int and float
```

4. Dynamic Polymorphism

Base Class Pointer: Pointer of base class pointing to derived class object.

```
class Base {
public:
   virtual void show() {
      cout << "Base";
   }
};
class Derived : public Base {
public:
   void show() override {
      cout << "Derived";
   }
};
Base* bptr = new Derived();
bptr->show(); // Output: Derived
```

Object Slicing: When derived class object is assigned to base class object, extra members are sliced off.

```
class Base {
public:
   int a;
};
class Derived : public Base {
public:
   int b;
};
```

Base obj = Derived(); // b is sliced off

Late Binding: Function is resolved during runtime (using virtual functions).

Method Overriding: Derived class provides its own version of a base class function.

```
class Base {
public:
   virtual void display() {
      cout << "Base Display";
   }
};
class Derived : public Base {
public:
   void display() override {
      cout << "Derived Display";
   }
};
```

5. Pure Virtual Functions & Abstract Class

Pure Virtual Function: Declared using `= 0`, must be overridden in derived class.

```
class Shape {
public:
   virtual void draw() = 0; // pure virtual function
};
```

Abstract Class: A class with at least one pure virtual function. Cannot be instantiated directly.

Short Questions

1. What is polymorphism in C++?

2. Differentiate between static and dynamic binding.

3. What is function overloading? Give an example.

4. What is ambiguity in overloading?

5. Explain object slicing with example.

6. Define pure virtual function.

7. What is an abstract class?

Unit 4

Unit Title: Operator Overloading

1. This Pointer

The `this` pointer is an implicit pointer available in all non-static member functions. It points to the current object.

```
class Demo {
   int x;
public:
   void setX(int x) {
      this->x = x;
   }
};
```

Applications of `this` pointer:

- Resolve naming conflicts between data members and parameters

- Return the current object from member function

2. Operator Function

Special function used to overload an operator. It must use the keyword `operator`.

```
class Complex {
   int real, imag;
public:
   Complex(int r, int i) : real(r), imag(i) {}
   Complex operator+(Complex c) {
      return Complex(real + c.real, imag + c.imag);
   }
};
```

3. Member and Non-Member Operator Function

Member Function: Defined inside the class using `operator` keyword.

```
class Number {
```

```
    int value;
public:
    Number(int v) : value(v) {}
    Number operator+(Number n) {
        return Number(value + n.value);
    }
};
```

Non-Member Function: Defined outside the class, sometimes declared as friend.

```
class Number {
    int value;
public:
    Number(int v) : value(v) {}
    friend Number operator-(Number a, Number b);
};

Number operator-(Number a, Number b) {
    return Number(a.value - b.value);
}
```

4. Operator Overloading

Operator overloading allows redefining the way operators work for user-defined types.

Operators that can be overloaded: +, -, *, /, ==, =, [], (), etc.
Operators that cannot be overloaded: ::, ., .*, sizeof, ?:

5. I/O Operators Overloading

Overload `<<` for output and `>>` for input using friend functions.

```
class Student {
    string name;
    int age;
public:
    Student(string n, int a) : name(n), age(a) {}
    friend ostream& operator<<(ostream &out, Student s);
};

ostream& operator<<(ostream &out, Student s) {
    out << "Name: " << s.name << ", Age: " << s.age;
```

```
    return out;
}
```

Short Questions

1. What is the purpose of the `this` pointer?

2. Write a program to demonstrate operator overloading.

3. What is the difference between member and non-member operator functions?

4. Which operators cannot be overloaded in C++?

5. Write an example of overloading I/O operators.

Unit 5

Unit Title: Exception Handling

1. Exception Handling in C++

Exception Handling in C++ is used to handle runtime errors and maintain the normal flow of the program.

C++ uses three keywords: `try`, `throw`, and `catch`.

2. try, throw, and catch

try: Block of code where exceptions may occur.
throw: Used to throw an exception.
catch: Handles the exception.

```
try {
    throw 10;
} catch (int e) {
    cout << "Caught exception: " << e;
}
```

3. Exceptions and Derived Classes

When catching exceptions, the base class handler should come after derived class handlers to avoid slicing.

```
class Base {};
class Derived : public Base {};
```

```
try {
   throw Derived();
} catch (Derived d) {
   cout << "Caught Derived";
} catch (Base b) {
   cout << "Caught Base";
}
```

4. Function Exception Declaration

You can declare what exceptions a function might throw using the `throw()` keyword (deprecated in modern C++).

```
void func() throw(int) {
   throw 5;
}
```

5. Unexpected Exceptions

If a function throws an exception not listed in its exception specification, it calls `unexpected()` handler.

In modern C++, it's recommended to use `noexcept` and avoid old-style specifications.

Short Questions

1. What is the use of try, throw, and catch in C++?

2. Explain how exceptions work with derived classes.

3. What is a function exception declaration?

4. What happens when an unexpected exception is thrown?

5. Write a simple program to handle integer exceptions.