CS304
Object Oriented Programming

Important subjective

Lec 23 - ACCESSING BASE CLASS MEMBER FUNCTIONS IN DERIVED CLASS

1. **What is the scope resolution operator and how is it used to access base class member functions in a derived class?**
   Answer: The scope resolution operator (::) is used to specify the scope of a member function or variable. To access a base class member function in a derived class, the derived class can use the scope resolution operator followed by the name of the function and the base class name.

   Can a derived class access a private member function of its base class? If so, how?
   Answer: No, a derived class cannot access a private member function of its base class directly. It can only access the base class member function through a public or protected member function or by declaring the derived class as a friend class of the base class.

   How does method overriding work in a derived class?
   Answer: Method overriding is when a derived class defines a member function with the same name as a member function in the base class. The derived class can modify the behavior of the base class member function by providing a new implementation. When the derived class object calls the member function, the derived class implementation is executed.

   Can a derived class modify a base class member function?
   Answer: No, a derived class cannot modify a base class member function directly. It can only modify the behavior of the base class member function by providing a new implementation in the derived class.

   What is virtual function and how is it used in a derived class to override a base class member function?
   Answer: A virtual function is a member function in a base class that can be overridden by a member function in a derived class. To override a base class virtual function, the derived class should provide a new implementation of the virtual function with the same signature as the base class virtual function.

   How is the order in which base classes are specified in a derived class declaration related to the order in which their constructors are called?
   Answer: The order in which base classes are specified in a derived class declaration determines the order in which their constructors are called. The base class constructors are called in the order they are specified in the derived class declaration, regardless of the order in which they are inherited.

   Can a derived class access private member variables of its base class?
   Answer: No, a derived class cannot access the private member variables of its base class directly. It can only access them through public or protected member functions or by declaring

the derived class as a friend class of the base class.

What is the difference between public, private, and protected inheritance?
Answer: Public inheritance makes public members of the base class accessible in the derived class, protected inheritance makes protected members of the base class accessible in the derived class, and private inheritance makes both public and protected members of the base class private in the derived class.

How does a derived class call a base class constructor?
Answer: A derived class can call a base class constructor explicitly in its own constructor by using the base class name followed by the constructor arguments in the member initializer list of the derived class constructor.

Can a derived class have its own private member functions and variables in addition to those inherited from the base class?
Answer: Yes, a derived class can have its own private member functions and variables in addition to those inherited from the base class.

Lec 24 - MODIFIED DEFAULT CONSTRUCTOR

1. **What is a modified default constructor?**
   Answer: A modified default constructor is a default constructor that has been customized to initialize the class data members to specific values.

   How is a modified default constructor defined?
   Answer: A modified default constructor is defined using the syntax ClassName::ClassName() { // code to initialize data members }

   Can a modified default constructor take arguments?
   Answer: No, a modified default constructor takes no arguments.

   What is the purpose of a default constructor?
   Answer: The purpose of a default constructor is to create an object of the class with default values for its data members.

   What happens if a modified default constructor is not defined for a class?
   Answer: If a modified default constructor is not defined for a class, the compiler will provide a default constructor that initializes the class data members to default values.

   Can a modified default constructor be overloaded?
   Answer: Yes, a modified default constructor can be overloaded to take different arguments or initialize data members in different ways.

   What is the difference between a default constructor and a modified default constructor?
   Answer: A default constructor initializes the class data members to default values, while a modified default constructor initializes them to specific values.

   Can a class have multiple constructors?
   Answer: Yes, a class can have multiple constructors, including default constructors, copy constructors, and others.

   What is the syntax for calling a modified default constructor?
   Answer: To call a modified default constructor, simply create an object of the class using the default constructor syntax: ClassName obj;

   When is a default constructor provided by the compiler?
   Answer: A default constructor is provided by the compiler when no other constructors are defined for the class.

Lec 25 - OVERLOADING VS. OVERRIDING

1. **What is overloading and overriding in object-oriented programming?**
   Answer: Overloading refers to creating multiple functions with the same name but different parameters, while overriding refers to redefining a function in a subclass that was originally defined in a superclass.

   What is the difference between overloading and overriding?
   Answer: Overloading is creating multiple functions with the same name but different parameters, while overriding is redefining a function in a subclass that was originally defined in a superclass.

   What is the purpose of overloading in object-oriented programming?
   Answer: Overloading allows a function to perform different tasks based on the parameters it is called with.

   What is the purpose of overriding in object-oriented programming?
   Answer: Overriding allows a subclass to provide a different implementation of a function defined in the superclass.

   Is overloading static or dynamic polymorphism?
   Answer: Overloading is an example of static polymorphism.

   Is overriding static or dynamic polymorphism?
   Answer: Overriding is an example of dynamic polymorphism.

   Can overloaded functions have different return types?
   Answer: Yes, overloaded functions can have different return types as long as their parameter lists differ.

   Can overridden functions have different return types?
   Answer: No, overridden functions must have the same return type as the function they are overriding.

   Can overloaded functions have different access modifiers?
   Answer: Yes, overloaded functions can have different access modifiers.

   Can overridden functions have different access modifiers?
   Answer: No, overridden functions must have the same access modifier as the function they are overriding.

