Question 1

Which of the following statements is true about creating a copy constructor and assignment operator?

1.  We need to create a copy constructor and assignment operator because the automatically generated copy constructor and assignment operator copy the data wrongly in certain situations.
2.  We need to create a copy constructor and assignment operator because the "canonical header file rules" dictates it.
3.  We need to create a copy constructor and assignment operator because the automatically generated copy constructor and assignment operator do nothing.
4.  We do not need to create a copy constructor and assignment operator because the automatically generated copy constructor and assignment operator do already a member copy.

Question 2

What statement is true about inheriting operators?

1.  When you don't create an assignment operator in the derived class, assigning two derived class objects will only copy the base class data.
2.  Operators from the base class cannot be overridden in the derived class.
3.  The equal compare operator is inherited but it will only compare the base class data.
4.  All operators are inherited.
5. You answered this question correctly.

Question 3

Which statement is true about the following C code?

Code (C++):

1. int i,j;
2. for (i=0, j=4; i<j;)
3. {
4. printf("%d, %d**\n**", i++, j--);
5. }

1.  This code does not compile because the *for* statement is missing the post iteration expression.
2.  This code executes the loop code two times.
3.  This code does not compile because the comma is not supported in the loop initialisation expression.
4.  This code is an infinite loop.

Question 4

What is the correct syntax to create a boost shared pointer to an *MyClass* object?

1.  boost:shared\_ptr<MyClass\*> mc(new MyClass);
2.  boost:shared\_ptr<MyClass> mc=new MyClass;
3.  boost:shared\_ptr<MyClass> mc(new MyClass);
4.  boost:shared\_ptr<MyClass> mc(MyClass());
5. You answered this question correctly.

Question 5

Which statement is false about iterators?

1.  An output iterator can only write to the current position once and must then be incremented. The current position cannot be read from.
2.  On a random access iterator you can use the square bracket operator [] to access elements a few steps before or after the current iterator position.
3.  An input iterator can only read from the current position once and must then be incremented. The current position cannot be written to.
4.  A forward iterator can read and write the current position multiple times. You can read what you just wrote.

Question 6

Which statement is false about iterators?

1.  Insert iterators are adaptors that transform an assignment (\*it=value) to an insert, push\_back or push\_front operation on a container.
2.  Stream iterators are adapters that allows us to use a stream as source or destination in code that uses iterators.
3.  On iterators you can only use the pre-increment (++it) operator and not the post-increment (it++).
4. On iterators you can use both pre- and post-increment version. Only the pre-increment version is often faster than the post increment.
5.  With the correct use of iterators (in combination with templates) you can write functions that work with every STL container.
6. You answered this question correctly.

Question 7

Which statement is true about the following program?

Code (C++):

1. int Swap(int, int);
3. int main()
4. {
5. int i1=10;
6. int i2=20;
7. int i3=Swap(i1, i2);
9. printf("%d %d %d**\n**", i1, i2, i3);
10. }
12. int Swap(int a, int b)
13. {
14. int tmp=a;
15. a=b;
16. b=tmp;
18. return a;
19. }

1.  The program does not compile because the *Swap()* function is declared twice.
2.  The program does not compile because the first *Swap()* does not define names for the input variables.
3.  The output of this program is: 20 10 20
4.  The output of this program is: 10 20 20

Question 8

What statement is false about function name overloading?

1.  Two functions can have the same name as long as the input arguments have different types.
2.  Two functions can have the same name as long as the number of input arguments are different.
3.  Two functions can have the same name as long as the output arguments have different types.
4.  Overloading works with both member functions and global functions.
5. You answered this question correctly.

Question 9

Which statement is false about random distributions?

1.  We are required to use a distribution in combination with a random number generator.
2.  The *uniform\_real\_distribution* transforms the random numbers to floating point values in a specific range.
3.  The *uniform\_int\_distribution* transforms the random numbers to integers in a range where the chance to get each integer value is the same.
4.  The *discrete\_distribution* transforms the random numbers to a set of numbers where the chance to get each value can be different.

Question 10

What is the output of the following program?

