noexcept specifier

OVERVIEW

A **noexcept specification** on a function is a method for a programmer to inform the compiler whether a function should throw exceptions.

The compiler can use this information to enable certain optimizations on non-throwing functions as well as enable the **noexcept operator**, which can check at compile time if a particular expression is declared to throw any exceptions.

noexcept will not call std::unexpected and may or may not unwind the stack.

DEFAULT FUNCTIONS

Compiler generated constructors and methods are noexcept.

Examples include:

- Default constructor
- Default destructor
- Default assignment operator

USER DEFINED FUNCTIONS

`noexcept` can be applied to custom methods.

- A promise that the method will not raise an exception
- If exception raised, std::terminate is called. This will cause an immediate termination.
- For this reason, all exceptions must be caught in the method.

```
Amphibious(Vehicle* _pVehicle,
          Boat* _pBoat) noexcept(true) {
    try {
         SetPointers(_pVehicle,
                     _pBoat);
    }
    catch (char* message) {
         cout << message << endl;
    }
}
```

NOEXCEPT(BOOLEAN)

- `noexcept` can deny or accept exception handling within a method.
- You provide a single Boolean expression as a parameter. Default is true.

```
// C'tor may throw
Amphibious(Vehicle* _pVehicle, Boat*
_pBoat) noexcept(false)
{
    SetPointers(_pVehicle,
                pBoat);
}
```

EXCEPTION OBJECTS

Exception objects are created with these methods:

- current_exception
- make_exception_ptr
- nested_exception::nested_ptr

The implementation of the exception object is environment specific.

The exception_ptr is a shared pointer for saving exception objects.

Rethrow using rethrow_exception(eptr)

```
// general purpose handler
void handle(std::exception_ptr eptr) {
  try
  {
    if (eptr) rethrow_exception(eptr);
  }
  catch(const std::exception& e)
  {
    cout << e.what() << "'\n";
  }
}
```