Lec 26 - BASE INITIALIZATION

1. **What is base initialization and how is it used in C++?**
   Answer: Base initialization is a mechanism used to initialize the data members of a base class in a derived class. It is done by calling the constructor of the base class in the initialization list of the derived class constructor.

   How is base initialization different from default initialization?
   Answer: Base initialization initializes the data members of the base class, while default initialization initializes the data members of the derived class.

   Why is base initialization important in C++?
   Answer: Base initialization is important because it ensures that the data members of the base class are properly initialized before the derived class constructor is called.

   Can base initialization be used to initialize data members of both the base class and derived class?
   Answer: No, base initialization can only be used to initialize the data members of the base class.

   What is the syntax for using base initialization in C++?
   Answer: The syntax for using base initialization in C++ is: derived_class::derived_class(arg1, arg2, ...): base_class(arg1, arg2, ...){...}

   What is the order of execution for constructors when base initialization is used?
   Answer: The order of execution for constructors when base initialization is used is: base class constructor(s) followed by derived class constructor.

   Can base initialization be used to initialize const data members in the base class?
   Answer: Yes, base initialization can be used to initialize const data members in the base class.

   When should base initialization be used in C++?
   Answer: Base initialization should be used in C++ when the base class has const data members that cannot be initialized in the derived class constructor, or when the derived class needs to initialize data members that are dependent on the values of the base class data members.

   How does base initialization improve performance in C++?
   Answer: Base initialization can improve performance in C++ by avoiding unnecessary default constructor calls for base class data members.

   What happens if a derived class constructor does not explicitly call the base class constructor using base initialization?
   Answer: If a derived class constructor does not explicitly call the base class constructor using base initialization, the default constructor of the base class is called.

Lec 27 - SPECIALIZATION (RESTRICTION)

1. **What is specialization in C++?**
   Answer: Specialization in C++ is a mechanism that allows programmers to define a different implementation of a template or function for a specific set of arguments.

   Why is specialization useful in C++?
   Answer: Specialization is useful in C++ when the default behavior of a template or function is not suitable for a particular data type or value.

   What are the restrictions of specialization in C++?
   Answer: Some of the restrictions of specialization in C++ include: you cannot partially specialize function templates, you cannot specialize function templates for built-in types, and specialization can lead to code duplication and maintenance issues.

   How does specialization help in C++?
   Answer: Specialization helps in C++ by allowing programmers to define a different implementation of a template or function for a specific set of arguments.

   What is partial specialization in C++?
   Answer: Partial specialization in C++ is a technique that allows programmers to define a specialized implementation of a template or function for a subset of the arguments.

   Can you partially specialize class templates in C++?
   Answer: Yes, you can partially specialize class templates in C++.

   What is the syntax for specialization in C++?
   Answer: The syntax for specialization in C++ is: template <>
   function_name<>(){/implementation/}

   What is explicit specialization in C++?
   Answer: Explicit specialization in C++ is a technique that allows programmers to provide a separate implementation for a specific set of template arguments.

   Can you specialize a non-template function in C++?
   Answer: No, you cannot specialize a non-template function in C++.

   What is a specialization hierarchy in C++?
   Answer: A specialization hierarchy in C++ is a set of specialized implementations of a template or function that are ordered from the most general to the most specific.

Lec 28 - VIRTUAL FUNCTIONS

1. **What is a virtual function in C++ and how is it declared?**
   Answer: A virtual function is a function that can be overridden by a derived class. It is declared using the virtual keyword before the function prototype in the base class.

   What is the difference between a virtual function and a non-virtual function in C++?
   Answer: A virtual function can be overridden by a derived class, while a non-virtual function cannot be overridden.

   Can a virtual function be static or friend in C++?
   Answer: No, a virtual function cannot be static or friend in C++.

   What is the purpose of a pure virtual function in C++ and how is it declared?
   Answer: A pure virtual function is a virtual function that has no implementation in the base class and must be overridden by the derived class. It is declared using the syntax virtual void functionName() = 0;

   Can a virtual function be defined outside of its class in C++?
   Answer: Yes, a virtual function can be defined outside of its class in C++.

   Can a constructor or destructor be virtual in C++?
   Answer: Yes, a constructor or destructor can be virtual in C++.

   What is a virtual function table (vtable) in C++?
   Answer: A virtual function table is a table that stores the addresses of all virtual functions in a class hierarchy.

   What is the difference between early binding and late binding in C++?
   Answer: Early binding is when the function call is resolved at compile-time based on the type of the object pointer, while late binding is when the function call is resolved at runtime based on the type of the object pointed to.

   What is the difference between function overloading and function overriding in C++?
   Answer: Function overloading is when multiple functions have the same name but different parameters, while function overriding is when a derived class provides its own implementation of a virtual function in the base class.

   What is the purpose of a virtual destructor in C++?
   Answer: The purpose of a virtual destructor is to ensure that the destructor of the derived class is called when a derived class object is deleted through a pointer to the base class.