Code (C++):

1. printf("%d**\n**", 6^12);

1.  18
2.  4
3.  14
4.  10
5. You answered this question correctly.

Question 11

Which statement is false about STL?

1.  The STL library provides no functionality for networking.
2.  The STL library should be installed separately before you can use it.
3. STL is part of the C++ specification and every C++ compiler must include an STL implementation. So you can use it without any additional installation.
4.  STL is a C++ library that uses templates for its implementation.
5.  The STL library provides among others various data structures, itrators, algorithms and allocators.
6. You answered this question correctly.

Question 12

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.  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.
3. At the end of the header file you can include the source so the source become part of the header file. E.g. *#ifndef MyClass\_cpp/#include "MyClass.cpp"/#endif*.
4.  Implement the template class a inline in the header file.
5.  When using a template class you can just include the header file as with regular classes.
6. You answered this question correctly.

Question 13

What statement is true about the following code?

Code (C++):

1. class A;

1.  'A' is a forward declaration. The body is implemented elsewhere.
2.  'A' is a local variable of type *class*.
3.  'A' is an empty class.
4.  This code does not compile.

Question 14

Which statement is true about the following program?

Code (C++):

1. int x=4.6;
2. printf("%d**\n**", x);

1.  The compiler issues an error because we assign a double value to an *int* variable.
2.  The compiler does not issue a warning or error but we get a runtime error when assigning the *double* value to the *int* variable.
3.  The compiler issues a warning because we assign a *double* value to an *int* variable but the program runs and outputs 5.
4.  The compiler issues a warning because we assign a *double* value to an *int*variable but the program runs and outputs 4.
5. You answered this question correctly.

Question 15

Which statement is true about the following code?

Code (C++):

1. delete[] x;

1.  The code is wrong because the [] are missing the size to delete.
2.  It deallocates an array pointed by variable *x*.
3.  It deallocates the first element of an array pointed by variable *x*.
4.  The code is wrong because the [] are not supported with delete.
5. You answered this question correctly.

Question 16

Which statement is true about destructors and inheritance?

1.  Destructors must be virtual so that derived class object in a base class variable will be deleted correctly.
2.  The system provided destructor is always called correctly.
3.  Destructors must be virtual so that the base class destructor is called when a derived class object is removed from memory.
4.  Destructors must be virtual because the canonical header file requires that.

Question 17

Which of the following options declares and allocates an array of *int* pointers?

1.  int\*[] array=new int\*[size];
2.  int[] array=new int\*[size];
3.  int\*\* array=new int\*[size];
4.  int\* array=new int\*[size];

Question 18

Which two statements are true about variables?

1.  Variables must be declared with name and data type.
2.  Variables can be assigned a value when declaring the variable or later.
3.  Variables have as default value 0.
4.  Multiple variables must be declared in separate statements.
5. You answered this question correctly.

Question 19

Which of the options below is the best operator declaration to add a two objects of type*Complex* (*Complex+Complex*)?

1.  Complex& operator + (const Complex& c1, const Complex& c2) const;
2.  Complex operator + (const Complex& c);
3.  Complex& operator + (const Complex& c) const;
4.  Complex operator + (const Complex& c) const;

Question 20

What statement is false about "pass by value" vs. "pass by reference"?

1.  Pass by value makes a copy of the argument.
2.  Pass by value is less efficient than pass by reference for objects.
3.  To pass an argument by reference you need to declare the input parameter with a '&'
4.  To pass an argument by reference you need to declare the input parameter with a '\*'.
5. You answered this question correctly.

Question 1

What statement is true about the following code?

Code (C++):

1. namespace
2. {
3. int x=20;
4. }

1.  Variable *x* is inaccessible because the namespace has no name.
2.  This does not compile because the namespace has no name.
3.  The variable *x* is a global variable only accessible in the current compilation unit and not as external global variable in another compilation unit.
4.  This code defines a local variable *x* in its own scope.

Question 2

Which statement is false about iterators?

