1. Explain main pillars of Object Oriented Programming.

Ans. Object-Oriented Programming (OOP) is a programming paradigm that is centered around the idea of objects, which are instances of classes. OOP focuses on the design and implementation of software systems by defining objects that encapsulate data and behavior. There are four main pillars of OOP, which are:

1. Encapsulation: Encapsulation refers to the practice of bundling data and behavior together within a class. This means that the data and behavior are hidden from the outside world and can only be accessed through methods provided by the class. Encapsulation helps to enforce data integrity and improves the security of the system by preventing external access to data.
2. Inheritance: Inheritance is the ability of a class to inherit properties and methods from another class. The class that is being inherited from is known as the parent class, while the class that inherits from it is known as the child class. Inheritance helps to promote code reuse and allows for the creation of more specialized classes based on existing ones.
3. Polymorphism: Polymorphism refers to the ability of objects to take on multiple forms. This means that an object can be treated as an instance of its own class or as an instance of one of its parent classes. Polymorphism allows for more flexible and extensible code.
4. Abstraction: Abstraction refers to the practice of hiding implementation details and focusing on the essential features of an object. This means that only the relevant information is presented to the user, while the underlying implementation details are hidden. Abstraction helps to reduce complexity and makes the system easier to understand and use.

These four pillars of OOP are interrelated and work together to provide a powerful and flexible programming paradigm for creating complex software systems.

2. Describe the need of header files in Object-Oriented Programming.

Ans. Header files in Object-Oriented Programming (OOP) serve an important purpose. They contain declarations of functions, classes, and other constructs that are needed by the program but are defined in other source files. The

need for header files arises from the fact that OOP programs are typically split across multiple source files, each containing a different class or set of related functions.

When a program is compiled, the compiler needs to know about all of the functions, classes, and other constructs that are used in the program. This information is stored in header files, which are included at the beginning of each source file that uses them. This allows the compiler to check that the program is using these constructs correctly, and to generate the appropriate code to implement them.

Header files also serve as a form of documentation. They provide a summary of the functions and classes that are defined in a program, and help developers to understand how these constructs are used. This is particularly important in large projects, where many developers may be working on different parts of the code. By including clear and informative header files, developers can easily understand how different parts of the program fit together and avoid introducing bugs or other problems.

In summary, header files are an essential part of OOP programming. They allow the compiler to check that a program is using functions, classes, and other constructs correctly, and serve as a form of documentation for developers. Without header files, it would be much more difficult to manage and understand complex OOP programs.

3. Compare C & C++ platforms.

Ans.

Sr.no	Features	C ++	C
1.	Object-Oriented Programming (OOP)	C++ is an object-oriented programming language. This means that C++ has features such as classes, inheritance, polymorphism, and encapsulation.	C is not oriented programming language. It doesn't have the features such as classes, inheritance, polymorphism, and encapsulation.
2.	Compiler	C++ is often considered a superset	C cannot compile C++ code. This means that

		of C, which means that C++ can compile C code.	developers who want to use C++ will need a C++ compiler, while those who only use C can use a C compiler.
3.	Memory Management	C++ offers a range of memory management features, such as dynamic memory allocation, which allows developers to allocate memory at runtime.	C does not have these features, and developers are responsible for managing memory manually.
4.	Standard Libraries	the C++ standard library is more extensive and includes features such as strings, streams, and containers.	They are not present in the C standard library.
5.	Performance	C++ can sometimes be slower than C, due to its more extensive runtime features and object-oriented design.	C is known for their high performance.

4. Define comment. Why is it important to write comments in programs?

Ans. In C++ OOP, a comment is a piece of text that is ignored by the compiler and is used to provide information about the code. There are two types of comments in C++:

1. Single-line comments: Single-line comments begin with two forward slashes (//) and continue until the end of the line. Anything after the // is ignored by the compiler.
2. Multi-line comments: Multi-line comments begin with /* and end with */. Anything between these two markers is ignored by the compiler.

Comments are an important part of programming because they provide information about the code that is not obvious from the code itself. Here are some reasons why it is important to write comments in programs:

1. Improve Code Readability: Comments can be used to explain the purpose of a function, class, or piece of code. This can make the code easier to read and understand, especially for developers who are new to the codebase.
2. Document Code Changes: Comments can be used to document changes to the codebase over time. This can help developers understand why a particular change was made and how it affects the code.
3. Debugging: Comments can be used to temporarily disable code during debugging. This allows developers to quickly test different parts of the code without having to delete or modify the code itself.
4. Collaboration: Comments can be used to communicate with other developers who are working on the same codebase. This can help to avoid misunderstandings and ensure that everyone is on the same page.

In summary, comments are an important part of programming in C++ OOP because they improve code readability, document code changes, aid in debugging, and facilitate collaboration among developers

5. in c++ Explain the concept of looping statements with example

Ans.

Looping statements in C++ are used to execute a block of code repeatedly until a certain condition is met. There are three types of looping statements in C++: for loop, while loop, and do-while loop.

Here are some examples of how to use each of these looping statements:

1. for loop: The for loop is used to iterate a specific number of times. It consists of three parts: initialization, condition, and increment/decrement. Here is an example of a for loop that prints the numbers from 1 to 10:

```
for(int i = 1; i <= 10; i++) {
    cout << i << " ";
}
```

Output: `1 2 3 4 5 6 7 8 9 10`

2. while loop: The while loop is used to iterate until a certain condition is met. Here is an example of a while loop that prints the numbers from 1 to 10:

```
int i = 1;
while(i <= 10) {
    cout << i << " ";
    i++;
}
```

Output: `1 2 3 4 5 6 7 8 9 10`

3. do-while loop: The do-while loop is similar to the while loop, but it executes the block of code at least once before checking the condition. Here is an example of a do-while loop that prints the numbers from 1 to 10:

```
int i = 1;
do {
    cout << i << " ";
    i++;
} while(i <= 10);
```

Output: `1 2 3 4 5 6 7 8 9 10`

These looping statements are commonly used in C++ programming to iterate through arrays, perform calculations, and implement control structures.