Lec 29 - ABSTRACT CLASSES

1. **What is an abstract class?**
   An abstract class is a class that cannot be instantiated and is used as a base class for creating derived classes. It contains one or more abstract methods, which have no implementation in the base class but are implemented in the derived classes.
   How is an abstract class different from a concrete class?
   An abstract class cannot be instantiated, whereas a concrete class can be instantiated.
   An abstract class may contain one or more abstract methods, whereas a concrete class does not contain any abstract methods.
   An abstract class may contain both abstract and non-abstract methods, whereas a concrete class contains only non-abstract methods.
   Can an abstract class have a constructor?
   Yes, an abstract class can have a constructor. However, it cannot be used to instantiate an object of the abstract class. The constructor is used to initialize the members of the class when a derived class object is created.
   How do you declare an abstract method in a class?
   To declare an abstract method in a class, you need to use the abstract keyword before the method declaration. For example: abstract void methodName();
   Can a concrete class inherit from an abstract class?
   Yes, a concrete class can inherit from an abstract class. However, the concrete class must implement all the abstract methods of the abstract class.
   Can an abstract class implement an interface?
   Yes, an abstract class can implement an interface. In this case, the abstract class must provide implementations for all the methods in the interface.
   What is the purpose of an abstract class?
   The purpose of an abstract class is to provide a common base for a set of related classes. It defines the common behavior and properties of the derived classes and provides a framework for implementing the behavior.
   Can an abstract class have a final method?
   Yes, an abstract class can have a final method. However, the method cannot be abstract.
   Can an abstract class have a static method?
   Yes, an abstract class can have a static method. However, the method cannot be abstract.
   Can an abstract class be marked as final?
   No, an abstract class cannot be marked as final because it is meant to be inherited by other classes.

Lec 30 - POLYMORPHISM – CASE STUDY: A SIMPLE PAYROLL APPLICATION

1. **What is the advantage of using polymorphism in a payroll application?**
   Answer: The advantage of using polymorphism is that it allows for flexibility and extensibility in the payroll application. By using polymorphism, new employee types can be added without having to modify the existing code.

   How can you implement polymorphism in a payroll application?
   Answer: Polymorphism can be implemented in a payroll application by using inheritance and virtual functions. The base class can have virtual functions that are overridden by the derived classes. This allows the application to call the appropriate function based on the type of the employee.

   What is the difference between static and dynamic polymorphism?
   Answer: Static polymorphism is resolved at compile-time, while dynamic polymorphism is resolved at run-time. In C++, static polymorphism is achieved through function overloading, while dynamic polymorphism is achieved through virtual functions.

   How can you implement a payroll system that handles different employee types, such as salaried and hourly employees?
   Answer: A payroll system that handles different employee types can be implemented using inheritance and polymorphism. A base class Employee can be created with virtual functions for calculating pay. Derived classes, such as SalariedEmployee and HourlyEmployee, can be created that inherit from the Employee class and override the pay functions.

   How can polymorphism improve the maintainability of a payroll application?
   Answer: Polymorphism improves the maintainability of a payroll application by making it easier to add new employee types and modify existing ones. With polymorphism, new employee types can be added by simply creating a new derived class that inherits from the Employee base class and overrides the necessary functions.

   What is the advantage of using a virtual destructor in a polymorphic class hierarchy?
   Answer: The advantage of using a virtual destructor in a polymorphic class hierarchy is that it ensures that the destructor of the derived class is called when an object of the derived class is destroyed through a pointer to the base class.

   How can you prevent slicing when passing objects of derived classes by value to functions that take parameters of the base class type?
   Answer: To prevent slicing, objects of derived classes should be passed by reference or by pointer to functions that take parameters of the base class type.

   What is the purpose of a pure virtual function in an abstract base class?
   Answer: A pure virtual function in an abstract base class is a function that has no implementation and must be overridden by any derived class. This ensures that any derived class is forced to provide an implementation for the function.

   What is the difference between an abstract class and a concrete class?
   Answer: An abstract class is a class that has at least one pure virtual function and cannot be instantiated, while a concrete class is a class that can be instantiated and does not have any pure virtual functions.

How can you implement a payroll system that handles overtime pay for hourly employees?
Answer: A payroll system that handles overtime pay for hourly employees can be implemented by adding a virtual function for calculating overtime pay to the HourlyEmployee derived class. The function can be called in the payroll calculation function to calculate the total pay for the employee.

Lec 31 - MULTIPLE INHERITANCE

1. **What is multiple inheritance and how does it differ from single inheritance?**
   Answer: Multiple inheritance is a feature in object-oriented programming that allows a child class to inherit properties and behaviors from multiple parent classes. In contrast, single inheritance only allows a child class to inherit from one parent class.

   What is the diamond problem in the context of multiple inheritance and how can it be resolved?
   Answer: The diamond problem is a conflict that arises when a child class inherits from two or more classes that have a common ancestor. It can be resolved using virtual inheritance, which ensures that only one instance of the common ancestor class is included in the object hierarchy.

   What is method resolution order (MRO) in multiple inheritance and how is it determined?
   Answer: Method resolution order (MRO) is the order in which a Python interpreter searches for methods in a multiple inheritance hierarchy. It is determined using the C3 algorithm, which takes into account the order of parent classes and their method definitions.

