C++11 introduced the noexcept keyword to identify a function as one that will not throw an exception.  This keyword informs the compiler that it can perform certain optimizations that would not be possible if uncaught exceptions could pass through the function.

If a function marked noexcept allows an uncaught exception to escape at runtime, the program terminates immediately.

|  |  |  |
| --- | --- | --- |
|  |  | Correct |
|  |  | Incorrect |

The abort() function terminates program execution by a SIGABRT signal.  This function, like terminate(), does not execute destructors for objects of automatic, thread, or static storage duration or call functions at addresses passed to atexit().

|  |  |  |
| --- | --- | --- |
|  |  | Correct |
|  |  | Incorrect |

int **count\_if**(InputIterator f, InputIterator l, Fn predicate)

count the occurences of t within range [f,l) that satisfy predicate

|  |  |  |
| --- | --- | --- |
|  |  | Correct |
|  |  | Incorrect |

For lambda Function the following capture list [=](...)   captures all non-local variables by reference

|  |  |  |
| --- | --- | --- |
|  |  | Correct |
|  |  | Incorrect |

Exceptions may NOT be thrown by the following expressions:

* a throw expression
* a dynamic\_cast expression
* a type\_id expression
* a new expression

|  |  |  |
| --- | --- | --- |
|  |  | Correct |
|  |  | Incorrect |

C++17 introduced new library facilities.  They include:

* std::string\_view (<string\_view>) - a read-only contiguous sequence of characters (see above)
* std::variant<T, ...>() (<variant>) - represents a type-safe union
* std::optional<T> (<optional>) - may or may not contain a value (beyond scope)
* std::any (<any>) - single values of any type (beyond scope)

|  |  |  |
| --- | --- | --- |
|  |  | Correct |
|  |  | Incorrect |

A shared smart pointer std::shared\_ptr can not copy construct or copy assign

|  |  |  |
| --- | --- | --- |
|  |  | Correct |
|  |  | Incorrect |

A reference-counted smart pointer std::shared\_ptr

* stores a raw pointer to its object or resource
* shares ownership of its object or resource with other smart pointers
* transfers ownership of the resource in move construction and move assignment
* copies itself in a copy construction
* releases itself from its object or resource and reattaches itself in a copy assign
* can be compared for equality
* calls its object's or resource's destructor only if it itself is being destroyed and no other smart pointer shares ownership of the object or resource

|  |  |  |
| --- | --- | --- |
|  |  | Correct |
|  |  | Incorrect |

A  thread object is joinable if it was default constructed

|  |  |  |
| --- | --- | --- |
|  |  | correct |
|  |  | incorrect |

C++11 Facilities

* std::ref() (<functional>) - returns an lvalue reference to its argument
* std::move() (<utility>) - returns an rvalue reference to its argument

|  |  |  |
| --- | --- | --- |
|  |  | Correct |
|  |  | Incorrect |

The istream class of the input-output hierarchy includes a member function that reads a stream in binary mode.  The prototype for this function is

std::istream& read(char\* data, std::streamsize nb);

|  |  |  |
| --- | --- | --- |
|  |  | Correct |
|  |  | Incorrect |

For lambda Function the following capture list [&](...)   captures all non-local variables by value

|  |  |  |
| --- | --- | --- |
|  |  | Correct |
|  |  | Incorrect |

A unique smart pointer std::unique\_ptr

* owns its object or resource
* stores a raw pointer to its object or resource
* transfers ownership of the object or resource in move construction and move assignment
* calls its object's or resource's destructor when it itself is destroyed
* cannot copy construct or copy assign

|  |  |  |
| --- | --- | --- |
|  |  | Correct |
|  |  | Incorrect |

**Reporting an Exception**

The throw expression reports an exception and takes the form

**throw** expression;

expression is an expression of a previously defined type.  A function that can includes this statement cannot be identified as a noexcept function.

**Handling the Exception**

The code for handling an exception consists of two parts: the set of statements that initiated the process and the set of statements that respond to an exception.  The keywords try and catch identify these complementary parts:

* a try block contains all of the code that initiates whatever might throw the exception
* one or more catch blocks contain the code that handles any exception that was thrown as a result of executing any statement within the try block

|  |  |  |
| --- | --- | --- |
|  |  | Correct |
|  |  | Incorrect |

try {

p = new char[strlen(s) + 1];

strcpy(p, s);

} catch (std::exception& e) { // called by std::bad\_alloc also

cout << "Standard Exception " << endl;

} catch (std::bad\_alloc) {

cout << "Insufficient memory " << endl;

}  
  
Is the second catch reachable ?

|  |  |  |
| --- | --- | --- |
|  |  | Yes |
|  |  | NO |

The try catch combination takes the form:

**try** {

// code that might generate exceptions - **throw exceptions**

} **catch (*Type identifier*)** {

// handler code for a specific type of exception

} **catch (*Type identifier*)** {

// handler code for another specific type of exception

} **catch (...)** {

// handler code for all other types of exception

}

|  |  |  |
| --- | --- | --- |
|  |  | Correct |
|  |  | Incorrect |

A  thread object is joinable if it has been moved from (either constructing another thread object, or assigning to it).

|  |  |  |
| --- | --- | --- |
|  |  | correct |
|  |  | incorrect |

Inheritance is a relationship between user-defined types.  These types may be:

* concrete - their representation is part of their definition and is known
* abstract - their representation is not part of their definition and is unknown

|  |  |  |
| --- | --- | --- |
|  |  | Correct |
|  |  | Incorrect |

Object-oriented languages require a specialized mechanism for identifying and handling errors.  Since the designer of an object is not necessarily aware of how their object will be used, the object itself may not be able to handle the error(s) that it identifies.  The preferred strategy in object-oriented programming is to identify the error as an ***exception*** to the normal execution process and to handle that error in some other as yet unknown object.

|  |  |  |
| --- | --- | --- |
|  |  | Correct |
|  |  | Incorrect |

Week 10 Code 1, had we neglected to join() the spwaned threads to the main threads

|  |  |  |
| --- | --- | --- |
|  |  | The result is defined and predictable |
|  |  | The result would be undefined |
|  |  | All of the above |
|  |  | None of the above |

The terminate() function terminates program execution as result of an error related to exception handling.  Cases include:

* the mechanism cannot find a handler for a thrown exception
* the handler encounters the body of a function with a noexcept specification
* the destruction of an object exits via an exception
* throw with no operand attempts to throw an exception when none exits

|  |  |  |
| --- | --- | --- |
|  |  | Correct |
|  |  | Incorrect |

Since a generic pointer cotains the type information of the object pointed to, the compiler can dereference the pointer and identify the region of memory starting at the address pointed to.

|  |  |  |
| --- | --- | --- |
|  |  | Correct |
|  |  | Incorrect |