**1. Key Differences between Procedural Programming and Object-Oriented Programming (OOP)**

| **Feature** | **Procedural Programming (POP)** | **Object-Oriented Programming (OOP)** |
| --- | --- | --- |
| **Approach** | Based on functions and procedures. | Based on objects and classes. |
| **Data Handling** | Data is global and can be accessed by any function. | Data is hidden inside classes (encapsulation). |
| **Reusability** | Code reusability is limited. | Code reusability is high through inheritance and classes. |
| **Security** | Less secure as data is globally accessible. | More secure due to data hiding and abstraction. |
| **Examples** | C, Pascal | C++, Java, Python |
| **Focus** | Focuses on “how to do” (functions). | Focuses on “who does it” (objects). |

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

1. **Encapsulation:**
   * Data and functions are bundled together in a class, protecting data from unauthorized access.
2. **Inheritance:**
   * Allows code reuse by creating new classes from existing ones.
3. **Polymorphism:**
   * Enables the same function or operator to behave differently based on context.
4. **Abstraction:**
   * Hides complex implementation details and shows only essential features.
5. **Modularity:**
   * Each class can be developed and tested independently, improving maintainability.
6. **Code Reusability and Maintainability:**
   * OOP reduces redundancy and makes updating or modifying code easier.

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

1. **Install an IDE (Integrated Development Environment):**
   * Common IDEs: *Dev C++*, *Code::Blocks*, *Turbo C++*, or *Visual Studio*.
2. **Install Compiler:**
   * IDEs like Code::Blocks come with the *GCC (GNU Compiler Collection)* for compiling C++ programs.
3. **Create a New Project or File:**
   * Open IDE → Click *New Project* or *New File* → Save as .cpp.
4. **Write the Program:**
   * Enter your C++ code in the editor window.
5. **Compile the Program:**
   * Click *Compile* or *Build* to convert source code into executable form.
6. **Run the Program:**
   * Execute the compiled file to see the output in the console.

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

C++ uses **cin** (for input) and **cout** (for output), both defined in the **iostream** library.

**✅ Example:**

#include <iostream>

using namespace std;

int main() {

string name;

int age;

cout << "Enter your name: "; // Output statement

cin >> name; // Input statement

cout << "Enter your age: "; // Output statement

cin >> age; // Input statement

cout << "Hello, " << name << "! You are " << age << " years old." << endl;

return 0;

}

**Explanation:**

* cout → Sends data to output screen.  
  Example: cout << "Hello";
* cin → Takes input from the user.  
  Example: cin >> name;
* endl → Ends the current line and moves to the next.

**5. What are the different data types available in C++? Explain with examples.**

C++ provides several **data types** that define the kind of data a variable can store.  
They are mainly classified into three categories:

**A. Primary (Built-in) Data Types:**

| **Data Type** | **Description** | **Example** |
| --- | --- | --- |
| int | Stores integers (whole numbers). | int age = 20; |
| float | Stores single-precision decimal numbers. | float price = 49.99; |
| double | Stores double-precision decimal numbers. | double pi = 3.14159; |
| char | Stores a single character. | char grade = 'A'; |
| bool | Stores true or false values. | bool isPass = true; |

**B. Derived Data Types:**

| **Type** | **Description** | **Example** |
| --- | --- | --- |
| Array | Collection of similar data items. | int marks[5]; |
| Pointer | Stores address of another variable. | int \*ptr; |
| Function | Block of reusable code. | void display(); |
| Reference | Alias for another variable. | int &ref = num; |

**C. User-Defined Data Types:**

| **Type** | **Description** | **Example** |
| --- | --- | --- |
| struct | Group of variables of different data types. | struct Student {int id; char name[20];}; |
| class | Defines data and functions together. | class Car {public: void start();}; |
| enum | Defines symbolic constants. | enum Day {Mon, Tue, Wed}; |

**6. Explain the difference between implicit and explicit type conversion in C++.**

| Type Conversion | Description | Example |
| --- | --- | --- |
| Implicit Type Conversion (Type Promotion) | Automatically performed by the compiler when different data types are used in an expression. | cpp int a = 5; float b = 2.5; float c = a + b; // a converted to float automatically |
| Explicit Type Conversion (Type Casting) | Manually performed by the programmer using casting operators. | cpp int a = 5; float b = 2.5; int c = (int)b + a; // b explicitly converted to int |

Summary:

* Implicit Conversion: Done automatically by the compiler.
* Explicit Conversion: Done manually by the programmer using (data\_type) syntax.

