AN IMPLEMENTATION OF IPL-V ON A SMALL COMPUTER

by Ned Chapin, Ph.D
Data Processing Consultant
Menlo Park, Calif.

Abstract

An implementation of IPL-V has been made that can be run on the computer most widely used by schools and colleges. This can facilitate the teaching of heuristic as well as the already available algorithmic oriented programing languages. For this implementation, the seven objectives selected lead to making the eight major choices that shaped this one-pass implementation. These choices mostly reflect decisions common to the implementation of any programing language. But in contrast to common practice, list processes were used in the implementation itself. Comparisons have been made of this implementation and other IPL-V implementations.

Objectives

The programing language IPL-V (Information Processing Language--Vth version) is an heuristic list processor. The language, which is an outgrowth of work done at Carnegie Institute of Technology and the RAND Corporation, has in the past been implemented almost exclusively on large computers such as the IBM-7090. But most schools and colleges which offer courses in programing do not have such large computers available; most have only a small computer.

Schools and colleges having only a small computer are limited therefore to teaching with algorithmic oriented languages, such as FORTRAN, ALGOL, and simple symbolic assemblers. No heuristic oriented languages are available. This limits the experience students of programing can acquire and largely precludes some stimulating and worthwhile problem areas, such as artificial intelligence.

In deciding to undertake an implementation to help fill this lack, objectives were given first attention. First, since the implementation was to be on a small computer, the prime objective selected was to provide as many "cells" as possible (terms with special meaning in IPL-V such as "cells" are discussed briefly in the Appendix to this paper). The experience of IPL-V users even with large computers has been that storage space is typically at a premium. To be able therefore to run even a small IPL-V program on a small computer would require the implementation to provide as many cells as possible.

The second objective selected was to specify a common computer configuration. The most widely used small computer on college campuses at present is the IBM-1620 with 20,000 positions of core storage, card input and output (250 and 125 cards per minute), and indirect addressing, but with no index registers, no printer, and no disk file. This is a variable word length computer, whereas all the IPL-V implementations to date had been made on fixed word length machines. In order to make a widely useable implementation, the implementation would have to be kept within the limits of the commonly found configuration of the small computer.

The third objective selected was to provide upward compatibility. A person who had tried out his IPL-V program on the small computer might wish to run it either in the same or in an expanded version on some larger computer. It was felt that if the user had to make few or no changes in his source language in going to the larger computer, then upward compatibility would be achieved. This would allow, in other words, anything which could be run on the proposed implementation to also be runable on an implementation on a larger computer with the same results, but would not necessarily provide the ability to run on the small computer implementation anything runable on the larger computer implementations.

The fourth objective selected was to assure the absence of any major impairment in the implementation. That is, the implementation should be essentially complete with no major parts of the IPL-V language lacking. It was correctly anticipated that it would not be possible or economic to implement all facets of the language on the small computer, but it was decided to attempt to implement as much of the IPL-V as the configuration would make possible.

The fifth objective selected was to provide a "programer-proof" implementation. In an academic environment, beginning and inexperienced programers are common, and typically make many programing mistakes. To help such programers avoid wasting machine time and their own time, the decision was made to build in protection against and detection of common source language mistakes.

The sixth objective selected was to seek speed of operation, when all else was equal, as a desired feature of the implementation. Speed of operation typically is not a prime consideration when working with a small computer in an academic environment, but where it was possible to choose alternative modes of implementation, the decision was made to favor those modes yielding the faster operating time, such as using a "one-pass" approach.

The seventh and last objective was to develop some experience on whether list processes were any more necessary or convenient in implementing a list processing language than in implementing an algorithmic language, which is also primarily a symbol manipulator, such as COBOL. Since a "one-pass" approach was used in this implementation, the comparison for fairness should be made with some other "one-pass" programs such as the CDC-1604 COBOL compiler.

Nature of the Implementation

The decision to attempt an implementation was made in February 1963 and was registered with the IPL-V secretary at the RAND Corporation. The RAND Corporation generously provided a list of the J routines, an annotated listing of the IBM-7090 implementation, a more extended version of the flow diagram for the interpreter than had appeared in the published manual, and updated copies of the RAND manual describing IPL-V (1). The work was then begun on the implementation.