6. Create a class staff having fields: Staff_id, name, salary. Write a menu driven program for :
   i) To accept the data
   ii) To display the data

Ans. Here's an example of a C++ program that creates a class named **`Staff`** with the given fields **`Staff_id`**, **`name`**, and **`salary`**. The program then uses a menu-driven approach to accept and display the data.

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

class Staff {
  private:
    int staff_id;
    string name;
    double salary;

  public:
    void accept_data();
    void display_data();
};

void Staff::accept_data() {
  cout << "Enter Staff ID: ";
  cin >> staff_id;
  cout << "Enter Name: ";
  cin.ignore();
  getline(cin, name);
  cout << "Enter Salary: ";
  cin >> salary;
}
```

```
void Staff::display_data() {
  cout << "Staff ID: " << staff_id << endl;
  cout << "Name: " << name << endl;
  cout << "Salary: " << salary << endl;
}

int main() {
  Staff s;
  int choice;

  do {
    cout << "\n*****MENU*****\n";
    cout << "1. Accept Data\n";
    cout << "2. Display Data\n";
    cout << "3. Exit\n";
    cout << "Enter your choice (1-3): ";
    cin >> choice;
```

```
    switch(choice) {
      case 1:
        s.accept_data();
        break;
      case 2:
        s.display_data();
        break;
      case 3:
        cout << "Exiting program...\n";
        break;
      default:
        cout << "Invalid choice. Please try again.\n";
    }
  } while(choice != 3);

  return 0;
}
```

Explanation:

The `Staff` class has three private fields: `staff_id`, `name`, and `salary`. It also has two public methods: `accept_data()` and `display_data()`.

The `accept_data()` method prompts the user to input the staff's ID, name, and salary using the `cin` object. It uses `getline()` to read in the name as a string with spaces.

The `display_data()` method simply prints out the values of the three fields.

In the `main()` function, the program uses a `do-while` loop to display a menu of options to the user: accept data, display data, or exit the program. It reads in the user's choice using `cin` and uses a `switch` statement to call the appropriate method of the `Staff` object `s`.

The loop continues until the user chooses to exit the program by selecting option 3.

7. Describe with examples Inline function and static data member

Ans.

Inline function in C++: An inline function is a function that is expanded in-line when it is called, rather than executing a function call. This can result in faster code execution and reduced overhead, but it may also increase the size of the resulting executable file. In C++, a function can be declared as `inline` by prefixing the function definition with the keyword `inline`.

Example:

```
inline int add(int x, int y) {
  return x + y;
}

int main() {
  int a = 5, b = 10;
  int result = add(a, b);
  return 0;
}
```

In this example, the `add()` function is declared as `inline` by using the `inline` keyword before the function definition. When the `add()` function is called in `main()`, the compiler

replaces the function call with the actual function code, resulting in faster code execution.

Static data member in C++: A static data member is a variable that is associated with the class rather than individual objects of the class. There is only one instance of the static data member that is shared by all objects of the class. To declare a static data member in C++, the `static` keyword is used before the data member's type.

Example:

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

class MyClass {
  public:
    static int count;
    int value;

    MyClass(int v) {
      value = v;
      count++;
    }
};

int MyClass::count = 0;

int main() {
  MyClass a(5);
  MyClass b(10);
  MyClass c(15);
  cout << "Number of objects created: " << MyClass::count << endl;

  return 0;
}
```

In this example, the `MyClass` class has a static data member `count` that is initialized to 0 outside of the class definition. The constructor of `MyClass` increments the `count` static data member each time an object is created.

In `main()`, three `MyClass` objects are created, each with a different value. The `cout` statement prints out the number of objects created, which is stored in the `count` static data member. Since there are three objects created, the output is:

```
Number of objects created: 3
```

Note that the static data member `count` is accessed using the scope resolution operator `::` with the class name `MyClass`. This is because the static data member is associated with the class itself, rather than individual objects of the class.

8. Write a program to print factorial of given number using special functions constructor & destructor.

Ans.

Here's an example program that calculates the factorial of a given number using a constructor and destructor in C++:

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

class Factorial {
  private:
    int num;
    long long result;

  public:
    Factorial(int n); // Constructor
    ~Factorial(); // Destructor
    void calculate(); // Method to calculate factorial
};

Factorial::Factorial(int n) {
  num = n;
  result = 1;
  cout << "Creating object for " << num << endl;
}
```

```cpp
Factorial::~Factorial() {
  cout << "Destroying object for " << num << endl;
}

void Factorial::calculate() {
  for (int i = 1; i <= num; i++) {
    result *= i;
  }
}

int main() {
  int n;
  cout << "Enter a positive integer: ";
  cin >> n;

