# University of Global Village (UGV), Barishal

## Practical Skill Development and Assessment Process form (PSDAP)

**Name of Subject:** Object Oriented Programming Sessional      **Department:** CSE

**Subject Code:** CSE-0613-2204      **Semester:**

**Student Name:**      **Session:**

**Student Id:**      **Section:**

**Credit:**      **Total Marks:**

**Obtain Marks:**

| SL No: | Title of Practical Skills | Details of Practical Skills | Date of Execute | Topic wise mark Allocation | Marks Obtain (Pass mark- 80%) | | Status | | Name of the Instructor/Lab Asst./Lab Expert | Signature of Instructor/Lab Asst./Lab Expert | Signature of External |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | Cont. Assessment | Final Assessment | C | NC | | | |
| **1.** | Implementation of struct data components of your project. | Design and define essential data components of your project using struct to represent core entities (e.g., customer, item, sensor). | | | | | | | | | |
| **2.** | Application of Union and Enumeration for Data Representation | Utilize union and enum to manage category-based data or variable data types within your project efficiently. | | | | | | | | | |
| **3.** | Exploration of Storage Classes and Constant Qualifiers | Integrate auto, static, and const in your project code to ensure controlled memory usage and variable persistence. | | | | | | | | | |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **4.** | Exception Handling Mechanism in Project Modules | Introduce error-handling using try, catch, and throw blocks to manage runtime errors or invalid input in your project functions. | | | | | | | | | |
| **5.** | File Handling for Persistent Data Management | Implement file operations for storing, retrieving, and updating data relevant to your project (e.g., records, logs, or configurations). | | | | | | | | | |
| **6.** | Implementation of Data Mapping and CRUD Operations | Use map and related STL containers to create, update, and delete project-specific records dynamically. | | | | | | | | | |
| **7.** | Function Overloading and Argument Handling in Modular Design | Apply call-by-reference, function overloading, and default arguments to develop reusable and flexible functions within your project. | | | | | | | | | |
| **8.** | Design and Implementation of Classes and Encapsulation | Create class structures to encapsulate project attributes and methods, maintaining data security and abstraction. | | | | | | | | | |
| **9.** | Advanced Class Features and Controlled Accessibility | Apply this pointer, static data members, and friend functions to optimize class-level behavior and access control. | | | | | | | | | |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **10.** | Constructors, Copy Constructors, and Destructors in Object Lifecycle Management | Implement constructors, copy constructors, and destructors to handle object creation and cleanup processes within your project. | | | | | | | | | |
| **11.** | Inheritance and Hierarchical Class Structures | Demonstrate different types of inheritance (single, multiple, multilevel, hierarchical, and hybrid) to promote reusability and modularity in your project. | | | | | | | | | |
| **12.** | Method Overriding and Ambiguity Resolution Techniques | Customize inherited class behaviors through method overriding and apply ambiguity resolution strategies where required. | | | | | | | | | |
| **13.** | Dynamic and Static Object Allocation | Create and manage project objects dynamically (heap) and statically (stack) to understand memory management and control. | | | | | | | | | |
| **14.** | Abstraction and Interface Design using Virtual Functions | Design abstract classes and interfaces with virtual functions to enhance modularity and abstraction in your project architecture. | | | | | | | | | |
| **15.** | Generalization and Specialization in Class Hierarchies | Apply class generalization and specialization principles to represent real-world | | | | | | | | | |

| | | relationships effectively in your project. | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **16.** | Modeling Class Relationships: Association, Composition, and Aggregation | Implement inter-class relationships (association, composition, and aggregation) to reflect logical dependencies in your project system. | | | | | | | | | **16.** |