WORKING PAPERS FOR THE LANGUAGE VARIATION
AND LIMITS TO COMMUNICATION PROJECT
Number 1

A USER'S MANUAL FOR PTP--
THE PROGRAMMABLE TEXT PROCESSOR

Gary Simons

Cornell University
and
Summer Institute of Linguistics

1977

A USER'S MANUAL FOR PTP--
THE PROGRAMMABLE TEXT PROCESSOR

Gary Simons

Cornell University
and
Summer Institute of Linguistics

0. INTRODUCTION

PTP, the Programmable Text Processor, is a high-level programming language that was written to fill a need for a specialized software package to do automatic text processing. It was developed in conjunction with the Cornell University research project, "Language Variation and Limits to Communication" (Dr. Joseph E. Grimes, Principal Investigator). The project is being partially supported by grant BNS 76-06031 from the National Science Foundation. The software was written for a portable microcomputer called the Electronic Text Processor, model ETP 8L. The ETP 8L was designed and built by William Hemsath of Cornell University, Department of Psychology. It is an Intel 8080 based 8-bit microcomputer with a 16 kilobyte memory (or 16K, roughly 16,000 characters). It is being used by the author during 1977 in the Solomon Islands to process the data resulting from linguistic field research.

PTP is written in the 8080 machine language. PTP operates in two modes--an immediate mode and a program mode. In the immediate mode, standard text editing commands are executed and the results displayed immediately on the monitor screen. In the program mode, predicates and control structure are added to make a powerful programming language. In the program mode text processing is done automatically according to a predefined program. Programs are interpreted, rather than compiled. All operation of PTP is on-line with the capability for interactive programs.

1. SPECIAL FEATURES OF PTP

The existing text processing software for the ETP 8L was a conventional command string text editor. This was inadequate for a number of reasons. The special features of PTP are discussed in terms of its advantage over the conventional command string editor.

(1) Space - The existing software took up 7K of the available 16K in memory. In writing PTP I always opted for space when there was a choice between saving time or saving space. The resulting code fits in 2K of memory.

(2) Simplicity - I felt that the command string editor could not be used by a beginner without a great deal of instruction and practice. One of the goals I had while writing PTP was to make it as easy as possible for the inexperienced operator to use. Thus the immediate mode editor

with constant buffer display was developed. PTP has proven successful in this respect; that is, people with no computer experience have been able to sit at the console and do successful text editing with only a few minutes of instruction and one written page of command descriptions. It has even been relatively easy for the person with no programming experience to write text processing programs. This is because the text processing program commands are the same commands used in immediate mode text editing. This allows the programmer to single step his program in the immediate mode before writing it or while debugging it.

(3) Program capability - A rudimentary program could be written with the command string editor but the only control structure available was the interative loop and the only predicate was the search function. By adding a whole series of predicates and a control structure which allows for conditional statements, a powerful programming language has been developed. The control structure is such that all PTP programs are "structured" programs and all of the basic program structures outlined by Dijkstra are possible. The expanded set of predicates allows for operations such as table lookup and sorting. A simple numerical capability has been added as well to permit counting frequency of occurrence. PTP thus operates in two modes--an immediate mode and a program mode. Programs are written and edited in the immediate mode using the basic text editing commands. In the immediate mode a program is just another text as far as PTP is concerned. In the program mode, PTP is instructed to take its commands out of a text buffer rather than from the keyboard.

(4) Multiple buffers - The command string editor was too restrictive for programming purposes because it had only one text buffer and an additional save buffer accessible by only two commands. For programming, and even some immediate mode applications such as searching a file to extract a subset of that file, a number of text buffers are needed in which all the commands can be used. For instance, the original data may be in one buffer while a subset of the data is extracted and compiled in another buffer; or, the program may be in one buffer, the data in another buffer, while the results are tabulated in a third buffer and a fourth buffer is used for interactive data entry. Sixteen buffers are available upon initialization of PTP. The sixteen buffers are identified by the numbers 1 through 16. They may be thought of as sixteen string variables, or as sixteen random access locations in memory. All of the commands work with equal effect in all of the buffers; that is, none of the buffers has a special predefined function. The buffers are contiguous and each is of variable length. The only restriction is that the length of all sixteen buffers combined must not

exceed the amount of available memory, in this case 13K. The other 3K of memory are taken up by a 1K ODT program (On-Line Debugging Technique) which is used as a monitor and the 2K PTP program. In addition to the buffer pointer which is used in standard text editors, two pointers--the auxiliary pointer and the save pointer--are maintained in order to facilitate operations between buffers.