1.  On a random access iterator you can use the square bracket operator [] to access elements a few steps before or after the current iterator position.
2.  A forward iterator can read and write the current position multiple times. You can read what you just wrote.
3.  An output iterator can only write to the current position once and must then be incremented. The current position cannot be read from.
4.  An input iterator can only read from the current position once and must then be incremented. The current position cannot be written to.
5. An input iterator can read the current position multiple times.
6. You answered this question correctly.

Question 3

Which of the operator declarations below is the best way to support the index operator for integer indices (*[int]*)?

1.  const Type& operator [] (int index) const; Type operator [] (int index);
2.  Type operator [] (int index) const;
3.  const Type& operator [] (int index) const; Type& operator [] (int index);
4.  Type& operator [] (int index) const;
5. You answered this question correctly.

Question 4

Which two statements are false about STL containers?

1.  Vectors generally allocate more memory than needed for the elements it stores.
2.  Elements stored in an STL container must be copyable (must provide a copy constructor).
3.  Arguments STL container operations are checked for correctness.
4. Not all container operations are safe. The user often has to make sure the arguments are correct. For example providing an invalid index to the vectors's square bracket operator will result in undefined behaviour and not an exception.
5.  STL containers cannot store pointers.
6. It is no problem to define a container that stores pointers. Just pass a pointer tpye as templater argument (e.g. *vector<int\*> v;*). Of course when using pointer, you are responsible for memory management of the objects.
7. You answered this question correctly.

Question 5

Which statement is false about functions?

1.  Functions can only have one return argument.
2.  Functions can have zero or more input arguments.
3.  When the function returns nothing, you don't need to specify a return type.
4.  Functions implemented in a source file can be called from another source file.
5. You answered this question correctly.

Question 6

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. STL is part of the C++ specification and every C++ compiler must include an STL implementation. So you can use it without any additional installation.
4.  STL is a C++ library that uses templates for its implementation.
5.  The STL library provides no functionality for networking.
6. You answered this question correctly.

Question 7

Which statement is false about iterators?

1.  Insert iterators are adaptors that transform an assignment (\*it=value) to an insert, push\_back or push\_front operation on a container.
2.  With the correct use of iterators (in combination with templates) you can write functions that work with every STL container.
3.  Stream iterators are adapters that allows us to use a stream as source or destination in code that uses iterators.
4.  On iterators you can only use the pre-increment (++it) operator and not the post-increment (it++).
5. On iterators you can use both pre- and post-increment version. Only the pre-increment version is often faster than the post increment.
6. You answered this question correctly.

Question 8

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.  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.  You can't use generic programming and object-oriented programming at the same time.
5. Generic programming can be used in combination with object-oriented programming without progrom. For example template classes can derive from each other.
6. You answered this question correctly.

Question 9

Which statement is false about an abstract class?

1.  Abstract classes can contain member data.
2.  Abstract classes enable us to work uniformly with a set of related classes.
3.  Abstract classes cannot be instantiated.
4.  Abstract classes can only have function declarations, no function implementations.
5. You answered this question correctly.

Question 10

What is the incorrect syntax to create a variant for a double, int or string or the syntax to extract the value?

1.  double d=boost::get<double>(v);
2.  boost::variant<double, int, string> v=40.0;
3.  boost::variant<double, int, string> v(40.0);
4.  double d=boost::get(v);

Question 11

What is the output of the following code?

Code (C++):

1. int size=3; int\* a=new int[size];
2. for (int i=0; i<size; i++) a[i]=10-i;
3. std::cout<<a[1]<<", "<<\*a<<", "<<(a+1)[0]<<", "<<\*a+1<<std::endl;
4. delete[] a;

1.  9, [address of variable *a*], 9, [address of variable *a* + *sizeof(int)*]
2.  9, 10, 9, 9
3.  9, 10, 11, 9
4.  9, 10, 9, 11
5. You answered this question correctly.

Question 12

Which statement is true about *const*?

1.  Only const variables can be passed to functions with const parameters.
2.  Const member functions make the current object state const during that function.
3.  Const member functions can't change any data.
4.  You cannot have a const and a non-const member function with the same name and input- and output-arguments.

