AUG 31 1968

PROGRESS REPORT
TO
ADVANCED RESEARCH PROJECTS AGENCY

Project:	PROJECT GENIE
Date:	August 16, 1968
Contract Number:	SD-185
Contractor:	University of California, Berkeley
Principle Investigator:	Melvin W. Pirtle
Phone:	(415) 642-7220

2.0 User Aids and Problem-oriented Programming Languages

A number of projects involving the design or implementation of problem-oriented programming languages and languages to otherwise aid the user of the system are underway. Our goals in these endeavors are to make actual use of the languages in subsequent work as well as to apply the best programming practice to subsystems running in our particular kind of system environment.

2.1 Question-Answering System

The HELP system and its related routines--all called QAS, the Question-Answering System--was originally developed in the project about two years ago. It was planned at that time that the system would be used to provide assistance to users of the time-sharing system at times when they would otherwise be forced to refer to system manuals which are always bulky, not well organized, and frequently out of date. The assumption that the user is somewhat familiar with a particular subsystem and its associated terminology restricts the questions which may be asked sufficiently that a relatively simple algorithm can be used to answer questions posed in unrestricted English form.

Unfortunately, the QAS written two years ago could not be fully implemented due to a lack of capability in the time-sharing system. The current TSS has undergone considerable revision since then, and a large-capacity disk file has been added to the system. Thus, the question-answering system developed two years ago has been reimplemented with a number of new features and with an encoding scheme for the data base which permits it to be stored much more compactly.

The systems consists of two programs, one for creating data bases and modifying them, and one for answering questions. A data base consists basically of a collection of messages and one or more lists of key words associated with each message. When the system is asked a question, it extracts all of the key words known to it from the question, making allowance for minor syntactical variations such as plurals and past tenses, and then searches for a message which is associated with the largest possible subset of those key words. This message is then typed out. If several messages are associated with key-word subsets of the same size, they are all typed. If it is not pertinent or if he is not interested, the user may silence the printing of one message and go on to the next.

A message is stored as an array of pointers to a dictionary which contains all the words used in any of the messages. Special characters, numbers, and other peculiarities of the message are encoded directly. A message may incorporate another message as part of it and may append another message to itself. In other words, the routine which types out messages interprets a simple language which includes transfer and subroutine jump commands, as well as direct and indirect references to operands. This scheme permits messages to be stored very compactly, which is important to successful use of this system since the amount of text in a data base can be quite extensive. The association of key-word lists with messages is recorded in a tree structure which can be searched rapidly.

Data bases have been constructed for the user interface to the time-sharing system (i.e., the system calls), for the graphic system, and for the editor. We are now in the process of improving these data bases and creating others. It is intended that every major subsystem will have a HELP command which will permit the user to call on the question-answering system and obtain information about the subsystem in a matter of a few seconds.

2.2 Conversational Algebraic Language

A new version of CAL, a language patterned on JOSS, has been written. The new implementation contains a small number of new language features, of which the most notable are the admission of six character names, random number functions, integer variables, and alphanumeric input-output. The most important features of the new implementation center around increased execution efficiency:

a) In the absence of any declaration, arrays are implemented by a hash table scheme which makes it unnecessary to search a list comparable in length to the number of elements in the array in order to find a particular element. In this scheme a hash table of some fixed size is allocated when the variable is first referenced with subscripts, and a chain is built from each entry of this hash table to hold those elements of the array which hash into that entry.

b) Arrays may be declared by a statement which specifies the number of subscripts and the upper and lower bounds of each subscript. The indicated amount of space is allocated for the array, and references to it thereafter proceed at high speed. An array may be reallocated at any time during execution of the program.

c) Actual machine instructions (or floating point 'pops' when necessary) are compiled for most operations. The major exceptions are: array references, which require a subroutine call to check the legality of the subscripts; function calls; and transfers of control, which require a search for the step to be transferred to. Code for each statement is compiled in a form which permits it to execute anywhere in memory. Relative branch and relative subroutine call and return 'POPs' are used to make this possible. The result, especially when integer variables are used as much as possible, is code which is less efficient than that produced by a good Fortran compiler by a factor of two or three, an enormous improvement over the older implementation, as well as over the interpretive versions of similar languages in use elsewhere.

The one other notable feature of the implementation is that all important tables and storage areas are paged, using routines similar to those originally developed with the 940 LISP system. This permits more or less indefinite expansion of the size of both program and data. The overhead associated with paging is less than 20 percent as long as no drum references are required. Allocation of storage is done in such a way as to keep adjacent statement and adjacent array elements together as much as possible, and to keep the source language text of the statement separate from the object code, since it is not required at run time. Scalar variables and constants are not paged.

A number of irritating points in the old implementation have also been cleaned up: the treatment of function arguments, restarting and continuing the program after an interruption or error, and the ability to type in a complete expression in response to a program request for input.

In the near future an extensive collection of commands for controlling the graphics system will be added. These will have two major purposes:

a) To permit unrestricted access to all facilities of the graphics system other than vector mode from the CAL language.

b) To provide a convenient mechanism for automatic scaling, labeling and plotting of a graph with a minimum of attention from the user.

2.3 SNOBOL4

A reasonably complete implementation of the SNOBOL4 language developed at Bell Laboratories has been completed. The implementation is most notable for providing:

a) Incremental creation and modification of the program, using an editing language which is a subset of QED, our standard text editor.

b) Completely paged storage of both data and program, allowing a total of about a quarter of a million words to be used for storage. All the features of the Bell Laboratories language have been included, with a few minor exceptions such as real numbers.

The entire system including editor, compiler and run-time routines, occupies about 8000 words of re-entrant code.

The system has just been completed and is in the final stages of debugging and trial by users. Since the language itself is extremely easy to use, and since the program construction facilities are familiar to all members of the project because of their similarity with the editor, it is expected that SNOBOL will find extensive applications in a large variety of string processing jobs, as well as being used for more difficult problems.

We are now beginning the design of a new editor which will be closely integrated with SNOBOL, so that users of the editor may conveniently call on the SNOBOL pattern matching facilities for searches and may construct SNOBOL programs to operate on all or part of the file being edited. We are attempting to make this interface as smooth as possible, so that users who are not programmers can still take advantage of at least the pattern matching facility.

PROGRESS REPORT

TO

ADVANCED RESEARCH PROJECTS AGENCY

Project:	PROJECT GENIE
Date:	August 16, 1968
Contract Number:	SD-185
Contractor:	University of California, Berkeley
Principle Investigator:	Melvin W. Pirtle
Phone:	(415) 642-7220