A few months after this, a group at the Statistical Laboratory of the University of Oregon in Eugene, Oregon, also began an implementation of IPL-V for a larger (40K) and not as widely available configuration of this computer. Then, in the fall of 1963, the Department of Psychiatry at the University of Toronto also began an implementation for the same computer.

The personnel who undertook the here-reported implementation under the writer's general direction and active contribution, consisted of undergraduate and graduate students at San Francisco State College and at the College of San Mateo. With the exception of the author, none were at the start professional programers, none had held programing jobs for pay, and all had been recently introduced to the art of programing. Those who contributed the most actively to the implementation have been Mrs. Margaret R. Buhn, a physicist with an interest in programing; Mr. Stanley Mazor, who is now a programer with Fairchild Instrument Co.; Mr. Michael Roos, a graduate student in English with an interest in programing; and Mr. Larry Selmer, an undergraduate student in engineering; and the author.

All those who worked on the project contributed their efforts on a volunteer basis and received no pay since the effort was not supported financially from any source. The group elected to use the IBM-1620 symbolic programing language and to meet as a group only when there were major issues to be discussed or major items of progress to report.

In their early discussions, the group decided that some basic decisions had to be made to limit and to define the character of the implementation to be made. Among the ideas that were debated and then discarded were making the cells of variable length (this would have precluded upward compatibility), working within the computer in IPL-V source language as closely as possible (this would have been prodigal with storage in this small computer), and changing the philosophy of implementation from that of an interpreter to that of a compiler (this would have altered part of the heuristic utility of the language).

The first major decision made by the group was to attempt a "one-pass" implementation but to break it into two overlays consisting of a loader, and an interpreter. The objective in doing this was to save storage by having in storage at the time of interpretation, the smallest size of interpreter program possible and the largest number of cells possible. In defining these overlays, the group recognized the need for two optional overlays, an editor, and a translator.

The editor overlay was to check for gross errors in the use of the IPL-V source language, such as clerical errors in assigning labels, to analyze the J routines called to check for their implementation status, and to correct some common format errors. The editor overlay was to accept the source IPL-V and produce a corrected IPL-V source deck and a set of diagnostics for the assistance of the programer. The use of the editor does not make the implementation into a "two-pass" system because the editor is not an essential overlay; it can be omitted entirely if the programer is careful in his use of the source language and in observing the operating rules for the implementation. This overlay is input-output bound in speed and uses about 5000 positions of storage.

The loader overlay was to translate IPL-V source language into a machine language form. The loader overlay, which is by far the longest and the most complex of the overlays was also to supply some redundant indicators with the translated material to increase the speed of operation of the following interpreter overlay. These were to flag additionally the P and Q digits of the IPL-V cell (see Figure 1) in order to mark a cell as regional, local, internal, or program; and to insert a T indicator digit (not called for in the source language) to indicate the regional, local, internal, or program status of the symbol in the cell. The loader occupies nearly one-half of the storage available (i. e., about 9700 positions) and for source programs of less than 95 source statements, is input-output bound in speed, although the speed is influenced by the number of regionals.

The interpreter overlay was to control the execution of the IPL-V program. The build-in trace specified in the IPL-V manual was to be also provided in the implementation. The interpreter reads in the J routines to be called, and provides execution of the program. The maximum number of cells on the space available list (H2) possible at the start of execution is 866. Each interpretation cycle requires an average of 18 milliseconds for basic operations, plus the time to execute the J routine or other process called. The shortest of these is less than half a millisecond, the average excluding output routines is about 60 milliseconds, except for the basic J routines which average about 20 milliseconds.

The fourth overlay, the translator, was to convert back from machine language into IPL-V source language. The reason for not incorporating these features into the interpreter was to shorten the interpreter and thus provide for more cells during the interpretation phase. This means that the output during the interpretation overlay and the trace output if any is in machine language, and has to be used as input for the translator overlay if the user desires to obtain an IPL-V source language output. This overlay uses about 5000 positions of storage, and is input-output board in speed. This overlay is not essential because the user can read the machine language output with the aid of the loader-produced symbol table.