   What is the role of the super() function in multiple inheritance?
   Answer: The super() function is used to call a method of a parent class in a child class. It is often used to access and modify the behavior of the parent class method in a child class.

   How can you prevent method name conflicts in multiple inheritance?
   Answer: Method name conflicts in multiple inheritance can be prevented by using unique method names, or by using the super() function to modify the behavior of the inherited methods.

   What is the difference between mixins and multiple inheritance?
   Answer: Mixins are a type of multiple inheritance that are used to add functionality to a class without creating a new class hierarchy. In contrast, multiple inheritance involves creating a new class hierarchy by inheriting from multiple parent classes.

   How does multiple inheritance affect code readability and maintainability?
   Answer: Multiple inheritance can make code more complex and difficult to read and maintain, especially when there are conflicts between inherited methods and attributes. However, it can also improve code flexibility and modularity when used appropriately.

   What is the order of constructor execution in multiple inheritance?
   Answer: The order of constructor execution in multiple inheritance is determined by the method resolution order (MRO). The constructor of the first parent class in the MRO is executed first, followed by the constructors of subsequent parent classes in the order specified by the MRO.

   What are the potential downsides of using multiple inheritance?
   Answer: The potential downsides of using multiple inheritance include increased complexity, potential conflicts between inherited methods and attributes, and reduced code readability and maintainability.

   How can you use multiple inheritance to create a more flexible class hierarchy?
   Answer: Multiple inheritance can be used to create a more flexible class hierarchy by allowing a child class to inherit properties and behaviors from multiple parent classes. This can help to create more modular and reusable code, as well as allowing for the creation of more complex

and diverse behavior. However, it is important to carefully design and implement the class hierarchy to avoid conflicts and maintain code readability and maintainability.

Lec 32 - GENERIC PROGRAMMING

1. **What is generic programming?**
   Answer: Generic programming is a programming paradigm that emphasizes writing reusable code that can be used with different data types.

   What is a template in C++?
   Answer: A template is a mechanism in C++ that allows generic programming. Templates define a blueprint for a class or function that can be used with different data types.

   How do templates work in C++?
   Answer: Templates work by defining a generic class or function that can be used with different data types. The template is instantiated with a specific data type when it is used in code.

   What are the advantages of generic programming?
   Answer: The advantages of generic programming include increased code reusability, improved code maintainability, and reduced code complexity.

   What is the difference between generic programming and object-oriented programming?
   Answer: Generic programming focuses on writing reusable code that can be used with different data types, while object-oriented programming emphasizes encapsulation, inheritance, and polymorphism.

   What is a generic algorithm?
   Answer: A generic algorithm is an algorithm that is written to work with different data types. The algorithm is typically written as a function or class template.

   What is type erasure in Java?
   Answer: Type erasure is a process in Java where the generic type information is removed from a generic class or method during compilation. This is done to maintain backward compatibility with older Java code.

   What is a type parameter in Java generics?
   Answer: A type parameter in Java generics is a placeholder for a specific data type that is used by a generic class or method.

   What is the syntax for defining a generic class in C++?
   Answer: The syntax for defining a generic class in C++ is:

   arduino
   Copy code
   template<typename T>
   class MyClass {
     // Class definition here
   };
   What is the syntax for defining a generic method in Java?
   Answer: The syntax for defining a generic method in Java is:

   typescript
   Copy code

```
public <T> void myMethod(T myParam) {
  // Method code here
}
```

Lec 33 - MULTIPLE TYPE ARGUMENTS

1. **What are multiple type arguments, and why are they useful in generic programming?**
   Answer: Multiple type arguments refer to the ability to define multiple data types for use with a generic class or function. They are useful in generic programming because they allow for increased flexibility and reusability of code.

   How are multiple type arguments defined in Java, and what is the default type argument?
   Answer: Multiple type arguments are defined using the < > syntax in Java, and the default type argument is Object.

   How are multiple type arguments defined in C++, and how many can be defined for a generic class?
   Answer: Multiple type arguments are defined using the template < > syntax in C++, and any number can be defined for a generic class.

   What is type erasure in the context of multiple type arguments?
   Answer: Type erasure is the process of removing the generic type information from a generic class or method during compilation, allowing for backward compatibility with older code that was not designed to use generics.

   Can multiple type arguments be used with functions in C++?
   Answer: Yes, multiple type arguments can be used with functions in C++.

   How do multiple type arguments improve code maintainability?
   Answer: By reducing the need for duplicate code, multiple type arguments can improve code maintainability by making it easier to modify and update code.

   How are multiple type arguments used in object-oriented programming?
   Answer: Multiple type arguments are used in object-oriented programming to create reusable code that can be used with different data types.

   What is the difference between single type arguments and multiple type arguments?
   Answer: Single type arguments can only be used with one data type, while multiple type arguments can be used with multiple data types.

   What is the syntax for defining multiple type arguments in Java?
   Answer: The syntax for defining multiple type arguments in Java is < type1, type2, ... >.

   What are some common use cases for multiple type arguments in generic programming?
   Answer: Common use cases for multiple type arguments include the creation of generic algorithms, data structures, and collections that can be used with different data types.

