SESSION 12: Invited Papers

Moderator: E. T. Irons

Institute for Defense Analyses, Princeton, N. J.

WPM 12.1: Toward Better Programming Languages

V. H. Yngve

Research Laboratory of Electronics—MIT

Cambridge, Mass.

THE STYLE of a programming language often has a great influence on the approach that a programmer takes to a particular problem. This is the *Whorf* hypothesis applied to programming languages. Most programming languages are general purpose in that anything that is programmable can be programmed in them. But to the extent that they differ, some programs are more easily expressed in one language than in another. We do not consider here the important considerations of the efficient use of computer memory and running time, since these are more related to the particular machine and compiler configuration used than to the structure of the programming language itself. We are more interested in factors of programming convenience, brevity, fluency, legibility, ease of learning, ease of checkout, etc., that are of more direct concern to the programmer.

One of the most important characteristics of a language, from the programmer's point of view, is the method of handling references to data. The machine instruction code itself usually allows reference to data by numerical address alone, although newer computers provide index registers and indirect addressing. In any case, the reference is usually to only a single register of the size provided by the hardware. Assembly programs provide conveniences like symbolic addressing and relative addressing, but reference is still to the individual register. One of the most important conveniences of *FORTRAN* and most other compilers is a method of referring to whole arrays without having to mention explicitly the address of each member of the array. Another important convenience of most compilers is the facility of indicating a computed address by writing an algebraic expression. Local addressing, or the ability to use the same name to refer to different data, is a convenience afforded by some systems: assembly programs provide heading features or separate assembly and relocation; *COBOL* provides a very convenient facility for this purpose by its system of qualification.

One of the biggest advantages of list processing languages is the opportunity they afford to refer with one name to a whole list, possibly also including many sublists. The elements comprising the lists usually are single registers; some may contain data and some may contain information specifying the organization of or interrelations between the data registers.

The reason why methods of data reference are so important for the style of a programming language is that a programmer tends to think in terms of the items that he can name in the language and manipulate.

It is in the area of data reference that the *COMIT* language is much different from other programming languages. *COMIT* was originally designed for convenient handling of alphabetic data—English and foreign words and sentences together with any temporary data needed for the operations of mechanical translation programs. *COMIT* has turned out, however, to be convenient for a number of other types of programs where the data are essentially non-numerical in character.

In *COMIT*, the items of reference are not computer registers, but problem-oriented items of any convenient size called constituents. A constituent may consist of any number of characters; thus it could be used for a letter, an English word of any lengh, or any other item of non-numerical data. The programmer may refer to these constituents directly, and so he does not have to take into consideration the particular computer word size. A constituent may also carry, in addition to the data part, any number of classificatory subscripts, and a numerical subscript useful for counting and simple arithmetic computations.

COMIT is rich in its methods of data reference. It includes several types of direct referencing, as well as direct address referencing and indirect address referencing. By direct referencing we mean the mention in the program of some or all of the data characters or classificatory subscripts. The desired data item or constituent is found by a linear search through a defined portion of the data. A fast alphabetic search is also available for table or dictionary look-up.

A convenient and much-used method of reference that is perhaps unique to *COMIT* is through context. It is possible, for example, to refer to the data on the right or on the left of a given item or to the data between given items. This is possible because data in *COMIT* can be conceived of as being arrayed along a one-dimensional problem space. The programmer thinks in terms of this problem-space and not in terms of computer registers. He is free to insert and delete items from the problem space without having to keep track of storage.

An example of the brevity and convenience that these features provide is the following program for reading an unknown amount of data, sorting it numerically, and printing it out in numerical order.

```
SET-UP $ = A + MARK/.32767 // *RSK1 *
INSERT $1 + $ + $1/.G.*1 = A + 2 + 1 + 3 // *RSK1
INSERT
PRINT A + $ + MARK = 2 // *WSM1 *
```

The first line initializes and reads in the first item. The second line is a loop that inserts an item in its correct numerical position and then reads in another item. The last line cleans up and prints the results.

The potential range of application of the digital computer is very wide. There is a wide diversity of problem areas involving a wide variety of items and structures to be referenced and manipulated. But computers are still relatively small so that it is at present unfeasible to supply all of the desirable features in one programming language that would make it simultaneously convenient for all problems. It is probably impossible to design the ideal general-purpose programming language because each useful programming language must be a judicious compromise between the various contradictory requirements of a problem area. The wider the problem area, the more difficult the compromise. A halting of the trend toward *Babel* through standardization would be disastrous. The continued development of new and especially of novel programming languages is required for continued progress—and necessary for a fuller realization of the potentialities of the digital computer.