The Preprocessor

All C/C++ compilers implement a stage of compilation known as the preprocessor. Part of the responsibility of the C++ preprocessor is to perform an intelligent search and replace on identifiers that have been declared using the #define or #typedef directives. Although most advocators of C++ discourage this usage of the preprocessor, which was inherited from C, it is still widely used by most C++ programmers. Most of the processor definitions in C++ are stored in header files, which complement the actual source code files.

The problem with the preprocessor approach is that it provides an easy way for programmers to inadvertently add unnecessary complexity to a program. Many programmers using the #define and #typedef directives end up inventing their own sublanguage within the confines of a particular project. This results in other programmers having to go through the header files and sort out all of the #define and #typedef information to understand a program, which makes code maintenance and reuse almost impossible. An additional problem with the preprocessor approach is that it is very weak when it comes to type checking and validation.

Java does not have a preprocessor. It provides similar functionality (#define, #typedef, and so on) to that provided by the C++ preprocessor, but with far more control. Constant data members are used in place of the #define directive, and class definitions are used in lieu of the #typedef directive. The end result is that Java source code is much more consistent and easier to read than C++ source code. Additionally, Java programs don't use header files; the Java compiler builds class definitions directly from the source code files, which contain both class definitions and method implementations.

Pointers

Most developers agree that the misuse of pointers causes the majority of bugs in C/C++ programming. Put simply, when you have pointers, you have the ability to trash memory. C++ programmers regularly use complex pointer arithmetic to create and maintain dynamic data structures. In return, C++ programmers spend a lot of time hunting down complex bugs caused by their complex pointer arithmetic.

The Java language does not support pointers. Java provides similar functionality by making heavy use of references. Java passes all arrays and objects by reference. This approach prevents common errors due to pointer mismanagement. It also makes programming easier in a lot of ways because the correct usage of pointers is easily misunderstood by all but the most seasoned programmers.

You might be thinking that the lack of pointers in Java will keep you from being able to implement many data structures such as dynamic arrays. The reality is that any pointer task can be carried out just as easily, and more reliably, with objects and arrays of objects. You then benefit from the security provided by the Java runtime system; it performs boundary checking on all array indexing operations.

111111111111111111111111
000000000000000000000000
111111111111111111111111
aaaaaaaaaaaaaaaaaaaaaaaa
cccccccccccccccccccccccc
eeeeeeeeeeeeeeeeeeeeeeee
gggggggggggggggggggggggg
iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii
kkkkkkkkkkkkkkkkkkkkkkk
mmmmmmmmmmmmmmmmmmm

000000000000000000000000000000000
111111111111111111111111111111111
000000000000000000000000000000000
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
ddddddddddddddddddddddddddddddd
fffffffffffffffffffffffffffffffffffffffffffffffffff
hhhhhhhhhhhhhhhhhhhhhhhhhhhhhh
jjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjj
lllllllllllllllllllllllllllllllllllllllllllllllllll
nnnnnnnnnnnnnnnnnnnnnnnnnnnnnn