1. How do you include and use the C++ Standard Library in a C++ program?
2. What are pre-processor directives in C++? Explain their role in the compilation process.
3. List and explain the purpose of common pre-processor directives used in C++.
4. How can you use pre-processor directives to prevent multiple inclusions of a header file?
5. Write a simple C++ program to print "Hello, World!" and explain each part of the code.
6. Write a C++ program to calculate the factorial of a number provided by the user.
7. How would you write a C++ program to read input from a user and display it? Illustrate with an example.
8. What are header files in C++ and why are they used?
9. Explain the concept of namespaces in C++. Why are they important?
10. How do you define and use a custom namespace in a C++ program? Provide an example.
11. What are library files in C++ and how do they differ from header files?
12. How can you link a library file to your C++ program during compilation?
13. Define the following object-oriented concepts: Objects, Classes, Data Abstraction, Encapsulation, Polymorphism, Inheritance, and Reusability.
14. Explain the concept of encapsulation and its benefits in object-oriented programming.
15. What are access modifiers in C++? List and explain the different types of access modifiers.
16. What is the difference between a structure and a class in C++?
17. Explain how class scope works in C++ with an example.
18. How do you define and access class members in C++? Provide a simple class definition and demonstrate how to create an object of that class and access its members.
19. What are member functions in C++? Give an example of how to define and call a member function.
20. Explain the concept of const member functions and their use cases.
21. What is the significance of the 'this' pointer in member functions?
22. What is encapsulation in C++ and why is it important?
23. How does information hiding relate to encapsulation? Provide an example demonstrating information hiding in a class.
24. What are the benefits of using encapsulation in object-oriented programming?
25. What are abstract data types (ADTs)? Give examples of ADTs in C++.
26. Explain how classes in C++ can be used to implement abstract data types.
27. What is the difference between an abstract data type and a concrete data type?
28. Define the terms "object" and "class" in the context of C++ programming.
29. How do objects and classes relate to each other in C++?
30. Illustrate with an example how to create an object of a class and access its members.
31. How do you declare and initialize an array of objects in C++? Provide an example.
32. Explain how member functions of objects in an array can be called.
33. What are some use cases for arrays of objects in C++?
34. What is a constructor in C++? Explain its purpose and how it is defined.
35. Describe the different types of constructors in C++ with examples.
36. What is a destructor? When and why is it used?
37. What are parameterized constructors in C++? Provide an example.
38. How do parameterized constructors differ from default constructors?
39. Explain the use of initializer lists in parameterized constructors.
40. What is a copy constructor in C++? Why is it needed?
41. Provide an example of a copy constructor and explain its functionality.
42. What happens if a copy constructor is not defined by the programmer?
43. What are dynamic constructors in C++? Explain with an example.
44. How do dynamic constructors differ from other constructors?
45. What are some common pitfalls when using dynamic constructors?
46. What is the role of a destructor in C++?
47. How is a destructor different from a constructor?
48. Provide an example of a class with a destructor and explain its functionality.
49. Explain the concepts of identity and behaviour in the context of objects in C++.
50. How do you differentiate between the identity and behaviour of an object with examples?

51. Why are the identity and behaviour of objects important in object-oriented programming?
52. How is garbage collection handled in C++?
53. What are some common practices for managing dynamic memory in C++?
54. Explain the role of destructors in the context of garbage collection.
55. How is dynamic memory allocation performed in C++? Provide examples using new and delete.
56. What are the advantages and disadvantages of dynamic memory allocation in C++?
57. Explain the concept of memory leaks and how they can be avoided.
58. What are explicit type conversions in C++? Provide examples.
59. How do explicit type conversions differ from implicit type conversions?
60. When and why would you use explicit type conversions in C++?
61. What is a static member function in C++? How does it differ from a non-static member function?
62. Provide an example of a static member function and explain how it is called.
63. What are some typical use cases for static member functions in C++?
64. What is a friend function in C++? Why and when would you use it?
65. Explain how friend functions can access private and protected members of a class with an example.
66. What is a friend class? Provide an example to illustrate its usage.
67. What are the potential risks or downsides of using friend functions and friend classes?
68. What is the 'this' pointer in C++? How is it used within a class?
69. Provide an example demonstrating the use of the 'this' pointer.
70. Explain how the 'this' pointer can be used in operator overloading.
71. What are container classes in C++? Give examples of different types of container classes.
72. Explain the role of iterators in C++. How do they interact with container classes?
73. Provide an example of using an iterator with a standard container class (e.g., vector or list).
74. What is function overloading in C++? How does it enhance a program's flexibility?
75. Provide examples of overloaded functions and explain how the compiler distinguishes between them.
76. What are some rules and best practices for function overloading in C++?
77. What is operator overloading in C++? Why is it used?
78. Explain the basic syntax of operator overloading with an example.
79. What are the benefits and potential drawbacks of operator overloading?
80. What are some operators that cannot be overloaded in C++? Why are these restrictions in place?
81. Discuss the limitations and rules for overloading operators in C++.
82. Explain why it is not possible to overload certain operators (e.g., the scope resolution operator ::).
83. Compare and contrast operator functions as class members and as friend functions.
84. Provide examples of overloading an operator as a class member and as a friend function.
85. Discuss the advantages and disadvantages of using friend functions for operator overloading.
86. How do you overload the binary operator + for a class in C++? Provide an example.
87. Explain the process of overloading the binary operator - with an example.
88. Demonstrate how to overload the binary operator * for a custom class.
89. Provide an example of overloading the binary operator /.
90. Explain how to overload the assignment operator = and discuss its significance in C++.
91. How do you overload the unary operator - for a class in C++? Provide an example.
92. Explain the steps involved in overloading the unary operator ++ (both prefix and postfix forms).
93. Provide an example of overloading the unary operator -- (both prefix and postfix forms).
94. Discuss the differences and nuances between overloading unary and binary operators.
95. What is inheritance in C++ and why is it used?
96. Explain the concept of a base class and a derived class with an example.
97. What are the key benefits of using inheritance in object-oriented programming?
98. List and explain the different types of inheritance supported in C++.

