Name: Jude Onyia

Student ID: V00947095

Course: ECE 596C

Due Date: June 19, 2020

Assignment 3: Non – Programming Exercise

6.1)

Due to exceptions that could be thrown for several reasons (i.e. division by zero, lack of memory, etc.), performing any necessary clean-up of an object must be put in a finalizer of that object. When exception is thrown, the program is violently removed from that code block, it does not execute any code further down the line from that exception. The only code guaranteed to execute is the code within the finalizer of that object. Therefore, any clean-up necessary must be done in the finalizer.

6.2 a)

During the stack unwinding process, objects are destroyed in the following sequence: die3, die, countdown, hello, i, bjarne, herb, dv, u, z.

6.2 b)

During the stack unwinding process, the only object destroyed is s.

6.3 a)

If there is insufficient memory to allocate for second buffer, an exception will be thrown, and the program will be violently ripped from the function without freeing the first buffer. To guarantee that both buffers are freed if an exception occurs, one can use a class for the buffers, where a char pointer is a data member. Overloads on operator= and operator[] can be used to assign the char pointer to a space in memory and access it, respectively. The finalizer of the class can be used to safely deallocate the memory if an exception is thrown. Both buf1 and buf2 will then be objects of this class. In the incident where buf1 is created and memory runs out while attempting to create buf2, as the exception is thrown, it is guaranteed that the finalizer of buf1 will be called to free up that space before leaving the function.

6.3 b)

If the formatting flags are changed and outputting the integer to the ostream causes an exception to be thrown, the program will be violently ripped from the function without running the code that restores the formatting flag. To prevent this, the line that changes the formatting flag and outputs the integer can be surrounded by a try clause. The body of the associated catch clause can restore the old formatting flags if an exception occurs.

6.3 c)

If the queue has a number of elements in it, and there is insufficient amount of memory left to push another element, q.push_back(value) might throws an exception. If it does and the user of the class does not catch it, the program could terminate without freeing the memory allocated for the elements already in the queue. To prevent this, a finalizer for the class can be explicitly defined to free

the queue when called. This will ensure that when an exception is thrown, the memory allocated for elements in the queue are freed before the object is destroyed and the program terminated.

6.5)

The function ‘analyze’ cannot throw an exception because the function has the noexcept specifier. This specifier indicates to the compiler that code for exceptions is not needed for this function, therefore, the compile will not generate code for exceptions for this function. This means the function is incapable of throwing an exception. If an attempt is made to throw an exception in this noexcept function, this will result in a fatal error.

The function ‘doWork’ does not have a noexcept specifier, therefore, it may or may not throw an exception. The compiler will generate code for exceptions for this function. Therefore, this function is capable of throwing an exception.