The second major decision that the group made was to save storage by separating the IPL-V regions and the actual allocation of storage. The implementations of IPL-V on the large computers permit the IPL-V programer to reserve areas of storage. In the present small computer implementations, the loader overlay loads only those regions to the extent which are actually used, and saves storage by floating the storage allocations actually made. Since the configuration of the small computer includes no auxiliary storage, and since the cells are linked, this can be done with no violence to the IPL-V compatibility.

The third major decision made by the group was to eliminate private termination cells and to alter the traditional character of the IPL-V termination cells. Since space was at a premium and since private termination cells are admitted to be wasteful of storage space, their elimination appeared desirable although it entailed some modification of a few J routines.

A true termination cell in the other implementations is a cell with zeros in it, but this is a prodigal use of storage. To avoid this, a way was found to additionally mark the T digit in the cell. Then any handling of such a marked cell is sensed by the computer as a list termination indication. Then, those J routines which can create or detect private termination cells were altered to return instead the cells to the H2 list and move the terminal indication up the list.

A fourth major decision made by the group was to redefine the IPL-V name of a cell. This was forced upon the group by the variable word length character of the small computer. In this implementation, a cell has several possible addresses; the question is which one is to be selected to serve as the name of the cell. The group decided that it would yield a faster operating interpreter to make the name of the cell act as the address of the link portion of the cell. This decision lead to real economies in certain commonly executed IPL-V operations, such as preserve and restore.

A fifth major decision made by the group was to include redundant data in the cells, in order to buy some faster speed of operation, even at the expense of lengthening them beyond the minimum necessary. This was a decision born mostly from the specific features of the small computer used—for example, a comparison cannot be made on a single character. The redundancy chosen was not used to displace data traditionally incorporated in the IPL-V cells, and therefore involved no loss of compatibility. The nature of this redundancy was noted above in describing the loader overlay (the T digit).

A sixth decision made by the group was to incorporate into the IPL-V cell specific indications of the status of the cell beyond those specified in the IPL-V manual. For example, the implementation provides a notation of the type of cell, be it regional, local, program, data term, etc. These specific indications are again over and beyond those required by the IPL-V manual but they do not violate compatibility, and were noted above in describing the loader overlay (the flag markings).

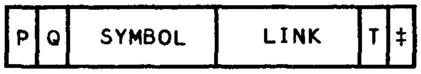
A seventh decision made by the group was to provide for continual changes in the form and length of the J routines. Some of the J routines are themselves written in IPL-V source language. Those not thus written must be implemented by other programing, and those in IPL-V may eventually be also implemented by other programing. Those written in IPL-V operate more slowly but occupy less storage space. As user needs change, some users may wish to change the J routines used to gain either speed or storage. To permit users to exercise this option has required providing for continued variation in the J routines.

An eighth decision made by the group and closely related to the previous one was to make all the J routines automatically relocatable upon loading, and to load only those J routines which actually are to be called by the interpreter. The decision to accept or reject a J routine is made in the editor overlay and finalized in the interpreter overlay. This leads to a considerable saving of storage space.

Results of the Implementation

In the more than a year's time since the implementation was started, a number of changes and reworkings have been made of this implementation of IPL-V. At present, the implementation is running on a field test basis.

Some comparisons may assist in viewing the



P is one digit
Q is one digit
Symbol is five digits
Link is five digits
T is one digit
‡ is one digit
Total is fourteen digits

