PROGRAMMING LANGUAGES AND

JOVIAL

C. J. SHAW

PROGRAMMING LANGUAGES AND JOVIAL/ From the very beginning, the effective utilization of the automatic digital computer has been hampered by the problems of man-machine communication. To the uninitiated, communicating with computers is quite as esoteric as communicating with spirits, and many authorities (notably Dr. Norbert Wiener) believe fundamental similarities exist between the two occupations. After all, what is an algorithm but a practical incantation?

Communication problems are always twofold: problems of information transmission; and problems of information coding. Unlike the spirit medium, the computer engineer has, according to the ads, finally solved the information transmission problem, and somewhere in the catalogue is an input-output device suitable for the most exacting user. Also unlike the medium, however, the computer programmer has yet to find as convenient a solution to the information coding problem as English, the language of the spirit world.

Basically, the automatic computer needs two kinds of information in order to function. It needs data for processing, and instructions to control the processing — and it needs these two kinds of information in its own machine language and in a physical representation accessible to its own circuitry. Since machine languages are the exotic creations of mystic engineers, full of 'Abra Clear and Add-abra,' it is not surprising that computer programmers have developed less intimidating and more abstract and symbolic programming languages for communicating with computers. Programs and data coded in such symbolic languages require translation into machine language before they can be utilized by the computer. This translation is done automatically, by the computer itself, under the direction of translation programs referred to, for lower level languages, as 'assemblers' and, for higher level languages, as 'compilers.' Just where the line between lower and higher level programming languages is drawn depends mainly on personal preference — and on the particular language being extolled.

Symbolic programming languages fall into three broad categories:

1. *Machine-oriented languages,* by which programs are described in terms of the individual machine instructions composing them. Machine language instructions are designed for ease of mechanization, and as a result, they produce such tiny portions of the total computation that they seldom correspond to operational units that are conceptually very meaningful to human beings. Consequently, machine-oriented programming languages are difficult both to learn and to use.

2. *Procedure-oriented languages,* by which programs are described in terms of the algorithms or computing rules they employ. Being liberated from the close correspondence with machine language that characterizes the machine-oriented language, such languages can utilize familiar notations in describing procedures, notations designed for the convenience of the human programmer, notations that do not require a deep knowledge of the computer being used.
3. *Problem-oriented languages,* by which programs are described in terms of available inputs and required outputs. Such languages are necessarily limited in scope to those classes of problems for which generalized computational procedures can be predetermined that bridge the gap between the inputs that can be specified and the results that can be requested. A wider applicability for the problem-oriented programming language concept must await the successful completion of research in the area of suitable formal languages for describing the behavior of data processing systems.

It is becoming quite apparent, as both problems and machines grow in complexity, that if humans are to cope with the computer's voracious appetite for information, then languages built for people, and not machines, must come into use. And at the present stage of development, it is the procedure-oriented programming language that offers the greatest promise.

Procedure-oriented languages vary in scope from the universal applicability afforded by machine-oriented languages to the limited applicability of problem-oriented languages, and a particular procedure-oriented programming language usually represents a compromise in favor of some specific area of application. Procedure-oriented languages are a fairly late development. Aside from some earlier experiments, it was not until 1956 that the first versions of such languages as FORTRAN and MATH-MATIC for numerical problems and FLOW-MATIC for business problems began to appear. The most significant recent developments in the field are the efforts toward language standardization. In 1958, committees from both the ACM (Association for Computing Machinery) and GAMM (German-Swiss Applied Mathematics Society) met in Zurich to propose a standard algorithmic language (ALGOL) for scientific numerical work. As a result of this, further meetings were held during 1959 between interested groups from other countries and in early 1960 an international conference was held in Paris which issued a revised version of the language. Concurrently, under the auspices of the Department of Defense, the Conference on Data Systems Languages prepared specifications for COBOL, a *CO*mmon

Business Oriented Language for business data processing problems.