2. THE CHARACTER SET

The character generator for the ETP 8L's display module has a 128 character set. This includes the standard 96 character ASCII set with upper and lower case Roman alphabet, the numerals, punctuation, and other common special characters. In addition to these, the ETP 8L can display the 24 lower case letters of the Greek alphabet and eight more special characters. In PTP, the Greek characters are assigned to the control characters on the basis of similarity of pronunciation between the Greek and Roman characters where possible. These characters are generated by holding down the control key and pressing the key for the appropriate letter. The position of all the characters on the keyboard is diagrammed in section 5.3.

By making use of the Greek characters in addition to the upper and lower case Roman characters, a phonetic alphabet has been devised in which every phonetic segment which we might expect to find in our studies of Austronesian languages can be uniquely represented with one character. This eliminates the need for common digraphs such as ng, gh, and th. Being able to represent all phones as a single character is essential if any computations such as counting phone co-occurrences are to be done. The alternative is, of course, complicated programming. The phonetic alphabet being used with PTP is given in two charts on the following page. The phonetic alphabet uses only 53 characters, less than half the number of available characters. Thus if it is necessary to introduce additional phonetic contrasts there are still a number of upper and lower case Roman characters available and many Greek characters. All of the numbers and special characters are available as well.

If the linguist prefers to type and store his phonetic data using digraphs, or if he is working with phonemic data written in an already existing orthography, then digraphs may certainly be used. However, when an analysis such as phone co-occurrences is going to be done, the linguist should precede it by a simple program that will search for all occurrences of each digraph and substitute a unique single character for them.

PTP Phonetic Alphabet for Vowels

		front	central	back
high	open	i	+	u
	close	ɪ		ʊ
mid	open	e	U	o
	close	ɛ		ɔ
low		æ	a	

PTP Phonetic alphabet for Consonants

		bilabial	labio-dental	alveolar	alveo-palatal	velar	glottal
Stops: voiceless							
	unaspirated	p		t	c	k	?
	aspirated	P		T	C	K	
voiced		b		d	j	g	
	prenasalized	B		D	J	G	
Nasals:		m		n	N		
Fricatives:	voiceless	ɸ	f	θ		χ	h
	voiced	β	v	ð		ɣ	
grooved							
	voiceless			s	ʃ		
	voiced			z	ʒ		
Laterals:							
laminal	voiced			l			
fricative	voiceless			ɬ			
	voiced			L			
Retroflexed:	flap			r			
	trill			R			
Semivowels:		w		y	y		

3. THE IMMEDIATE MODE

3.0 General

In the immediate mode, PTP is a standard text editor with the functions of buffer pointer movement, character deletion, and character insertion. Commands for copying material from one location to another, for pointer manipulation, and for cassette input and output are available as well.

In the immediate mode, each command is executed immediately. That is, as soon as a command key is depressed, the command is executed and the display is refreshed to show the resulting buffer contents or pointer position. The sign of the immediate mode is an asterisk in the upper left hand corner of the display. This asterisk indicates that PTP is in the immediate mode and ready to accept the next command.

The display of the ETP 8L has eight lines. The first line contains the asterisk and the characters of the current command. The remaining seven lines display the buffer contents. The first line which is displayed is the one containing the buffer pointer. The remaining lines of the screen are filled with the lines of text following the line which contains the buffer pointer. The buffer pointer always appears on the screen as an up arrow (↑) preceding the character to which the buffer pointer is pointing.

The explanation of the immediate mode commands is organized into six groups: buffer pointer movement, character deletion, character insertion, pointer manipulation, copy and save commands, and cassette input and output. Each command is designated by a three letter mnemonic code. These codes in turn designate a single command key on the keyboard. After the discussion of all the commands, the assignment of codes to keys is given.