FIGURE 1. FORMAT OF AN IPL-V CELL

```
FIRST ALTERNATIVE                     1 NAME  PQSYMBL LINK
MOVE KEY AND PLACE TO W0 AND W1         L1    00J51
PUSH DOWN LIST NAMED IN W9                    41W9
MOVE KEY INTO H0                              11W1
INSERT KEY ON LIST NAMED IN W9                21W9
PUSH DOWN LIST NAMED IN W9                    41W9
MOVE PLACE INTO H0                            11W0
INSERT PLACE ON LIST NAMED IN W9              21W9
CLEAN W0 AND W1 LISTS                         00J31    0
                                      1
SECOND ALTERNATIVE                    1 NAME  PQSYMBL LINK
MOVE KEY AND PLACE TO W0 AND W1         L2    00J51
PUT NAME OF LIST INTO H0                      11W9
PUT NAME OF FRONT CELL IN H0                  00J60
GET KEY FROM W1 LIST                          11W1
INSERT KEY ON LIST                            00J63
PUT NAME OF LIST INTO H0                      11W9
PUT NAME OF FRONT CELL IN H0                  00J60
GET PLACE FROM W0 LIST                        11W0
INSERT PLACE ON LIST                          00J63
CLEAN W0 AND W1 LISTS                         00J31    0
                                      1
FIGURE 2. ALTERNATIVE IPL-V SOURCE    1
  VERSIONS OF LOADER UNFOUND          1
  SYMBOL SAVE OPERATIONS              1
```

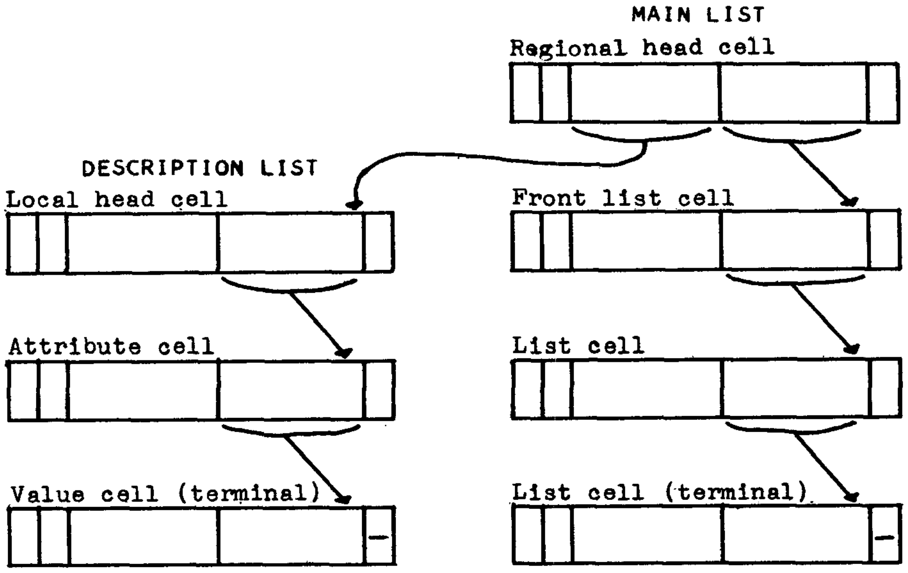


FIGURE 3. FORMAT OF A LIST STRUCTURE IN IPL-V

results of the implementation attempt. The University of Oregon group which began field test of its implementation in November, 1963, makes available a maximum H2 list of 121 cells. This means that IPL-V problems of larger size can be run on the present implementation than on the University of Oregon implementation. In spite of its being more "programer-proof," the present implementation is also faster, for some of the work that must be done in the University of Oregon interpreter phase already has been done in the present implementation in the loader or editor overlays, at the cost of a lengthened run set-up time. Also, because of its translator overlay, the present implementation reduces the amount of work to be performed by the interpreter.

In the present implementation, the IPL-V source language statements are screened twice, once by the editor overlay and a second time by the loader overlay. These editing checks are more extensive than those in the University of Oregon implementation. But the additional phases included in the present implementation involve more card handling than the University of Oregon implementation, and the convenience of IPL-V source language output directly from the interpreter overlay is lacking from the present implementation.

A comparison with the IBM-7090 implementation is also illuminatory. The accuracy of the implementation of each of the J routines in the present implementation has been checked by comparisons of traces of the same routines in the IBM-7090 implementation. Where differences have been detected, the present implementation has been changed to conform to the results obtained with the IBM-7090 implementation. Of the nearly 200 J routines commonly available with the IBM-7090 implementation, not all are included in the present implementation. The J routines omitted are those requiring magnetic tape units, auxiliary storage, or a high speed printer on line.

