Dynamic memory management and exception handling

1. (3p) Dynamic memory allocation.
   Create a C-style array of integers with 1000000 elements on the stack. Run/debug the program and you should get a stack overflow exception (if not, increase the number of elements).
   Comment out that part of the code and instead create a pointer to an integer, and dynamically allocate an integer array of 1000000 elements to it (use the “new” expression). You should now be able to run the program without an exception. Remember to free the dynamically allocated memory at the end of the program.
2. (2p) Dynamic memory management with smart pointers.
   The file **assignment-smart-pointers.cpp** contains the function “demonstrateRawPointers”, which allocates integers dynamically with raw pointers. If they are not explicitly deleted, they will remain in memory until the program finishes running. You don’t have to modify the “demonstrateRawPointers” function, which is there only for demonstration.
   Implement the definition of the function “demonstrateSmartPointers” where you allocate integers dynamically with smart pointers (in which case you don’t have to free the memory yourself).
   You may use any smart pointer type you want.
   Your only task here is to create a new dynamic pointer pointing to an integer value, which you can implement in a single line inside the demonstrateSmartPointers() function because you don’t have to delete it yourself. If you debug the code, you know the program works if it doesn’t continuously reserve more and more memory as it runs.
3. (5p) Throwing and catching exceptions.
   Write a class “UnitVector”. The constructor should take two floating-point values as its arguments and assign those to its private variables x and y, which represent the elements of the vector. In the constructor, if the length of the object is not 1, throw an std::invalid_argument exception that has an error message in string format.
   Catch the exception in the main function when constructing a UnitVector of invalid size and print the string error message from the exception.
   Hint: direct comparison can be difficult with floating-point types, so instead of comparing a double or a float representing the length of the vector to 1, you can instead subtract 1 from the length of the vector and check if the result is smaller than some very small number (so that the vector length is close to 1).