1. Create three **const int** values, then add them together to produce a value that determines the size of an array in an array definition. Try to compile the same code in C and see what happens (you can generally force your C++ compiler to run as a C compiler by using a command-line flag).

2. Prove to yourself that the C and C++ compilers really do treat constants differently. Create a global **const** and use it in a global constant expression; then compile it under both C and C++.

3. Create example **const** definitions for all the built-in types and their variants. Use these in expressions with other **consts** to make new **const** definitions. Make sure they compile successfully.

4. Create a **const** definition in a header file, include that header file in two **.cpp** files, then compile those files and link them. You should not get any errors. Now try the same experiment with C.

5. Create a **const** whose value is determined at runtime by reading the time when the program starts (you’ll have to use the **<ctime>** standard header). Later in the program, try to read a second value of the time into your **const** and see what happens.

6. Create a **const** array of **char**, then try to change one of the **char**s.

7. Create an **extern const** declaration in one file, and put a **main( )** in that file that prints the value of the **extern const**. Provide an **extern const** definition in a second file, then compile and link the two files together.

8. Write two pointers to **const long** using both forms of the declaration. Point one of them to an array of **long**. Demonstrate that you can increment or decrement the pointer, but you can’t change what it points to.

9. Write a **const** pointer to a **double**, and point it at an array of **double**. Show that you can change what the pointer points to, but you can’t increment or decrement the pointer.

10. Write a **const** pointer to a **const** object. Show that you can only read the value that the pointer points to, but you can’t change the pointer or what it points to.

11. Remove the comment on the error-generating line of code in **PointerAssignment.cpp** to see the error that your compiler generates.

12. Create a character array literal with a pointer that points to the beginning of the array. Now use the pointer to modify elements in the array. Does your compiler report this as an error? Should it? If it doesn’t, why do you think that is?

13. Create a function that takes an argument by value as a **const**; then try to change that argument in the function body.

14. Create a function that takes a **float** by value. Inside the function, bind a **const float&** to the argument, and only use the reference from then on to ensure that the argument is not changed.