PRELIMINARY SAMPLES OF THE ALPHA LANGUAGE

By: S. A. Schmitt
J. R. King

Date: September 9, 1959

Company Confidential

IBM Research Laboratory
Yorktown Heights, New York

Alpha is an attempt to provide as convenient a Harvest programming language as possible, within the existing restrictions of delivery time and manpower. It is modest in its objectives. It is to be used by programmers, not by analysts. It is procedure-oriented, not problem-oriented. That is, the programmer must state the method by which the problem is to be solved; Alpha does not do automatic problem analysis. Alpha will not rearrange a problem to use the Harvest stream unit as efficiently as possible, but a programmer can state his procedure in such an order as to cause Alpha to use the stream unit efficiently. Thus the higher-level decisions are left with the programmer. On the other hand, Alpha relieves him of the burdensome parts of programming - translating procedure statements into detailed machine instructions, and allocating storage for these detailed operations.

This report gives a general description of the Alpha language and system. Many of the details are still vague, for they will have to be worked out as the system is written. However, the plan described here is one that the IBM Farm Boy group believes to be within its ability to implement in the time available.

The clearest demonstration of the nature and capabilities of Alpha is afforded by the sample programs submitted along with this report. The report itself should be considered as a supplement to the programs.

The most distinctive characteristic of an Alpha-language program is its resemblance to an English-language description of a procedure. The vocabulary is large enough to allow statements to be made in reasonable-sounding English sentences, and the order of statements is one that might be used in a verbal explanation. In spite of this freedom, however, the language has been chosen so as to be unambiguously interpretable by the compiler.

It is striking how much of each program is devoted to data description and the selection of particular pieces and parts of the data. We believe that this is characteristic of the problems with which Harvest will be confronted, and the Alpha language is therefore aimed toward making data specification as clear and simple as possible. In this respect these recent efforts on Alpha complement the previous work on Transcript, which was aimed primarily at a specification of the operations required. Transcript can then be considered to be a part of the over-all Alpha system, specifying the common operations for which subroutines will have to be written.

Alpha will achieve much of its flexibility and readability by use of a large vocabulary. This will include both words and phrases; an initial pass of the compiler will identify each word and group the words into phrases wherever possible. To avoid possible confusion between words of the language and programmer-assigned names, Alpha will use the expanded alphabet that is to be available on Harvest input-output equipment. All words of the Alpha language will be written in lower case, while all symbolic names chosen by the programmer will be written in upper case.

No matter how large a vocabulary a programming language has, there will always be occasions when it is desirable to add to the language. This can be done in three ways: First, a programmer may, as in any other programming system, write a subroutine and refer to it by a symbolic name. Second, the Alpha language includes terms that make it possible to define new words. That is, a new word may be incorporated into the language by explaining, in Alpha language, how the word is to be interpreted. In its first pass the compiler will use the explanatory information to replace the new word and its context by equivalent statements in standard Alpha language. This is an important feature, not only because it allows any programmer to develop special-purpose extensions of the language but also because it enables the developers and maintainers of the Alpha system to build up the language much more rapidly than would be possible if every new term required the writing of a detailed subroutine. Finally, the third method of extending the language is to make available a system for writing generator subroutines for macro-instructions, as in the IBM 7070 Autocoder system. This is another bootstrapping technique that should simplify the task of the system maintenance group.

An editing problem

We have a file of an unknown number of records of varying unspecified lengths. (Max. no. of records = 100, max. length of a record = 500). The elements of a record are the characters A to Z. It is desired to edit each record so that when it is divided into pairs, no pair will consist of a doubled letter. This may be accomplished by breaking up an offending doublet as it arises and inserting an X if the doublet is ≠XX and a Q otherwise. If an odd letter remains at the end of a record, attach an X (or a Q if the odd letter is X). The file of edited records will later be put through a table conversion process. The original file need not be retained in the machine. The order of records within the file is not important, and there is no correlation in the processing from record to record.

```
1  input
2  FILE :      file 1634 ; max 100 RECORDS
3          RECORD : record
4                  SERIAL :   4d
5                  TEXT : →  ;  max 500 CHARACTERS
6                         CHARACTER : domain = alphabet 1
7
8
```

```
9  input
10 FILE      : file MOSES ; max 100 RECORDS
11         RECORD : record
12                 SERIAL : 4d
13                 TEXT : →; max 500 CHARACTERS
14                        CHARACTER : domain = alphabet 1
15 card : (running time)
           1-4 : 4d = MOSES
16
17
18
19
20
21
22
23
24
25
```