The following conventions are used in the following sections. The buffer containing the buffer pointer is called the current buffer. Buffer pointer is abbreviated to simply BP. Auxiliary pointer is abbreviated to Aux pointer. CRLF is used as an abbreviation of carriage return/line feed. CRLF is stored as one character in memory (not two). Thus it is a single character when all the character oriented commands--MFC, MBC, DFC, DBC, CPC, and SVC--are executed. The letter n preceeding a code indicates that the command may have an optional decimal numeric argument. The number must be in the range of 1 to 255. If no argument is given an argument of one is generated. The symbol ⌊_____⌋ is used to represent an argument string of any number or combination of characters from the PTP character set.

3.1 Buffer pointer movement commands

nMFC	Move the buffer pointer (BP) Forward n Characters
nMFL	Move the BP Forward n Lines
nMBC	Move the BP Back n Characters
nMBL	Move the BP Back n Lines

For all four commands a buffer boundary blocks any further movement. That is, a move forward command can never move past the end boundary of the current buffer and a move back command can never move past the beginning boundary of the current buffer. A move command thus causes the buffer pointer to move until it has gone the specified number of characters or lines, or until it reaches a buffer boundary, whichever occurs first.

For the MFL command, whether the BP is at the beginning or middle of a line, the BP is always moved to the beginning of the following line. For the MBL command, if the BP is at the beginning of a line, the BP is moved to the beginning of the previous line. If the BP is in the middle of a line, it is moved to the beginning of the same line.

3.2 Character deletion commands

nDFC	Delete Forward n Characters following the BP
nDFL	Delete Forward n Lines following the BP
nDBC	Delete Back n Characters preceding the BP
nDBL	Delete Back n Lines preceding the BP

For all four commands a buffer boundary blocks any further deletion. That is, a deletion command can never delete a buffer boundary or beyond it. If a buffer boundary is in the scope of a deletion command, all characters up to, but not including, the buffer boundary are deleted and the deletion terminates at the boundary.

End of Tape

For the nDFL command, whether the BP is at the beginning or middle of a line, all the characters from the BP up to and including the nth CRLF following the BP are deleted. For the nDBL command, if the BP is at the beginning of a line, the previous n lines are deleted. If the BP is in the middle of a line, every character from the BP up to but not including the nth CRLF preceding the BP is deleted.

3.3 Character insertion command

INS ␣␣␣␣ ESC INSert the given argument string into the buffer at the BP

When the INS command is given, the asterisk prompt character disappears indicating that PTP is not ready to accept another command. Then, everytime a key is pressed that character is inserted into the buffer at the point where the BP is. The display is refreshed after each character so that the operator can always see exactly what is in the buffer. After a character is inserted, the BP is updated to follow that character. When the wrong character is inserted, the CNCL (cancel) key is used to correct the error. Every time the CNCL key is pressed, the character preceding the BP is deleted. The display is refreshed to show that the character has been deleted and the BP is updated. The correct characters can then be typed and the insert proceeds.

3.4 Pointer manipulation commands

nSBP	Set the Buffer Pointer to the beginning of buffer n
SAX	Set the AuXiliary pointer to the BP
SSV	Set the SaVe pointer to the BP
TRA	TRAde the auxiliary pointer and the buffer pointer
~~TRS~~	~~TRade the Save pointer and the BP~~

In the immediate mode, three different pointers are maintained: the buffer pointer, the auxiliary pointer, and the save pointer. Their uses will be discussed below in the discussion of the copy and save commands and the predicates of the program mode. At this point we simply consider how to set these pointers.

All pointer setting is done by means of the BP. The SBP command sets the BP to the beginning of the desired buffer. If further positioning of the BP is required, the BP movement commands are used. The SAX command sets the auxiliary pointer (Aux) to point to the same location the BP does. After executing the SAX command, both the Aux pointer and the BP point to the same place. The TRA command trades the Aux pointer and the BP; that is, after execution the BP points to where the Aux pointer previously pointed and the Aux pointer points to where the BP previously pointed. These two commands are the only means available for changing the Aux pointer.