In the IBM-7090 implementation, the computer apparently executes in the non-trace mode approximately 55 instructions per average interpretation cycle exclusive of J routine execution. In the present implementation, the small computer executes approximately 50 instructions per average interpretation cycle, but this includes some instructions which are required only in the trace mode. These are executed even though the trace mode is not specified in order to make a shorter interpreter at the expense of a longer execution time. This is clearly an operating inefficiency but is consistent with the choice of objectives noted earlier.

A comparison of the present implementation with the COBOL compiler for the CDC-1604 illuminates the use of list processes in symbol manipulating programs. As an implementation aid, list processes were found convenient for use in this implementation only during the loader overlay, and there served primarily as compensation for the lack of a second pass of the IPL-V source statements. The first pass of a two-pass system typically is used to create a table of symbol equivalents. In the loader overlay of the present implementation, when a symbol cannot be found in the symbol table, an entry of the symbol (key) and of the location (place) where its equivalent is needed is made on a push down list (IPL-V source equivalents of such an operation are shown in Figure 2). Then later when the symbol equivalents are known, the list is successively popped up and each equivalent determined from the completed symbol equivalents table and stored where needed. This proved to be faster and more economic of storage than establishing a "missing equivalent" table.

By contrast, the CDC-1604 COBOL compiler, a one-pass system, uses no list processes during compilation. To provide for missing symbol equivalents, a table is set up and stimulated indirect (second level) addressing is used. The closest thing to a list operation is the control pattern used in generating code for nested IF statements, but this is done by a table technique used in a last-in, first-out manner. In the execution of a COBOL object program, the CDC-1604 uses a list-like process for maintaining an inventory of stacked I-O priority requests, and for controlling the execution of a PERFORM. But these operations are actually done as a hybrid between a list process and a table technique.

The present IPL-V implementation is essentially complete for the small computer configuration originally selected. However, many users of these configurations are acquiring disk files. This suggests the desirability of producing another version of the implementation to incorporate auxiliary storage operations and to implement additional J routines that use auxiliary storage. Since most users of the disk file with this small computer have available a monitor (executive) routine, it would be a convenience to be able to call from auxiliary storage, the editor, loader, interpreter, or translator overlays as needed.

Appendix on IPL-V

IPL-V is a list processing language that is usually used in an heuristic manner for seeking solutions to non-mathematical problems. The lists are composed of cells each linked to the next down on the list (see Figure 3). The first cell on each list, the head cell, may be given a regional or local name to identify the list uniquely, and each list is ended by a terminal cell. Because of this one-direction-only linkage, deletion of cells from the end of the list results in the creation of extra (private) termination cells.

Each cell in an IPL-V list may carry the name of some constant (called a data term in IPL-V), or the name of some list, or a J routine, or may carry no name (that is, be empty). Some lists carry the program to be executed by interpretation; others carry the lists of symbols to be operated upon. The non-program lists are usually list structures. Each of these consists of a list of cells and one or more associated description lists. Each description list consists of pairs of cells carrying information about the characteristics of the basic list. The program list cells usually name reserved lists (called H and W lists) to be operated upon, or J routines to be executed, or routines written in IPL-V source language to be performed. The J routines are designed to do such things as to find symbols on list, to insert and delete symbols on lists, to perform arithmetic operations, and to create and erase lists.

Two common IPL-V operations are to restore one list and preserve another. In these operations, conceptually the top cell on the list to be restored is removed from the list and the next cell "popped up" into the top position. The cell thus freed is then inserted as the new top cell on the list to be preserved and the other cells in that list "pushed down." The cell moved has its contents changed to make it a copy of the old top cell on the pushed down list.

References

1. Allen Newell, editor, IPL-V Programmer's Reference Manual Memorandum RM-3739-RC (Santa Monica, Calif.: RAND Corp., June 1963), 131 pp.

2. Allen Newell, editor, Information Processing Language-V Manual (Englewood Cliffs, N. J.: Prentice-Hall Inc., 1961), 244 pp.

3. Allen Newell and Hugh S. Kelly, editors, Information Processing Language-V Manual Second Edition (Englewood Cliffs, N. J.: Prentice-Hall, Inc., 1964), 267 pp.