1. What is the purpose of function templates in C++?
   1. To improve program performance
   2. To reduce the size of the executable file
   3. **To create generic functions that work with different data types**
   4. To hide the implementation details of a function
2. How do you define a function template in C++?
   1. **Using the keyword "template" followed by the template parameter list**
   2. Using the keyword "generic" followed by the function definition
   3. Using the keyword "function" followed by the template parameter list
   4. Using the keyword "template" followed by the function name
3. What symbol is used to specify a template parameter in C++?
   1. \*
   2. $
   3. #
   4. **typename or class**
4. When are function templates instantiated in C++?
   1. **At compile time**
   2. At runtime
   3. When the program starts
   4. When the template is defined
5. Which of the following is an example of a function template?
   1. int add(int a, int b) { return a + b; }
   2. void printArray(T arr[], int size) { /\* implementation \*/ }
   3. float divide(float a, float b) { return a / b; }
   4. **template <typename T> T max(T a, T b) { return (a > b) ? a : b; }**
6. What does a class template define in C++?
   1. **A generic class that works with different data types**
   2. A class with only member functions
   3. A class with private access specifier
   4. A class with static member variables
7. How do you define a class template in C++?
   1. Using the keyword "class" followed by the template parameter list
   2. Using the keyword "template" followed by the class definition
   3. Using the keyword "generic" followed by the class name
   4. **Using the keyword "template" followed by the class name**
8. Can a class template have member functions that are not templates?
   1. **Yes**
   2. No
9. What is the purpose of overloading template functions in C++?
   1. To make the program compile faster
   2. To allow the same function template to work with different data types
   3. **To create multiple template functions with the same name but different implementations**
   4. To prevent template instantiations
10. How is a template function overloaded in C++?
    1. **By providing multiple template parameter lists**
    2. By providing different return types for the same template function
    3. By providing different template parameter names
    4. By providing different function names
11. What happens if you try to call a template function without providing the template arguments explicitly or deducible from the function arguments?
    1. The compiler will automatically choose the template arguments based on the function arguments.
    2. **The compiler will issue an error, and you must provide the template arguments explicitly.**
    3. The compiler will assume default template arguments.
    4. The compiler will promote the function arguments to the closest matching data type.
12. Can you use a non-type template parameter for a class template in C++?
    1. **Yes**
    2. No
13. What is the keyword used for specifying a non-type template parameter?
    1. value
    2. constant
    3. **non-type**
    4. typename
14. Which of the following is an example of a non-type template parameter?
    1. **template <int size> class Array { /\* implementation / };**
    2. template <typename T, T value> void printValue() { / implementation / }
    3. template <typename T> T add(T a, T b) { return a + b; }
    4. template <bool flag> void process() { / implementation \*/ }
15. Can you overload a function template based on the const-qualification of the template arguments?
    1. **Yes**
    2. No
16. What is the purpose of the "typename" keyword in a template function definition?
    1. To specify the return type of the function
    2. **To indicate that the following identifier is a template parameter**
    3. To specify the type of the template arguments
    4. To indicate that the function is a template function
17. In C++, can you provide default template arguments for function templates?
    1. **Yes**
    2. No
18. What is the purpose of providing default template arguments?
    1. To make the function template non-generic
    2. To specify the default return value of the template function
    3. **To allow the function template to be used without explicitly providing template arguments**
    4. To enforce the template parameters to have specific default values
19. Can you specialize a function template for a specific data type in C++?
    1. **Yes**
    2. No
20. What is function template specialization?
    1. Defining a template function inside another template function
    2. Overloading a template function based on the const-qualification of the function arguments
    3. **Providing an alternate implementation for a template function for a specific data type**
    4. Creating a function template without any template parameters
21. How is a function template specialization identified in C++?
    1. By providing the "specialize" keyword
    2. By enclosing the specialization with curly braces { }
    3. **By providing the template arguments in angle brackets < >**
    4. By using the "template" keyword followed by the data type
22. Can you partially specialize a class template in C++?
    1. **Yes**
    2. No
23. What is partial class template specialization?
    1. Providing a default implementation for the class template
    2. **Specifying a partial list of template arguments for the class template**
    3. Specializing the member functions of the class template
    4. Creating a class template with multiple template parameters
24. Can you have a function template and a non-template function with the same name in C++?
    1. **Yes**
    2. No
25. What is the function resolution order when calling a function with the same name but different types in C++?
    1. The non-template function is preferred over the template function.
    2. **The template function is preferred over the non-template function.**
    3. The compiler will randomly choose one of the functions.
    4. The program will not compile due to ambiguity.
26. How do you define a member function of a class template outside the class definition?
    1. By using the keyword "member" followed by the function definition
    2. By providing the template parameters before the member function definition
    3. By using the "template" keyword followed by the class name and function definition
    4. **By using the class template name followed by the member function definition**
27. Can you have a function template and a member function template with the same name in the same class?
    1. **Yes**
    2. No
28. How do you specify template arguments explicitly when calling a template function in C++?
    1. By using the "template" keyword followed by the template arguments
    2. By enclosing the template arguments in square brackets [ ]
    3. **By providing the template arguments in angle brackets < >**
    4. By using the "template" keyword followed by the function name and arguments
29. What is the purpose of using the "typename" keyword when defining a member function of a class template outside the class definition?
    1. To indicate that the function is a member function
    2. To specify the return type of the member function
    3. **To indicate that the following identifier is a template parameter**
    4. To disambiguate between member function and non-member function templates
30. Can you use type deduction for function templates with non-template function arguments?
    1. **Yes**
    2. No
31. What happens when you overload a function template and a non-template function with the same name and arguments in C++?
    1. The program will not compile due to ambiguity.
    2. **The non-template function is preferred over the template function.**
    3. The template function is preferred over the non-template function.
    4. The program will compile, but the behavior is undefined.
32. What is the syntax for explicitly specifying template arguments when calling a member function of a class template?
    1. object->function<arg1, arg2>()
    2. **object.function<arg1, arg2>()**
    3. object.function(arg1, arg2)
    4. object.function::arg1, arg2
33. What is the output of this program?