  Factorial f(n); // Create object

  f.calculate(); // Calculate factorial

  cout << "Factorial of " << n << " is " << f.result << endl;

  return 0;
}
```

Explanation:

The `Factorial` class has two private fields: `num` and `result`. `num` holds the number for which the factorial is to be calculated, and `result` holds the calculated factorial. The class has a constructor that initializes `num` to the input argument and `result` to 1. The constructor also prints a message indicating that an object has been created for the given number. The class also has a destructor that prints a message indicating that the object has been destroyed.

The `calculate()` method of the `Factorial` class uses a `for` loop to calculate the factorial of the number stored in `num` and stores the result in `result`.

In `main()`, the program prompts the user to enter a positive integer and stores the input in `n`. It then creates an object of the `Factorial` class with `n` as the argument. The

`calculate()` method is called on the object to calculate the factorial. Finally, the program prints the calculated factorial.

When the program finishes executing, the destructor of the `Factorial` object is automatically called, which prints a message indicating that the object has been destroyed.

9. Write a C++ program to overload binary operators '>' and '<' to compare two strings.

Ans.

Here's an example program that overloads the `>` and `<` operators to compare two strings in C++:

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

class String {
  private:
    string str;

  public:
    String(string s) {
      str = s;
    }

    bool operator>(String s) {
      if (str > s.str) {
        return true;
      } else {
        return false;
      }
    }
```

```
    bool operator<(String s) {
      if (str < s.str) {
        return true;
      } else {
        return false;
      }
    }
};

int main() {
  String s1("apple");
  String s2("banana");

  if (s1 > s2) {
    cout << s1 << " is greater than " << s2 << endl;
  } else if (s1 < s2) {
    cout << s1 << " is less than " << s2 << endl;
  } else {
    cout << s1 << " is equal to " << s2 << endl;
  }

  return 0;
}
```

Explanation:

The `String` class has one private field: `str`, which stores the string value. The class has a constructor that initializes `str` to the input argument.

The `>` and `<` operators are overloaded using member functions of the `String` class. The `>` operator compares the `str` value of the calling object with the `str` value of the input object `s`. If the `str` value of the calling object is greater than the `str` value of `s`, the function returns `true`. Otherwise, it returns `false`. The `<` operator works similarly, but returns `true` if the `str` value of the calling object is less than the `str` value of `s`.

In `main()`, two `String` objects `s1` and `s2` are created with the values "apple" and "banana", respectively. The `>` and `<` operators are used to compare the two objects. If `s1` is greater than `s2`, the program prints a message indicating that `s1` is greater than `s2`. If `s1` is less than `s2`, the program prints a message indicating that `s1` is less than `s2`. Otherwise, the program prints a message indicating that `s1` is equal to `s2`.

10. Explain the types of polymorphism with example.

Ans.

In C++, polymorphism is the ability of an object to take on multiple forms. There are two types of polymorphism in C++: compile-time polymorphism (also known as static polymorphism or function overloading) and runtime polymorphism (also known as dynamic polymorphism or function overriding).

1. Compile-time polymorphism:

Compile-time polymorphism is a type of polymorphism in which the compiler determines which function to call based on the number, types, and order of arguments passed to the function. This is achieved through function overloading, operator overloading, and template functions.

Function Overloading Example:

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

void print(int num) {
  cout << "Integer number: " << num << endl;
}

void print(double num) {
  cout << "Double number: " << num << endl;
}

int main() {
  print(5);
  print(3.14);

  return 0;
}
```

In this example, we have defined two functions named `print`. One function takes an integer argument and the other function takes a double argument. We call these functions with different arguments, and the compiler determines which function to call based on the argument types. This is an example of compile-time polymorphism.

2. Runtime polymorphism:

Runtime polymorphism is a type of polymorphism in which the function call is resolved at runtime based on the actual type of the object being pointed to or referred to. This is achieved through virtual functions and inheritance.

Virtual Function Example:

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

class Animal {
  public:
    virtual void sound() {
      cout << "This is an animal sound." << endl;
    }
};

class Dog : public Animal {
  public:
    void sound() {
      cout << "The dog barks." << endl;
    }
};

class Cat : public Animal {
  public:
    void sound() {
      cout << "The cat meows." << endl;
    }
```

```
};

int main() {
  Animal* animal;
  Dog dog;
  Cat cat;

  animal = &dog;
  animal->sound(); // Calls Dog's sound() function

  animal = &cat;
  animal->sound(); // Calls Cat's sound() function

  return 0;
}
```

In this example, we have defined a base class **`Animal`** with a virtual function **`sound`**. We also define two derived classes **`Dog`** and **`Cat`** that override the **`sound`** function. We then create a pointer of type **`Animal`** and assign it to a **`Dog`** object and a **`Cat`** object. When we call the **`sound`** function through the pointer, the function call is resolved at runtime based on the actual type of the object being pointed to or referred to. This is an example of runtime polymorphism.

11. Differentiate between multiple and multilevel inheritance in C++?

Ans.

In C++, inheritance is a feature that allows a class to inherit properties and behavior from another class. There are different types of inheritance in C++, such as single, multiple, multilevel, and hierarchical inheritance.

Here's the difference between multiple and multilevel inheritance in C++:

1. Multiple Inheritance:

Multiple inheritance is a type of inheritance in which a class can inherit properties and behavior from multiple base classes. In multiple inheritance, a derived class can inherit members from two or more base classes.

Example:

```
class A {
public:
  void printA() {
    cout << "A" << endl;
  }
};

class B {
public:
  void printB() {
    cout << "B" << endl;
  }
};

class C : public A, public B {
public:
  void printC() {
    cout << "C" << endl;
  }
};
```

```
int main() {
  C c;
  c.printA(); // A
  c.printB(); // B
  c.printC(); // C
  return 0;
}
```

In this example, class `C` is derived from two base classes `A` and `B`. This means that `C` inherits members from both `A` and `B`. The `printA()` and `printB()` functions can be called through an object of class `C`.

2. Multilevel Inheritance:

Multilevel inheritance is a type of inheritance in which a class is derived from a base class, which is itself derived from another base class. In multilevel inheritance, a derived class inherits from a base class that is itself a derived class.

