**QUE 1 : What are the key differences between Procedural Programming and Object- Oriented Programming (OOP)?**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Procedural Programming** | **Object-Oriented Programming (OOP)** |
| **Approach** | Follows a **step-by-step** approach with procedures (functions). | Organizes code using **objects** that combine data and behavior. |
| **Structure** | Code is divided into **functions**. | Code is divided into **classes and objects**. |
| **Focus** | Focuses on **how to do** things (procedures/steps). | Focuses on **what things are** (objects and their interactions). |
| **Data Handling** | Data is usually **separate** from functions and is **passed around**. | Data and functions are **bundled together** inside objects. |
| **Examples** | C, Pascal, BASIC | Java, Python (OOP style), C++ |
| **Reusability** | Harder to reuse code; duplication is common. | Promotes **code reusability** using inheritance, polymorphism, etc. |
| **Security** | Less secure, as data can be easily accessed and modified. | **More secure**, with **encapsulation** (hiding data inside classes). |
| **Real-World Modeling** | Hard to model real-world entities directly. | Easy to model real-world entities (e.g., Car, BankAccount as objects). |

**QUE 2 : List and explain the main advantages of OOP over POP.**

**1. Modularity**

* In OOP, code is organized into **classes and objects**.
* Each class is like a **self-contained unit**.
* This makes it easier to **manage, update, and debug** parts of the code separately.

Example: You can work on the "Car" class without worrying about the "Person" class.

**2. Reusability**

* OOP uses **inheritance**, which allows a class to **reuse** properties and methods of another class.
* You don't have to **rewrite** the same code again.

Example: You can create a "SportsCar" class by reusing (inheriting) features from the "Car" class.

**3. Scalability and Maintainability**

* OOP makes it easier to **extend** a program.
* You can add new features with **less risk** of breaking existing code.
* **Maintaining** large projects becomes simpler because everything is well-organized.

Example: Adding a new type of user in a banking app is easy if you already have a "User" class.

**4. Data Security (Encapsulation)**

* OOP allows you to **hide** sensitive data inside objects.
* Data can be accessed only through **public methods** (getters and setters).
* This protects the code from **accidental changes**.

Example: A "BankAccount" object hides the balance and only allows it to be modified through deposit/withdraw methods.

**5. Real-World Modeling**

* OOP closely matches **how we think about real life**.
* Objects represent **real-world entities** with attributes (data) and behaviors (methods).

Example: "Dog" object with properties like breed, and behaviors like bark().

**6. Polymorphism**

* In OOP, the **same operation** can behave **differently** on different classes.
* This helps in **writing flexible and dynamic code**.

Example: A "draw()" method could draw a circle, square, or triangle based on the object.

**7. Easier Troubleshooting**

* When something goes wrong, you can **trace** the problem to a specific object or class.
* Makes **debugging and testing** faster.

Example: If the "Engine" object fails, you check only the "Engine" class, not the entire program.

**QUE 3 : Explain the steps involved in setting up a C++ development environment.**

**1. Install a C++ Compiler**

* A **compiler** converts your C++ code into a program the computer can run.
* Popular compilers:
  + **GCC** (GNU Compiler Collection) – mostly for Linux/Mac, also available for Windows.
  + **MinGW** – lightweight GCC version for Windows.
  + **MSVC** (Microsoft Visual C++) – for Windows, part of Visual Studio.
  + **Without a compiler, you can't turn code into an actual program!**

**2. Choose and Install an IDE or Text Editor**

* An **IDE (Integrated Development Environment)** helps you **write, compile, and debug** programs easily.
* Popular IDEs and Editors:
  + **Code::Blocks** (free and simple for beginners)
  + **Visual Studio** (powerful, especially for Windows users)
  + **CLion** (professional, paid, from JetBrains)
  + **VS Code** (lightweight editor with C++ extensions)

🔹 IDEs often include a compiler, so installation becomes easier!

**3. Configure the IDE**

* After installation:
  + **Link the compiler** to the IDE (if it’s not auto-detected).
  + Set the **path** to the compiler executable in IDE settings.

🔹 Example: In Code::Blocks, you set the compiler under "Settings → Compiler".

**4. Write Your First C++ Program**

* Create a **new project** or **new file.**

**5. Build and Run the Program**

* Click **Build** or **Compile** to translate code into an executable file.
* Click **Run** to see the output (your program in action).

**QUE 4 : What are the main input/output operations in C++? Provide examples.**

**1. Input using cin**

* **cin** is used to **take input** from the user.
* It stands for **"console input"**.
* It reads input from the **keyboard**.

**2. Output using cout**

* **cout** is used to **display output** to the user.
* It stands for **"console output"**.
* It sends output to the **screen**.

**3. Using cin and cout Together**

* Often, you will **take input** with cin and **show output** with cout.

**4. Handling Multiple Inputs**

* You can use **multiple cin** or **chain inputs** together.

**5. Using getline() for Input**

* **cin** stops at spaces, but **getline()** reads a **full line**, including spaces.