line 1:	'input'	:	a) an explanatory word for readability b) a word signifying that what follows describes data being supplied from the outside
line 2:	'FILE: '	:	a symbolic name for the data down to the next name at the same hierarchical level
	'file n'	:	means "this is the contents of a certain physical file called n on the master list held by the supervisor." it does not include the introductory and final data on the file which describe the file and give checks - only the logical content
	'max n SYMB'	:	a bound for the amount of input for the space allocation routine (when it is written)
line 3:	'RECORD'	:	the first - and here the only - breakdown of the structure FILE
	'record'	:	means it is a logical record. Whether in this file the records are distinguished by EOR marks or counts is a concern of the supervisor but not the programmer
line 4:	'SERIAL'	:	the first breakdown of RECORD
	'4d'	:	4 decimal digits. If this is ever to be treated as a number without other specs it is a 4-digit number.
line 5:	'TEXT'	:	The second breakdown of RECORD
	→	:	means all the remainder until the next symbolic name on the same hierarchical level.
	'max n SYMB'	:	more for the space allocator
line 6:	'CHARACTER'	:	the first (and only) breakdown of TEXT
	domain =	:	these items (CHARACTERS) come from
	alphabet n	:	an alphabet is an established collating sequence. in this case alphabet 1 is A → Z and the domain definition could have been omitted, this one thereby being implied. (This is true of the bottom element of a hierarchy.)

lines 9-16 give an alternative formulation when the program is to be run at various times with the particular file specified at running time

line 10: Here MOSES is a symbolic name for the file number. The compiler is faced with three alternatives:

a) MOSES will be given by an = statement somewhere later in the input statement

b) MOSES will be listed as an input given at running time and the object program will have to arrange to pick it up (this is the alternative shown here)

or c) MOSES will be calculated by an = statement operation later on. In this program since only one file is being read in, the input would be unspecified and the program thrown off the machine.

```
 1  program
 2  "We first edit the texts so that the resulting sequences of pairs contain no
 3  doublets"
 4
 5  take a RECORD
 6  work with TEXT
 7  identify by SERIAL
 8           take two CHARACTERS
                     tree:
 9                   if only one remains then
10                            put it out
11                            if it is not 'X' then put out 'X'; else put out 'Q'
12                            end of processing CHARACTERS
13                   if they are unequal then put them out
14                   if they are equal then
15                            put out first CHARACTER
16                            if it is 'X' then put out 'Q'; else put out 'X'
17                            put back second CHARACTER
18           do this for all CHARACTERS
19  call the result : new TEXT
20  call ( new TEXT, old SERIAL ) : PROCESSED @ RECORD
21  do this for all RECORDS
22
23  " " We now perform a transformation"
24                                  continuation
25
```

line 1: 'program' :

a) an explanatory and for readability

b) a signal that input description is ended

c) a marker to break the hierarchical structure of 'do' loops. This could as easily have been accomplished by the following comment in accordance with the rules of line 23.

line 2-3 " " : a comment with no effect on the program. An X in the right hand column means that the statement continues on the next line. No hyphen should be used to divide a word at the end since the key puncher probably would be forced to divide the line differently, anyhow.

line 4 : spaces may be left for readability

line 5 'take an / the / a / n SYMB' :

this 1) indicates the greatest amount of material that needs to be present for the current processing. When this is the highest hierarchy it gives info about how much need be read into memory at one time for the space allocator.

2) reads in the data from outside if it is not already in the machine

3) sets up indexing to obtain the sequence of SYMBs

4) if > 1 item is taken. sets up implied references to the members as -1, -2, ... and/or as 'first', 'second', ...