Example:

```cpp
class A {
public:
  void printA() {
    cout << "A" << endl;
  }
};

class B : public A {
public:
  void printB() {
    cout << "B" << endl;
  }
};

class C : public B {
public:
  void printC() {
    cout << "C" << endl;
  }
};
```

```cpp
int main() {
  C c;
  c.printA(); // A
  c.printB(); // B
  c.printC(); // C
  return 0;
}
```

In this example, class `B` is derived from class `A`, and class `C` is derived from class `B`. This means that `C` inherits members from both `A` and `B`. The `printA()`, `printB()`, and `printC()` functions can be called through an object of class `C`.

In summary, multiple inheritance allows a class to inherit from two or more base classes, while multilevel inheritance involves a chain of derived classes where each class is derived from the previous one.

12. Write a C++ program to overload increment and decrement operator

Ans.

To overload the increment and decrement operators in C++, we need to define member functions of the class. Here's an example program that demonstrates how to overload the prefix increment and decrement operators:

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

class Counter {
private:
    int count;
public:
    Counter() {
        count = 0;
    }

    void display() {
        cout << "Count: " << count << endl;
    }

    Counter operator++() {
        count++;
        return *this;
    }

    Counter operator--() {
        count--;
```

```
        return *this;
    }
};

int main() {
    Counter c;
    c.display(); // Count: 0
    ++c;
    c.display(); // Count: 1
    ++c;
    c.display(); // Count: 2
    --c;
    c.display(); // Count: 1
    return 0;
}
```

In this example, we have defined a class `Counter` with a private member variable `count`. We have defined two member functions, `operator++()` and `operator--()`, which increment and decrement the `count` variable, respectively.

To overload the increment and decrement operators, we have used the operator functions `operator++()` and `operator--()`. These functions return a `Counter` object that has been incremented or decremented. In this example, we have overloaded the prefix increment and decrement operators.

In the `main()` function, we have created an object of the `Counter` class and called the `display()` function to display the initial count value. Then, we have used the overloaded increment and decrement operators to modify the count value and display it again using the `display()` function.

Output:

```
Count: 0
Count: 1
Count: 2
Count: 1
```

13. Explain the use of Friend Function in OOPL with example.

Ans.

In C++, a friend function is a function that is declared in a class and has access to its private and protected members. It can be used to provide external functions or classes with access to the private or protected members of a class.

Here's an example that demonstrates the use of a friend function in C++:

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

class MyClass {
private:
    int x;
public:
    MyClass(int a) {
        x = a;
    }

    friend void display(MyClass obj);
};

void display(MyClass obj) {
    cout << "The value of x is: " << obj.x << endl;
}

int main() {
    MyClass obj(10);
    display(obj); // The value of x is: 10
    return 0;
}
```

In this example, we have defined a class `MyClass` with a private member variable `x`. We have also declared a friend function `display()` inside the class. This means that the function `display()` has access to the private member `x`.

In the `main()` function, we have created an object of the `MyClass` class and passed it to the `display()` function. The `display()` function uses the `x` member of the `MyClass` object to display its value.

Output:

```
The value of x is: 10
```

14. What is Dynamic Binding?

Ans.

Dynamic binding is a mechanism in C++ that allows the selection of the appropriate method implementation at runtime, rather than at compile time. It is also known as late binding or runtime polymorphism. Dynamic binding is achieved through the use of virtual functions and pointers or references to objects.

In dynamic binding, the function call is resolved at runtime based on the actual type of the object pointed to by the pointer or reference. This allows us to write more flexible and extensible code because we can use a single function interface to handle different types of objects.

Here's an example that demonstrates dynamic binding in C++:

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

class Shape {
public:
    virtual void draw() {
        cout << "Drawing a shape." << endl;
    }
};

class Circle : public Shape {
public:
    void draw() {
        cout << "Drawing a circle." << endl;
    }
};

class Rectangle : public Shape {
public:
    void draw() {
        cout << "Drawing a rectangle." << endl;
```

```
    }
};

int main() {
    Shape* shape;
    Circle circle;
    Rectangle rectangle;

    shape = &circle;
    shape->draw(); // Drawing a circle.

    shape = &rectangle;
    shape->draw(); // Drawing a rectangle.

    return 0;
}
```

In this example, we have defined a base class `Shape` with a virtual function `draw()`. We have also defined two derived classes, `Circle` and `Rectangle`, that override the `draw()` function to draw a circle and a rectangle, respectively.

In the `main()` function, we have created two objects of the `Circle` and `Rectangle` classes and assigned their addresses to a pointer of type `Shape`. We have then called the `draw()` function through the pointer. The actual function that is called depends on the type of the object pointed to by the pointer. This is an example of dynamic binding.

Output:

```
Drawing a circle.
Drawing a rectangle.
```

In summary, dynamic binding is a powerful mechanism in C++ that allows us to write more flexible and extensible code. It is achieved through the use of virtual functions and pointers or references to objects.

15. in c++ What is the use of this pointer?

Ans.

In C++, `this` is a keyword that is used to refer to the current object within a member function of a class. It is a pointer to the object whose member function is being executed.

The primary use of the `this` pointer is to avoid naming conflicts between local variables and member variables of a class. When a member variable and a local variable have the same name, the local variable takes precedence over the member variable. To access the member variable, we can use the `this` pointer to refer to the current object.

Here's an example that demonstrates the use of the `this` pointer in C++:

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

class MyClass {
private:
    int x;
public:
    void setX(int x) {
        this->x = x;
    }

    void printX() {
        cout << "x = " << this->x << endl;
    }
};

int main() {
    MyClass obj;
    obj.setX(10);
    obj.printX(); // x = 10

    return 0;
}
```

In this example, we have defined a class `MyClass` with a private member variable `x`. We have also defined two member functions `setX()` and `printX()` that set and print the value of `x`, respectively. Inside these functions, we have used the `this` pointer to refer to the current object.

