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

Procedural Programming focuses on functions and procedures that operate on data, while Object-Oriented Programming (OOP) organizes code around objects which combine data and functions.

Key Differences:
- POP emphasizes procedures; OOP emphasizes objects.
- In POP, data is global and shared; in OOP, data is encapsulated within objects.
- OOP supports inheritance, polymorphism, and abstraction which are not directly available in POP.
- OOP is more secure as data access is controlled through access specifiers.

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

Advantages of OOP:
1. Encapsulation – Improves security by bundling data and functions.
2. Inheritance – Promotes reusability of code.
3. Polymorphism – Provides flexibility through multiple forms of functions/operators.
4. Abstraction – Hides unnecessary details, showing only relevant features.
5. Modularity – Makes programs easier to manage and maintain.

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

Steps:
1. Install a Compiler (e.g., GCC, MinGW).
2. Choose an IDE (e.g., Code::Blocks, Dev-C++, VS Code).
3. Configure PATH environment variable (if required).
4. Write code and save as .cpp file.
5. Compile using g++ program.cpp -o program.
6. Run the compiled program.

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

C++ uses cin and cout for input/output.

Example:

```
#include<iostream>
using namespace std;
int main() {
  int age;
  cout << "Enter your age: ";
  cin >> age;
  cout << "You are " << age << " years old.";
  return 0;
}
```

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

C++ data types:
- int: Integer values (e.g., int a = 10;)
- float: Decimal values (e.g., float pi = 3.14;)
- double: Large floating-point numbers (e.g., double d = 19.99;)
- char: Single character (e.g., char grade = 'A';)
- bool: Boolean values (e.g., bool flag = true;)
- string: Sequence of characters (e.g., string name = "John";)

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

- Implicit conversion (Type Casting): Automatically performed by the compiler. Example: int x = 5; float y = x; // int converted to float.
- Explicit conversion: Manually done using cast operators. Example: double d = 9.8; int x = (int)d; // double converted to int.

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

Types of Operators:
- Arithmetic: +, -, *, /, % (e.g., a+b)
- Relational: ==, !=, >, <, >=, <= (e.g., a > b)
- Logical: &&, ||, ! (e.g., if(a>0 && b>0))
- Assignment: =, +=, -= (e.g., a+=5)
- Increment/Decrement: ++, -- (e.g., a++)
- Bitwise: &, |, ^, ~, <<, >>

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

- Constants are fixed values that cannot be changed during execution. Declared using const keyword.

Example: const int MAX = 100;
- Literals are direct values assigned to variables. Example: int x = 10; (10 is a literal).

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

Conditional statements allow decision-making in C++.

- if-else: Executes different code blocks depending on condition.

Example:
```
if(a > b) {
  cout << "a is greater";
} else {
  cout << "b is greater";
}
```

- switch: Compares a variable against multiple constant values.

Example:
```
switch(choice) {
  case 1: cout << "One"; break;
  case 2: cout << "Two"; break;
  default: cout << "Invalid";
}
```

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

- for loop: Used when number of iterations is known.
  Example: for(int i=0;i<5;i++) cout<<i;
- while loop: Checks condition before execution.
  Example: while(i<5) {cout<<i; i++;}
- do-while loop: Executes at least once since condition is checked later.
  Example: do{cout<<i; i++;}while(i<5);

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

- break: Terminates loop immediately.
  Example: for(int i=0;i<5;i++){ if(i==3) break; cout<<i; }
- continue: Skips current iteration and moves to next.
  Example: for(int i=0;i<5;i++){ if(i==2) continue; cout<<i; }

12. Explain nested control structures with an example.

When one control structure is placed inside another.

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

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

- Function: A block of code that performs a specific task.
- Declaration: Tells the compiler about function (prototype).
  Example: int add(int, int);
- Definition: Actual body of the function.
  Example: int add(int a,int b){ return a+b; }
- Calling: Executes the function.
  Example: cout<<add(5,3);

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

- Scope defines the accessibility of variables.
- Local variables: Declared inside functions, accessible only within that function.
- Global variables: Declared outside all functions, accessible throughout the program.

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

Recursion: A function calling itself.

Example:

```
int factorial(int n){
  if(n==0) return 1;
  return n*factorial(n-1);
}
```

Calling factorial(5) gives 120.

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

Function prototype declares the function before its actual definition.
It tells compiler about return type and parameters.
Example: int add(int, int);
Used to allow calling functions before their definition.

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

- Array: Collection of elements of same data type stored in contiguous memory.
- Single-dimensional array: One row (e.g., int arr[5]).
- Multi-dimensional array: Multiple rows/columns (e.g., int mat[3][3]).

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

String handling is done using string class or C-style character arrays.
Example (C++ string): string name = "John"; cout<<name.length();
Example (C-style): char name[] = "John"; cout<<strlen(name);

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

- 1D Array: int arr[5] = {1,2,3,4,5};
- 2D Array: int mat[2][2] = {{1,2},{3,4}};

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

- Common string operations:
  length(): Returns size.
  append(): Concatenates.
  substr(): Extracts substring.
  compare(): Compares two strings.

Example: string s="Hello"; s.append(" World"); cout<<s;

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

- Key concepts:
  1. Encapsulation – Wrapping data & methods together.
  2. Abstraction – Hiding details, showing only essentials.
  3. Inheritance – Reusing existing classes.
  4. Polymorphism – One function/operator with multiple behaviors.

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

- Class: Blueprint of objects.
- Object: Instance of a class.

Example:
```
class Car{
  public: string brand;
  void start(){cout<<"Car started";}
};
Car c1; c1.start();
```

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

Inheritance allows a class (derived) to acquire properties of another (base).

Example:
```
class Person{public: string name;};
class Student: public Person{public:int roll;};
Student s; s.name="John"; s.roll=101;
```

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

Encapsulation is binding data and methods together, restricting direct access.
Achieved using classes and access specifiers (private, protected, public).

Example:

```
class Account{
 private: int balance;
 public: void setBalance(int b){ balance=b; }
 int getBalance(){ return balance; }
};
```