**7. What are the different types of operators in C++? Provide examples of each.**

Operators are special symbols that perform operations on variables and values.

**1. Arithmetic Operators**

Perform basic mathematical operations.

+, -, \*, /, %

Example: int sum = a + b;

**2. Relational Operators**

Compare two values and return true (1) or false (0).

==, !=, >, <, >=, <=

Example: if(a > b)

**3. Logical Operators**

Used to combine multiple conditions.

&& (AND), || (OR), ! (NOT)

Example: if(a > 0 && b > 0)

**4. Assignment Operators**

Used to assign values to variables.

=, +=, -=, \*=, /=, %=

Example: x += 5; // x = x + 5

**5. Bitwise Operators**

Operate on bits of data.

&, |, ^, ~, <<, >>

Example: c = a & b;

**6. Increment/Decrement Operators**

Increase or decrease a variable’s value by one.

++, --

Example: a++; b--;

**7. Conditional (Ternary) Operator**

Used as a shortcut for an if-else statement.

(condition) ? value\_if\_true : value\_if\_false;

Example: result = (a > b) ? a : b;

**8. Miscellaneous Operators**

| **Operator** | **Description** | **Example** |
| --- | --- | --- |
| sizeof | Returns size of a data type or variable | sizeof(int) |
| & | Address of operator | &num |
| \* | Pointer (value at address) | \*ptr |
| -> | Accesses class members via pointer | ptr->display() |

**8. Explain the purpose and use of constants and literals in C++.**

**A. Constants:**

A **constant** is a fixed value that cannot be changed during program execution.

**Types of Constants:**

1. **Integer Constants:** Whole numbers  
   Example: int a = 100;
2. **Floating-point Constants:** Decimal numbers  
   Example: float pi = 3.14;
3. **Character Constants:** Single character enclosed in quotes  
   Example: char grade = 'A';
4. **String Constants:** Sequence of characters enclosed in double quotes  
   Example: string name = "Chetan";

**Defining Constants:**

const float PI = 3.14159; // using const keyword

#define MAX 100 // using preprocessor directive

**B. Literals:**

Literals are **actual fixed values** assigned to variables or constants in the program.  
Example:

int age = 21; // 21 is an integer literal

char grade = 'A'; // 'A' is a character literal

string city = "Surat"; // "Surat" is a string literal

**Purpose:**

* Make code more readable and reliable.
* Prevent accidental modification of fixed values.
* Improve program safety and maintainability.

**9. What are conditional statements in C++? Explain the if-else and switch statements.**

**Definition:**

Conditional statements are used to **make decisions** in a program.  
They allow the program to **execute different code blocks** based on certain conditions (true or false).

**A. if-else Statement**

Used when you want to execute one block of code if a condition is true, and another if it’s false.

**Syntax:**

if (condition) {

// Executes when condition is true

} else {

// Executes when condition is false

}

**Example:**

#include <iostream>

using namespace std;

int main() {

int marks = 85;

if (marks >= 50)

cout << "You passed!" << endl;

else

cout << "You failed!" << endl;

return 0;

}

**Output:**

You passed!

**B. switch Statement**

Used when you need to compare a single variable against multiple values.

**Syntax:**

switch (variable) {

case value1:

// Code for case 1

break;

case value2:

// Code for case 2

break;

default:

// Code if no case matches

}

**Example:**

#include <iostream>

using namespace std;

int main() {

int day = 3;

switch (day) {

case 1: cout << "Monday"; break;

case 2: cout << "Tuesday"; break;

case 3: cout << "Wednesday"; break;

default: cout << "Invalid day";

}

return 0;

}

**10. What is the difference between for, while, and do-while loops in C++?**

oops allow a set of statements to be executed repeatedly until a condition is false.

| **Loop Type** | **Syntax** | **Condition Check** | **Use Case** |
| --- | --- | --- | --- |
| **for loop** | for(initialization; condition; update) | At the beginning of each iteration | When the number of iterations is **known** |
| **while loop** | while(condition) | Before each iteration | When the number of iterations is **unknown** |
| **do-while loop** | do { } while(condition); | After each iteration | When the loop must run **at least once** |

**11. How are break and continue statements used in loops? Provide examples.**

**A. break Statement**

Used to **exit** a loop immediately, even if the loop condition is still true.

**Example:**

#include <iostream>

using namespace std;

int main() {

for (int i = 1; i <= 5; i++) {

if (i == 3)

break; // Stops the loop

cout << i << " ";

}

return 0;

}

**Output:**

1 2

**B. continue Statement**

Used to **skip the current iteration** and move to the next iteration of the loop.

**Example:**

#include <iostream>

using namespace std;

int main() {

for (int i = 1; i <= 5; i++) {

if (i == 3)

continue; // Skips when i = 3

cout << i << " ";

}

return 0;

}

**12. Explain nested control structures with an example.**

**Definition:**

A **nested control structure** means using one control structure (like if, for, or while) **inside another**.

**Example: Nested Loop**

#include <iostream>

using namespace std;

int main() {

for (int i = 1; i <= 3; i++) { // Outer loop

for (int j = 1; j <= i; j++) { // Inner loop

cout << "\* ";

}

cout << endl;

}

return 0;

}

**13. What is a function in C++? Explain the concept of function declaration, definition, and calling.**

A function in C++ is a block of code that performs a specific task.  
It helps in reusing code, improving readability, and reducing redundancy.

Functions divide a large program into smaller, manageable parts called modules.

Types of functions:

1. Library functions (e.g., sqrt(), pow(), printf() etc.)
2. User-defined functions (functions created by the programmer)

Concepts of a Function:

1. Function Declaration (Prototype):  
   It tells the compiler about the function name, return type, and parameters before its use.  
   Example:
2. int add(int, int); // Function declaration
3. Function Definition:  
   It contains the actual body of the function where the task is performed.  
   Example:
4. int add(int a, int b) {
5. return a + b;
6. }
7. Function Calling:  
   When a function is used inside the main program, it is called.  
   Example:

int sum = add(5, 10); // Function call

**14. What is the scope of variables in C++? Differentiate between local and global scope.**

**Scope** of a variable means the part of the program where that variable can be accessed or used.

**Types of Scope:**

| **Type** | **Description** | **Example** |
| --- | --- | --- |
| **Local Scope** | Variable declared inside a function or block; accessible only within it. | cpp int main() { int x = 10; } |
| **Global Scope** | Variable declared outside all functions; accessible throughout the program. | cpp int x = 20; |

**15. Explain recursion in C++ with an example.**

**Recursion** is a process in which a function calls itself directly or indirectly to solve a problem.

Each recursive call reduces the problem into smaller subproblems until a **base condition** is met.

**Example: Factorial using Recursion**

#include <iostream>

using namespace std;

int factorial(int n) {

if (n == 0 || n == 1)

return 1; // Base case

else

return n \* factorial(n - 1); // Recursive call

}

int main() {

int num = 5;

cout << "Factorial of " << num << " is: " << factorial(num);

return 0;

}

**Output:**

Factorial of 5 is: 120

**Explanation:**

* factorial(5) → 5 \* factorial(4) → 5 \* 4 \* 3 \* 2 \* 1 = 120

**16. What are function prototypes in C++? Why are they used?**

A **function prototype** (or **function declaration**) is a statement that tells the compiler about a function’s **name**, **return type**, and **parameters** before it is used.

It appears **before the main() function** or before the function is called.

**Syntax:**

returnType functionName(parameterType1, parameterType2, ...);

**17. What are arrays in C++? Explain the difference between single-dimensional and multi- dimensional arrays.**

An **array** in C++ is a **collection of elements of the same data type** stored in **contiguous memory locations**.  
Arrays allow us to store and access multiple values using a single variable name with an index.

**Syntax:**

data\_type array\_name[size];

**Example:**

int numbers[5] = {10, 20, 30, 40, 50};

**Types of Arrays:**

| **Type** | **Description** | **Example** |
| --- | --- | --- |
| **Single-Dimensional Array (1D)** | Stores data in a single row or column. | int marks[5] = {85, 90, 75, 88, 92}; |
| **Multi-Dimensional Array (2D, 3D, etc.)** | Stores data in a table (rows and columns). | int matrix[2][2] = {{1,2}, {3,4}}; |

**Example of Each:**

**1D Array Example:**

int arr[3] = {1, 2, 3};

**2D Array Example:**

int mat[2][2] = { {1, 2}, {3, 4} };

**18. Explain string handling in C++ with examples.**

A **string** is a sequence of characters. In C++, there are **two main ways** to handle strings:

**1. Character Array Method**

Uses the char data type and stores characters terminated by a null character ('\0').

**Example:**

#include <iostream>

using namespace std;

int main() {

char name[10] = "Chetan";

cout << "Name: " << name;

return 0;

}

**Output:**

Name: Chetan

**2. String Class (C++ Standard Library)**

The <string> header provides the string class with built-in functions for easier string manipulation.