There will generally be a hierarchy of takes.

line 6 ' work with SYMB' :

partly for readability, partly to help the compiler to set up indexing for getting this field. [There is some question of just 'taking TEXT'. However, I feel that the more elaborate procedure helps the compiler enough to make it worth while] . TEXT now replaces RECORD as the implied symbol if any is referred to.

line 7 'identify by SERIAL' :

a) specifies a field which is sufficient to identify the item worked with for future reference.

b) sets up a table (implicit, explicit?) between order of processing and identification to the outside.

c) an identification can not be referenced implicitly. Thus TEXT is still the implicit reference.

line 8 'take n SYMB' :

This is subordinate to TEXT so it sets up indexing to secure (the next) two characters. CHARACTER(S) is now the implicit reference and the two can be identified as -1, -2, or first, second. The compiler makes a note that it must have n. If there are none it goes back up to the top of the next higher hierarchical level. If there are some but not enough it will look for instructions how to proceed. If no such instructions are forthcoming it will drop the elements and procede as if there were none. If there are enough it will look for instructions on how to handle them.

line 9 'if' 'only n remain(s)' :
'≤ n remain'
'< n remain'
'partial field'

. . . . then
. . . ; else

'if' introduces a condition extending up to the word 'then'. If the condition is satisfied the operations following 'then' are performed. If the conditions are not satisfied the statements following 'else' are executed. If only one positive action and one negative action are specified the whole thing should be written as one line. If several positive actions are specified, they should be indented under the 'if' while the 'else' is put under the 'if'. If ≥ 3 conditions are involved they are aligned and the appropriate actions indented under each. If there is an 'else' covering others it falls under the 'ifs'.
The last named symbol CHARACTER- is implied here.

line 10	'put SYMB out' pro 'put out SYMB' pro	:	'put out' means form into a result stream. It does not automatically imply writing on tape or printing, tho the writeout might happen automatically if memory became full. The pronoun 'it' refers to CHARACTER.
line 11	'if . . . then. . ;else. . .	:	See line 9. 'it' again refers to CHARACTER This is a subordinate 'if' and is tested only if 'only one remains'.
line 12	'end of processing SYMB'	:	determines at what hierarchical level SYMB is being processed. Looks for ops at that level from this place downward. Sets trigger saying this processing level is finished. Backs up to top of next higher level.
line 13	'they' 'them'	:	both refer to CHARACTER(S). If in the indentations immediately above a new implicit name had been generated we would nevertheless have reverted to an implicit 'CHARACTER(S)' since this 'if' is on the same level as the 'if' on line 9.
line 14	'they'	:	as in line 13
line 15	'first CHARACTER'	:	first refers to the implicit numbering of line 8. Actually here and in line 17 'CHARACTER' could have been omitted. 'first CHARACTER' is now implicit reference
line 16	'it'	:	refers to first character
line 17	'put back SYMB' 'put SYMB back' 'second'	:	back up appropriate indexing so these items will be at the head same as 'first' in line 15
line 18	'do this'	:	indexing for this hierarchical level. repetition starts from the top statement in this level.
	'for'	:	introduces extent of repetition
	'all SYMB'	:	look at the extent of SYMB defined elsewhere and exhaust it in order
(other possibilities)	['first n SYMB']	:	for minimum of (all , n)
	['until']	:	introduces method of stopping
	['[the] result']	:	built-in name for output stream
	['has length n SYMB']	:	control for stopping is on the output. Of course it stops sooner if the input is exhausted.

line 19 'call SYMB:' : a labelling statement
'the result' : see line 18
'new' : has meaning only if SYMB has occurred before. Then it transfers the label from the unprocessed material to the processed. This is logically done item by item. The old material is no longer labelled; exactly when it can be destroyed for new storage space is not altogether clear.

line 20 '(new TEXT, old SERIAL)': this means the adjoining in this order of the two fields. 'new' I believe is unnecessary tho probably mnemonic. 'old' means the unprocessed material or fields associated with it.

PROCESSED ❹ RECORD : I would have liked to call them 'new RECORDS' but I am uncertain yet what would happen in the next statement. A hypothesis is given in line 21.

line 21 'do this' : specifies repetition at the level of RECORD
for all RECORDS unless we specifically say 'new'. The labels in indexing 'do' statements refer to the original labels at the top of the hierarchical level, since for the unprocessed material the old label is the only applicable one.

line 22 " " ... " : A comment with extra quote mark at the front is a marker to say that subsequent processing is logically distinct. If the comment here (or in lines 2-3) is itself to contain a quote, use single quotes inside. A comment may provide a label for the following statement by capitalizing some words in it. The first contiguous set of all-capital words not separated by punctuation except ❹ is the label.
e.g. "STEP ONE : Go Jump❹"