JOVIAL / In mid-1958, shortly before the publication of the ALGOL-58 specifications, the System Development Corporation initiated a research project to investigate the problems of automatic coding. This project resulted in the development of CLIP, a Compiler Language for Information Processing.

As soon as preliminary work on the CLIP project had assured the practicality of the scheme, it was decided to develop a similar, procedure-oriented programming for the SACCS (Strategic Air Command Control System) computer programs. This language, named JOVIAL, was like CLIP patterned after ALGOL, and was adapted to the programming of large-scale, computer-based, command control systems by the incorporation of certain features found desirable from experience gained in the development of the SAGE air defense system of computer programs. Subsequently, however, because of the success of the first primitive working versions of the language and due to a growing realization of its wide potential scope, a decision to standardize on JOVIAL as a corporate procedure-oriented programming language was made, and further development has proceded on this basis.

Prime motivation for the development of JOVIAL is the wish to have a common, powerful, easily understandable, and mechanically translatable programming language suitable for a very wide range of applications. Since the Corporation's activities center on the design, development, and implementation of large-scale information processing systems utilizing a variety of computers, such a language must be machine-independent, with a power of expression in logical operations and symbol manipulation, as well as in numerical computation. One of the further requisites of any programming language, intended for large-scale data processing systems, is that it include the capability of designating and manipulating system data, as described in a Communication Pool (COMPOOL) of system configuration information. Though highly desirable for any data processing system, a COMPOOL is a vital necessity for large-scale systems where otherwise problems of coordination between programmers are apt to be unsolvable.

None of the existing problem-oriented languages were entirely adequate for the purpose of programming command control systems. (COBOL is intended primarily only for business applications; ALGOL, only for scientific-numeric applications.)

JOVIAL was developed to incorporate the necessary features mentioned above. It thus belongs to the ALGOL family of procedure-oriented programming languages. Because of its strong family resemblance, the extensive literature of ALGOL procedures that is arising, with the most simple of transliteration, also can be applied to JOVIAL.

A JOVIAL program describes a particular solution to a data processing problem, intended to be incorporated by translation into a machine language program. As in ALGOL, the two main elements of this description are:

1. A set of declarations, describing the data to be processed.
2. A set of statements, describing the algorithms or processing rules.

These two sets of descriptions are, to a great extent, mutually independent, so that changes in one do not necessarily entail changes in the other.

JOVIAL'S ADVANTAGES *JOVIAL is a general purpose programming language.* Because of the wide range of problems encountered, any programming language intended for large-scale command control data processing systems must be truly universal in scope. As such a language, JOVIAL is well suited for a variety of different applications since it provides consistent notations for the designation and manipulation of: numeric values (in both fixed-point and floating-point representation); dual or complex numeric values; alphanumeric values; status values; Boolian values; table of values; and multi-dimensional array of values.

Scientific and engineering applications involving problems of numerical analysis, business applications involving much input-output movement of data, and logically complex applications involving non-numeric data are all conveniently expressible in JOVIAL.

JOVIAL is a readable programming language, utilizing the familiar notations of algebra and symbolic logic. Where no standard conventional notation corresponds to the symbols of the language, English words are used, making JOVIAL largely self-explanatory. In addition, JOVIAL has no format restrictions, and with the ability to intermix comments among the symbols of a program and to define notational additions to the language, the only limit to expressiveness is the ingenuity of the programmer. A JOVIAL program may thus serve as its own documentation, allowing program maintenance and revision to be performed with relative ease by programmers other than the original author.

The convenient subordination of detail without its loss in JOVIAL also contributes to readability, and greatly expedites the task of writing programs. One simple JOVIAL statement can result in the generation of scores of machine instructions which might normally take hours to code in a machine oriented language. This reduction in program size proportionally reduces the opportunity for purely typographic errors which, due to JOVIAL's readability, are much more obvious when they do occur. Once the logical or structural type of error has been eliminated, confidence in the resulting machine language program can be high, because all the minor details of code are machine generated and presumably error free.