In the `main()` function, we have created an object of the `MyClass` class and called the `setX()` function to set the value of `x` to 10. We have then called the `printX()` function to print the value of `x`. The `this` pointer is used to access the member variable `x`.

Output:

```
x = 10
```

In summary, the `this` pointer is a powerful tool in C++ that allows us to refer to the current object within a member function of a class. It is primarily used to avoid naming conflicts between local variables and member variables of a class.

16. Explain the concept of Virtual Function.

Ans.

n C++, a virtual function is a member function that is declared in a base class and can be overridden in a derived class. When a virtual function is called through a pointer or reference to a base class, the actual function that is called is determined at runtime based on the type of the object pointed to by the pointer or reference. This is known as dynamic binding or late binding.

The primary use of virtual functions is to enable polymorphism, which allows a single function interface to handle different types of objects. Virtual functions are declared in the base class with the `virtual` keyword and are overridden in the derived classes with the `override` keyword.

Here's an example that demonstrates virtual functions in C++:

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

class Shape {
public:
    virtual void draw() {
        cout << "Drawing a shape." << endl;
    }
};

class Circle : public Shape {
public:
    void draw() override {
        cout << "Drawing a circle." << endl;
    }
};

class Rectangle : public Shape {
public:
    void draw() override {
        cout << "Drawing a rectangle." << endl;
    }
};
```

```cpp
int main() {
    Shape* shape;
    Circle circle;
    Rectangle rectangle;

    shape = &circle;
    shape->draw(); // Drawing a circle.

    shape = &rectangle;
    shape->draw(); // Drawing a rectangle.

    return 0;
}
```

In this example, we have defined a base class `Shape` with a virtual function `draw()`. We have also defined two derived classes, `Circle` and `Rectangle`, that override the `draw()` function to draw a circle and a rectangle, respectively.

In the `main()` function, we have created two objects of the `Circle` and `Rectangle` classes and assigned their addresses to a pointer of type `Shape`. We have then called the `draw()` function through the pointer. The actual function that is called depends on the type of the object pointed to by the pointer. This is an example of dynamic binding.

Output:

```
Drawing a circle.
Drawing a rectangle.
```

In summary, virtual functions are a powerful mechanism in C++ that allow us to achieve polymorphism and dynamic binding. They are declared in the base class with the `virtual` keyword and are overridden in the derived classes with the `override` keyword. Virtual functions enable us to write more flexible and extensible code by using a single function interface to handle different types of objects.

17. Why templates are used in C++? How many types of templates are there in C++?

Ans.

Templates are used in C++ to create generic functions and classes that can work with different data types without the need to write multiple functions or classes for each data type. Templates provide a way to parameterize types, allowing us to write code that is more flexible, reusable, and efficient.

There are two types of templates in C++: function templates and class templates.

1. Function templates: A function template is a generic function that can work with different data types. It is defined with a type parameter, which can be used to specify the type of the arguments and the return type of the function. Here's an example of a function template that calculates the maximum of two values:

```
template<typename T>
T max(T a, T b) {
    return (a > b) ? a : b;
}
```

In this example, `typename T` is the type parameter that specifies the data type of the arguments and the return type of the function. The `max()` function can be called with any data type that supports the comparison operator (`>`), such as `int`, `float`, `double`, etc.

2. Class templates: A class template is a generic class that can work with different data types. It is defined with a type parameter, which can be used to specify the data type of the member variables and the member functions of the class. Here's an example of a class template that implements a stack:

```
template<typename T>
class Stack {
private:
    T data[100];
    int top;

public:
    Stack() {
        top = -1;
    }

    void push(T value) {
        data[++top] = value;
    }

    T pop() {
        return data[top--];
    }

    bool empty() {
        return top == -1;
    }
};
```

In this example, `typename T` is the type parameter that specifies the data type of the stack elements. The `Stack` class can be instantiated with any data type, such as `int`, `float`, `double`, etc.

In summary, templates are used in C++ to create generic functions and classes that can work with different data types. Function templates and class templates are the two types of templates in C++. Templates provide a way to write code that is more flexible, reusable, and efficient.

18. Write a C++ Program to find Largest among two numbers using function template.

Ans.

Here's an example program in C++ that uses a function template to find the largest among two numbers:

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

// Function template to find the largest among two numbers
template<typename T>
T findLargest(T a, T b) {
    return (a > b) ? a : b;
}

int main() {
    int num1, num2;
    cout << "Enter two numbers: ";
    cin >> num1 >> num2;

    // Call the function template to find the largest among two numbers
    int largest = findLargest(num1, num2);

    cout << "The largest number is: " << largest << endl;

    return 0;
}
```

In this example, we have defined a function template `findLargest()` that takes two arguments of the same data type and returns the largest among them. The `typename T` specifies the type parameter that can be used to represent any data type.

In the `main()` function, we have declared two integer variables `num1` and `num2` and read their values from the user. We then call the `findLargest()` function template by passing the two integer variables as arguments. The `findLargest()` function template automatically resolves to the `int` data type because we have passed integer arguments. Finally, we display the largest number using the `cout` statement.

Note that the function template can work with other data types, such as `float`, `double`, etc., as long as we pass the same data type arguments.

19. in C++ Explain how exception handling mechanism can be used for debugging a program.

Ans.

Exception handling mechanism in C++ provides a way to handle runtime errors that may occur during the execution of a program. It allows you to catch and handle exceptions gracefully, instead of terminating the program abruptly.

Using exception handling mechanism, you can also debug your program by identifying and handling the runtime errors that occur during program execution. When an exception is thrown, the program can catch and handle it in a way that can help you identify the cause of the error.