Question 13

Which statement is false about Boost?

1.  Boost is cross-platform and has support for most modern C++ compilers.
2.  The boost library is largely implemented using templates.
3.  Boost is an open-source library and can be used freely in non-commercial and commercial applications.
4.  Boost is a C++ library that is standard available in C++ compilers.
5. Boost is not included in C++ compilers and need to be installed seperately. However, some parts of boost will become part of the standard library in the upcoming C++ 2011 standard.
6. You answered this question correctly.

Question 14

What statement is true about the following class?

Code (C++):

1. // In the header file: "MyClass.hpp"
2. class MyClass
3. {
4. int m\_data1;
6. MyClass();
8. private:
9. double m\_data2;
10. }
12. // In the source file: "MyClass.cpp"
13. #include "MyClass.hpp"
14. MyClass::MyClass()
15. {
16. }

1.  Client code can instantiate this class because it defines a default constructor.
2.  The initial value of *m\_data2* is 0.0.
3.  *m\_data1* is a public data member.
4.  This class declaration does not compile.
5. You answered this question correctly.

Question 15

Which statement is false about inline functions?

1.  Member functions implemented within the class definition must also have the keyword *inline* to be compiled as inline.
2.  Functions declared as inline may not be compiled as regular function.
3.  The implementation of an inline function must be available at compile time. The function cannot be inlined when the function implementation is only available at link time.
4.  Inline functions can be executed faster than non inline functions.
5. You answered this question correctly.

Question 16

What statement is true about the following code?

Code (C++):

1. #include <iostream>
3. void Swap(int& a, int& b)
4. {
5. int tmp=a;
6. a=b;
7. b=tmp;
8. }
10. int main()
11. {
12. int i1=10;
13. int i2=20;
15. Swap(i1, i2);
17. std::cout<<"i1="<<i1<<", i2="<<i2<<std::endl;
18. }

1.  The parameters to the *Swap* function are passed as pointer.
2.  The program does not compile.
3.  The parameters to the *Swap()* are passed by value.
4.  The parameters to the *Swap* function are passed by reference.
5. You answered this question correctly.

Question 17

Which statement is true about the following code?

Code (C++):

1. int i=0;
2. while (i<2);
3. {
4. printf("%d ", i);
5. }

1.  The program infinitely print: 0 0 0 0 0 0 0 0 0
2.  The program prints nothing and hangs.
3.  The program will print: 1 2
4.  The program will print: 0 1
5. You answered this question correctly.

Question 18

Which two statements are true about variables?

1.  Variables have as default value 0.
2.  Variables must be declared with name and data type.
3.  Multiple variables must be declared in separate statements.
4.  Variables can be assigned a value when declaring the variable or later.
5. You answered this question correctly.

Question 19

Which statement is false about polymorphism?

1.  When a derived class object is stored in a base class variable, the derived class object functions are hidden.
2.  Polymorphism is automatically turned on in C++.
3.  When calling a function via reference/pointer to a base class object, polymorphism enables variation of behaviour depending on the actual derived class object stored in the variable.
4.  Polymorphism only works with pointer or reference variables.

Question 20

Which statement is false about sequence containers?

1.  Sequence containers store their data linearly.
2.  The *list<T>::pop\_front()* function does only remove the first element but does not return the first element.
3. To get the first element you need to use the *front()* function.
4.  Sequence containers order their data.
5.  A vector is like an array that can add elements at the end but not at the beginning of the array.

Question 1

Which statement is true about data hiding?

1.  Data hiding hides the internal data of a class from users of the class so the internal structure can be changed without affecting the users of a class.
2.  Data hiding is mandatory in C++.
3.  Data hiding ensures the data cannot be changed.
4.  Data hiding saves memory space.

Question 2

Which of the following statements states three key features of object oriented programming?

1.  Classes, objects and variables.
2.  Data hiding, classes and encapsulation.
3.  Functions, variables and classes.
4.  Encapsulation, data hiding and inheritance.
5. You answered this question correctly.

Question 3

Which statement is false about friends?

