# Operator overloads

## Given class X, which expression calls the std::string conversion operator?

class X {

std::string str;

public:

X(const char \* cstr) : str(cstr) {}

operator std::string() const { return str; }

};

1. X foo {"foo"}; print("{}\n", foo);
2. X foo {"foo"}; X s = foo;
3. X foo {"foo"}; const char\* s = foo;
4. X foo {"foo"}; std::string s = foo;

## When would you need to define non-member operators?

1. You define non- member operators when you anticipate objects of the given class on the right-hand side of an operator.
2. Non-member operators are always better than member operators.
3. Use non-member operators when your code doesn't otherwise work as expected.
4. There's no good reason to use non-member operators.

## What is the purpose of the function operator?

1. to provide an object that works like a function
2. to provide parameters to a function
3. to call a function
4. to return a function

# Move semantics

## What is an rvalue?

1. An rvalue may be on either side of an assignment.
2. An rvalue can only be on the right-hand side of an assignment.
3. An rvalue can be moved.
4. An rvalue can only be moved.

## Given class X, What is the signature for a move constructor?

1. X(const X& rhs);
2. X(X&& rhs);
3. X(X& rhs);
4. X(X&& rhs) noexcept;

## What does the std::move() function do?

1. casts a value to an rvalue reference, if possible
2. tests if a value is movable
3. moves a value
4. forces a value to be movable

## What is the value of the copy-and-swap idiom?

1. It has no additional value.
2. It both copies and swaps.
3. It's faster than a copy operator.
4. It is both more efficient and more succinct.

## Given class X, What is the signature for a move assignment operator?

1. X& operator=(const X& rhs);
2. X& operator=(X& rhs);
3. X& operator=(X&& rhs);
4. X& operator=(X&& rhs) noexcept;

## How does a copy/swap operator double as a move assignment operator?

1. The rhs parameter is changed by the compiler to an rvalue reference.
2. It calls your move assignment operator when the operand is movable.
3. It doesn't. You still need a move assignment operator.
4. The copy/swap operator calls the move constructor when the operand is movable.

## What is an lvalue?

1. An lvalue can never be assigned to.
2. An lvalue may be on either side of an assignment.
3. An lvalue is any object that may appear on the left-hand side of an assignment.
4. An lvalue can be moved.

## What is the difference between a copy and a move operation?

1. Move is safer.
2. Copy makes a copy of the value to a new object; Move re-associates the value with a different object.
3. Copy is safer.
4. Move is always better.

## When should you implement the rule of three and five?

1. always define all of the relevant member functions
2. when you support move semantics
3. It's not really important.
4. when you have defined any of the following member functions: destructor, copy constructor, copy assignment operator, or the move equivalents

# Lambda expressions

## What is a closure?

1. Any lambda is a closure.
2. A lambda expression that uses a captured value or values.
3. A lambda is a closure when it's used as a predicate.
4. A lambda that returns a value.

## When are the parameter parentheses () optional in a lambda definition?

1. if there are no parameters
2. always optional
3. if there are no parameters and no specifiers or trailing return type
4. never optional

## How would I declare a map of lambdas, given a key type of char?

1. map<char,void<>>
2. map<char,lambda>
3. map<char,void(\*)()> or map<char,std::function<void()>>
4. map<char,[](){}>

## What is a predicate?

1. A lambda expression that takes two parameters.
2. Any lambda expression is a predicate.
3. A lambda expression that works with an algorithm.
4. A lambda expression that returns a boolean value of true or false.

# Constraints and concepts

## In the following template header, what does the requires keyword do?

template<typename T> requires std::integral<T>

1. specifies the constraint
2. specifies a concept or boolean expression for use as a constraint
3. specifies an expression for use as a constraint
4. specifies a concept for use as a constraint

## What does a constraint do?

1. constrains a parameter to a set of values
2. specifies the requirements of a template parameter
3. specifies the parameters for a function
4. specifies the requirements of a function parameter

## In the following code, what does the requires keyword do?

template<typename T> concept Numeric = requires(T a) { a + 1; a \* 1; };

1. introduces a function that expresses a concept
2. introduces a block of code that defines a concept
3. introduces a template that defines a concept
4. introduces a boolean requirement for satisfying a user-defined concept

# Optionals and Variants

## Does the variant class supercede the primitive union type?

1. Yes, variant is a type-safe version of union.
2. No, the primitive union provides overlapping storage for multiple types. The variant class holds one strongly-typed value at a time.
3. No, but union is not type-safe so you should never use it.
4. Yes, variant is simply a modern, object-oriented replacement for union.

