Introducing the `-Wall`, `-Weffcc++`, and `-Wextra`.

- `-Wall`: This enables all construction warnings, which are questionable for some users. These warnings are easy to avoid or modify, even in conjunction with macros. It also enables some language-specific warnings described in C ++ Dialect Options and Objective-C/C ++ Dialect Options.
- `-Wextra`: As its name implies, it examines certain extra warning flags that are not checked by `-Wall`. Warning messages for any of the following cases will be printed:
    - A pointer is compared against integer zero with the `<`, `<=`, `>`, or `>=` operands.
    - A non-enumerator and an enumerator show up in a conditional expression.
    - Ambiguous virtual bases.
    - Subscripting a `register` type array.
    - Using the address of a `register` type variable.
    - A derived class' copy constructor does not initialize its base class.

Note that points from 2 to 6 are C++ only.

- `-Weffc++`: It checks the violations of some guidelines suggested in *Effective and More Effective C++*, written by Scott Meyers. These guidelines include the following:
    - Define a copy constructor and an assignment operator for classes with dynamically allocated memory.
    - Prefer initialization over assignment in constructors.
    - Make destructors virtual in base classes.
    - Have the `=` operator return a reference to `*this`.
    - Don't try to return a reference when you must return an object.
    - Distinguish between prefix and postfix forms of increment and decrement operators.
    - Never overload `&&`, `||`, or `,`.

To explore these three options, let's look at the following example at https://github.com/PacktPublishing/Expert-C-2nd-edition/blob/main/Chapter14/ch14_static_analysis.cpp.
First, let's build this without any options:

```
g++ -o ch14_static.out ch14_static_analysis.cpp
```

This can be built successfully, but if we run it, as expected, it will crash with a **segmentation fault (core dumped)** message.
Next, let's add the `-Wall`, `-Weffc++`, and `-Wextra` options and rebuild it:

```
g++ -Wall -o ch14_static.out ch14_static_analysis.cpp
g++ -Weffc++ -o ch14_static.out ch14_static_analysis.cpp
g++ -Wextra -o ch14_static.out ch14_static_analysis.cpp
```

Both `-Wall` and `-Weffc++` give us the following message:

```
ch14_static_analysis.cpp: In function 'int& getVal()':
ch14_static_analysis.cpp:9:6: warning: reference to local variable
'x'
returned [-Wreturn-local-addr]
int x = 5;
^
```

Here, it's complaining that, in the `int & getVal()` function (*line 9* of the `ch14_static_analysis.cpp` file), a reference to a local variable is returned. This will not work because once the program goes out of the function, `x` is garbage (The lifetime of `x` is only limited in the scope of the function). It does not make any sense to reference a dead variable.
`-Wextra` gives us the following message:

```
ch14_static_analysis.cpp: In function 'int& getVal()':
ch14_static_analysis.cpp:9:6: warning: reference to local variable
```

```
'x'
returned [-Wreturn-local-addr]
int x = 5;
^
ch14_static_analysis.cpp: In function 'int main()':
ch14_static_analysis.cpp:16:10: warning: ordered comparison of
pointer
with integer zero [-Wextra]
if( x> 0 ){
^
```

The preceding output shows that `-Wextra` not only gives us the warning from `-Wall` but also checks the six things we mentioned earlier. In this example, it warns us that there is a comparison between a pointer and integer zero in file `ch14_static_analysis.cpp`, *line 16* of the code. Now that we know about how to use the static analysis options during compile time, we'll take a look at dynamic analysis by executing a program.