Lec 34 - GENERIC ALGORITHMS

1. **What are generic algorithms, and why are they important in programming?**
   Answer: Generic algorithms are algorithms designed to work with any data type, providing a reusable and adaptable solution to programming problems.

   How are generic algorithms different from regular algorithms?
   Answer: Regular algorithms are designed to work with specific data types, whereas generic algorithms can work with any data type.

   What are the benefits of using generic algorithms in programming?
   Answer: Generic algorithms provide increased reusability and adaptability of code, reducing development time and code complexity.

   What are some examples of generic algorithms?
   Answer: std::sort, std::transform, std::find_if are some examples of generic algorithms in C++.

   What is the syntax for using generic algorithms in Java?
   Answer: The syntax for using generic algorithms in Java is < >.

   What is the purpose of the std::transform algorithm in C++?
   Answer: The std::transform algorithm applies a function to each element in a range, transforming it into a new value.

   Can generic algorithms be used with user-defined data types?
   Answer: Yes, generic algorithms can be used with user-defined data types.

   What is the difference between a generic algorithm and a template function?
   Answer: A generic algorithm is a specific type of template function designed to work with any data type, whereas a template function can be specialized for specific data types.

   How can generic algorithms help reduce code duplication?
   Answer: By creating a generic algorithm that can work with any data type, developers can avoid writing the same code multiple times for different data types.

   What is type erasure, and how is it related to generic algorithms?
   Answer: Type erasure is the process of removing the generic type information from a generic class or method during compilation. It allows for backward compatibility with older code that was not designed to use generics and is related to generic algorithms in that it is a fundamental concept in generic programming.

Lec 35 - MEMBER TEMPLATES

1. **What are member templates in C++?**
   Answer: Member templates are templates that are defined inside a class or struct.

   What is the advantage of using member templates?
   Answer: Member templates provide increased reusability and adaptability of code, reduced code complexity, and improved code performance.

   Can member templates access the data members and methods of the class they are defined in?
   Answer: Yes, member templates can access the data members and methods of the class they are defined in.

   What is the syntax for defining a member template?
   Answer: The syntax for defining a member template is: template<typename T> void MyClass<T>::myFunction(T arg) { //function body }

   Can member templates be specialized for specific data types?
   Answer: Yes, member templates can be specialized for specific data types.

   What is the difference between a member function template and a regular member function?
   Answer: A member function template can work with any data type, whereas a regular member function can only work with specific data types.

   What is the purpose of a member function template?
   Answer: The purpose of a member function template is to provide a generic member function that can work with any data type.

   Can member templates be used to provide generic constructors?
   Answer: Yes, member templates can be used to provide generic constructors.

   What is template specialization?
   Answer: Template specialization is the process of defining a specialized version of a template for a specific data type.

   What is the advantage of using a member function template over a regular member function?
   Answer: The advantage of using a member function template is increased reusability and adaptability of code, reduced code complexity, and improved code performance.

Lec 36 - MEMBER TEMPLATES REVISITED

1. **What is the difference between regular member function templates and member function templates revisited?**
   Answer: Member function templates revisited use template argument deduction, whereas regular member function templates require explicit template arguments to be provided.

   How does template argument deduction work in member templates revisited?
   Answer: Template argument deduction infers the template arguments from the function arguments.

   What is the syntax for using member templates revisited?
   Answer: template<typename T> auto Class<T>::func(args...)

   What is the purpose of using member templates revisited?
   Answer: To simplify the syntax of member function templates and reduce code duplication.

   Can member templates revisited be used with non-static member functions?
   Answer: Yes, they can be used with non-static member functions.

   How does member templates revisited improve code readability?
   Answer: It reduces the amount of code required and makes it easier to understand the function's purpose.

   Can member templates revisited be specialized for specific data types?
   Answer: Yes, they can be specialized.

   How does member templates revisited improve code adaptability?
   Answer: It makes it easier to modify the code for different data types and use cases.

   What is the advantage of using member templates revisited over regular member function templates?
   Answer: It reduces the amount of code required and makes the code more concise and readable.

   How does member templates revisited improve code reusability?
   Answer: It allows the same function to be used with different data types, making the code more versatile and reusable.

Lec 37 - RESOLUTION ORDER

1. **What is resolution order, and why is it important in programming?**
   Answer: Resolution order is the set of rules that determine the order in which conflicting claims or issues will be addressed. It is important in programming because it helps resolve conflicts between variables or functions with the same name or value.

   How is resolution order determined in Python classes?
   Answer: The default resolution order for Python classes is determined using a depth-first search of the inheritance hierarchy.

   What is the diamond problem, and how is it solved in programming languages that support multiple inheritance?
   Answer: The diamond problem is a scenario that arises in languages that support multiple inheritance, where a class inherits from two or more classes that have a common ancestor. It is solved using various approaches, such as method resolution order (MRO) or virtual inheritance.

   What is the order of precedence for CSS styles?
   Answer: The order of precedence for CSS styles is inline styles, embedded styles, and external styles.

   What is the resolution order for overloaded operators in C++?
   Answer: The resolution order for overloaded operators in C++ is undefined, meaning that it is up to the compiler to decide.