Here's an example of how exception handling can be used for debugging a program:

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

int main() {
    int num1, num2;

    try {
        cout << "Enter two numbers: ";
        cin >> num1 >> num2;

        if (num2 == 0) {
            throw "Division by zero";
        }

        int result = num1 / num2;
        cout << "Result: " << result << endl;
    }
    catch (const char* msg) {
        cerr << "Error: " << msg << endl;
    }

    return 0;
}
```

In this example, we have used a `try` block to catch any exceptions that might occur during the program execution. Within the `try` block, we have taken two integer inputs from the user and checked if the second number is zero. If it is zero, we have thrown an exception with a message "Division by zero".

In the `catch` block, we have caught the exception and displayed an error message "Error: Division by zero". Using this mechanism, we can handle the exception gracefully instead of letting the program terminate abruptly.

By using exception handling mechanism, we can debug our program by identifying the runtime errors that occur during program execution. We can also add multiple `catch` blocks to handle different types of exceptions, and perform appropriate actions based on the type of exception caught.

20. What is generic programming? How is it implemented in C++?

Ans.

Generic programming is a programming paradigm in which the algorithms and data structures are written in such a way that they can be used with different data types without the need for the programmer to write separate code for each data type. It allows code reuse, reduces code duplication, and improves program efficiency.

In C++, generic programming is implemented using templates. A template is a mechanism that allows a programmer to define a class or a function with generic types, which can be used with different data types. Templates are defined using the `template` keyword, followed by the template parameter list, and then the class or function definition.

Here's an example of a template function that can be used with different data types:

```
template<typename T>
T getMax(T a, T b) {
    return (a > b) ? a : b;
}
```

In this example, we have defined a function template `getMax()` that takes two arguments of the same data type and returns the largest among them. The `typename T` specifies the type parameter that can be used to represent any data type.

To use this function template, we can call it with different data types, such as `int`, `float`, `double`, etc., as shown below:

```
int num1 = 10, num2 = 20;
int maxNum = getMax(num1, num2);
cout << "The maximum number is: " << maxNum << endl;

float float1 = 3.14, float2 = 2.71;
float maxFloat = getMax(float1, float2);
cout << "The maximum float number is: " << maxFloat << endl;
```

In this way, we can use the same function template `getMax()` with different data types without writing separate code for each data type. This is the essence of generic programming in C++.

21. Discuss the various forms of get() function supported by the input stream. How are they used in C++?

Ans.

The `get()` function is a member function of the input stream classes in C++, such as `istream`, `ifstream`, and `istringstream`. It is used to extract characters from the input stream. There are several forms of the `get()` function that are supported by the input stream, which are as follows:

1. `get()` with no arguments: This form of the `get()` function is used to extract the next character from the input stream and returns it as an integer value. It can be used as follows:

```
char ch = cin.get();
```

This statement extracts the next character from the standard input stream and assigns it to the variable `ch`.

2. `get(char& c)`: This form of the `get()` function is used to extract the next character from the input stream and stores it in the variable `c`. It returns a reference to the input stream object, which can be used to chain input operations. It can be used as follows:

```
char ch;
cin.get(ch);
```

This statement extracts the next character from the standard input stream and stores it in the variable `ch`.

3. `get(char* s, streamsize n)`: This form of the `get()` function is used to extract up to `n-1` characters from the input stream and stores them in the character array `s`, followed by a null character. It returns a reference to the input stream object, which can be used to chain input operations. It can be used as follows:

```
char str[20];
cin.get(str, 20);
```

This statement extracts up to 19 characters from the standard input stream and stores them in the character array `str`, followed by a null character.

4. `get(streambuf& sb)`: This form of the `get()` function is used to extract the next character from the input stream and inserts it into the stream buffer `sb`. It returns an integer value that represents the extracted character. It can be used as follows:

```
streambuf sb;
cin.get(sb);
```

This statement extracts the next character from the standard input stream and inserts it into the stream buffer `sb`.

These different forms of the `get()` function provide flexibility in extracting characters from the input stream in different ways, depending on the requirements of the program. They can be used to read input from the standard input stream, files, or other input sources, and to parse and process the input data in various ways.

22. What is a file mode? Describe the various file mode options available

Ans.

In C++, a file mode is used to specify the mode in which a file should be opened. The file mode determines whether the file will be opened for reading, writing, or both, as well as whether the file should be created or truncated if it already exists.

There are several file mode options available in C++, which are specified as a string parameter to the `fstream` constructor or the `open()` method. The following are the various file mode options available in C++:

1. `ios::in`: This mode is used to open a file for input operations, which means that the file will be read-only. If the file does not exist, an error will occur.
2. `ios::out`: This mode is used to open a file for output operations, which means that the file will be write-only. If the file does not exist, it will be created. If the file already exists, its contents will be truncated before writing.
3. `ios::app`: This mode is used to open a file for output operations in append mode, which means that the output will be appended to the end of the file. If the file does not exist, it will be created.
4. `ios::binary`: This mode is used to open a file in binary mode, which means that the file will be read or written in binary format, rather than in text format. This is useful for handling non-text data, such as images, audio, and video.
5. `ios::ate`: This mode is used to open a file with the initial position at the end of the file, which means that any output operations will be appended to the end of the file. This mode is often used for appending data to a file without overwriting its contents.
6. `ios::trunc`: This mode is used to truncate an existing file to zero length before opening it for output operations. If the file does not exist, it will be created.

These file mode options provide flexibility in opening and manipulating files in different ways, depending on the requirements of the program. They can be used to read input from files, write output to files, append data to files, and handle binary data in various formats.

23. What is input stream and output stream? Explain various methods to open a file

Ans.

In C++, an input stream and an output stream are objects that are used to read input from and write output to external files or devices.

An input stream is an object that is used to read data from a file or other input source. The `istream` class provides input stream functionality in C++, and it provides various methods for reading input, such as `get()`, `getline()`, `ignore()`, and `peek()`.

An output stream is an object that is used to write data to a file or other output destination. The `ostream` class provides output stream functionality in C++, and it provides various methods for writing output, such as `put()`, `write()`, and `flush()`.

To open a file in C++, there are several methods available, which are described below:

1. `fstream` constructor: This method is used to open a file using a `fstream` object, which can be used for both input and output operations. The `fstream` constructor takes two parameters: the name of the file to be opened, and the mode in which the file should be opened. For example, to open a file named "example.txt" for input and output operations, the following code can be used:

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

int main() {
    fstream file("example.txt", ios::in | ios::out);
    // ... use file for input and output operations
    file.close();
    return 0;
}
```