**Example:**

#include <iostream>

#include <string>

using namespace std;

int main() {

string str1 = "Hello";

string str2 = "World";

string result = str1 + " " + str2; // Concatenation

cout << "Combined String: " << result;

return 0;

}

**Output:**

Combined String: Hello World

**19. How are arrays initialized in C++? Provide examples of both 1D and 2D arrays.**

Array initialization means assigning values to array elements at the time of declaration.

**1D Array Initialization**

int marks[5] = {85, 90, 75, 88, 92};

If fewer values are given, remaining elements are set to **zero**:

int numbers[5] = {10, 20}; // {10, 20, 0, 0, 0}

**2D Array Initialization**

int matrix[2][3] = {

{1, 2, 3},

{4, 5, 6}

};

or simply:

int matrix[2][3] = {1, 2, 3, 4, 5, 6};

**Access Example:**

cout << matrix[1][2]; // Output: 6

**20. Explain string operations and functions in C++.**

The **string class** in C++ provides several built-in operations and functions for handling text.

**Common String Operations:**

| **Operation** | **Example** | **Description** |
| --- | --- | --- |
| **Concatenation** | s3 = s1 + s2; | Joins two strings |
| **Length** | s1.length() | Returns number of characters |
| **Character Access** | s1[0] | Accesses specific character |
| **Comparison** | s1 == s2 | Checks equality |
| **Substring** | s1.substr(0,3) | Extracts part of a string |
| **Find** | s1.find("Hello") | Finds position of substring |
| **Insert** | s1.insert(5, "C++") | Inserts text into string |
| **Erase** | s1.erase(3,2) | Removes characters |
| **Append** | s1.append(" World") | Adds to the end of string |

**Example Program:**

#include <iostream>

#include <string>

using namespace std;

int main() {

string str = "C++ Programming";

cout << "String: " << str << endl;

cout << "Length: " << str.length() << endl;

cout << "Substring (0,3): " << str.substr(0, 3) << endl;

cout << "Find 'Pro': " << str.find("Pro") << endl;

return 0;

}

**Output:**

String: C++ Programming

Length: 15

Substring (0,3): C++

Find 'Pro': 4

**21. Explain the key concepts of Object-Oriented Programming (OOP).**

Object-Oriented Programming (OOP) is a programming approach that organizes software design around **objects** instead of functions and logic.  
The main **key concepts** of OOP are:

1. **Class** – A blueprint or template for creating objects.
2. **Object** – An instance of a class that has data and behavior.
3. **Encapsulation** – Binding data and functions together and restricting direct access to the data.
4. **Inheritance** – The mechanism by which one class can acquire the properties and behaviors of another class.
5. **Polymorphism** – The ability of one function or operator to behave differently based on the context.
6. **Abstraction** – Hiding complex details and showing only the essential features.

**22. What are classes and objects in C++? Provide an example.**

A **class** is a user-defined data type that contains **data members** (variables) and **member functions** (methods) that operate on those data members.

An **object** is an **instance of a class** that represents a real-world entity.

**23. What is inheritance in C++? Explain with an example.**

**Inheritance** is the process by which one class (called the **derived class**) acquires the properties and behaviors of another class (called the **base class**).  
It allows **code reusability** and establishes a relationship between classes.

**Example:**

#include <iostream>

using namespace std;

class Person {

public:

string name;

int age;

void show() {

cout << "Name: " << name << ", Age: " << age << endl;

}

};

class Student : public Person { // Student inherits Person

public:

int studentId;

void display() {

cout << "Student ID: " << studentId << endl;

}

};

int main() {

Student s;

s.name = "Chetan";

s.age = 21;

s.studentId = 101;

s.show();

s.display();

return 0;

}

**Explanation:**  
Here, Student inherits from Person, so it can access name, age, and show() from the base class.

**24. What is encapsulation in C++? How is it achieved in classes?**

**Encapsulation** is the process of **wrapping data (variables)** and **functions (methods)** together into a single unit — a **class** — and restricting direct access to the data from outside the class.

It is achieved by:

1. Declaring data members as **private**.
2. Providing **public methods (getters and setters)** to access or modify the data safely.

**Example:**

#include <iostream>

using namespace std;

class BankAccount {

private:

double balance; // private data member

public:

void deposit(double amount) {

balance += amount;

}

void showBalance() {

cout << "Balance: " << balance << endl;

}

};

int main() {

BankAccount acc;

acc.deposit(1000);

acc.showBalance();

return 0;

}