1.  Friends violate the information hiding principle of object-oriented programming.
2.  A class cannot access the private members of its friend classes.
3.  Friend functions can access the private members of the class they are friend of.
4.  Friend functions can access the '*this*' pointer.

Question 4

Which statement is false about STL?

1.  The STL library should be installed separately before you can use it.
2. STL is part of the C++ specification and every C++ compiler must include an STL implementation. So you can use it without any additional installation.
3.  The STL library provides among others various data structures, itrators, algorithms and allocators.
4.  The STL library provides no functionality for networking.
5.  STL is a C++ library that uses templates for its implementation.
6. You answered this question correctly.

Question 5

Which statements are false about algorithms?

1.  Removing algorithms are a special kind of mutating algorithms.
2. Removing algorithms are a special kind of modifying algorithms, not mutating algorithms.
3.  Modifying algorithms can modify the elements of data structures and change the order.
4. Modifying algorithms do not change the order of the data in a cotainer.
5.  Mutating algorithms change the order of elements but not the elements themself.
6.  STL algorithms accept a start- and end-iterator instead of the complete container.
7. You answered this question correctly.

Question 6

Which statement is true about the following code?

Code (C++):

1. delete[] x;

1.  The code is wrong because the [] are not supported with delete.
2.  It deallocates the first element of an array pointed by variable *x*.
3.  It deallocates an array pointed by variable *x*.
4.  The code is wrong because the [] are missing the size to delete.
5. You answered this question correctly.

Question 7

What is the correct syntax to create a boost shared pointer to an *MyClass* object?

1.  boost:shared\_ptr<MyClass> mc(MyClass());
2.  boost:shared\_ptr<MyClass> mc=new MyClass;
3.  boost:shared\_ptr<MyClass\*> mc(new MyClass);
4.  boost:shared\_ptr<MyClass> mc(new MyClass);
5. You answered this question correctly.

Question 8

Which statement is false about the following code?

Code (C++):

1. // Predicate determining if the value satisfies a criteria.
2. struct Predicate
3. {
4. bool operator()(int v)
5. {
6. return (v%2)>0;
7. }
8. };
10. int main()
11. {
12. vector<int> v(5);
13. v[0]=10; v[1]=14; v[2]=9; v[3]=15; v[4]=8;
15. // Find the first number satisfying the given criteria.
16. vector<int>::iterator result=find\_if(v.begin(), v.end(), Predicate());
17. cout<<"First number found: "<<\*result<<endl;
18. }

1.  Instead of a class with an operator round bracket (function object) we can also pass a global function to the *find\_if()* function.
2.  The predicate determines what element will be found.
3.  This code finds the first even number in the vector.
4. The code finds the first odd number, not the first even number.
5. You answered this question correctly.

Question 9

What statement is true about the following code?

Code (C++):

1. #include <iostream>
3. void Swap(int& a, int& b)
4. {
5. int tmp=a;
6. a=b;
7. b=tmp;
8. }
10. int main()
11. {
12. int i1=10;
13. int i2=20;
15. Swap(i1, i2);
17. std::cout<<"i1="<<i1<<", i2="<<i2<<std::endl;
18. }

1.  The program does not compile.
2.  The parameters to the *Swap()* are passed by value.
3.  The parameters to the *Swap* function are passed as pointer.
4.  The parameters to the *Swap* function are passed by reference.
5. You answered this question correctly.

Question 10

Which statement is true about *const*?

1.  Const member functions make the current object state const during that function.
2.  Const member functions can't change any data.
3.  Only const variables can be passed to functions with const parameters.
4.  You cannot have a const and a non-const member function with the same name and input- and output-arguments.
5. You answered this question correctly.

Question 11

What is the output of the following program?

Code (C++):

1. class A
2. {
3. public:
4. virtual std::string F() { return "A"; }
5. };
7. class B: public A
8. {
9. public:
10. std::string F() { return "B"; }
11. };
13. int main()
14. {
15. A a; B b;
16. A\* ap;
18. std::cout<<a.F()<<", ";
19. std::cout<<b.F()<<", ";
21. ap=&a; std::cout<<ap->F()<<", ";
22. ap=&b; std::cout<<ap->F()<<std::endl;
24. return 0;
25. }

1.  This code does not compile because the derived class function F() must also be virtual.
2.  A, B, A, B
3.  A, A, A, A
4.  A, B, A, A

Question 12

Which statements are true about the following code?