2. `ifstream` and `ofstream` constructors: These methods are used to open a file using separate `ifstream` and `ofstream` objects, which can be used for input and output

operations respectively. The `ifstream` constructor takes a file name parameter and opens the file for input operations, while the `ofstream` constructor takes a file name parameter and opens the file for output operations. For example, to open a file named "input.txt" for input operations and a file named "output.txt" for output operations, the following code can be used:

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

int main() {
    ifstream input("input.txt");
    ofstream output("output.txt");
    // ... use input and output objects for reading and writing
    input.close();
    output.close();
    return 0;
}
```

3. `open()` method: This method is used to open a file using an existing `fstream`, `ifstream`, or `ofstream` object. The `open()` method takes two parameters: the name of the file to be opened, and the mode in which the file should be opened. For example, to open a file named "data.bin" in binary mode using an existing `fstream` object, the following code can be used:

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

int main() {
    fstream file;
    file.open("data.bin", ios::binary | ios::in | ios::out);
    // ... use file object for binary input and output operations
    file.close();
    return 0;
}
```

These are some of the methods available for opening files in C++. By using these methods, files can be opened and read from or written to using input and output streams, providing a powerful and flexible means of working with external data sources.

24. Write a C++ program to read a list containing item name, item code, and cost interactively and produce a three column output as shown below

Item Name	Item Code	Cost
Database	1006	550.95
Java Programming	905	99.70

Ans.

Here is the C++ program that reads a list containing item name, item code, and cost interactively and produces a three-column output:

```cpp
#include <iostream>
#include <iomanip>
#include <string>

using namespace std;

int main() {
    string name, code;
    double cost;

    cout << "Enter item details (name, code, cost):" << endl;

    // Read input data until end-of-file (EOF) is encountered
    while (cin >> name >> code >> cost) {
        // Display the item details in the specified format
        cout << setw(20) << left << name
             << setw(10) << left << code
             << setw(10) << right << fixed << setprecision(2) << cost
             << endl;
    }

    return 0;
}
```

Here's an example input and output:

```
Enter item details (name, code, cost):
Item1 Code1 10.50
Item2 Code2 20.00
Item3 Code3 5.25
Item4 Code4 12.75
```

Output:

```
Item1           Code1          10.50
Item2           Code2          20.00
Item3           Code3           5.25
Item4           Code4          12.75
```

25. Explain terms - Polymorphism, Inheritance, Abstraction

Ans.

Polymorphism, inheritance, and abstraction are three fundamental concepts of object-oriented programming (OOP) that help developers to create complex and efficient software applications.

1. Polymorphism: Polymorphism means "many forms" and refers to the ability of an object to take on multiple forms or behaviors. In C++, polymorphism is implemented through virtual functions and function overloading. Virtual functions allow a derived class to provide its own implementation of a function that is already defined in the base class, while function overloading allows multiple functions with the same name to be defined, but with different parameter types.
2. Inheritance: Inheritance is a mechanism in which one class (called the derived class) can inherit properties and methods from another class (called the base class). In C++, inheritance is implemented through the `class` keyword, where the derived class inherits the members (i.e., variables and functions) of the base class using the `public`, `private`, or `protected` access specifiers.
3. Abstraction: Abstraction is a process of hiding complex details and presenting only the essential features to the user. It is achieved in C++ through the use of abstract classes and interfaces. An abstract class is a class that cannot be instantiated but can be used as a base class for other classes, while an interface is a collection of abstract methods that define a set of behaviors that a class must implement.

In summary, polymorphism allows objects to take on multiple forms, inheritance allows a derived class to inherit properties and methods from a base class, and abstraction allows for the hiding of complex details and presentation of only the essential features

to the user. These three concepts are fundamental to the design and implementation of effective object-oriented software systems.

26. in C++ Create a class staff having fields: Staff_id, name, salary. Write a menu driven program for : i) To accept the data

ii) To display the data

Ans.

Here's a sample code that implements the menu-driven program for accepting and displaying data for the `Staff` class:

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

class Staff {
private:
    int Staff_id;
    string name;
    double salary;
public:
    void acceptData();
    void displayData();
};

void Staff::acceptData() {
    cout << "Enter Staff ID: ";
    cin >> Staff_id;
    cout << "Enter name: ";
    cin >> name;
    cout << "Enter salary: ";
    cin >> salary;
}
```

```
void Staff::displayData() {
    cout << "Staff ID: " << Staff_id << endl;
    cout << "Name: " << name << endl;
    cout << "Salary: " << salary << endl;
}

int main() {
    int choice;
    Staff staff;

    do {
        cout << "\nMENU\n";
        cout << "1. Accept data\n";
        cout << "2. Display data\n";
        cout << "3. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch(choice) {
            case 1:
                staff.acceptData();
                break;
```

