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

Ans:

- Approach:
  - Procedural-Oriented Programming follows top-down approach.
  - Object-oriented Programming follow bottom-up approach.
- Focus:
  - Procedural-Oriented Programming focus on functions.
  - Object-oriented Programming focus on objects.
- Data Security:
  - Procedural-Oriented Programming have low security.
  - Object-oriented Programming have high security (Encapsulation).
- Code Reusability:
  - Procedural-Oriented Programming have less code reusability.
  - Object-oriented Programming have high code reusability (Inheritance).

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

Ans:

- Encapsulation (Data Security)
  i) OOP binds data and functions together in a class
  ii) Data is hidden using private/protected access
  iii) Prevents accidental data modification
- Data Hiding
  i) Internal details are hidden from the user
  ii) Only required functionality is exposed
- Code Reusability
  i) OOP supports inheritance
  ii) Existing class features can be reused in new classes
- Modularity
  i) Program is divided into objects
  ii) Each object handles a specific task
- Easy Maintenance
  i) Changes in one class do not affect others
  ii) Debugging and updating is easier

Q: 3 What are the main input/output operations in C++? Provide examples.

Ans:

- `cin` (Standard Input)
    - i) Used to take input from the keyboard
    - ii) Works with the extraction operator >>

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

int main()
{
    int a;
    cout << "Enter a number: ";
    cin >> a;
    return 0;
}
```

- `cout` (Standard Output)
    - i) Used to display output on the screen
    - ii) Works with the insertion operator <<.

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

int main()
{
    int a = 10;
    cout << "Value of a = " << a;
    return 0;
}
```

Q: 4 What are the different data types available in C++? Explain with examples.

Ans:

- Basic Data Type
    - i) Int
    - ii) Float
    - iii) Char
    - iv) Double
    - v)

```
int a = 10;
float b = 3.14;
char ch = 'X';
bool isTrue = false;
```

- Derived Data Types
  - i) array
  - ii) pointer
  - iii) function
  - iv) reference

```
int arr[3] = {1,2,3};
int *p = &arr[0];
```

- User-Defined Data Types
  - i) struct
  - ii) union
  - iii) class

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

- Void Data Types

Q: 5 Explain the difference between implicit and explicit type conversion in C++.

Ans:

- Implicit Type Conversion:
  Implicit type conversion is an automatic conversion performed by the compiler without programmer intervention. Implicit type conversion is an automatic conversion performed by the compiler without programmer intervention.

```
int a = 10;
double b;

b = a;
```

- Explicit Type Conversion:

  Explicit type conversion is a manual conversion done by the programmer using type casting.

```
double x = 9.8;

int y;

y = (int)x;
```

Q: 6 What are the different types of operators in C++? Provide examples of each

Ans:

- Arithmetic Operators

```
int a = 10, b = 3;
cout <<a+b;
cout<<a-b;
cout<<a*b;
cout<<a/b;
cout<<a%b;
```

- Relational (Comparison) Operators

```
int a=10,b=3;

cout << (a > b);
cout<<(a<b)
cout<<(a>=b)
cout<<(a<=b)
cout<<(a==b)
cout<<(a!=b)
```

- Logical Operators

```
if(a > 0 && b > 0)
    cout << "Both positive";
```

- Bitwise Operators
- Assignment Operators
- Conditional (Ternary) Operator

```
int max = (a > b) ? a : b;
```

Q: 7 Explain the purpose and use of constants and literals in C++.

Ans:

- Constant:
  Constants are variables whose values cannot be changed once they are defined during program execution.

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

int main()
{
    const float PI = 3.14;
    float radius = 5;
    float area = PI * radius * radius;
    cout << "Area = " << area;

    return 0;
}
```

Q: 8 What are conditional statements in C++? Explain the if-else and switch statements.

Ans: Conditional statements are used to make decisions in a program. They execute different blocks of code based on conditions.

Types of Conditional Statements in C++

i) if

ii) if-else

iii) else-if ladder

iv) switch

-
  ```
  #include <iostream>
  using namespace std;
  int main()
  {
      int num = 10;
      if (num % 2 == 0)
          cout << "Even number";
      else
          cout << "Odd number";
      return 0;
  }
  ```

-
  ```
  #include <iostream>
  using namespace std;
  int main()
  {
      int choice = 2;
  ```

```
    switch (choice)
    {
        case 1:
            cout << "Addition";
            break;
        case 2:
            cout << "Subtraction";
            break;
        case 3:
            cout << "Multiplication";
            break;
        default:
            cout << "Invalid choice";
    }
    return 0;
}
```

Q : 9 What is the difference between for, while, and do-while loops in C++?

Ans :

- **`for`** Loop
  i) Best when number of iterations is known in advance
- **`while`** Loop
  i) Best when number of iterations is not known
  ii) Condition checked before loop body
- do-while Loop
  i) Loop body must execute at least once
  ii) Condition checked after loop body

Q : 10 How are break and continue statements used in loops? Provide examples

Ans :

- **`break`** statement :
  i) Immediately terminates the loop.
  ii) Control moves outside the loop.