Code (C++):

1. // Print the list contents.
2. template <typename T>
3. void Print(const T& ds)
4. {
5. // Typedef for the iterator to simplify code.
6. typedef T::const\_iterator iterator;
8. // Print the list elements.
9. cout<<"Data: ";
10. iterator end=ds.end();
11. for (iterator it=ds.begin(); it!=end; it++) cout<<\*it<<", ";
12. cout<<endl;
13. }
15. int main()
16. {
17. vector<int> v(5);
18. v[0]=10; v[1]=14; v[2]=9; v[3]=15; v[4]=8;
19. Print(v);
20. }

1.  Instead of *it!=end* we can also use *it<end* because the iterator of vector supports the < operator
2.  For the typedef we can also use *list<T>::iterator* instead of *const\_iterator*.
3.  To make this code more flexible, you can change the *Print()* function to accept two iterators.
4. The current *Print()* function prints the whole data structure. If two iterators are accepted then also a sub range of a data structure can be printed (start- and end-iterator).
5.  Instead of an std::vector, you can pass an std::list or any other data structure that supports input iterators.
6. Since the *Print()* function is templated for the data structure type, it can accept anything that supports the input iterator concept.
7. You answered this question correctly.

Question 13

Which of the two statements below are true about the *break* and *continue* keywords?

1.  The continue keyword exits a looping statement.
2.  The *break* keyword exits a looping statement.
3.  The break keyword can only be used in a *switch* statement.
4.  The *continue keyword aborts the current loop iteration and continues the next iteration.*
5. You answered this question correctly.

Question 14

Local variables can be defined at?

1.  Only at the beginning of a function in both C and C++.
2.  Everywhere in a function in both C and C++.
3.  Only at the beginning of a function in C and everywhere in a function in C++.
4.  C and C++ don't have local variables.

Question 15

What statement is false about constructors and destructors?

1.  You can have multiple constructors and multiple destructors.
2.  A constructor initialises an object.
3.  A destructor is used to clean up resources.
4.  A destructor is automatically called when an object gets out of scope.
5. You answered this question correctly.

Question 16

Which statement is false about the boost random library?

1.  The numbers generated by boost random number generators are really random.
2. The numbers are not really random. The numbers are a (very long) calculated sequence of numbers that look random but eventually repeat.
3.  To get random numbers in a specific range, you need to use a distribution in combination with the random generator.
4.  The boost random library provides various algorithms for generating random numbers.
5.  To get different random numbers each time you need to set the seed of a random number generator.
6. You answered this question correctly.

Question 17

Which statement is true about operator overloading?

1.  The overload operator is a special operator to enable function overloading.
2.  With operator overloading we can give a different meaning to existing operators e.g. when using the + operator on two doubles.
3.  Operator overloading enable us to define a special member function that is called when you use an operator on an instance of the class.
4.  Operator functions must be member functions.

Question 18

What is false about templates?

1.  It is no problem to use lots of functionality on the template type.
2. Template classes should in general require as less as possible from the template type. If you require lots of functionality from the template type, then you will limit the number of types that are suitable as template argument. When more functionality is required, often you can create a sub class that is more specialised. For example, an *Array<T>* class requires no functionality on T and can store any type. A *NumericArray<T>* derived from*Array<T>* adds numeric functionality and requires that T supports mathematical operations. This limits the types that can be stored in the numeric array but those types can still be used in the base array class.
3.  When you use functionality on the template argument, you must document what you expect from the from the type passed as template argument.
4.  Template source files should also have an *#ifndef* construct.
5.  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.
6. You answered this question correctly.

Question 19

What are the building steps from C/C++ source code to executable?