   What is the difference between left-to-right and right-to-left resolution order in programming?
   Answer: Left-to-right resolution order evaluates expressions from left to right, while right-to-left resolution order evaluates expressions from right to left.

   How is the resolution order for conflicting domain name records determined?
   Answer: The resolution order for conflicting domain name records is determined by the order of precedence of the record types, which is CNAME, A, and then MX.

   How is the resolution order for conflicting method calls determined in Java?
   Answer: The resolution order for conflicting method calls in Java is determined by the class hierarchy, where methods in the superclass take precedence over methods in the subclass.

   What is method resolution order (MRO) in Python, and how is it determined?
   Answer: Method resolution order (MRO) in Python is the order in which methods are searched for and executed in a class hierarchy. It is determined using the C3 linearization algorithm, which is a modified depth-first search of the inheritance graph.

   What is the difference between breadth-first search and depth-first search, and how are they used in resolution order?
   Answer: Breadth-first search explores all the neighbors of a node before moving on to the next level, while depth-first search explores as far as possible along each branch before backtracking. Both algorithms can be used in resolution order, depending on the context and the desired outcome.

Lec 38 - FUNCTION TEMPLATE OVERLOADING

1. **What is function template overloading in C++?**
   Answer: Function template overloading is a technique in C++ that allows programmers to define multiple functions with the same name but different argument types.

   How is function template overloading achieved in C++?
   Answer: Function template overloading is achieved by defining a function template with placeholders for argument types that can be instantiated with different types at compile time.

   What is the purpose of function template overloading?
   Answer: The purpose of function template overloading is to improve code flexibility and reusability by creating a single function that can be used with different data types.

   Can function templates be overloaded based on the return type?
   Answer: No, function templates cannot be overloaded based on the return type.

   What is the difference between function overloading and function template overloading?
   Answer: Function overloading creates multiple functions with the same name and argument types, while function template overloading creates multiple functions with the same name but different argument types.

   Can function templates be overloaded based on the number of arguments?
   Answer: Yes, function templates can be overloaded based on the number of arguments.

   What are the advantages of function template overloading?
   Answer: Function template overloading improves code flexibility and reusability by creating a single function that can be used with different data types.

   Can function templates be overloaded based on the constness of the arguments?
   Answer: Yes, function templates can be overloaded based on the constness of the arguments.

   How many function templates can be defined for a given set of argument types?
   Answer: Multiple function templates can be defined for a given set of argument types.

   Can function templates be overloaded based on the type of argument?
   Answer: Yes, function templates can be overloaded based on the type of argument.

Lec 39 - TEMPLATES & STATIC MEMBERS

1. **What is a template in C++?**
   Answer: A template is a feature of C++ that allows for the creation of generic functions and classes that can work with different data types.

   What is a static member in C++?
   Answer: A static member is a member of a class that is shared by all instances of the class and can be accessed without creating an object.

   What is the purpose of using templates in C++?
   Answer: The purpose of using templates in C++ is to create generic functions and classes that can work with different data types, improving code reusability, flexibility, and efficiency.

   How is a static member variable initialized in C++?
   Answer: A static member variable is initialized outside the class definition, typically in the source file, using the scope resolution operator and the class name.

   Can a template class have static members?
   Answer: Yes, a template class can have static members that are shared by all instances of the class.

   What is template specialization in C++?
   Answer: Template specialization is a feature of C++ that allows for the creation of specialized versions of a template function or class for a specific data type.

   Can a static member function access non-static members of a class?
   Answer: No, a static member function cannot access non-static members of a class.

   Can a template function be overloaded in C++?
   Answer: Yes, a template function can be overloaded with different argument types.

   What is the syntax for declaring a static member function in C++?
   Answer: The syntax for declaring a static member function in C++ is to use the "static" keyword before the function name in the class definition.

   How can a template function be defined outside the class definition in C++?
   Answer: A template function can be defined outside the class definition by specifying the template parameter list and using the "template" keyword followed by the function signature.

Lec 40 - CURSORS

1. **What is a cursor in a database management system?**
   Answer: A cursor is a database object that allows the traversal of a set of rows retrieved from a query result set.

   How do you declare a cursor in SQL?
   Answer: To declare a cursor in SQL, you use the DECLARE statement followed by the cursor name and query that will be used to populate the cursor.

   What are the types of cursors available in SQL?
   Answer: The types of cursors in SQL are static, dynamic, and keyset-driven.

   How do you fetch data from a cursor in SQL?
   Answer: To fetch data from a cursor in SQL, you use the FETCH statement, which retrieves the next row from the result set associated with the cursor.

   How do you close a cursor in SQL?
   Answer: To close a cursor in SQL, you use the CLOSE statement followed by the cursor name.

   What is the purpose of a cursor in database programming?
   Answer: Cursors allow you to manipulate individual rows of data returned from a query, making it possible to perform complex data operations that are not possible with simple SQL statements.

   What is a forward-only cursor?
   Answer: A forward-only cursor is a type of cursor that can only be scrolled forward through the rows of the result set.

   What is a keyset-driven cursor?
   Answer: A keyset-driven cursor is a type of cursor that is based on a unique key value or set of values, making it possible to quickly search through the result set.