```
int i = 1;
while(i <= 5)
{
    if(i == 4)
        break;
    cout << i << " ";
    i++;
}
```

- **`continue`** statement :
    i) Skip the current iteration of the loop.
    ii) Continue the loop from next iteration.

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

int main()
{
    for(int i = 1; i <= 5; i++)
    {
        if(i == 3)
            continue;
        cout << i << " ";
    }
    return 0;
}
```

Q : 11 Explain nested control structures with an example.

Ans : A nested control structure is a control statement placed inside another control statement. It is used to handle complex decision-making or repeated operations where one condition or loop depends on another.

Types of Nested Control Structures

i) Nested if-else (if inside another if)

ii) Nested loops (loop inside another loop)

iii) If inside a loop / loop inside an if

- Nested if-else :

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

int main()
{
    int num;
    cout << "Enter a number: ";
    cin >> num;

    if (num > 0)
    {
        if (num % 2 == 0)
            cout << "Positive Even Number";
        else
            cout << "Positive Odd Number";
    }
    else
    {
        cout << "Number is not positive";
    }

    return 0;
}
```

- Nested Loop :

```
for(int i = 1; i <= 3; i++)
{
    for(int j = 1; j <= 3; j++)
    {
        cout << "* ";
    }
    cout << endl;
}
```

Q : 12 What is a function in C++? Explain the concept of function declaration, definition, and calling.

Ans : A function in C++ is a self-contained block of code that performs a specific task. Functions help in code reusability, modularity, and better readability of a program.

- Function Declaration :
  A function declaration tells the compiler about:
  i) Function name
  ii) Return type
  iii) Number and type of parameters
  It is also called a function prototype.

- Function Definition :
  A function definition contains the actual code that executes when the function is called.

- Function Calling :
  A function call executes the function by passing required arguments.

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

// Function declaration
int add(int, int);

// Function definition
int add(int a, int b)
{
    return a + b;
}

int main()
{
    int sum;

    // Function calling
    sum = add(5, 3);

    cout << "Sum = " << sum;
    return 0;
}
```

Q : 13 What is the scope of variables in C++? Differentiate between local and global scope.

Ans : The scope of a variable refers to the region of the program where the variable is accessible and can be used. Scope determines the visibility and lifetime of a variable in a program.

Types of Variable Scope in C++

1. Local Scope
2. Global Scope

1. Local Scope

A local variable is declared inside a function or a block. It is accessible only within that function or block.

Characteristics

- Created when the function/block starts
- Destroyed when the function/block ends
- Cannot be accessed outside its scope

2. Global Scope

A global variable is declared outside all functions. It can be accessed throughout the program.

Characteristics

- Exists for the entire program execution
- Accessible by all functions
- Can be accessed using the scope resolution operator :: if needed

Q : 14 Explain recursion in C++ with an example.

Ans : Recursion is a programming technique in which a function calls itself to solve a problem. It breaks a complex problem into smaller sub-problems of the same type until a base condition is reached.

```
#include <iostream>
using namespace std;
int factorial(int n)
{
    if (n == 0)
        return 1;
    else
        return n * factorial(n - 1);   // recursive call
}
int main()
{
    int num = 5;
    cout << "Factorial = " << factorial(num);
    return 0;
}
```

Q : 15 What are function prototypes in C++? Why are they used?

Ans : A function prototype is a declaration of a function that tells the compiler about:

- the function name
- the return type

- the number and types of parameters

```
#include <iostream>
using namespace std;
// Function prototype
int add(int, int);

int main()
{
    cout << add(5, 3);
    return 0;
}
// Function definition
int add(int a, int b)
{
    return a + b;
}
```

Q : 16 What are arrays in C++? Explain the difference between single-dimensional and multi- dimensional arrays.

Ans : An array in C++ is a collection of elements of the same data type stored in contiguous memory locations. Each element is accessed using an index, starting from `0`.