Determining the cycles of a permutation

We are given 100 permutations, each of 35 elements, and wish to calculate the cycle structure. The printout should exhibit the original permutation as well as the cycle structure.

```
1   input
2   FILE : file 1984, 100 RECORDS
3         RECORD : record ; 35 ELEMENTS
4               ELEMENT : domain = 1(1)35
5   program
6   take a RECORD
7   form a table T : 1(1) 35// RECORD . i for i = 1(1)35
8   form a list L : 1 (1)35 ; initially 1(1)35
9   "This is a list of the elements not yet manipulated"
10  form a list CC : lists C
11  "This is a list of the cycles, C"
12        form a list C : 1(1)35 ; where
13              C.1 = L.1
14              C.i = T (C.(i-1)) for i = 2(1)-
15              C.j = T (C.j) = C.1
16        delete all C.i from L
17        do this until L is empty
18  order CC by length of C's
19  print : 11-115 : 1(1) 35 ; format (s d)
20          11-115 : ELEMENTS , format (s d)
21          double space
22          11- : Ci ; format (s d)
23          6-ple space
24  do this for all RECORDS
25  end
```

line 1	'input'	:	marks beginning of description of data entering the program from outside
line 2	'FILE'	:	a symbolic name - here never needed in the program
	'file n'	:	a physical file named and referred to by the supervisor
	'100 RECORDS'	:	for storage allocation
line 3	'RECORD'	:	the subdivision of FILE
	'record'	:	a physical record
	'35 ELEMENTS'	:	for storage allocation
line 4	'ELEMENT'	:	the subdivision of RECORD
	'domain ='	:	points out extent of variable
	'1(1)35'	:	means it can be 1, 2, 3, ... 35
line 5	'program'	:	marks the beginning of processing
line 6	'take $\overset{a}{n}$ SYMB'	:	set up indexing to pick out the sequence of RECORDS; work with $\overset{one}{n}$ at a time
line 7	'form a'	:	create the object mentioned and label it for reference
	'table T:'	:	'table' implies a transformation. Whenever T( ) is encountered it will do this conversion
	' — // — '	:	on the left are the arguments separated by commas; on the right the entries
	'RECORD.i'	:	SYMB. $\overset{i}{\underset{k,}{\underset{l,}{j,m,n,}}}$ is a dummy variable enumerating the subdivision of SYMB.
	'for i = 1(1)35'	:	says the subdivisions of RECORD are taken in 1, 2, . . . , 35 order. Now that i has been enumerated it is released for further use.
line 8	'form a'	:	as in line 7
	'list L:'	:	a list is just that. If items are deleted from it, it closes up.
	'1(1)35 '	:	in this position it gives the type of elements on the list
	'initially'	:	says what is on the list to start with
	'1(1)35'	:	the list L is originally 1, 2, . . . , 35
line 9	" "	:	a comment

line 10	'lists C'	:	the elements of CC are themselves lists called C. (actually to be referred to as C.1, C.2, ...)
line 11	" "	:	a comment
line 12			This is indented because the statements refer to list CC.
	"where "	:	whereas 'for' gives an enumeration, 'where' introduces a combined enumeration-calculation in the next lower level of the hierarchy.
line 13	'C.1 = L.1'	:	the first element on C is the first element on L. [If the domains of C and L had not allowed this an error indication should come up.]
line 14	C.i = T(C.(i-1))	:	gives the rule of calculation. The i^{th} element of C is the transform by table T of the $(i-1)^{st}$.
line 15	'C.j'	:	j means the last element of .[.(j-1) means the next to the last
	'=:'	:	is the element such that
	'T(C.j) = C.1'	:	the transform of the last element of C equals the first element of C.
line 16	'delete...from)'	:	from an ordinary list this means strike out and close up
	'all C.i'	:	all elements of C
	'from L'	:	from list L
line 17	'do this'	:	indication of repetition back to beginning of this hierarchy - in this case, line 12.
	'until'	:	points to condition for stopping
	'is empty'	:	there are no elements
line 18	'order ... by / in'	:	place the elements in the order to be specified if the word 'descending' does not occur in this statement, the order is ascending.
	'length'	:	a built-in operation which results in categorizing each element by the number of subelements. Here the number of elements in each C would be counted. The 'order by length' would put the shortest one first, etc.
line 24		:	establishes repetition for level of RECORDS
line 25	'end'	:	signifies end of complete program