Programming Staff Note 16

Computation Center
Dec. 16, 1963

FROM: F. J. Corbató

SUBJECT: Programming Style and Exposition

First, a few lofty remarks. The perfectly written program would be one where the procedure were not only lucidly stated but accomplished with the most effective use of computer time and space resources. These somewhat contradictory objectives require a compromise among the factors of brevity, completeness, and suggestiveness as well as the ever-present concern for space-time efficiency.

In general, to present programming method, specification and usage ideas effectively is of course a problem of exposition. A description of a program which may or may not be embedded in a formal program should resemble a section of a textbook, in that by organized material, the reader is guided and shown the proper sequence for absorbing the ideas present. This is often done by an introductory summary establishing the scope and context as well as the essential points to be made; indications of the relative importance of the different sections to be read are of further assistance. A not unreasonable guide is to assume that the liklihood of a description being read to completion is inversely proportional to the length, so that it is a contest for you, the writer, to gain for a given degree of clarity, maximum reader impact and interest as briefly as possible. For in the end the all-too-typical reader will only read that which he has been persuaded is worth his spending his time on. Having made this point it nevertheless remains to be emphasized that lack of clarity is more of a fault then lack of brevity.

Reapproaching the practical world of writing CTSS system programs, the following guidelines, many of which are typographical in nature, are offered:

1. All programming ideas should be organized and broken down so that they can be expressed in modules of BSS subprograms. This modularity gives assurance that any single piece can be understood or rewritten reasonable quickly and that there is complete independence between subprograms except for the explicit "calls" and the implicit "common" pool. Further benefits result from the rapidity and machine-time efficiency with which a module can be redone and replaced. At present the lack of generality and the slight inefficiency of the BSS form do not seem to be a problem.

2. All modules should be as machine invariant as possible, and in the MAD language except when excessive clumsiness or inefficiency result. It is important that our programming ideas be in a state to be rapidly transferable to the different computers of the future.

3. Unless good reasons exist, no MAD subprogram should exceed about 200 statements. FAP subprograms should be kept as small as possible unless they are a high-efficiency version of a MAD module; in any case, single FAP subprograms should not exceed a few hundred cards.

4. Programs should have vertical punctuation (by means of blank Remark cards in MAD, * cards in FAP) such that significant sections are set off as "paragraphs". This particularly applies to before and after the scope of THRØUGH statements.

5. Every MAD or FAP program should contain interspersed comments. A MAD program which is between 20 to 35 percent remark cards (including blank "punctuation" remark cards) is reasonable. (All remark cards in MAD should be blank in column 12 at least.) In FAP, 25 to 50 percent of the cards with comments is reasonable. FAP programs should always have preceding remark cards giving a sample calling sequence.

6. Extended remarks of greater length are tolerable (and in general desirable) provided they are all in one block at either the beginning or end of the program. In either case, there should be an initial remark stating the scope (i.e. line numbers) of the extended remarks so that typewriter users can skip over them if desired. (A special version of PRINTF which filters out remarks might be useful during debugging.)

7. To show loop and conditional nesting, in MAD, all loops and compound conditional statements should be clarified by indentation of 2 spaces (or 1 space in heavily-nested cases). This indentation should apply to all statements in the respective scopes except THRØUGH, WHENEVER, ØR WHENEVER, ØTHERWISE and END ØF CØNDITIONAL. (Use circled digits for spaces when key punching is being done by others.)

8. To set off key phrases punctuate with a space between the comma and the FØR in a THRØUGH statement; punctuate with a space after the comma in a simple WHENEVER statement.

9. All programs should have their declarations collected at either the beginning or the end.

10. The above points are not rigid rules and exceptions (especially in the direction of greater explicitness) should be made whenever important or tricky algorithms are being done. Maximum speed of communication of ideas with minimum reader time and effort is the overall criterion by which to judge alternatives.