   What is a dynamic cursor?
   Answer: A dynamic cursor is a type of cursor that allows you to change the underlying query associated with the cursor while it is still open.

   Can cursors be used in all database management systems?
   Answer: No, cursors are not available in all database management systems and their usage and implementation may vary between systems that support them.

Lec 41 - STANDARD TEMPLATE LIBRARY

1. **What is the Standard Template Library (STL) in C++?**
   Answer: The Standard Template Library is a library of generic algorithms, data structures, and iterators that are part of the C++ standard library.

   What is an iterator in STL?
   Answer: An iterator is a type of object that points to an element in a container in STL and allows us to access and modify its value.

   What are the advantages of using STL in C++ programming?
   Answer: The advantages of using STL in C++ programming are that it provides a set of reusable and interchangeable components, reduces development time, improves code quality, and makes the code more maintainable.

   What is a container in STL?
   Answer: A container is a class that stores and manages a collection of objects in STL, such as vectors, lists, sets, and maps.

   What is the difference between a vector and a list in STL?
   Answer: A vector is a sequence container that stores objects in contiguous memory, while a list is a doubly linked list container that stores objects in non-contiguous memory.

   What is an algorithm in STL?
   Answer: An algorithm is a set of operations that can be performed on a range of elements in a container in STL, such as sorting, searching, and copying.

   What is a template in STL?
   Answer: A template is a C++ feature that allows us to write generic code that works with different data types, and it is used extensively in STL to provide a generic framework for algorithms and containers.

   What is the difference between a stack and a queue in STL?
   Answer: A stack is a container that provides LIFO (last-in-first-out) access to elements, while a queue is a container that provides FIFO (first-in-first-out) access to elements.

   What is the complexity of finding an element in a map in STL?
   Answer: The complexity of finding an element in a map in STL is O(log n), where n is the number of elements in the map.

   What is the difference between a set and a multiset in STL?
   Answer: A set is a container that stores unique elements in sorted order, while a multiset is a container that stores multiple copies of the same element in sorted order.

Lec 42 - ITERATORS

1. **What is an iterator in C++?**
   Answer: An iterator in C++ is an object used to traverse through the elements of a container and perform operations on them.

   What is the purpose of an iterator?
   Answer: The purpose of an iterator is to provide a way to access the data stored in a container without exposing its internal representation, making the container more flexible and reusable.

   What is the difference between an input iterator and an output iterator?
   Answer: An input iterator is used to read data from a container, while an output iterator is used to write data to a container.

   What is the difference between a forward iterator and a bidirectional iterator?
   Answer: A forward iterator can only move forward in a container, while a bidirectional iterator can move both forward and backward.

   What is the difference between a random access iterator and a bidirectional iterator?
   Answer: A random access iterator provides constant time access to any element in a container, while a bidirectional iterator only provides constant time access to the next or previous element.

   How is the complexity of an iterator defined?
   Answer: The complexity of an iterator is defined by the amount of time it takes to perform certain operations, such as incrementing or decrementing the iterator.

   What is the difference between a container and an iterator?
   Answer: A container is a data structure that stores elements, while an iterator is an object used to traverse through the elements of a container.

   What is the purpose of the std::begin() and std::end() functions?
   Answer: The std::begin() function returns an iterator pointing to the first element in a container, while the std::end() function returns an iterator pointing to the end of the container.

   What is the difference between a constant iterator and a regular iterator?
   Answer: A constant iterator is used to iterate through a container without allowing modifications to the elements, while a regular iterator allows modifications to the elements.

   How are iterators used in algorithms from the Standard Template Library?
   Answer: Iterators are used as arguments to algorithms in the Standard Template Library to specify which elements in a container to operate on.

Lec 43 - EXAMPLE – ABNORMAL TERMINATION

1. **What is abnormal termination in computer programming?**
   Abnormal termination refers to the unexpected termination of a program due to an error or exception. It occurs when a program encounters an unforeseen problem and cannot continue to execute normally.

   What is the difference between an exception and an error?
   An error is a problem that occurs at runtime and halts program execution, while an exception is a problem that occurs at runtime but can be handled by the program through the use of exception handling techniques.

   What is the purpose of exception handling?
   The purpose of exception handling is to provide a mechanism for handling errors and exceptions in a program gracefully. It allows the program to detect and respond to errors without crashing or terminating unexpectedly.

   What is a try-catch block?
   A try-catch block is a programming construct used in exception handling. The try block contains the code that might throw an exception, while the catch block contains the code that handles the exception if one is thrown.

   What is the syntax of a try-catch block?
   The basic syntax of a try-catch block is as follows:

   ```
   try {
   // Code that might throw an exception
   }
   catch (ExceptionType e) {
   // Code to handle the exception
   }
   ```

   What is the purpose of the finally block in a try-catch-finally statement?
   The finally block is used to contain code that is guaranteed to execute regardless of whether or not an exception is thrown. It is typically used for cleanup operations such as closing files or releasing resources.

   What is the difference between a checked exception and an unchecked exception?
   A checked exception is a type of exception that the compiler requires the program to handle or declare, while an unchecked exception is a type of exception that the compiler does not require the program to handle or declare.

