* [**Home**](https://www.quantnet.com/)
* [**Forum**](https://www.quantnet.com/forum/)
* [**2015 MFE Rankings**](https://www.quantnet.com/mfe-programs-rankings/)
* [**Tracker**](https://www.quantnet.com/tracker/)
* [**Resources**](https://www.quantnet.com/resources/)
* [**Events**](https://www.quantnet.com/events/)
* WEI WU
* Member

Top of Form





Bottom of Form

* [**Your Profile Page**](https://www.quantnet.com/members/wei-wu.30273/)
* [**Personal Details**](https://www.quantnet.com/account/personal-details)
* [**Contact Details**](https://www.quantnet.com/account/contact-details)
* [**Privacy**](https://www.quantnet.com/account/privacy)
* [**Preferences**](https://www.quantnet.com/account/preferences)
* [**Alert Preferences**](https://www.quantnet.com/account/alert-preferences)
* [**Avatar**](https://www.quantnet.com/account/avatar)
* [**External Accounts**](https://www.quantnet.com/account/external-accounts)
* [**Password**](https://www.quantnet.com/account/security)
* [**Keyword Alert**](https://www.quantnet.com/account/keyword-alert)
* [**Your News Feed**](https://www.quantnet.com/account/news-feed)
* [**Likes You've Received**](https://www.quantnet.com/account/likes)
* [**Your Content**](https://www.quantnet.com/search/member?user_id=30273)
* [**People You Follow**](https://www.quantnet.com/account/following)
* [**People You Ignore**](https://www.quantnet.com/account/ignored)
* [**Account Upgrades**](https://www.quantnet.com/account/upgrades)
* [**Your Bookmarks**](https://www.quantnet.com/account/bookmarks)
* [**Log Out**](https://www.quantnet.com/logout/?_xfToken=30273%2C1450814881%2C7917d0dcda45f440f47b8ecce5cc00cb9b64100b)

Top of Form

** Show online status**

Bottom of Form

[**Conversations**](https://www.quantnet.com/conversations/)

* [**Start a New Conversation**](https://www.quantnet.com/conversations/add)
* [**Show All...**](https://www.quantnet.com/conversations/)

[**Alerts**](https://www.quantnet.com/account/alerts)

* [**Alert Preferences**](https://www.quantnet.com/account/alert-preferences)
* [**Show All...**](https://www.quantnet.com/account/alerts)
* [https://www.quantnet.com/styles/uiflex/uiflex/quantnet.logo.png](https://www.quantnet.com/)
* [**HOME**](https://www.quantnet.com/)
* [**FORUM**](https://www.quantnet.com/forum/)
* [**2015 MFE RANKINGS**](https://www.quantnet.com/mfe-programs-rankings/)
* [**TRACKER**](https://www.quantnet.com/tracker/)
* [**RESOURCES**](https://www.quantnet.com/resources/)
* [**EVENTS**](https://www.quantnet.com/events/)
* [**https://www.quantnet.com/data/avatars/s/30/30273.jpg?1445650467 WEI WU**](https://www.quantnet.com/account/)

[*QUIZ 6*](https://www.quantnet.com/quiz/18/view)>

**Attempt Result**

Top of Form

Question 1

When writing a template class, were should you use the class name with the template type (*MyClass<T>*)?

1.  When the template class is used as output argument.
2.  In the source file of a template class before the scope operator.
3.  When the template class is used as input argument.
4.  In the name of the constructor and destructor.

Question 2

Which statement is false about exceptions?

1.  Every type can be thrown. Not only primitive types but also your own class types.
2.  If a *try* block does not have any *catch* block, the exception will be passed to an outer *try* block or the system.
3.  One *try* block can have many *catch* blocks.
4.  If a *try* block does not have a matching catch block for the thrown exception, the exception will be passed to an outer *try* block or the system.
5. You answered this question correctly.

Question 3

Which statement is false about generic programming?