99. Provide examples for each type of inheritance: single, multiple, multilevel, hierarchical, and hybrid inheritance.
100. What are the potential problems associated with multiple inheritance and how can they be resolved?
101. What is a virtual base class in C++? Why and when is it used?
102. Provide an example illustrating the use of a virtual base class to solve the diamond problem.
103. How does virtual inheritance differ from non-virtual inheritance?
104. Explain the concept of casting base class pointers to derived class pointers with an example.
105. What is dynamic_cast and when should it be used?
106. Discuss the differences between static_cast, dynamic_cast, const_cast, and reinterpret_cast.
107. How do you call base class member functions from a derived class in C++? Provide an example.
108. What is function overriding in the context of inheritance?
109. Explain how to prevent a member function from being overridden in derived classes.
110. What does it mean to override a base-class member in a derived class? Provide an example.
111. How does C++ handle function overriding and how does it differ from function overloading?
112. What is the significance of the virtual keyword in function overriding?
113. Explain the differences between public, protected, and private inheritance with examples.
114. How do access specifiers affect member access in derived classes?
115. Provide a scenario where protected inheritance would be preferred over public or private inheritance.
116. How are constructors and destructors called in a derived class?
117. Explain the order of constructor and destructor calls in inheritance with an example.
118. What is the purpose of using an initializer list in a derived class constructor?
119. Compare and contrast composition and inheritance. Provide examples illustrating when to use each.
120. What are the advantages of using composition over inheritance in certain situations?
121. Explain the concept of "has-a" versus "is-a" relationships with examples.
122. Differentiate between function overloading and function overriding in C++.
123. Provide examples to illustrate function overloading and function overriding.
124. Discuss the rules and limitations associated with function overriding in C++.
125. What is run time polymorphism in C++ and how is it achieved?
126. Explain the role of virtual functions in implementing run time polymorphism.
127. Provide an example demonstrating run time polymorphism using base and derived classes.
128. What is a virtual function in C++? Why is it used?
129. How do virtual functions support polymorphism in C++?
130. Provide an example of a virtual function in a base class and its override in a derived class.
131. What is a pure virtual function? How is it declared?
132. Explain the purpose of pure virtual functions in abstract base classes.
133. Provide an example of a pure virtual function and its implementation in a derived class.
134. Define abstract base class and concrete class in C++.
135. Why are abstract base classes used in C++? Provide an example.
136. Explain the process of inheriting and implementing abstract base classes.
137. What is dynamic binding in C++ and how does it differ from static binding?
138. Provide an example demonstrating dynamic binding using virtual functions.
139. Discuss the advantages of dynamic binding in object-oriented programming.
140. What is a virtual destructor in C++ and why is it important?
141. Provide an example to illustrate the need for a virtual destructor in a base class.

142. What can happen if a base class destructor is not declared virtual in a polymorphic base class?
143. What are I/O streams in C++ and how are they used for file operations?
144. Explain the difference between ifstream, ofstream, and fstream classes in C++.
145. Provide an example of how to open, read, and close a file using ifstream.
146. How do you open a file for writing using ofstream? Provide an example.
147. What are the various file opening modes in C++? Explain with examples.
148. Describe the process of checking if a file has been successfully opened in C++.
149. Explain the use of the getline function for reading from a file. Provide an example.
150. How do you write data to a file using the write function in C++? Provide an example.
151. What are stream manipulators and how are they used in file I/O operations? Provide examples.
152. Explain the different stream format states and how they affect file I/O operations.
153. Describe the various stream error states and how to handle them in C++. Provide examples.
154. What are the key differences between formatted and unformatted I/O in C++?
155. Explain how to use the read and write functions for unformatted I/O in C++.
156. Provide an example of using cin and cout for standard input and output operations.
157. What are some common stream manipulators used for formatting output in C++? Provide examples.
158. Describe how to set and reset stream format states using stream manipulators.
159. What is a template in C++ and why is it used?
160. Explain the syntax for defining a function template with an example.
161. How do you overload template functions in C++? Provide an example.
162. What are class templates and how are they defined? Provide an example.
163. Explain the concept of non-type parameters in class templates with an example.
164. How do templates interact with inheritance in C++? Provide an example.
165. What is the role of friend functions in template classes? Provide an example to illustrate.
166. What are the basics of exception handling in C++? Explain the try, throw, and catch keywords.
167. Provide an example of throwing and catching an exception in C++.
168. How do you re-throw an exception in C++? Provide an example.
169. Explain how to process unexpected exceptions in C++ with an example.
170. Describe the role of constructors and destructors in exception handling.
171. What are some best practices for using exceptions in C++?
172. Explain the difference between standard exceptions and user-defined exceptions in C++.
173. How can you define and use custom exception classes in C++? Provide an example.
174. Discuss the impact of exceptions on resource management and how RAII (Resource Acquisition Is Initialization) helps in this context.
175. What are stream manipulators and how do they affect input/output operations in C++?
176. Provide examples of commonly used stream manipulators for formatting output.
177. How do you change the width and precision of floating-point numbers using stream manipulators?
178. Explain the purpose of the setf and unsetf functions in managing stream format states.
179. How can you check and clear stream error states in C++? Provide examples.
180. What are the common error states for streams and how do they differ from each other?