Computer users are often faced with the necessity of producing large numbers of computer programs in short periods of time. Such a language as JOVIAL will alleviate the heavy burden this places on the existing programming staff and thus tend to lessen the requirement for quick augmentation with relatively inexperienced programmers.

JOVIAL will also simplify and speed up the related problem of training personnel in the design of data processing systems and the development of computer programs for such systems because, although JOVIAL was designed primarily as a tool for professional programmers, its readability makes it easy for non-programmers to learn and use, and should also help to broaden the base of JOVIAL users beyond those engaged in actual programming.

JOVIAL is a machine-independent programming language, answering the pressing need for a common standard of communication between the users of many different computers. As a common programming language, JOVIAL serves both as a means of communicating information processing methods between people and as a means of realizing a stated process on a number of different computers. Consequently, JOVIAL will significantly reduce the retraining problem encountered in shifting programming personnel to projects based on new or unfamiliar computers.

Translation programs, referred to as compilers, are currently being written for the 709 & 7090, the AN/FSQ-7 & Q-8, the AN/FSQ-31, and the 2000 computers. Translating between JOVIAL and machine or machine-oriented language, these compilers will allow the efficient translation of JOVIAL programs from one computer to another. Thus, through JOVIAL, it becomes possible to develop data processing systems for existing computers that can operate on future, more powerful computers with minimal conversion costs.

THE JOVIAL COMPILERS / The compilers translating between JOVIAL and the various machine languages all consist of a pair of sub-programs performing two separate and distinct phases of translation. The first phase is concerned largely with codifying the data description declarations and with determining appropriate sequences of elementary operations conforming to the algorithms described in the JOVIAL statements. This first phase of translation, performed by a program known as the "Generator," is entirely machine independent, producing as its output an "Intermediate Language" (IL), also machine-independent, this functions as a limited sort of Universal Computer Oriented Language (UNCOL). The program translated from JOVIAL to IL in the first phase, is translated from IL to a machine language during the second phase by a program known, simply, as the "Translator." Figure 1 illustrates the structure of a JOVIAL compiler. Notice that the Generator accepts COMPOOL declaration describing system data, and that a JOVIAL program may contain portions coded in a machine-oriented language.

Each JOVIAL compiler has its own unique Translator. The Generator and Intermediate Language, being machine-independent, however, are common to all, as shown in Figure 2.

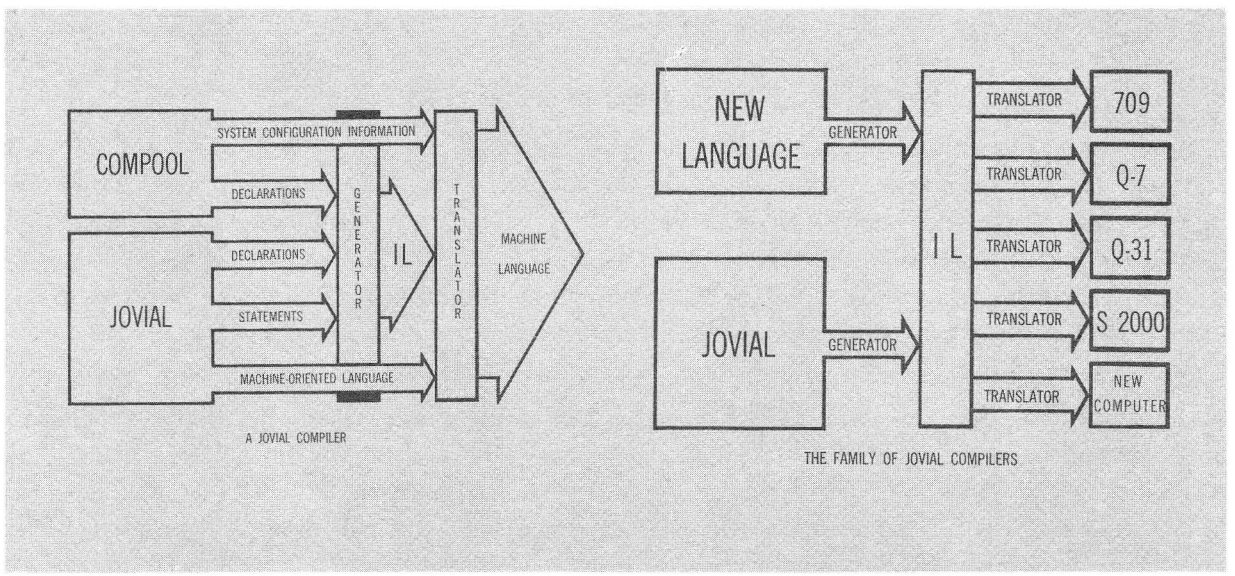


Figure 1

Figure 2.

This arrangement has several convenient features:

- A single Generator means that control over the form of the JOVIAL language is centralized, eliminating otherwise irresistible tendencies toward the growth of diverging dialects.
- One Generator eliminates the duplication of effort involved in writing a unique Generator for each compiler.
- A common IL allows a family of compilers to be produced for some new problem-oriented language merely by writing a new Generator to translate to the IL.

A brief example, of a matrix multiplication routine, will serve to illustrate the general appearance and some of the properties of the JOVIAL language.

$$\gamma_{ij} = \sum_{k=0}^{n-1} \alpha_{ik}\,\beta_{kj} \qquad \begin{array}{l}\text{For } i = 0, 1, \ldots, x\text{-}1 \\ \text{For } j = 0, 1, \ldots, y\text{-}1\end{array}$$

Computational Scheme for Matrix Multiplication.

```
          MATRIX' MULTIPLICATION. "THIS ROUTINE MULTIPLIES ALPHA, AN X-ROW,
                  N-COLUMN, FLOATING POINT NUMERIC MATRIX BY BETA, AN N-ROW,
                  Y-COLUMN, SIMILAR MATRIX TO OBTAIN GAMMA, AN X-ROW, Y-COLUMN,
                  PRODUCT MATRIX."
                  BEGIN
                  "THE FOLLOWING DEFINITIONS IMPROVE READABILITY."
DEFINE FLOATING   "F" $
DEFINE THRU       ",1," $
                  "THE FOLLOWING DEFINITIONS SPECIFY DIMENSIONS AND REQUIRE THAT
                  X, Y, AND N BE REPLACED WITH INTEGERS BY THE ROUTINE'S USER."
DEFINE ROW        "X" $
DEFINE COLUMN     "Y" $
DEFINE NORM       "N" $

ARRAY ALPHA       ROW NORM      FLOATING $
ARRAY BETA        NORM COLUMN   FLOATING $
ARRAY GAMMA       ROW COLUMN    FLOATING $

                  FOR I = Ø THRU ROW-1 $
                    BEGIN
                    FOR J = Ø THRU COLUMN-1 $
                      BEGIN
                      GAMMA ($I,J$) = Ø $
                      FOR K = Ø THRU NORM-1 $
                        GAMMA ($I,J$) = GAMMA ($I,J$) + ALPHA ($I,K$) *BETA ($K,J$) $
                      END
                    END
                  END
```

THE CLIP CONTRIBUTION / CLIP (Compiler Language for Information Processing), began in 1958 as a research activity in automatic coding. Bob Bosak, then in charge of the Programming Research and Development Group, encouraged this exploratory effort, which was undertaken by Jules Schwartz and Erwin Book. While they were still in the early stages of their work, the ALGOL-58 specifications were published. They discovered that there were many similarities to their own effort and since ALGOL was being so widely circulated and accepted, they adopted the ALGOL-58 notation.

While studying the problem of how to build a compiler, Jules and Erwin found they were developing a language that could be used to express the compiling process. At about this time (January, 1959), Jules was transferred to SDC's SACCS Division in New Jersey to work on the utility programs for this system and took with him the first results of the automatic coding research activity. At SACCS, concentration was on defining the procedure-oriented language, which became JOVIAL, while at SDC Santa Monica, the effort continued on developing the techniques of the compiling process, or CLIP, with a continual exchange of information going on between both groups.

CLIP was designed to be a tool for experimentation with advanced coding techniques. A CLIP compiler was written and checked out. At about the same time CLIP was being used to compile itself, the JOVIAL common generator project was set up, under the auspices of the SDC Programming Languages Committee, headed by Bob Bosak,

At first, this generator was being written in its own language, JOVIAL, but as the task progressed, it became apparent that it wasn't necessary to use all the facilities offered by the JOVIAL language just for the purpose of writing the generator. In fact, project members discovered that what they were using of JOVIAL pretty well corresponded to CLIP. Some minor modifications to the CLIP compiler were made by Harvey Bratman and Erwin Book. They realized they could write the JOVIAL generator in CLIP and use the CLIP compiler to produce a 709 version of the JOVIAL generator — so bypassing the tedious task of hand-translation. As a result of this decision, the JOVIAL generator is already being checked out, months ahead of schedule.

In addition to those already mentioned, members of the CLIP Project include Ellen Clark, Don Englund, Harold Isbitz, Howard Manelowitz and Ellis Myer.

Present plans are to use CLIP to produce 709 versions of the Q-7 and 2000 translators as a first step in producing translators for these computers. Thereafter, while JOVIAL will be the standard corporate programming language, CLIP will continue to be used as a research and experimental tool.

NAMES AND JOVIAL/Ever since its inception as a problem-oriented language for SACCS early in 1959, JOVIAL has been closely associated with the name of Jules Schwartz. Indeed, "JOVIAL" itself is an acronym standing for "Jules' Own Version of the International Algebraic Language" (IAL, later renamed ALGOL), and was coined in happy preference to "OOVIAL"—"Our Own Version of the IAL." Jules was then in charge of the original compiler project located with SDC's SACCS Division in Lodi, New Jersey (later, in Paramus), and the language was named during his absence on a business trip.

In May of 1960 when the decision was made to standardize on JOVIAL as a common programming language for SDC, a Generator project was set up, headed by Erwin Book. Currently, four Translator projects exist: the Q-7, Q-8 Translator project, headed by Cal Jackson; the 2000 Translator project, headed by Ellen Clark; and the 709, 7090 and Q-31 Translator projects, headed by Jules Schwartz in Paramus. Working compilers for all four computers are scheduled for operation at about the same time —in the spring of 1961.

Robert Bosak/B.A., Mathematics, UCLA, 1949; graduate work in Game Theory and Statistics also at UCLA. Joined RAND in 1948 to work on analysis and programming of scientific problems. Joined Lockheed Aircraft Corp., in 1951, where, with its Georgia Division, he organized and directed

the Mathematical Analysis Group. Returned to RAND in 1956, heading the SAGE Program Development Group and later was appointed to SDC's Advanced Planning Staff. Became head of Programming Research and Development Group in 1958, responsible for directing pure and applied programming research. In various posts with SDC has been continuously associated with development of programming techniques.

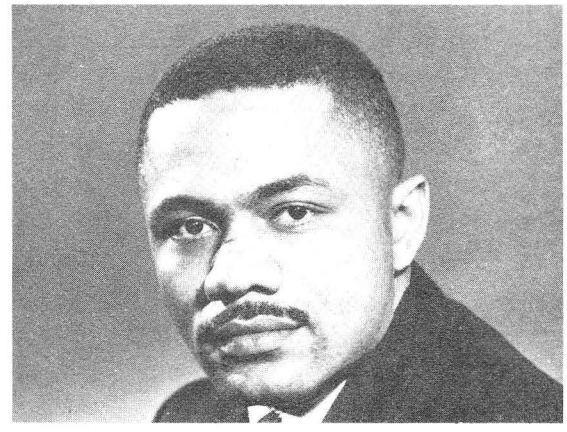
Erwin Book / B.S. degree, CCNY, with a major in statistics. Joined RAND in November, 1955. Worked on writing programs and testing the initial version of the SAGE system. Later, went into utility programming area and worked on design of the COMPASS system. Wrote some programs for COMPASS, including the assembly program. He then turned his attention to problem oriented languages and automatic programming. Headed the CLIP project in the Research Directorate. Presently in charge of the common generator project for JOVIAL.

Harvey Bratman / B.A., Mathematics, UCLA, 1950. Statistician at U.S. Naval Ordnance Test Station, China Lake, California, June 1950 through February 1952. Joined Mathematics Department of Lockheed Aircraft Corporation, working on a variety of

aeronautic engineering problems. In 1956 became responsible for systems and subroutines used at Lockheed. Member of the SHARE 709 System (SOS) Committee. In December, 1958, joined SDC and has been working in the field of automatic coding. He is one of the designers of CLIP and has coded and checked out part of the 709 CLIP generator.

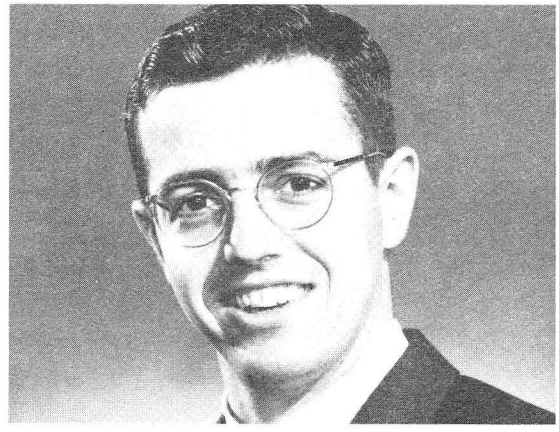
Ellen Clark/B.A., Mathematics and Physics, University of Kentucky. Mathematics instructor, Vanderbilt University and North Georgia College. Started programming engineering and managerial problems with the Mathematical Analysis Group at Lockheed Aircraft's Georgia Division. Designed and implemented general purpose output system for IBM 704. Joined SDD of RAND in 1957 for work on SAGE. As member of SDC's Research Directorate, worked on CLIP Compiler project and is currently on loan to SSRL to design JOVIAL translator for Philco 2000.

Calvin Jackson/B.S., Mathematics, Morehouse College; graduate study at Atlantic University. Formerly with the Mathematical Analysis Department of Lockheed Aircraft Corp., Marietta, Georgia. While at Lockheed, his work included the integration of Flutter analysis programs into an automatic system and the installation and maintenance of Lockheed's Automatic Operating System. Joined SDC in January,



1960; assigned to the JOVIAL project since February, 1960. Currently in charge of the Q-7 translator project.

Jules Schwartz / B.S., Mathematics, Rutgers University, 1951. Graduate work in Mathematics and Mathematical Statistics at the University of Southern California, UCLA and Columbia University, where he is presently doing part-time graduate study. Joined The RAND Corporation Numerical Analysis Department in 1954. In January, 1956, transferred to the System Development Division, which later became SDC. Since early 1959, has been on loan to the SACCS Division from SDC's Research Directorate.



Christopher Shaw / Studied mathematics at UCLA. With SDC since June 1957, working on checkout and maintenance of SAGE computer programs until assignment to the Training Section in May, 1958, as Programmer Instructor. After completion of a 709 Programmer's Manual in July, 1959, began work on a JOVIAL training manual which, with interruptions, was continued until transfer to the JOVIAL project in March, 1960. Currently working on JOVIAL documentation with the Generator Project.

SYSTEM DEVELOPMENT CORPORATION / 2500 COLORADO AVENUE / SANTA MONICA, CALIFORNIA

BR-3/11-60