1.  Generic programming enables us to create type-safe data structures for a certain type without creating a new class for each type.
2.  You can't use generic programming and object-oriented programming at the same time.
3.  With generic programming, a generic data type is used in the code which at compile-time will be replaced by a specific type that is provided by the user of the generic code.
4.  Generic programming can be used as an alternative to polymorphic functions.
5. You answered this question correctly.

Question 4

Which statement is false about template functions?

1.  A C++ class can have template member functions with a different template type as the class.
2.  When calling a template function we always need to specify the template argument.
3.  C++ does only support template classes but not template global functions.

Question 5

Which statement is false about composition and exceptions?

1.  When using composition, it is preferred that the parent object does not catch the exceptions thrown in the composite object.
2.  When using composition, it is preferred that the parent object catches the exceptions thrown in the composite object and converts it to another exception object.
3.  When using composition, unhandled exceptions thrown in the composite object make that the client must know of the composite object.
4.  When using composition, the exceptions thrown in the composite object can be re-thrown with extra information.
5. You answered this question correctly.

Question 6

Which statement is false about sub-type polymorphism and parametric polymorphism?

1.  Sub-type polymorphism is done at run-time while parametric polymorphism is done at compile-time.
2.  Sub-type polymorphism depends on virtual functions.
3.  Parametric polymorphism does not work when the classes are not derived from a common base class.
4.  Parametric polymorphism is faster than inheritance polymorphism.

Question 7

Which statement is false about template classes?

1.  If a template class is defined as *template <typename T> class MyClass* and variables *a* and *b* are declared as *MyClass<int> a; MyClass<double> b;*, then variables *a* and *b* are of the same type.
2.  The template types are part of the class name.
3.  A template class is a description of a regular class.
4.  A class is an instance of a template class analog to that an object is an instance of a class.

Question 8

What is false about templates?

1.  When you use functionality on the template argument, you must document what you expect from the from the type passed as template argument.
2.  Template source files should also have an *#ifndef* construct.
3.  The member functions of a template class are only compiled when used. You won't get any potential compiler errors inside the function unless you use that function.
4.  It is no problem to use lots of functionality on the template type.

Question 9

Which statement is false about iterators?

1.  Iterators are used to traverse data structures in a data structure independend way.
2.  Iterators are a nested type of the data structure they iterate.
3.  In a loop that traverses a data structure, you cannot compare the current iterator with the end iterator using the < operator
4.  A *std::list<T>* supports a random access iterator.

Question 10

Which statement is false about STL?

1.  The STL library provides among others various data structures, itrators, algorithms and allocators.
2.  The STL library should be installed separately before you can use it.
3.  The STL library provides no functionality for networking.
4.  STL is a C++ library that uses templates for its implementation.
5. You answered this question correctly.

Result

Score:

4 correct answers out of 10 questions. You need at least 8 correct answers to pass this quiz.

Pass/Fail:

Failed ([Take this quiz again!](https://www.quantnet.com/quiz/18/take))

### Question 1

Which statement is false about composition and exceptions?

1.  When using composition, the exceptions thrown in the composite object can be re-thrown with extra information.
2.  When using composition, unhandled exceptions thrown in the composite object make that the client must know of the composite object.
3.  When using composition, it is preferred that the parent object does not catch the exceptions thrown in the composite object.
4.  When using composition, it is preferred that the parent object catches the exceptions thrown in the composite object and converts it to another exception object.
5. You answered this question correctly.

### Question 2

Uninstanciated templates cannot be stored in an .o file for the linker. Thus when using a template class, the implementation of the template class must be known at compile while with regular classes the implementation of the class can be linked later at lik time from an .o file. How can you make sure that the template implementation in known at compile-time?

1.  When using a template class, include the source file instead of the header file.
2.  When using a template class you can just include the header file as with regular classes.
3.  Implement the template class a inline in the header file.
4.  The template class header file can include the template class source file if that was not yet done. Then the user of the template class can just include the header file which in his turn includes the source file.
5. You answered this question correctly.

### Question 3

Which of the following data structures are provided by STL directly (thus is not an adapter)?