The TRS works exactly like the TR/T command, except the save pointer is tracked with the BP rather than the Aux.

The SSV works exactly like the SAX command, except that the save pointer is set instead of the Aux. In addition to using the SSV command, the save pointer can be changed using the SVC and SVL commands (see next section). The save pointer is automatically updated after execution of SVC or SVL.

3.5 Copy and save commands

nCPC	CoPy the n Characters following Aux to the BP (BP updated)
nCPL	CoPy the n Lines following Aux to the BP (BP updated)
nSVC	SaVe the n Characters following BP to the save pointer (Save updated)
nSVL	SaVe the n Lines following BP to the save pointer (Save updated)

For all off these commands there is no restriction as to where the pointers are pointing. They may be in different buffers, the same buffer, the same line, even to the same character. The essential difference between the copy commands and the save commands is that the copy commands copy material from somewhere else and insert it at the BP; the save commands copy material from the BP and insert it somewhere else. In each case the pointer where the material is inserted is updated to follow the inserted material; the other pointer remains unaffected. That is, for the copy commands the Aux pointer is unaffected while the BP is updated to point to the character immediately following what was inserted. For the save commands, the BP is unaffected while the save pointer is updated to point to the character immediately following what was inserted. After material is copied it is not deleted; it remains unaffected.

The usefulness of the SVL command can be illustrated with a sample application. In one buffer, a linguist has a sorted list of vocabulary items. He suspects that reduplicated stems have a restricted distribution of syllable final consonants. To test this hypothesis it is necessary to extract all the reduplicated stems from the list and analyze them separately. With a card file system one would thumb through the cards one by one and pull out the cards for reduplicated stems. In PTP, the save pointer is first set to an empty buffer via the SSV command. Then the BP is set to the beginning of the vocabulary file. By successively pressing the MFL key, the linguist "thumbs" through the file one item at a time. Whenever he sees a reduplicated stem, the SVL key is pressed to copy that stem

into a list being compiled in the save buffer. Since the save pointer is updated whenever a line is saved, the items will be copied in the same sequential order in which they occur in the master file. When the linguist has gone through the entire vocabulary file, he simply sets the BP to the buffer in which the reduplicated stems were saved and finds a complete sorted list. The original data is still intact and there is no need to refile cards that have been pulled out.

3.6 Cassette input and output

cv — cassette view--display the contents of the tape

cr — cassette read--read a file from the tape and insert into buffer

cw — cassette write--write the contents of a buffer onto tape

The ETP 8L uses a conventional cassette tape recorder with standard audio cassette tapes for long term storage of data files and PTP programs. The digital data is encoded on tape in a tone/no tone scheme. Files are written and read at the rate of thirty characters per second.

The cassette view command is used to position the tape. When the view command is given, every character that is read from the tape is echoed on the display screen. The buffers and pointers are not affected. In the view mode, the tape drive may be started, stopped, rewound, or fast forwarded at will. When the tape is finally positioned, pressing the ESC key returns PTP to the immediate mode. The contents of the current buffer are displayed and the asterisk prompt character appears.

The cassette write command is used to save a data or program file on cassette tape. The data is written out in blocks of 256 characters. The data is written on tape beginning with the character following the BP. Thus before the cw command is given, the BP should be positioned to the beginning of the data to be written on tape. The write routine recognizes only two special characters--a form feed and the end of buffer character. A form feed is inserted into the text buffer by pressing the 'y' key while holding down the control key. It prints on the display as a square root sign (√). When the write routine encounters the form feed it fills the remainder of the current block on the tape with ASCII character 8, which prints as an iota (ι). The next block then begins with the character

following the form feed. The form feed is used to mark the end of a header block. The first block of a data file is generally a header block which identifies the contents of the file, the date it was generated, and any other pertinent information. By using the form feed to delimit the header block, one ensures that the data will begin a new block and not begin in the middle of a block. The write routine continues writing out the data in blocks of 256 characters until the end of buffer character is detected. At this point, the block being written is filled with ASCII character 26 (→) until 256 characters have been written. When the final block has been written on tape, PTP returns to the immediate mode and is ready to accept a new command. As a file is written out, the display is refreshed each time a new block is begun so that the operator may monitor the progress of the output. The display is refreshed such that the first character of the block currently being written is the first character of the display. The BP is not affected by the command. When the cw command has been fully executed the BP is still in the same place as when the output was begun.