1.  Pre-processing, compiling and linking.
2.  Pre-processing, compiling, linking and post-processing.
3.  Pre-processing and compiling.
4.  Compiling and linking.

Question 20

Which statement is true about destructors and inheritance?

1.  The system provided destructor is always called correctly.
2.  Destructors must be virtual because the canonical header file requires that.
3.  Destructors must be virtual so that the base class destructor is called when a derived class object is removed from memory.
4.  Destructors must be virtual so that derived class object in a base class variable will be deleted correctly.

Question 1

Which statement is false about template classes?

1.  The template types are part of the class name.
2.  A class is an instance of a template class analog to that an object is an instance of a class.
3.  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.
4. For each type that is used as template argument, a new class is generated. So*MyClass<int>* and *MyClass<double>* are two different types.
5.  A template class is a description of a regular class.
6. You answered this question correctly.

Question 2

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. E.g.: *MyClass<T> F() const;*
3.  When the template class is used as input argument.
4. E.g.: *void F(const MyClasss<T> d);*
5.  In the source file of a template class before the scope operator.
6. E.g.: *void MyClass<T>::F(){}*
7.  In the name of the constructor and destructor.
8. You answered this question correctly.

Question 3

Which statement is false about the following code assuming class *MyClass* has a member function called *F()*?

Code (C++):

1. MyClass\* mc=new MyClass;

1.  Function *F()* can be called as follows: *(\*mc).F();*
2.  Function *F()* can be called as follows: *mc->F();*
3.  The code is wrong because you need to add round brackets when dynamically creating an object: *new MyClass()*
4.  The code creates a single *MyClass* object on the heap using the default constructor.

Question 4

Which statement is false about the colon syntax?

1.  The colon syntax must be used to initialise reference data members.
2.  The colon syntax must be used to initialise const data members.
3.  The colon syntax is more efficient when initialising complex data members.
4.  The colon syntax is just another way to initialise your data members. No performance advantages.
5. You answered this question correctly.

Question 5

What statement is true about the following code?

Code (C++):

1. class A;

1.  'A' is a local variable of type *class*.
2.  'A' is a forward declaration. The body is implemented elsewhere.
3.  This code does not compile.
4.  'A' is an empty class.
5. You answered this question correctly.

Question 6

Which statement is true about aggregation (open diamond shape in UML) and composition (filled diamond shape in UML)

1.  With aggregation the lifetime of the embedded object is the same as the 'outer' object while with composition the lifetime can be different.
2.  Aggregation embeds multiple objects while composition only embeds one object.
3.  There is no difference between aggregation and composition.
4.  With composition the lifetime of the embedded object is the same as the 'outer' object while with aggregation the lifetime can be different.
5. You answered this question correctly.

Question 7

Which statement is false about interfaces?

1.  Interfaces can be used instead of inheritance when there is no 'ISA' relation.
2.  Interfaces are allowed to provide default implementations.
3.  There is not interface construct in C++ but they can be emulated using abstract classes/functions.
4.  A function working with an interface can work with any set of unrelated classes as long as they implement that interface.
5. You answered this question correctly.

Question 8

Which of the operator declarations below is the best way to support the index operator for integer indices (*[int]*)?

1.  const Type& operator [] (int index) const; Type& operator [] (int index);
2.  Type operator [] (int index) const;
3.  const Type& operator [] (int index) const; Type operator [] (int index);
4.  Type& operator [] (int index) const;
5. You answered this question correctly.

Question 9

Which statement is false about boost variants?

1.  When retrieving a value from the variant, we can use the *get<T>()* global function.
2.  A variant can contain one value of a given collection of types.
3.  A variant is like a type-safe C union and boost variant can contain class types while a union can only contain the build-in data types.
4.  A variant can contain one value of any type.
5. A variant can only contain a value of the types that were specified as template arguments when creating the variant.
6. You answered this question correctly.

Question 10

Which two statements are true about looping statements?

1.  The *for* loop can always be used to replace a *while* loop.
2.  The *do...while loop is executed as long an expression is true.*
3.  The *while* loop is executed one or more times.
4.  The *repeat...until* loop is executed until an expression becomes true.