## Why use the optional class instead of a struct or other solution?

1. The optional class is made for flexibility and simplicity.
2. The optional class is modern and object-oriented.
3. The optional class is correct and everything else is incorrect.
4. The optional class is the only solution that works.

## What is the purpose of the any class?

1. The any class can store multiple objects that coexist in the same space.
2. The any class can store many objects of many types.
3. The any class can store untyped values.
4. An any object stores one type-safe instance of an object of any type.

# Concurrency

## What is the difference between sleep\_for and sleep\_until?

1. sleep\_for doesn't require a thread.
2. sleep\_for requires the chrono library.
3. They are the same.
4. sleep\_for will sleep for a specified interval; sleep\_until will sleep until a specified point in time.

## How does async return values from a thread?

1. async cannot return values
2. async returns values in a future object
3. async returns values in a promise object
4. async returns values on the stack

## What does the thread::join() method do?

1. The join() method blocks execution of the caller until the thread completes.
2. The join() method joins the thread with the caller.
3. The join() method executes the caller before the thread.
4. The join() method gives priority to the thread.

## What types qualify for use with std::atomic?

1. std::atomic requires a primitive type.
2. std::atomic only works with integral types.
3. std::atomic works with any simple type.
4. std::atomic requires a trivial type. All primitive types are trivial, including bool, int, float, and double.

## What is the purpose of a mutex?

1. A mutex shares data among multiple threads.
2. A mutex locks data.
3. A mutex allows multiple threads concurrent access to resources.
4. A mutex is used to coordinate mutually exclusive access to resources by multiple threads of execution.

## What is the purpose of the producer-consumer idiom?

1. to facilitate threads that produce data and threads that consume data using a single common and coordinated container
2. to produce data
3. to produce and consume data
4. to consume data

# Other subjects

## Which standard random number generator uses the hardware RNG engine (if available)?

1. std::knuth\_b
2. std::default\_random\_engine
3. std::mt19937
4. std::random\_device

## Which type and value of b1 after the following code snippet?

std::tuple<int, double, const char\*> x {42, 73.2, "hello, earthlings!"};

const auto [a, b, c] = x;

1. b is type std::tuple<int, double, const char\*> and has the value (42, 73.2, "hello, earthlings!")
2. b is type int and has the value 42
3. b is type double and has the value 73.2

## What is the purpose of the std::filesystem::path class?

1. It represents a path that must exist on your file system.
2. It represents a path on a file system independent of the conventions of a specific operating system.
3. It represents a path as if it were on a UNIX-based operating system.
4. It represents a path to a specific file or object.

## Which std::chrono clock is monotonic?

1. std::chrono::high\_resolution\_clock
2. std::chrono::utc\_clock
3. std::chrono::system\_clock
4. std::chrono::steady\_clock

## What is the result of this braced initialization?

int x {5.5};

1. It will create a float instead of an int.
2. It will round the value to an integer 6.
3. It will fail to compile because it would require a narrowing conversion, which is not allowed with braced initialization.
4. It will truncate the value to an integer 5.

# Other C++ 20 new features

## What is the primary purpose of the three-way comparison (spaceship) operator?

1. to simplify comparisons
2. to use with lambda predicates
3. The purpose of the spaceship operator is to provide a unified comparison operator for classes.
4. to make comparisons easier to read

## Assuming the value of inta is 47, what is the string returned by the following format() call?

auto x = format("inta is [{:0>10}]\n", inta);

1. inta is [47\*\*\*\*\*\*\*\*]
2. inta is [0000000047]
3. inta is [ 47 ]
4. inta is [\_\_\_\_47\_\_\_\_]

## What is the purpose of the std::span class?

1. It has no effect.
2. It makes a C-array faster.
3. It adds STL container capabilities to a primitive C-array.
4. It makes a C-array better.

## What is the output of the following code snippet?

int x {-3};

unsigned y {7};

if (x < y) print("true\n");

else print("false\n");

if (std::cmp\_less(x, y)) print("true\n");

else print("false\n");

1. false true
2. false false
3. true true
4. true false