1.  Doubly linked list
2.  A last-in/first-out (LIFO) data structure.
3.  A first-in/first out (FIFO) data structure.
4.  Array that can grow.

### Question 4

Which statement is false about sub-type polymorphism and parametric polymorphism?

1.  Sub-type polymorphism depends on virtual functions.
2.  Parametric polymorphism does not work when the classes are not derived from a common base class.
3.  Sub-type polymorphism is done at run-time while parametric polymorphism is done at compile-time.
4.  Parametric polymorphism is faster than inheritance polymorphism.
5. You answered this question correctly.

### Question 5

Which statement is false about generic programming?

1.  You can't use generic programming and object-oriented programming at the same time.
2.  With generic programming, a generic data type is used in the code which at compile-time will be replaced by a specific type that is provided by the user of the generic code.
3.  Generic programming can be used as an alternative to polymorphic functions.
4.  Generic programming enables us to create type-safe data structures for a certain type without creating a new class for each type.
5. You answered this question correctly.

### Question 6

Which statement is false about template functions?

1.  When calling a template function we always need to specify the template argument.
2.  A C++ class can have template member functions with a different template type as the class.
3.  C++ does only support template classes but not template global functions.
4. You answered this question correctly.

### Question 7

Which statement is false about STL?

1.  STL is a C++ library that uses templates for its implementation.
2.  The STL library provides no functionality for networking.
3.  The STL library provides among others various data structures, itrators, algorithms and allocators.
4.  The STL library should be installed separately before you can use it.
5. You answered this question correctly.

### Question 8

When using the throw specification to specify that a function "F" can throw an exception of type "X", what statement is true about the throw specification?

1.  A caller of function "F" should catch the exception type "X" else we get a compiler error.
2.  The function "F" can only throw the exception type "X" else we get a compiler error.
3.  The throw specification is only for the user, the compiler does not use the information in the throw specification.
4.  The function "F" must have a throw statement for the exception type "X" else we get a compiler error.

### Question 9

Which statement is false about iterators?

1.  Iterators are used to traverse data structures in a data structure independend way.
2.  Iterators are a nested type of the data structure they iterate.
3.  In a loop that traverses a data structure, you cannot compare the current iterator with the end iterator using the < operator
4.  A *std::list<T>* supports a random access iterator.
5. You answered this question correctly.

### Question 10

When having the following code, what is the catch syntax for combining a polymorphic net with an explicit net?

Code (C++):

1. class E
2. {
3. public:
4. virtual std::string GetMessage() { return "Exception E"; }
5. };
7. class EA: public E
8. {
9. public:
10. virtual std::string GetMessage() { return "Exception EA"; }
11. };
13. class EB: public E
14. {
15. public:
16. virtual std::string GetMessage() { return "Exception EB"; }
17. };
19. void F()
20. {
21. throw EA();
22. }
24. int main()
25. {
26. try
27. {
28. F();
29. }
31. // Catch code here!
33. return 0;
34. }

1.  catch (E& ex) { std::cout<<ex.GetMessage()<<std::endl; } catch (EA& ex) { std::cout<<"Handling EA"<<std::endl; }
2.  catch (E& ex) { std::cout<<ex.GetMessage()<<std::endl; } catch (...) { std::cout<<"Unknown Exception"<<std::endl; }
3.  catch (EA& ex) { std::cout<<"Handling EA"<<std::endl; } catch (E& ex) { std::cout<<ex.GetMessage()<<std::endl; }
4.  catch (EA ex) { std::cout<<"Handling EA"<<std::endl; } catch (E ex) { std::cout<<ex.GetMessage()<<std::endl; }

Question 1

What statement is true about catch handlers?

1.  A catch handler that catches all possible exceptions should catch the Exception base class: catch (Exception& ex) {}
2.  A catch handler that catches all possible exceptions should catch "...": catch (...) {}
3.  A catch handler that catches all possible exceptions is not possible in C++.
4.  A catch handler that catches all possible exceptions should use the default keyword:*catch default { }*

Bottom of Form