The cassette read command is used to read the data from a file on tape and insert it into the text buffer. The data is inserted beginning at the location pointed to by the BP. Thus before the command is given, the BP should be positioned to the location where the data is to be inserted. The BP is updated after every character is read in, thus when the file has been completely read in the BP is located at the very end of the file. Each block begins with a sequence of start characters. When the cr command is given, the tape is scanned until the start characters are detected, then the data begins to be inserted into the buffer. Thus the cr command may be given while the tape is running in the middle of the header block. The first character of the next block, the first character of the actual data, will be the first to be inserted into the buffer. During the read routine, every character which is read from the tape is echoed on the screen. This allows the operator to monitor the progress of the reading operation. As each block is read in, a checksum is calculated to determine if the block has been read correctly. If there is a reading error, a sequence of eight question marks is inserted into the text buffer following the last character of the block. These question marks also appear on the display screen. If a reading error has occurred, the first thing the operator does when the read terminates is to search for the eight question marks in the text buffer. This can be done by executing ps????????$% in the program mode. Then the 256 characters immediately preceding can be examined to find and correct the incorrect character or characters. The reading operation may be aborted at any time by pressing the ESC key. PTP will immediately return to the immediate mode and everything inserted so far remains in the buffer.

3.7 Assignment of commands to keys

In addition to a standard ASCII keyboard, the ETP 8L has a block of sixteen keys to the right of the main keyboard. The majority of the immediate mode commands that have been discussed are assigned to these keys as diagrammed in section 5.3. Note that all the move commands are in the upper left corner and all the delete commands are in corresponding positions in the upper right corner. Three of the commands are assigned to keys on the regular keyboard: TRA to t, CPC to h, and SVC to j. The assignment of these commands is also diagrammed in section 5.3. The cassette commands are typed as a sequence of the two characters cv, cw, or cr.

4. PROGRAM MODE

4.0 General

In the program mode the PTP does its text processing completely under its own control. All of the actual text processing commands are the same commands as described under the immediate mode. To these processing commands are added predicates, control structure, program control commands, and numerical operations. The predicates allow for the execution of a string of commands or the transfer of control on the basis of the outcome of testing a condition. The control structure directs the flow of execution within the program. The control structure is strictly block structured and all resulting programs are structured programs. The program control commands have to do with commencing and terminating program execution and communication between the program and the operator. The numerical operations allow for the counting of frequency of occurrence and for displaying decimal numbers. After these program mode commands are discussed, the insert command in the program mode and the assignment of new commands to keys are discussed.

A PTP program is nothing more than a text. A program is entered using the INS command of the immediate mode. When a command key is pressed, it generates a single lower case character which is inserted into the text buffer (see section 5.2). Once it is in the buffer, the program can be edited using the text editing commands in the immediate mode. When the program is finally edited into its correct form, the command to execute the program is given. Only then does the text really become a program. The immediate mode has another very important use in program development and debugging. Since all the basic text processing commands are also immediate mode commands, a program can be single stepped in the immediate mode by issuing

the commands one by one in the order in which they would be executed in the program mode. The operator must simulate all the predicates by asking himself if thay are true or false and simulate the control stuucture by transferring control backwards or forwards in the program as indicated. In this way a program can be tested to see if it will work properly, or it can be debugged to find the point at which the program logic fails.