1. Single-Dimensional Array

A single-dimensional array stores elements in a linear sequence using one index.

```
int marks[5] = {70, 80, 90, 85, 75};
for(int i = 0; i < 5; i++)
{
    cout << marks[i] << " ";
}
```

2. Multi-Dimensional Array

A multi-dimensional array stores data in tabular or matrix form using two or more indices.

```
int matrix[2][3] = {

    {1, 2, 3},

    {4, 5, 6}

};

for(int i = 0; i < 2; i++)

{

    for(int j = 0; j < 3; j++)

    {

        cout << matrix[i][j] << " ";

    }

    cout << endl;

}
```

Q : 17 Explain string handling in C++ with examples.

Ans : In C++, strings can be handled in two main ways:

1. Using C-style strings (char arrays)
2. Using the string class.

- Using C-style strings :

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

int main()
{
    char name[20];
    cout << "Enter name: ";
    cin >> name;          // no spaces
    cout << name;
    return 0;
}
```

- Using the string class. :

```
#include <iostream>

#include <string>

using namespace std;

int main()

{

    string s1 = "Hello";

    string s2 = "World";

    string s3 = s1 + " " + s2;

    cout << s3 << endl;

    cout << "Length: " << s3.length();

    return 0;

}
```

Q : 18 How are arrays initialized in C++? Provide examples of both 1D and 2D arrays.

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

- 1-D Array :
  ```
  int A[5] = {10, 20, 30, 40, 50};
  ```

- 2-D Array :

```
int matrix[2][3] = {

    {1, 2, 3},

    {4, 5, 6}

};
```

Q : 19 Explain the key concepts of Object-Oriented Programming (OOP).

Ans : Object-Oriented Programming (OOP) is a programming paradigm that organizes a program using objects and classes. It focuses on modelling real-world entities and improves code reusability, security, and maintainability.

1. Class

A class is a blueprint or template used to create objects.
It defines the data members (variables) and member functions (methods).

2. Object

An object is an instance of a class. It represents a real-world entity and occupies memory.

3. Encapsulation

Encapsulation means wrapping data and functions together into a single unit (class) and restricting direct access to data using access specifiers.

4. Abstraction

Abstraction means showing only essential details and hiding implementation details.

5. Inheritance

Inheritance allows a derived class to reuse properties of a base class, improving code reusability.

6. Polymorphism

Polymorphism means one function name having different behaviours.

Types:

- Compile-time (Function Overloading)
- Run-time (Function Overriding)

Q : 20 What are classes and objects in C++? Provide an example.

Ans : A class in C++ is a user-defined data type that acts as a blueprint for creating objects. It contains data members (variables) and member functions (methods) that define the properties and behaviour of an object.

An object is an instance of a class. It represents a real-world entity and occupies memory. Objects use the data and functions defined in the class.

```
#include <iostream>
using namespace std;
class Student
{
private:
    int roll;
    string name;
public:
    void getData(int r, string n)
    {
        roll = r;
```

```
            name = n;
        }
        void showData()
        {
            cout << "Roll No: " << roll << endl;
            cout << "Name: " << name << endl;
        }
    };

    int main()
    {
        Student s1;
        s1.getData(1, "Rahul");
        s1.showData();

        return 0;
    }
```

Q : 21 What is inheritance in C++? Explain with an example.

Ans : Inheritance is an important feature of Object-Oriented Programming (OOP) in C++ that allows a derived class to reuse the properties and functions of a base class. It helps in code reusability, extensibility, and hierarchical organization of classes.

```
#include <iostream>
using namespace std;
class Person
{
protected:
    string name;
public:
    void setName(string n)
    {
        name = n;
    }
};
class Student : public Person
{
private:
    int roll;
public:
    void setRoll(int r)
    {
        roll = r;
    }
    void showData()
    {
        cout << "Name: " << name << endl;
        cout << "Roll No: " << roll << endl;
    }
};
int main()
{
    Student s1;
    s1.setName("Rahul");
    s1.setRoll(101);
    s1.showData();

    return 0;
}
```

Q : 22 What is encapsulation in C++? How is it achieved in classes?

Ans : Encapsulation is an Object-Oriented Programming (OOP) concept that means wrapping data (variables) and the functions (methods) that operate on that data into a single unit called a class.

Why Encapsulation is Important

- Provides data security
- Prevents unauthorized access
- Improves maintainability
- Supports data hiding

How Encapsulation is Achieved in C++

Encapsulation is achieved using:

1) Classes
2) Access specifiers (private, public, protected)
3) Public member functions to access private data

```
#include <iostream>
using namespace std;
class Account
{
private:
    int balance;
public:
    void setBalance(int b)
    {
        balance = b;
    }
    int getBalance()
    {
        return balance;
    }
};
int main()
{
    Account acc;
    acc.setBalance(5000);
    cout << acc.getBalance();
    return 0;
}
```