A bit of background:

At the time of this article (1952), the computational community was in the midst of a debate on ***how*** software, in general, would be organized.

All programs at the time were simply sequences of assembly instructions written and tailored to accomplish a particular task.

The idea of writing reusable functions was new and under debate: some (like von Neumann) didn't like the idea. Hopper calls them "subroutines" or "type A routines"

The idea of having the computer itself generate reusable functions from higher-level code was only a very early idea -- and one Hopper helped form. She calls these "type B routines." "C" and "D" represent yet higher-level routines, left unspecified.

Up to this point in the article, Hopper has described arguments and technical details in favor of her "type A routines" (reusable functions) and "type B routines" (computer-compiled reusable functions). She concludes by jumping into the debates of the day:

It is here that the question can best be answered concerning a liking for or an aversion to subroutines. Since the use of subroutines in this fashion increases the abilities of the computer, the question becomes meaningless and transforms into a question of how to produce better subroutines faster. However, balancing the advantages and disadvantages of using subroutines, among the advantages are:

1. relegation of mechanical jobs such as memory allocation, address modification, and transcription to the UNIVAC;
2. removal of error sources such as programming errors and transcription errors;
3. conservation of programming time;
4. ability to operate on operation;
5. duplication of effort is avoided, since each program in turn may become a subroutine.

Only two disadvantages are immediately evident. Because of a standardization, a small amount of time is lost in performing duplicate data transfers which could be eliminated in a tailor-made routine. In base load problems, this could become serious. Even in this case, however, it is worthwhile to have UNIVAC produce the original program and then eliminate such duplication before rerunning the problem. The second disadvantage should not long remain serious. It is the fact that, if a desired subroutine does not exist, it must be programmed and added to the library. This will be most likely to occur in the case of input and output editing routines until a large variety is accumulated. This situation also emphasizes the need for the greatest generality in the construction of subroutines.

Several directions of future developments in this field can be pointed out. It is to be hoped that reports will be presented on some of them next September.

More type A compiling routines will be devised; those handling commercial rather than mathematical programs; some special purpose compiling routines such as a routine which will compute approximate magnitudes as it proceeds and select sub-routines accordingly. Compiling routines must be informed of the average time required for each sub-routine so that they can supply estimates of running time with each program. Compiling routines can be devised which will correct the computational procedure submitted to produce the most efficient program. For example, if both sin θ and cos θ are called for in a routine, they will be computed more rapidly simultaneously. This will involve sweeping the computer information once to examine its structure.

Type B routines at present include linear operators. More type B routines must be designed. It can scarcely be denied that type C and D routines will be found to exist adding higher levels of operation. Work is already in progress to produce the formulas developed by type B routines in algebraic form in addition to producing their computational programs.

Thus by considering the profesional programmer (not the mathematician), as an integral part of the computer, it is evident that the memory of the programmer and all information and data to which he can refer is available to the computer subject only to translation into suitable language. And it is further evident that the computer is fully capable of remembering and acting upon any instructions once presented to it by the programmer.

With some specialized knowledge of more advanced topics, UNIVAC at present has a well grounded mathematical education fully equivalent to that of a college sophomore, and it does not forget and does not make mistakes. It is hoped that its undergraduate course will be completed shortly and it will be accepted as a candidate for a graduate degree.