Question 11

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.
5. Template classes should in general require as less as possible from the template type. If you require lots of functionality from the template type, then you will limit the number of types that are suitable as template argument. When more functionality is required, often you can create a sub class that is more specialised. For example, an *Array<T>* class requires no functionality on T and can store any type. A *NumericArray<T>* derived from*Array<T>* adds numeric functionality and requires that T supports mathematical operations. This limits the types that can be stored in the numeric array but those types can still be used in the base array class.
6. You answered this question correctly.

Question 12

Which statement is false about sequence containers?

1.  A dequeu can insert elements at the beginning and extract elements from the end but can also insert elements at the end and extract elements from the beginning.
2. A deque is a double ended queue so it supports pushing and popping element on both sides. In contrast a regular queue supports pushing only at the beginning and popping only at the end.
3.  Inserting elements in an *std::list* is faster than inserting elements in an *std::vector*.
4.  The *std::list* class supports the square bracket [] operator.
5.  Traversing to a certain element in a vector is faster than in a list.

Question 13

What is the output of the following program?

Code (C++):

1. int x=4;
2. printf("%d**\n**", (x\*=4, x+2));

1.  This program does not compile.
2.  16
3.  6
4.  18
5. You answered this question correctly.

Question 14

Local variables can be defined at?

1.  Everywhere in a function in both C and C++.
2.  C and C++ don't have local variables.
3.  Only at the beginning of a function in both C and C++.
4.  Only at the beginning of a function in C and everywhere in a function in C++.
5. You answered this question correctly.

Question 15

Which statement is false about sequence containers?

1.  A vector is like an array that can add elements at the end but not at the beginning of the array.
2.  Sequence containers order their data.
3. Data in a sequence container are not automatically sorted.
4.  The *list<T>::pop\_front()* function does only remove the first element but does not return the first element.
5.  Sequence containers store their data linearly.
6. You answered this question correctly.

Question 16

Which of the options below is the best description of a canonical header file?

1.  A header file with minimal a default constructor, copy constructor, destructor and assignment operator.
2.  A header file with minimal a constructor, destructor, assignment operator and equal compare operator.
3.  A header file with minimal a default constructor, copy constructor, destructor and equal compare operator.
4.  A header file with minimal a constructor, destructor, equal compare operator and not equal compare operator.
5. You answered this question correctly.

Question 17

Which of the following statements is true about creating a copy constructor and assignment operator?

1.  We need to create a copy constructor and assignment operator because the "canonical header file rules" dictates it.
2.  We need to create a copy constructor and assignment operator because the automatically generated copy constructor and assignment operator copy the data wrongly in certain situations.
3.  We need to create a copy constructor and assignment operator because the automatically generated copy constructor and assignment operator do nothing.
4.  We do not need to create a copy constructor and assignment operator because the automatically generated copy constructor and assignment operator do already a member copy.
5. You answered this question correctly.

Question 18

Which statement is false about random distributions?

1.  The *discrete\_distribution* transforms the random numbers to a set of numbers where the chance to get each value can be different.
2.  The *uniform\_int\_distribution* transforms the random numbers to integers in a range where the chance to get each integer value is the same.
3.  We are required to use a distribution in combination with a random number generator.
4. A random number generator can also be used without distribution but then you get every possible value while in a lot of situations we want to limit the possible values.
5.  The *uniform\_real\_distribution* transforms the random numbers to floating point values in a specific range.
6. You answered this question correctly.

Question 19

What statement is true about the following code?

Code (C++):

1. namespace A::B
2. {
3. class MyClass
4. {
5. };
6. }

1.  This code defines a class in the single namespace called *A::B*.
2.  This code defines a class in namespace *A* that is nested in namespace *B*.
3.  This code defines a class in namespace *B* that is nested in namespace *A*.
4.  This code does not compile.

Question 20

What is the output of the following program?

Code (C++):

1. double x=5/2;
2. printf("%.1f**\n**", x);

1.  2.5
2.  2.5\n
3.  2
4.  2.0
5. You answered this question correctly.