```
PUT SKIP EDIT ('AREA UNDER THE CURVE',
               'BY THE TRAPEZOIDAL RULE')
               (X(9), A, SKIP, X(7), A);
```

and the bizarre

```
PUT SKIP EDIT(' ') (A(1));
```

changed into a simple PUT SKIP. And there is no reason to specify character-string lengths in the A format items; computers count much better than people do.

The purpose of the assignment

```
M = 1 / K;
```

is unclear. Does it defend against some mysterious conversion? Is it to convey geometrical insight? Or does the programmer worry that computers divide more slowly than they multiply? It is a rare program that can be speeded up significantly by changing divisions into multiplications, and this is not one of them — M appears only twice. Efficiency cannot be of grave importance anyway, not when the code contains the statement

```
AREA1 = .5 * M * (2 * L);
```

which has two superfluous multiplications (but no divisions!). M can be eliminated. Similarly, N, LMTS, L and AREA1 vanish as the obvious substitutions are made.

We can now remove all those declarations with the strange precisions needed for intermediate results. The remaining declarations consist of just two different types. A close look reveals that K is not declared, even though all other arithmetic variables are. By default K will be FIXED BINARY so a number of type conversions will occur, to no advantage. K should be included in the declarations.

With all the extraneous assignments removed, it is easier to see the underlying structure. It is also easy to see that the indentations reflect little of what is going on. But what is the purpose of the variable I? It is laboriously kept equal to J so that OUT can be called at the end of the last iteration. Clearly I is not needed, for J could be used for the test. But the test is not needed; OUT could be called just after the inner DO loop has terminated. But OUT need not be called at all, for its code could just as well appear in the one place it is invoked. The structure simplifies remarkably.

Now we can see that the summing variable AREA is supposed to be initialized at the beginning of each loop on K. This is much better practice than clearing it before entering the loop and again at the end of each iteration — in a remote procedure at that. Our major criticism of the procedure OUT is not its existence, since it was there for pedagogical reasons, but that it changes AREA and uses LMTS when it does not have to. Destroying modularity in this fashion, referring to seemingly local variables in unexpected places, is an invitation to future bugs. When code is rearranged, or the use of such non-local variables is changed, errors are almost certain to be introduced.

Putting all our improvements together gives: