DOCUMENT RESUME

ED 027 743 EM 007 134

By-Fasana, Paul J., Ed.; Shank, Russell, Ed.

Tutorial on Generalized Programming Language s and Systems. Instructor Edition.

American Society for Information Science, Washington, D.C.
Spons Agency-National Science Foundation, Washington, D.C.
Pub Date Jul 68
Grant-F-NSF-GN-657
Note-65p.; Manual based on materials prepared and presented by Thomas K. Burgess and others at the Annual Convention of the American Society for Information Science (New York, October 22-26, 1967)
EDRS Price MF-$0.50 HC-$3.35
Descriptors-*Computer Science, *Computer Science Education, Information Retrieval, Information Storage, *Manuals
Identifiers-COBOL, FORTRAN, PL I, SNOBOL

This instructor's manual is a comparative analysis and review of the various computer programing languages currently available and their capabilities for performing text manipulation, information storage, and data retrieval tasks. Based on materials presented at the 1967 Convention of the American Society for Information Science, the manual describes FORTRAN, a language designed primarily for mathematical computation; SNOBOL, a list-processing language designed for information retrieval application; COBOL, a business oriented language; and PL/I, a new language incorporating many of the desirable features of FORTRAN and COBOL but as yet implemented only for the IBM 360 computer system. (TI)

TUTORIAL ON

GENERALIZED PROGRAMMING LANGUAGES AND SYSTEMS

Instructor Edition.

Edited by
Paul J. Fasana
Columbia University Libraries
New York, N. Y.
and
Russell Shank
The Smithsonian Institution
Washington, D. C.

Based on materials prepared and presented
at the 1967 ASIS Annual Convention

Principal Tutor: Thomas K. Burgess

Partially supported by a grant from the
National Science Foundation (GN-657)

Tutorial Subcommittee
Conference Program Committee
American Society for Information Science
(formerly American Documentation Institute)
Annual Convention (1967)
New York, 1968

ADI Tutorial Manuals

PREFACE

A prime responsibility of a professional society is to foster continuing education activities covering new developments in topics of importance to the work of its members. This is particularly true in rapidly expanding and highly complex technologies such as those in the field of information science.

With this view, the 1967 Conference Planning Committee of the American Documentation Institute (now the American Society for Information Science) chaired by Paul Fasana of the Columbia University Libraries, established a Tutorial Subcommittee to organize training sessions for presentation at the Conference. The Subcommittee, under the direction of Russell Shank, then Associate Professor at the Columbia University School of Library Service, agreed to develop three workshop tutorials for the following areas: elements of information systems; electronic data processing concepts; and generalized programming languages and systems.

The tutorials on these topics ran concurrently. Each began with a general session in which the tutorial leader gave an overview of the topic to be covered. The participants were then formed into smaller workshop or seminar groups for detailed instruction by a team of tutors. Each tutorial lasted the entire day. The general sessions were limited to about 100 people; seminar groups were limited to about 25 people. In the seminar groups each of the tutors either covered the entire topic simultaneously, or presented a part of the information to be covered, repeating their presentations as groups were rotated among them.

In planning the tutorials it was apparent that syllabi or workbooks were needed to assure that the basic information to be presented was uniform and organized for the instructors of each of the groups. It was assumed that if syllabi were carefully prepared they could be made generally available and be useful in similar courses at other national and local meetings of information science groups.

Three manuals, covering the three different topics, were prepared and used experimentally at the Conference. Initially, it was hoped that each manual would contain a comprehensive outline of the topics to be presented, a display of the illustrations and visual material used in the lectures, glossaries, bibliographies, and problems. It was further assumed that the sessions would be more meaningful if an instructor's version and a student's outline (with sufficient space for notes) were prepared. The instructor's edition would have enough detailed information to allow other instructors to present the course.

The variation among the approaches to the three topics treated in the tutorials made it difficult to attain this objective of uniformity in style of presentation, at least on the first attempt. All three manuals have been extensively revised for publication. This material will undoubtedly be improved through refinements as the tutorials are presented elsewhere in the future.

This package contains both an instructor's and a student's version of the syllabus for each topic. Undoubtedly other instructors will wish to make modifications of these manuals to suit local needs and instructor's talents. These manuals are offered, therefore, primarily as examples for

for those who might be planning similar tutorials. The student edition may be produced in quantity locally as required.

Very briefly, the scope of each of the three sessions of the 1967 tutorials was as follows:

Tutorial I. Elements of Information Systems.

Paul L. St. Pierre, Principal Tutor

An introductory course for those with no previous experience or formal training in systems analysis. Objective will be to familiarize participants with the techniques (file analysis, record analysis, flow-charting, costing) and the terminology of systems analysis. Those who complete this tutorial should have a knowledge of what systems analysis is, what function it serves and how it can be applied.

Tutorial II. Electronic Data Processing Concepts.
Bruce Stewart, Principal Tutor

For those with little or no experience with EDP equipment. Emphasis will be on a functional description of various types of equipment. Participants will be given an understanding of how such equipment works. They will not be trained to operate particular machines.

Tutorial III. Generalized Programming Languages and Systems.
Thomas K. Burgess, Principal Tutor

For those with considerable systems analysis and programming experience. Presentation of a comparative analysis and review of the various programming languages currently available, especially as they apply to information storage and retrieval. The relative merits of different program languages for use in textual analysis, file structure, file manipulation, and similar topics will be stressed. Program languages to be covered include FORTRAN, COBOL, PL 1, SNOBOL, and ALGOL.

Russell Shank
Washington, D.C.
July, 1968

TABIE OF CONTENTS

TABIE OF FIGURES

Generalized Programming Languages and Systems

A Review of Programming Languages with Reference to Information Storage and Retrieval

Introduction

As the title implies, this manual is designed to discuss higher level programming languages and their capabilities for performing text manipulation and data retrieval tasks. It assumes that the student has some knowledge and experience with a programming language and with information retrieval activities. It is not necessary for the student to be familiar with all the languages discussed, but he should have a working knowledge of at least one higher level language. It was difficult for the authors to reduce the amount of information available on each of these languages to a useful amount of information which could be presented in a short period of time. Therefore, we have had to limit our discussions to talking about the particular structures of the various languages useful for IR activities. No attempt is made to teach programming in any of these languages; this can be accomplished through the use of the bibliography and self-study on the part of the student.

Separate consideration of the languages provides an examination of the pros and cons of each. However, the decision is left to the user as to which language to use. This decision must not only be based upon the availability of the languages but also on the computer environment in which he will be operating and the particular applications which he is considering. We have chosen to discuss FORTRAN and COBOL because of their wide availability on all kinds of equipments and PL/I and SNOBOL because of their particular characteristics and ease in performing information retrieval tasks.

Preliminary Assessment

Until recently, the design of programming languages took no consideration of information retrieval requirements and applications. As a consequence, advancement of information retrieval methods has been retarded. In addition, the design of hardware has not fully considered information retrieval requirements. Many operating systems for various computers are not designed to handle information retrieval requirements. Because of time limitations, no discussions of hardware and operating systems are included in this presentation. However, hardware design must be considered when evaluating programming languages for use in information retrieval applications. Many of the

advantages or disadvantages of the various programming languages which will be considered can be improved or diminished by either the operating system and/or the computer on which the language runs. Therefore, there is more involved in deciding which higher level programming language to use than mere choice of language.

FORTRAN, which is one of the earliest compiler languages to be developed, was primarily designed for mathematical computation. FORTRAN is not an easy language to use since it lacks string manipulation capabilities, does not handle variable data, and has limited input and output operations. It is, however, one of the most widely implemented languages and is available in nearly every computer center.

SNOBOL is a list-processing language and was designed specifically for information retrieval application. It is an easy language to learn but uses a large amount of core storage and requires long production run time. Although it is used in a large number of installations, it has not been generally implemented.

COBOL is a business oriented language but was not designed specifically for information retrieval activities. Although many

information retrieval tasks can be performed with COBOL, it is awkward and difficult to do so. COBOL structures are rigid, and string-manipulation and text-editing is difficult to accomplish. COBOL is widely implemented and should be available in most computer centers.

PL/I is the newest language to be discussed and incorporates many of the desirable features of both FORTRAN and COBOL, as well as the list-processing languages. PL/I has only been implemented for the IBM 360 computer system; therefore, at present there is limited availability. Current experience with PL/I indicates that the compiler is inefficient with a large amount of overhead in the programs. Undoubtedly, as time goes on, PL/I compilers will become more efficient.

FORTRAN

FORTRAN

Characteristics of Fortran

FORTRAN is procedures oriented and was intended to compile programs for solving scientifically oriented problems. Thus, programs feature:

"Words" which are fixed in length

Arithmetic operations

Array handling

Matrix manipulation

Conditional branching

Algorithmic problem solution

IR Functions

The question to be considerd for information retrieval by the programmer is whether or not the features listed above will satisfy his needs in a particular program. These include the following common functions:

(English) word matching

Word replacement

String manipulation

Sorting and editing

List processing

Phrase analysis

Syntax analysis

Concept associations

Hierarchical subject arrangements

Characteristics of an IR System

An analysis of these functions points to a system design which features:

Character manipulation

Variable length "word"

List manipulation

Extensive input-output

Array capability

Arithmetic capability

Matrix manipulation

Conditional branching

Algorithmic problem solution

Additional Fortran Features

Comparing the desired features of the system with those of the FORTRAN compiled programs, it is quickly seen that all of the features of FORTRAN programs are needed, except the fixed "word" length orientation. Not actually featured by FORTRAN but programmable are these important characteristics:

Variable length "word"

Character manipulation

List manipulation

Extensive input-output

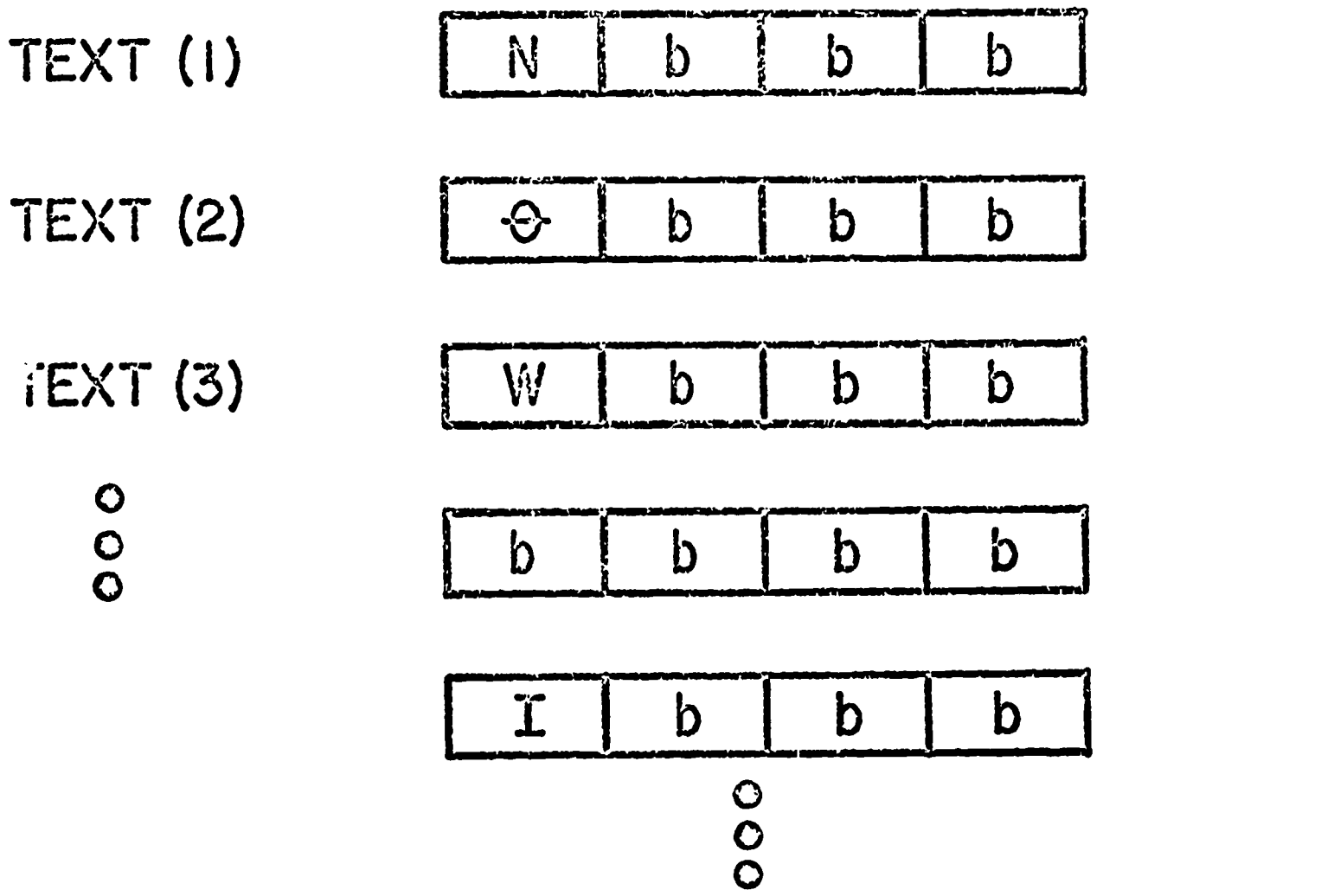
Character Manipulation

To gain character manipulation and variable word length ability is a costly technique and wasteful of storage in that an entire FORTRAN word is used to store a simple alphabetic character. For example:

CHARACTER MANIPULATION

Adequate Storage

NOW IS THE TIME FOR ALL ⁱⁱⁱⁱ




```
READ (5,1) (TEXT(I), I = 1,80)
1 FORMAT (80A1)
```

Greater efficiency can be attained if FORTRAN is used in a binary or hexadecimal computer, since every bit configuration represents a valid numerical quantity. In this case, FORTRAN arithmetic can be used to isolate, replace, and generally manipulate characters. This is shown in the following two illustrations:

CHARACTER (AND BIT) ISOLATION
Packed Hexadecimal Words

NOW IS THE TIME FOR ALL !!!!

TEXT (1)

N	Θ	W	b

TEXT (2)

I	S	b	T

TEXT (3)

H	E	b	T

o
o
o

o
o
o

NNNN = TEXT (2) / 16**4

NNNN = NNNN * 16**6

NNNN

0 0	0 0	I	S

NNNN

S	0 0	0 0	0 0

CHARACTER (AND BIT) REPLACEMENT
Packed Hexadecimal Words

NNNN

T	⊖	N	E

I

I	00	00	00

J

b	00	00	00

N(1) = NNNN / 16**6

00	00	00	T

N(1) = N(1) * 16**6

T	00	00	00

N(2) = NNNN / 16**6

00	T	⊖	N

N(2) = N(2) * 16**6

N	00	00	00

N(2) = N(2) / 16**4

00	00	N	00

N(3) = I / 16**2

00	I	00	00

N(4) = J / 16**6

00	00	00	b

NNNN = N(1) + N(2) + N(3) + N(4)

T	I	N	b

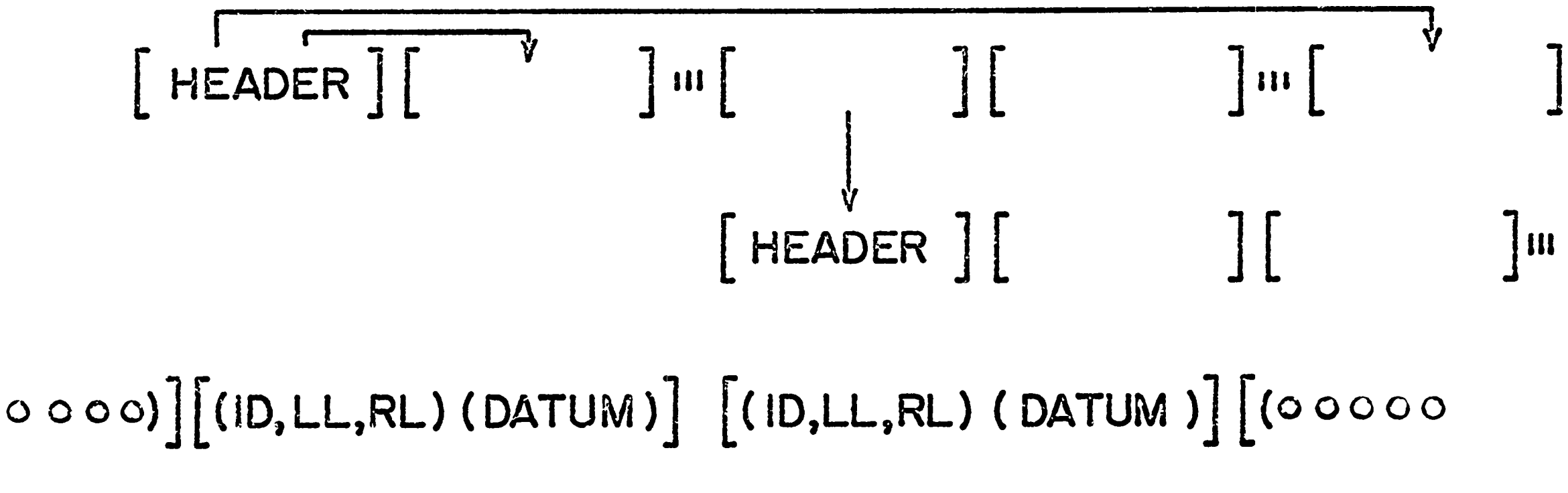
Supplementary Aids

FORTRAN programs can be used for information retrieval but only at the cost of decreased efficiency. By using supplementary aids, the amount of inefficiency can be reduced. The aids can be specially written subroutines in assembly language. These are designed to provide the characteristics missing from FORTRAN, and can be used with a FORTRAN program when needed.

There are published and unpublished subroutines which are available, and before writing an original assembly program the programmer should check the library at his own installation.

The use of SLIP (Symmetric List Processing; see reference list) can be useful. Subroutines in SLIP are called "primitives" which can be manipulated to build more meaningful subroutines. These may be called into FORTRAN programs.

When SLIP is used, character lists are built with sufficient headers, identification, and pointers to permit extreme flexibility in manipulations within these lists, their sublists, sub-sub-lists, etc. These include sort sequence routines and push-up and push-down storage capability.



SLIP LIST

SNOBOL

Background

SNOBOL is a high-level programming language specifically designed for manipulating groups ('strings') of characters of varying lengths. It was developed at Bell Telephone Laboratories (Holmdel, New Jersey) and has been implemented there in several versions for a variety of different computer configurations.

This report assumes that the reader is familiar with SNOBOL 3. SNOBOL 4 is implemented (at present) only for OS/360, requires a 360/40 or larger, and requires about 240 K (bytes) of core. This may pose difficulties for some potential users.

A version of SNOBOL 3 for OS/360 is now in the final stages of development by Dr. Luther Haibt, IBM, Yorktown Heights, New York. Unlike other versions, this SNOBOL 3 is a compiler, not an interpreter. It requires something over 128K (i.e., 192K) to implement. This sounds as if it will be a useful and interesting tool. It has features not in Bell Laboratories SNOBOL 3, but also lacks certain features of SNOBOL 3 (i.e., back referencing).

Various versions of SNOBOL have been implemented (as of 1966) for the following equipment:

IBM 1620
IBM 7040/44
IBM 7094
RCA 601/604
SDS 930-940
CDC 3100

At Columbia, SNOBOL 3 has been used on the IBM 7040-7090, Direct Coupled System, under IBSYS and IBJOB, as well as on the 7090 alone. SNOBOL 4 is being used on the 360/50 - 360/75 combination, running under ASP. SNOBOL 4 is also available for paired 360/62's, running under HASP.

An interesting SNOBOL variant for OS/360 has been implemented (in two versions) at MIT.[1]

Uses - General

Structurally, SNOBOL is closely related to the Markov algorithm language,[2] this makes it a powerful and general language. The majority of users employ it in logical, algebraic, and mathematical analyses. It is used to simulate compiler and other computer languages, and to translate from one language to another.

Although considered powerful and general, SNOBOL seems to have been explicitly designed to deal with problems in information science and librarianship, such as language analysis, bibliographic work, index analysis, etc.

Operations

The fundamental operations of SNOBOL are:

1) The ability to name strings of characters of varying length (TITLE 'WAR AND PEACE'; AUTHOR 'TOLSTOY.')

2) The ability to concatenate named strings; for example, if the computer is told to print out: 'THE AUTHOR OF ' TITLE ' IS ' AUTHOR, it emits: "THE AUTHOR OF WAR AND PEACE IS TOLSTOY."

3) The ability to make pattern-matches of strings. (For example, to answer yes or no to the question whether the string of characters named AUTHOR matches the literal string of characters 'TOLSTOY.')

SNOBOL permits a number of other logical operations on strings; one of its most important values is that it takes care of common problems in dealing with alphabetic information of varying length. In other languages their solution can be tedious and lengthy to program, increasing the chances of logical, syntactic, and punching error, and making debugging difficult and time-consuming.

The syntax, logic and form of expression of SNOBOL are clear and readily grasped, but more abstract than those of such languages as PL/I or COBOL.

SNOBOL operates with the basic assumption that all strings of characters are of variable length unless told otherwise (valuable and important from the programmer's point of view).

SNOBOL also permits indirect referencing and has a well-designed system for permitting the programmer to write and call his own generalized subroutines, functions, or procedures.

As is the case with any high-level language, SNOBOL trades ease of use and flexibility against efficiency of computer time. Some operations (e.g., concatenation and pattern matching) seem comparatively quite efficient. Other operations (e.g., character-by-character sorting) are quite inefficient. Prototype programs can be written rapidly in SNOBOL with the more inefficient sections replaced later by machine language. Input/output (at least in SNOBOL 3) is somewhat clumsy, as it was not really written for and is not as much used for information science as for other purposes. (Dr. Haibt's 360 SNOBOL 3, being mainly a compiler, promises to be (relatively) more efficient than other versions.)

SNOBOL is ideally designed for manipulating natural language information without resorting to explicit coding of input - combining, comparing, substituting, deleting, correcting, permuting, etc.

Uses - Specific

At Columbia, a number of programs or program segments for bibliographic data manipulation for both experimental and production purposes have been written. A brief description of some of this work might help the reader to determine if the language can be of use to him more than a more detailed description of the language itself.

One major area of concern at Columbia has been to develop language-independent unit operations (as in chemistry) for information handling; these can be used for analysis of problems, flow-charting, writing procedures in various computer languages, assisting in language development, and better understanding of bibliographic and indexing operations in rigorous terms.

Sixty or so of such operations have been written and tested in SNOBOL, and about forty inserted in a procedures or function library. These include format and layout routines (column formation, page layout, paragraphing, line justification) as well as routines more directly linked with information science theory.

Two programs for book catalogs have been written - one simple one by Jessica L. Harris was used for the production of a computer-based school library catalog (Farmingdale, Long Island - 10,000 titles). The second, by Brian Aveney, is a prototype program, demonstrating how conventional bibliographic elements used as access points may be recognized, and conventional entries, in various types of arrangements, can be produced without explicit tagging of elements in input.

Another program produces book indexes using human-created entries, but provides by program all subordination and consolidation

of references, repetition of subjects continued from one page to the next, page layout, page numbering, running heads, elimination of widows, capitalization, insertion of most function codes for printing, and supplying index statistics.

The corrections, additions, and deletions routine used with this program is completely general in that it may be used with any type of material and requires no coding or use of line numbers. The index program itself is completely general in that line width, page size, and so on may be defined in terms of the needs of a particular index. A still more general program, permitting up to four levels of subordination, up to four columns per page, and insertion of pronunciation, etc., with entries, is now being debugged.

Other examples of programming which might be indicative include experimentation in automatic indexing, programs for three different types of KWOD indexes, a generalized search routine independent of the input data format, a tentative general bibliographic program, prototype journal (in the style of the H.W. Wilson Reader's Guide), index programs, frequency count routines, routines for internally tagging words or parts of entries, and so on.

Also written are routines for expanding index entries to permit combining compatable existing indexes, a quite elaborate program (by Jessica L. Harris) for the analysis of thesaurus and subject headings, and some work on directory production problems.

SNOBOL has allowed all of these programs and routines to be written within 10 months with never more than three people programming and with all of these people working full-time on other projects. Up until 10 months ago, too, two of these three had had no computer programming experience of any kind.

Summary

SNOBOL, while not an efficient processing language, is easy to learn and debug, ideally suited to information problems, quite flexible, very powerful, and likely to grow in use, efficiency, and importance.

FOOTNOTES

[1]Adelbert G. Goff, OS/360 SNOBOL User's Manual, Boston, June, 1967.

[2]Markov, A.A. Theory of Algorithms. Akad. Nauk SSSR, 1954. (English edition OTS-USDC 1961) (citation taken from Rosen, Saul, ed. Programming Systems and Languages. New York, McGraw-Hill, 1967, p. 454.

APPENDIX I

Installations using SNOBOL (1966)

Dr. Hsu, in his Introduction, gives a list of installations using SNOBOL. Some of these installations may have locally produced manuals or text materials:

AVCO Corporation
Bellcom, Inc.
Bell - White Sans
The Boing Company
Brigham Young University
University of California at Berkeley
University of California at Los Angeles
University of Chicago
Columbia University
University of Florida
FMC Corporation
Grumman Aircraft
Harvard University
University of Hawaii
University of Houston
IBM France
University of Illinois
Imperial College (U.K.)
Lockheed Missile and Space Company
University of Maryland
Massachusetts Institute of Technology
University of Michigan
National Security Agency
New York University
Ohio State University
Polytechnic Institute of Brooklyn
Princeton University
Purdue University
Service Bureau Corporation
Stanford University
Technology Incorporated
USAF Academy
University of Utah

University of Washington
Washington State University
Thomas J. Watson Research Center
University of Wisconsin
Yale University

The list would doubtless be much longer if it were recompiled today; including, for example, the Bronx (N.Y.) High School of Science, the University of Delaware, and many others.*

*Robert Hsu and Laura Gould, A Linguist's Introduction to SNOBOL, Honolulu, October 1966.

COBOL

Recommendation: If you have not had any experience with COBOL, it is recommended that you read McCracken's "A Guide to COBOL Programming."[1]

Background

Historically, COBOL grew out of the desire to develop a procedure-oriented language that would be closer to the common English used in the business world. A second goal was to get a source language compatible with any computer. Work began on COBOL in 1959 under the direction of a group of computer experts, many of whom had worked on somewhat similar "English-Language" systems. The maintenance and development of the language was established as one of the functions of a group called the Conference ON DAta SYstems Languages (CODASYL).

The first formal report defining COBOL was issued in 1960; that version of COBOL is referred to as COBOL-60. The next year a more refined version was described, called COBOL-61, and is probably the most familiar one in use today.

COBOL-61

COBOL-61 consists of two aspects of language elements: required and elective. The original idea was to establish certain

requirements that a source language must meet to be called COBOL. The elective aspects of the language was left up to the discretion of the implementor. In addition, many computer manufacturers have added their own extensions to COBOL.

COBOL-65

In 1965 the CODASYL Committee revised COBOL again and came out with COBOL-65, which is the version being used in most third-generation equipment. This version also has been accepted by the USA Standards Institute Committee on Common Programming Languages.[2]

STRUCTURE

COBOL consists of the following structural divisions

Identification Division

Environment Division

Data Division

File Section

Working-Storage Section

Report Section

Procedure Division

Identification Division identifies the source program and the output of compilation. In addition, any other information such as date, name of programmer, etc., may be included.

Environment Division specifies a standard method of expressing those aspects of a data processing problem that are dependent upon the physical characteristics of the users particular computer.

Data Division describes the data that the object program is to accept as input, to manipulate, to create, or to produce as output. Data to be processed falls into two categories:

1. Data contained in files (input, output, report) and enters or leaves the internal memory of the computer from a specified area or areas.
2. Data developed internally and placed into intermediate or working-storage or placed into specific format for output reporting purposes.

Procedure Division contains declaratives and procedures. Declaratives allow certain procedural changes to be made during processing.

EXAMPLE: The USE verb allows us to do special processings at the beginning of files. Procedures are composed of paragraphs or sections in which all the operations of the data are performed.

ADVANTAGES OF COBOL FOR INFORMATION RETRIEVAL

1. The tabular representation of Information in the Data Division (the level structure concept) provides an excellent capability for retrieving parts of established files.

2. The capability of sorting and merging within COBOL provides a powerful tool for organizing files for retrieval of information.
3. The familiarity of the language provides for easier use by persons required to use the language.
4. The availability of compilers on many computers make it especially attractive. (Availability of COBOL files already in use).
5. The ease of using subroutines for conversational access via terminals.
6. The advantage of report writing capabilities within the language.

DISADVANTAGES OF COBOL FOR INFORMATION RETRIEVAL

1. The language was not designed for Information Retrieval and thus is often awkward and wordy.
2. The rigid level structure concept does not provide ease of searching string data (text editing, string manipulation, etc.).
3. The response time on conversational system is often not as fast as other languages.

CURRENT COBOL SYSTEMS FOR INFORMATION RETRIEVAL

Because of COBOL'S file structure and universality, it has been chosen for specific information systems. An example is the programming of the MEDLARS system (Medical Literature Analysis and Retrieval System) in COBOL for use on the IBM 7094 at UCLA.

An important reason for choosing COBOL for an on-line retrieval system is the prior establishment of many files required by the system in previous batch processing system. This has been helped by the expansion and use of COBOL in the direct access storage field over the last few years. Two such systems tied into COBOL that are currently being used are General Electric's Integrated Data Store (IDS) and IBM's Index Sequential Access Method (ISAM).

GE's IDS Retrieval System

Integrated Data Store (IDS) is a new information oriented method of integrating the operating functions of an information retrieval system. It allows an efficient system for the storing and retrieval of data. Some of the specific advantages claimed for IDS include:

1. Shorter time for design and programming of information systems.
2. Reduced costs in design and programming of information systems.

3. More efficient use of disk storage capacity.
4. Reduced record maintenance, updating, and retrieval time.

The IDS language provides a simplified means for record processing in the environment of mass random access storage and extends the range of COBOL. This extension lies basically in four additional IDS instructions: STORE, RETRIEVE, MODIFY, and DELETE. These macro-instructions work in conjunction with, and supplement, the normal COBOL language in handling files, records or fields. Such extensions make COBOL into a more powerful retrieval language.

IBM ISAM Retrieval System

This system developed by IBM for use on the 360 series is aimed at processing files either sequentially or randomly. It basically operates on complete records and is organized so that rapid sequential processing is possible. Indexed sequential organization by reference to indexes associated with the file, makes it possible to quickly locate individual records for random processing.

In this method of organization, the programming system has control over the location of the individual records. The user, therefore, needs to do little I/O programming; the programming system does almost all of it since the characteristics of the file are known.

BACKGROUND

In 1963, the Advanced Language Development Committee of the SHARE FORTRAN project was formed to recommend the successor language for the currently available FORTRANs. The Committee was made up of three SHARE members and three IBM representatives. The goals of the Committee were to provide a language which would encompass more users while still remaining a useful tool to the engineer.

The Committee found that many parts of the current FORTRAN language were outmoded (such as overlapped I/O and processing, and asynchronous operations), since hardware capabilities had increased substantially since the development of the language. The Committee published a report defining a (expanded) FORTRAN system in March, 1964 and in June, 1967, the second SHARE report came out defining data structure, the report generator, and removing some of the system 360 restrictions. In March 1965 IBM announced its PL/I which contained major revisions of the expanded FORTRAN defined by the SHARE reports. The resulting definitions in the PL/I available today contained many features that would look familiar to FORTRAN, ALGOL and COBOL programmers, and several features alien to all three.

The first PL/I compiler available from a manufacturer was Release I of the F level compiler by IBM for use under OS/360 in 1966, though

it contained only a subset of the fully defined language and lacked many of the features that make PL/I a flexible programming language. In the spring of 1967 Release II of the PL/I containing nearly the full language, in particular the RECORD oriented input/output capabilities which gave it versatility for file heading in the use of the direct access devices, but without list processing and asynchronous capabilities. In the summer of 1967 PL/I D level for DOS/360, the Disk Operating System, was released by IBM as a subset of the PL/I language. While IBM is the only manufacturer with current releases of the language, other implementations are in progress or have been announced. Digitec is said to be producing a PL/I compiler for the GE 635/645 and the Sigma 7 computers. RCA is producing a subset for its Spectra 70, and UNIVAC is also said to be working on a PL/I subset.

General Language Structure

PL/I provides program segmentation capability, which gives it its modular program structure. Programs are organized into procedures or blocks and may be made up of one or many blocks. These may be separated from one another (external) or nested within one another (internal).

Blocks

Blocks provide two important logical functions: 1) they define the scope of applicability of data variables and of other names so that the same name may be used for different purposes in different blocks without ambiguity; and 2) they allow storage for data variables to be assigned only during an execution of the block and freed for other uses at the termination of the block.

Certain blocks, called "procedure blocks", may be invoked remotely from different places in the program and provide means to handle arguments and return values.

Example of Procedure Blocks

```
MAIN:    PROCEDURE OPTIONS(MAIN);
                  .
                  .
                  .
         CALL A;
A:       PROCEDURE;
                  .
                  .
                  .
         CALL B;
          B:  PROCEDURE;
                        .
              CALL C;
                        .
                        .
                        .
              END B;
         END A;
C:       PROCEDURE;
                  .
                  .
                  .
         END C;
                  .
                  .
                  .
        END MAIN;
```

Desirable PL/1 I-R System Features

1. Large Scope of Applicability
2. Fixed and Varying Length Character String Variables
3. Character and Bit String Manipulation
4. Dynamic Storage Allocation
5. Complex Data Structure Specification
6. List and Table Processing
7. Extensive File Manipulation
8. Flexible Input and Output for Report Generating
9. Arithmetic Capability
10. Matrix and Array Manipulation
11. Macro-Language Facility
12. Asynchronous Program Execution
13. Procedure Oriented Language

Character Strings

PL/I has the ability to describe a wide variety of data types, including floating point decimal numbers, and floating point binary numbers of varying precision, fixed decimal and fixed binary numbers of varying precision and complex numeric data. Of particular interest to information storage and retrieval programming is the ability to define and manipulate fixed and varying length character strings and fixed and varying length bit strings. With varying length string data the length of the string is kept automatically when performing string lengthening or string shortening functions. Data variables of the type mentioned above can be grouped by using either arrays, structures, structures of arrays, or arrays of structures. An array is composed of elements of the same characteristics and each may have up to 32 dimensions in the current IBM implementation of PL/I. The data structure is a collection of variables and arrays not necessarily alike in characteristics. Structures may also contain other structures. Individual items of an array are referred to by "subscripted names"; individual items of a structure are referred to by names that may sometimes have to be qualified to avoid ambiguity.

In PL/I array names and structure names can be used as variables. Either name may be used as an operand of an array expression or of a structure expression and it returns an array or a structure result.

Input/Output Capabilities

PL/I contains two distinct types of I/O facilities; stream-oriented input/output and record-oriented input/output.

Stream-Oriented I/O

In stream-oriented input/output the input is considered as one stream or continuous string of characters, with all data conversion done on input. Similarly, on output, data conversion is done to convert everything to character strings and the output is one continuous string. There are three types of string I/O: data directed, list directed, and edit directed. The first two provide free-form input and output with little format control, while edit directed input/output is much like FORTRAN type input/output statements with format specifications. However, unlike FORTRAN, the format specifications are not rigid and allow variables and expressions to be contained within the format list, allowing more flexibility than is available with FORTRAN I/O. Stream input/output of the edit type may also be used on internal files as well as the conventional external file type.

Record-Oriented I/O

Record-oriented input/output offers both speed and versatility in file handling and is oriented to reading or writing logical records from

a peripheral unit. This method gives the ability for either sequential or direct access of records which may be unblocked or blocked. Files may be opened for input/output or for update and processed with record I/O.

Special Features of PL/I: ARITHMETIC CAPABILITY

Arithmetic operations and matrix and vector manipulations are roughly the same as those found in FORTRAN, having approximately the same routines such as SIN, SQRT, etc. available in the PL/I library. The same operators are available and expressions are formed in the same manner as in FORTRAN. However, variable types contained within the expressions can be any of the data types allowed in PL/I including bit and character strings. Conversion will automatically take place upon evaluation of the expression, although there is a considerable time and space overhead when writing mixed expressions such as these.

ARITHMETIC CAPABILITY

1. Very similar to fortran or algol
2. /,+ , - ,*, ** operations

   EXAMPLE :

   A = B + C* (D** 2) - E ;
3. Array operations
   (a) Up to 32 dimension in an array
   (b) A = B + C Matrix addition
   (c) A = B(1 ,*)* C(*, 1) ; Vector multiplication
   (d) Similarly substraction , division , and exponentiation can be preformed
   (e) Array operations are preformed on an element by element basis

Text Handling Features

PL/I contains basic but versatile character string manipulation facilities, including:

- Comparison of character strings.
- A concatenation operator.
- Extracting and setting substrings of character or of bit strings, (SUBSTR).
- Scanning a character or bit string for certain character or bit configurations, (INDEX).
- Converting character strings to bit strings and bit strings to character strings, and assigning bit configurations to character strings, (UNSPEC).

TEXT MANIPULATION

1.	DECLARE	CSTRING CHARACTER (length) VARYING ;
2.	=	String Moving A = B ;
3.	||	Concatenation A = B C ;
4.	=, >, ≥, <, ≤	Comparison If A < B Then
5.	SUBSTR	EXTRACTING Substrings A = SUBSTR (B, n, m) ;
		Setting Substrings SUBSTR (A, n, m,) = B ;
6.	INDEX	PATTERN Matching J = INDEX (STR PSTR) ;
		String Searching
7.	LENGTH	Current Length Inquiry J = LENGTH (STR) ;
8.	UNSPEC	Converting Bit String to Character String UNSPEC (A)='11001010'B ;
		Converting Character String to Bit String BSTR = UNSPEC (A) ;

Example 1

```
SEARCH:   PROCEDURE(STR,INCL);
          /* PROCEDURE TO SEARCH FOR SOMETHING INCLOSED IN PARENTHESIS
             IN THE CHARACTER STRING, STR, AND EXTRACT IT AND PLACE
             IT IN THE CHARACTER STRING INCL */
          DECLARE STR CHARACTER(*),
                  INCL CHARACTER(*);
          J = INDEX(STR,'(' );  /* SEARCH FOR LEFT PARENTHESIS */
          K = INDEX( SUBSTR( STR,J ), ')' ); /* SEARCH FOR RIGHT PARENTHESIS
                                                AFTER THE LEFT */
          INCL = SUBSTR( STR, J+1, K-2 );  /*EXTRACT THE SUBSTRING*/
          END SEARCH;
```

File Handling Capabilities

Most of PL/I file handling capabilities come with RECORD I/O or a combination of RECORD I/O and internal file capabilities of stream I/O. PL/I implemented under OS/360 provides the ability to use all access methods available under the operating system which includes support of both sequential and directly accessed devices and is able to use variable length record input/output as well. Sequential and direct access methods, as well as all supported peripheral devices can read and write variable length records.

Sequential Access

Sequential access may be done with either blocked or unblocked records and may be buffered or unbuffered. PL/I supports (Index Sequential Access Method) ISAM, which provides a complete filing system. Under ISAM, records are identified by alpha-numeric keys of user-defined length, and indexes to all records in the file are automatically kept. The file may be accessed directly by these keys or sequentially starting from any point in the file. Records may be added or deleted and updated in either sequential or direct modes of processing. Records are placed in the file corresponding to the collating sequence of their key. Under direct access methods, there are three principle methods of referring to records in the direct access file:

1) the relative record number;

2) the relative record number plus a key which may be alpha-numeric;

3) relative track number plus a key which may be alpha-numeric;

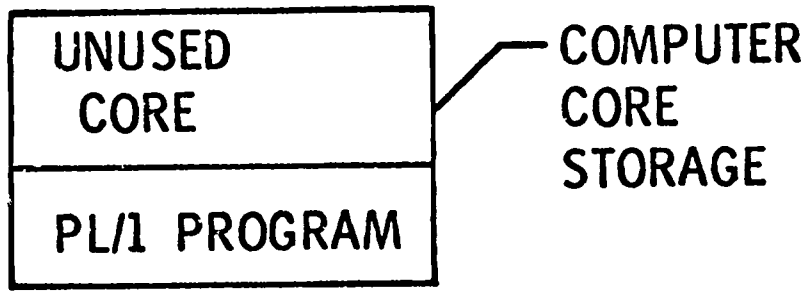
In 2 & 3 duplicate keys may be added to a file by specifying that the record be placed in the first available space after the relative record number if a record is already written in that spot. Similarly extended search option is also available for the so-placed records. Updating of files is available in a direct mode of access only.

Storage Control

Computer storage for any data variable in a PL/I program may be assigned statically for the entire execution of the program or dynamically during execution. Dynamic storage allocation within the program permits more efficient use of variable size data areas. Two classes of automatic storage are available in PL/I: automatic and controlled. When the variable has the controlled storage attributes, the programmer may allocate or free storage for that variable at any time. Storage for the variable with automatic storage attribute is allocated upon entry to a block and freed upon exit.

DYNAMIC STORAGE ALLOCATION

1. Programmer controlled allocation of storage for variables.




```
DECLARE    DICTIONARY    data attributes    CONTROLLED ,
           FOUND-LIST    data attributes    CONTROLLED ,
ALLOCATE   DICTIONARY ;
           :
           : process with dictionary
           :
FREE       DICTIONARY ;
ALLOCATE   FOUND-LIST ;
           :
           : process with found-list
           :
FREE       FOUND-LIST ;
```

Interrupt Handling

PL/I handles interrupts generated by system or user generated conditions such as data conversion errors, file conditions, page conditions, I/O errors, etc. The interrupt handling routine gives the programmer the opportunity to either correct the condition that caused the interrupt, or to do processing prior to closing of the program. It also gives him the ability to supercede system defined action for such cases.

List Processing

PL/I provides in its definition primitive facilities for list processing, although they have not been implemented yet. They are said to be similar to the facilities offered by L[6].

Asynchronous Operations and Tasks

PL/I allows tasks to be created by the programmer and provides facilities for synchronizing, testing for completion, and assigning priorities. By using the asynchronous operations which are to be implemented in Release III of the PL/I compiler, programmers can use computer facilities which can operate simultaneously (such as input/output channels and multiple central processing units). Programs

Example 3

Example of overlapping processing with input/output activities.

```
A:      PROCEDURE;
        DECLARE  ONE  EVENT,    /* ONE AND TWO ARE DECLARED AS */
                 TWO  EVENT;    /* EVENT VARIABLES */

                 .
                 .
                 .
        READ FILE (FILE#1) INTO (MAINREC) KEY (KEY#1) EVENT(ONE);
        READ FILE (FILE#2) INTO (SUBREC)  KEY (KEY#2) EVENT(TWO);
        /* DO PROCESSING DURING THE INPUT TIME*/

                 .
                 .
                 .
        WAIT(ONE);       /* WAIT FOR COMPLETION OF THE FIRST READ */
        WAIT(TWO);       /* WAIT FOR COMPLETION OF THE SECOND READ */

                 .
                 .
                 .
        END A;
```

may be written in which input/output units initiate or complete transmission at unpredictable times such as those found in disc operations and terminal operations and which effectively overlap these operations.

Compile Time Facility

The compile time facilities are a macro-language facility that can be used to perform several functions. These include:

1) modification of a source program to change variable names;
2) inclusion of strings of text into the source program where the strings of text reside in the user or system library;
3) conditional compilation of sections of the source program;
4) generation of in-line code.

Limitations

The current implementation of PL/I by IBM contains certain overheads in terms of speed and total program size. The large size of PL/I programs is almost directly due to the extensive use of library

may be written in which input/output units initiate or complete transmission at unpredictable times such as those found in disc operations and terminal operations and which effectively overlap these operations.

Compile Time Facility

The compile time facilities are a macro-language facility that can be used to perform several functions. These include:

1) modification of a source program to change variable names;
2) inclusion of strings of text into the source program where the strings of text reside in the user or system library;
3) conditional compilation of sections of the source program;
4) generation of in-line code.

Limitations

The current implementation of PL/I by IBM contains certain overheads in terms of speed and total program size. The large size of PL/I programs is almost directly due to the extensive use of library

modules by the program such as those for error-monitoring, type conversion, file opening and closing and many others. Speed of execution is greatly affected by techniques used to accomplish a given job, or by basic construction of the program (e.g., extensive use of procedures or of control storage). A list of pitfalls for the programmer to avoid which adversely affect the speed of execution of a program can be found in Attachment 1 and also in the PL/I Programmer's Guide IBM Form C28-6594-1. Release 3 of PL/I from IBM has been developed to reduce overhead connected with many of these areas. It may also offer the asynchronous and list processing capabilities and is to be released in the 3rd quarter of 1967.

FORTRAN

Some helpful reading material:

1. G. Salton, "Data Manipulation and Programming Problems in Automatic Information Retrieval." PP 204-210, V9, N3, March 1966, Communications of the ACM.

2. Charles T. Meadow "The Analysis of Information Systems." John Wiley and Sons, Inc. New York, 1967.

3. Charles Philip Lecht "The Programmer's FORTRAN II and IV.: McGraw-Hill Book Company. New York, 1966.

4. J. Weizenbaum "Symmetric List Processor" PP 524-544, V6, No. 9, September 1963, Communications of the ACM.

5. R. M. Lee "A Short Course in FORTRAN IV Programming" McGraw-Hill Book Company, New York, 1967 (This is one of many FORTRAN primers available).

ANNOTATED BIBLIOGRAPHY

Desautels, E.J., and Douglas K. Smith. "An Introduction to the String Processing Language SNOBOL." in Rosen, Saul, ed. Programming Systems and Languages. New York, McGraw-Hill, 1967. (McGraw-Hill Computer Science Series) pp. 419-454,

A complete general manual for SNOBOL 3. The Rosen book in which it appears is quite useful for the comparison of languages and for much incidental information.

Farber, D.J., R.E. Griswold, and I.P. Polonsky. "SNOBOL, A String Manipulation Language." Journal of the Association for Computing Machinery, vol. 11, no. 2, January, 1964, pp. 21-30.

Covers the initial version of the language, SNOBOL, and includes a discussion of the desirable aspects of a language for string manipulation.

Farber, D.J., R.E. Griswold, and I.P. Polonsky. "SNOBOL 3 Programming Language." Bell System Technical Journal, vol. SLV, no. 6, July-August 1966, pp. 895-944.

Forte, Allen. SNOBOL 3 Primer: an introduction to the Computer Programming Language. Boston, MIT Press, 1967. $3.95 (paper).

Goff, Adelbert G. OS/360 SNOBOL User's Manual. Boston, Brown University, June, 1967 (mimeo).

A variant for OS/360 which has been implemented in two versions at MIT.

Griswold, R.E., J.F. Poage, and I.P. Polonsky. Preliminary Report on the SNOBOL 4 Programming Language. Holmdel, N.J., Bell Telephone Laboratories, November 22, 1967. (S4D4) (offset)

Hsu, Robert, and Laura Gould. A Linguist's Introduction to SNOBOL. Honolulu, Pacific and Asian Linguistics Institute, October, 1966. (mimeo).

Covers only the most basic and important features of SNOBOL 3. Possible users should not be deterred by the fact that the examples are based on linguistic comparison of Proto-Oceanic and Trukese.

Simon, A.H. and D.A. Walters. RCA SNOBOL Programmers Manual. Princeton, N.J., RCA Laboratories, December, 1964.

According to Desautels, this version has some useful extensions of the language not in Bell Laboratories SNOBOL 3.

Wilson, David L. SNOBOL 3, a List Processing Language. IBM Document no. 1.4.024, 1620 General Program Library, November, 11, 1966.

For the IBM 1620. This version is slow and has no provision for programmer defined functions.

BIBLIOGRAPHY

BOOKS: COBOL

COBOL 1961 REVISED SPECIFICATIONS FOR A COMMON BUSINESS-ORIENTED LANGUAGE, Washington D.C., U.S. Government Printing Office, 1961.

General Electric, GE 400 SERIES COBOL LANGUAGE, Phoenix, GE Computer Department, 1965.

General Electric, INTRODUCTION TO INTEGRATED DATA STORE, Phoenix, GE Computer Department, 1965.

IBM, IBM SYSTEM/360 OPERATING SYSTEM COBOL LANGUAGE, New York, IBM Programming Systems Publications, 1967.

IBM, IBM SYSTEM/360 OS COBOL (F) PROGRAMMER'S GUIDE, New York, IBM Programming Systems Publications, 1967.

McCracken, Daniel D., A GUIDE TO PROGRAMMING, New York, John Wiley & Sons, Inc., 1963.

COBOL INFORMATION BULLETIN #9, New York, United States of America Standards Institute, 1967.

PERIODICALS: COBOL

"A Detailed Description of COBOL," Annual Review in Automatic Programming, Volume 2, 1961, p. 197.

"A Critical Discussion of COBOL," Annual Review in Automatic Programming, Volume 2, 1961, p. 293.

"General Views on COBOL," Annual Review in Automatic Programming, Volume 2, 1961, p. 345.

"A Critical Appraisal of COBOL," The Computer Bulletin, Volume 4, 1961.

"Why COBOL," Communications of the Association for Computing Machinery, Volume 5, 1962, p. 236.

BIBLIOGRAPHY - PL/I

(1) Irwin, Larry. Implementing Phrase Structure Productions in PL/I, Comm. ACM 10, 7 (July, 1967) 424.

(2) Lawson, Harold W. Jr. PL/I List Processing. Comm. ACM 10, 6 (June 1967) 358, 367

(3) PL/I: Language Specifications. IBM Corp. C28-6571-4

(4) Mitchell, R. W.; Christensen, Carlos; Myszewski, Mathew; Sampson, Carol. An Informal PL/I Roundtable, Collection One. Massachusetts Computer Associates, Inc. Technical Report CA-6704-0511, April 5, 1967.

(5) Salton, G. Data Manipulation and Programming Problems in Automatic Information Retrieval. Comm. ACM 9, 3 March 1966 204, 210.

(6) Raphael, B. Survey of Computer Languages for Symbolic and Algebraic Manipulations. DDC, AD-649401 (March 1967)

(7) SHARE Advanced Language Committee, Report of the. (March 1, 1964.) Revised Edition.

(8) Special Interest Group of Programming Languages (SIGPLAN) of the Los Angeles Chapter of the Association for Computing Machinery. PL/I Bulletin No. 3.

(9) Weinberg, Gerald M., PL/I Programming Primer. McGraw Hill Book Co., New York, 1966. 278 pages.

(10) Weiss, Eric A., The PL/I Converter. McGraw Hill Book Co., 1966 116 pages.