```
            case 2:
                staff.displayData();
                break;
            case 3:
                cout << "Exiting...\n";
                break;
            default:
                cout << "Invalid choice! Try again.\n";
        }
    } while (choice != 3);

    return 0;
}
```

In this program, the `Staff` class has three private data members: `Staff_id`, `name`, and `salary`. It also has two public member functions: `acceptData()` and `displayData()`, which are used to accept and display the data respectively.

In the `main()` function, a `Staff` object is created, and a menu is displayed to the user using a `do-while` loop. Depending on the user's choice, the corresponding function is called. The program continues to loop until the user chooses to exit.

27. Write a program to print factorial of given number using special functions constructor & destructor.

Ans.

Here is a C++ program to print the factorial of a given number using constructor and destructor:

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

class Factorial {
    int num;
    int fact;

    public:
        Factorial(int n) { //constructor to calculate factorial
            num = n;
            fact = 1;
            for (int i = 1; i <= num; i++) {
                fact *= i;
            }
        }

        void display() { //function to display factorial
            cout << "Factorial of " << num << " is: " << fact << endl;
        }

        ~Factorial() { //destructor
            cout << "Object is destroyed." << endl;
```

```
        }
};

int main() {
    int num;
    cout << "Enter a number: ";
    cin >> num;
    Factorial f(num); //creating object of Factorial class
    f.display(); //displaying factorial
    return 0;
}
```

Explanation:

- We have created a class `Factorial` with two private data members `num` and `fact`.
- The constructor `Factorial(int n)` is used to calculate the factorial of the given number `n` using a `for` loop and stores the value in `fact`.
- The public function `display()` is used to display the factorial value.
- The destructor `~Factorial()` is used to display a message when the object is destroyed.
- In the `main()` function, we accept a number from the user and create an object of the `Factorial` class by passing the number as a parameter.
- Then we call the `display()` function to display the factorial value and the destructor is called when the program terminates.

28. Explain how exception handling mechanism can be used for debugging a C++ program

Ans.

In C++, exception handling is a mechanism that allows you to handle runtime errors in a structured manner. With exception handling, you can catch and handle errors that occur during the execution of a program.

Exception handling can be used for debugging a C++ program in the following ways:

1. Identifying the location of the error: When an exception is thrown, the program stops executing normally and control is transferred to the exception handler. By examining the exception information, you can identify the location of the error in the program.

2. Providing useful error messages: When an exception is caught, you can display a useful error message to the user, explaining the nature of the error and suggesting how to correct it.
3. Providing graceful recovery: Exception handling allows you to gracefully recover from errors and continue the execution of the program. This can prevent the program from crashing and allow the user to continue working with the program.

Here's an example of how exception handling can be used for debugging a C++ program:

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

int main() {
   int x, y;
   cout << "Enter two numbers: ";
   cin >> x >> y;

   try {
      if (y == 0) {
         throw "Division by zero error";
      }
      int result = x / y;
      cout << "Result = " << result << endl;
   }
   catch (const char* error) {
      cerr << "Error: " << error << endl;
   }
   return 0;
}
```

In the above example, if the user enters 0 as the second number, a "Division by zero error" exception is thrown. The exception is caught in the catch block, which displays an error message to the user. This way, the user is informed of the error and can correct the input values.

28. Discuss the various forms of get() function supported by the input stream. How are they used C++ program?

Ans.

In C++, the get() function is a member function of the istream class and is used to extract characters from the input stream. The get() function can be used in different forms based on the input parameters passed to it.

The various forms of the get() function are:

1. get(): This form of the get() function is used to extract a single character from the input stream.
2. get(char& ch): This form of the get() function is used to extract a single character from the input stream and store it in the variable 'ch'.
3. get(char* str, int n, char delim): This form of the get() function is used to extract a string of characters from the input stream and store it in the character array 'str'. It extracts 'n' characters or until the delimiter character 'delim' is encountered, whichever comes first.
4. getline(char* str, int n): This form of the get() function is used to extract a line of characters from the input stream and store it in the character array 'str'. It extracts 'n' characters or until the end of the line is encountered, whichever comes first.

These different forms of the get() function can be used in a C++ program to extract input from the user through the console and process it accordingly. For example, the get() function can be used to read user input and store it in a string variable, as shown below:

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

int main() {
   char name[20];
   cout << "Enter your name: ";
   cin.get(name, 20);
   cout << "Your name is: " << name << endl;
   return 0;
}
```

In the above program, the get() function is used to read the user's name from the console and store it in the 'name' character array. The name is then printed to the console using cout.

29. What is input stream and output stream? Explain various methods to open a file.

Ans.

In C++, an input stream is a flow of data from an input source (such as a file, keyboard, or network connection) into a program, while an output stream is a flow of data from a program to an output destination (such as a file, console, or network connection).

To open a file in C++, there are various methods:

1. Using the constructor of the file stream class: You can create an object of the ifstream (for input) or ofstream (for output) class by specifying the file name and the file mode. For example:

```
ifstream inputFile("filename.txt", ios::in);
ofstream outputFile("filename.txt", ios::out);
```

Here, `ios::in` and `ios::out` are the file modes for input and output, respectively.

2. Using the open() function: You can use the `open()` function of the file stream class to open a file. For example:

```
ifstream inputFile;
inputFile.open("filename.txt", ios::in);
ofstream outputFile;
outputFile.open("filename.txt", ios::out);
```

3. Using the fopen() function: You can also use the `fopen()` function from the C standard library to open a file. For example:

```
FILE* inputFile;
inputFile = fopen("filename.txt", "r");
FILE* outputFile;
outputFile = fopen("filename.txt", "w");
```

Here, "r" and "w" are the file modes for reading and writing, respectively.

It's important to note that when using files in C++, you should always close the file after use, using the `close()` function for file stream classes or `fclose()` function for `FILE` pointers.