#include <iostream>

using namespace std;

template <typename T>

T maximum(T x, T y)

{

return (x > y)? x : y;

}

int main()

{

cout << maximum(3, 7) << std::endl;

cout << maximum(3.0, 7.0) << std::endl;

cout << maximum(3, 7.0) << std::endl;

return 0;

}

1. **Compiler Error in last cout statement as call to maximum is ambiguous**
2. Compiler Error in all cout statements as data type is not specified
3. 7 7.0 7.0
4. None of the above
5. Which of the following statement is correct about the program given below?

#include <iostream>

using namespace std;

template <class T, class U>

class A {

T x;

U y;

static int count;

};

int main() {

A<char, char> p;

A<int, int> q;

cout << sizeof(p) << endl;

cout << sizeof(q) << endl;

return 0;

}

* 1. Compiler Error: There can not be more than one template arguments.
  2. 6 12
  3. 8 8
  4. **2 8**

1. Which of the following is true about the following program

#include <iostream>

using namespace std;

template <class P, class Q, class R>

class A {

P x;

Q y;

R z;

static int count;

};

int main()

{

A<int, int, int> m;

A<char, char, char> n;

cout << sizeof(m) << endl;

cout << sizeof(n) << endl;

return 0;

}

* 1. Compiler Error: template parameters cannot have default values
  2. 12 6
  3. **12 3**
  4. 6 3

1. What will be the output of this program?

#include <iostream>

using namespace std;

template <int i>

void fun()

{

i = 20;

cout << i;

}

int main()

{

fun<10>();

return 0;

}

* 1. **Compile Error**
  2. 10
  3. 20
  4. 15

1. What will be the output of this program?

#include <iostream>

using namespace std;

template<int n> struct funStruct

{

static const int val = 2\*funStruct<n-1>::val;

};

template<> struct funStruct<0>

{

static const int val = 1 ;

};

int main()

{

cout << funStruct<10>::val << endl;

return 0;

}

* 1. Compiler Error
  2. 1
  3. 2
  4. **1024**