   How can you create your own exception class?
   You can create your own exception class by extending the Exception class or one of its subclasses. Your custom exception class should provide constructors that allow you to pass any necessary information to the superclass.

   What is the purpose of the throw keyword?
   The throw keyword is used to explicitly throw an exception from a method or block of code. It

allows you to create and throw your own custom exceptions or to re-throw exceptions that have been caught by a catch block.

What is the purpose of the throws keyword?
The throws keyword is used in a method declaration to indicate that the method may throw one or more types of exceptions. It allows the calling code to handle the exceptions that might be thrown by the method.

Lec 44 - STACK UNWINDING

1. **What is stack unwinding in C++?**
   Answer: Stack unwinding is a process in C++ that happens when an exception is thrown. It involves removing all the objects in the call stack in reverse order of their creation until the corresponding catch block is found.

   How does stack unwinding work?
   Answer: When an exception is thrown, the C++ runtime system starts unwinding the call stack to find a catch block that can handle the exception. It does this by destroying all the objects created in the call stack in reverse order of their creation until the catch block is found.

   What is the role of catch blocks in stack unwinding?
   Answer: Catch blocks in C++ are used to handle exceptions that are thrown by a program. When an exception is thrown, the runtime system starts unwinding the call stack to find a catch block that can handle the exception. Once the catch block is found, the exception is handled and the program continues execution.

   Can stack unwinding cause memory leaks?
   Answer: Yes, stack unwinding can cause memory leaks if the objects created on the stack were not properly deleted before the exception was thrown. In this case, the objects will not be deleted and will cause memory leaks when the stack is unwound.

   How can you prevent memory leaks in stack unwinding?
   Answer: To prevent memory leaks in stack unwinding, it is important to properly manage memory in your program. You should use smart pointers, such as shared_ptr or unique_ptr, to manage objects on the heap. You should also ensure that all objects created on the stack are properly deleted before an exception is thrown.

   What is the difference between stack unwinding and stack trace?
   Answer: Stack unwinding is a process that happens when an exception is thrown in C++. It involves removing all the objects in the call stack in reverse order of their creation until the corresponding catch block is found. A stack trace, on the other hand, is a record of the function calls that led to a particular point in a program's execution.

   Can stack unwinding be disabled in C++?
   Answer: Stack unwinding cannot be disabled in C++. It is a fundamental mechanism that is used to handle exceptions in the language. However, you can use techniques such as RAII (Resource Acquisition Is Initialization) to ensure that resources are properly managed in your program and prevent memory leaks.

   What is the impact of stack unwinding on performance?
   Answer: Stack unwinding can have a significant impact on performance in C++. This is because it involves the destruction of all the objects in the call stack in reverse order of their creation. However, the impact can be minimized by properly managing memory in your program and using techniques such as RAII.

   Can stack unwinding cause data corruption?
   Answer: Yes, stack unwinding can cause data corruption if the objects created on the stack were not properly deleted before the exception was thrown. In this case, the objects will not be

deleted and can cause data corruption when the stack is unwound.

How does C++ ensure that objects are properly deleted during stack unwinding?
Answer: C++ ensures that objects are properly deleted during stack unwinding by automatically calling the destructors of objects created on the stack as the stack is unwound. This is done in reverse order of the objects' creation to ensure that they are properly cleaned up.

Lec 45 - RESOURCE MANAGEMENT

1. **What is resource management in C++?**
   Resource management in C++ refers to the techniques used to handle and manage system resources, such as memory, files, network connections, etc.

   What is dynamic memory allocation in C++?
   Dynamic memory allocation is a mechanism provided by the language to allocate memory at runtime. The memory is allocated using operators such as new and delete.

   What is a smart pointer?
   A smart pointer is a class that provides automatic memory management for dynamically allocated objects. It automatically deletes the object it points to when it is no longer needed.

   What is RAII?
   RAII (Resource Acquisition Is Initialization) is a technique in C++ used to manage resource acquisition and release in a scoped manner. It ensures that resources are acquired and released in a predictable and safe way.

   How can you handle exceptions in C++?
   Exceptions can be handled using try-catch blocks. The code that might throw an exception is placed in the try block, and the catch block catches the exception and handles it appropriately.

   What is the difference between throw and throw() in C++?
   throw is used to throw an exception, while throw() is used to specify that a function does not throw any exceptions.

   What are the benefits of using smart pointers in C++?
   Smart pointers help prevent memory leaks and improve code safety by automatically releasing resources when they are no longer needed. They also simplify code by eliminating the need for manual memory management.

   What is the role of destructors in C++?
   Destructors are used to release resources acquired by an object when it is no longer needed. They are called automatically when an object is destroyed.

   What is the use of the std::unique_ptr class in C++?
   std::unique_ptr is a smart pointer that provides exclusive ownership of the object it points to. It automatically deletes the object when it is no longer needed.

   How can you prevent resource leaks in C++?
   Resource leaks can be prevented by using RAII, smart pointers, and exception handling. These techniques ensure that resources are acquired and released in a predictable and safe way, even in the presence of exceptions.