4.1 Predicates

All predicates are represented by a sequence of two lower case characters, p followed by a character which uniquely identifies the particular predicate. In the following descriptions two conventions are used. An n preceding the p indicates that the predicate can have a numerical argument in the range of 1 to 255. If no argument is given, 1 is assumed. The symbol ⌊_____⌋ is used to represent a string argument of any length. The string may contain any characters except the following: dollar sign, semicolon, or a use of unbalanced parentheses.

pa — Does the aux pointer equal the buffer pointer? That is, do they point to the same location? The pointers are not affected.

npe — Are the n characters following the aux pointer equal to the n characters following the BP? That is, are the n characters at the two locations identical? The pointers are not affected.

npf — Are the n lines following the aux pointer equal to the n lines following the BP? That is, are the n lines identical character by character?

pi⌊_____⌋$ — is the string following the BP equal to the argument string? The dollar sign character closes the argument string. The number of characters in the argument string is the number of characters following the BP that are compared. All characters compared must be identical. The pointers are not affected.

pl Is the line following the aux pointer lexically greater than the line following the BP? The standard ASCII collating sequence is used. The Greek characters precede all other characters. They are ranked in the alphabetical order of the Greek alphabet. The pointers are not affected.

pm Is there more left in the current buffer? Returns false only when BP is at the end buffer boundary. The pointers are not affected.

nps,___$ search for the nth occurrence of the argument string. The dollar sign character closes the argument string. This predicate returns true if n occurrences of the argument string are found. The search begins at the BP. If true, the BP is positioned at the end of the nth occurrence of the string. If false, the BP is positioned at the end of the buffer.

4.2 Control structure

The control sturcture is that used by MacIntosh in his Regular Expression Compiler. The symbols are only five: left and right parentheses, colon, semicolon, and question mark. The parentheses define procedures. The question mark follows a predicate and sets up a condition. The semicolon separates the then-clause from the else-clause. The colon is used to direct control back to the beginning of a procedure and thus sets up a loop.

(Enter a procedure

) Leave a procedure

: Jump to the beginning of the current procedure and execute again

; Jump to the end of the current procedure and fall through to the character following the right parenthesis

? If predicate is true, fall through to the character following the question mark. If predicate is false, jump to the character following the next semicolon in the current procedure. Semicolons in nested procedures are disregarded.

4.3 Program control commands

nEXC	EXeCute the program located in buffer n
OEXC	Argument zero indicates execute program which begins at the BP
IMM	Call the IMMediate mode processor from a program
#	Terminate the current EXC or IMM command
MON	MONitor the current buffer
"	Balanced double quotes delimit a comment

The EXC command is given in the immediate mode to enter the program mode. It is given in the program mode to begin execution of a subroutine. When an argument of 1 through 16 is given, the program counter is set to the beginning of that buffer and execution in the program mode begins. If an argument of 0 (zero) is given, the program counter is set equal to the current position of the BP and execution in the program mode begins. This feature is useful when a whole library of programs is stored in one buffer. Using the MFL command the operator can search for the program he wants to execute and then position the BP at its beginning. The OEXC command will execute it.

The IMM command offers the capability for an interactive program with data entered when the program requests it. It can also be used in an error routine so that the operator can investigate the cause of an error in execution and even edit the data in order to recover from the error. (This can be done only if the data buffer follows the program buffer.) It is even possible to change the program while it is being executed, as long as all changes follow the IMM command or do not change the number of characters in the program. The # command given in immediate mode will return control to the program with the character following the IMM command.

The double cross must appear at the end of all programs. It is the command to pop the program counter stack. If one is in program mode with no nesting of subroutines, this results in a return to immediate mode. If a subroutine program is being executed through an EXC command within a program, control returns to the calling program with the character following the EXC command. If one is operating in the immediate mode through an IMM command executed under program control, control is returned to the program at the character following the IMM command.

Execution of the MON command causes the current buffer within the environment of the BP to be displayed on the screen. In the program mode, INS is the only command which automatically gives a display. Thus MON is used to force a display so that the operator can monitor the progress of the program's execution.

The double quote is used to delimit a comment. When a double quote is encountered during program execution, the program is scanned until the next double quote is found. All characters in between are ignored and execution continues with the character following the second double quote. The comment is especially useful for labelling the separate programs and tables in a program buffer.

4.4 Numerical operations

INC	INCrement the byte at the BP
NUM	Convert the byte at BP to a three digit decimal NUMber

The INC command gives PTP the capability of counting the number of occurrences of a specified thing. Using INC one can count from 0 to 250. When the count reaches 250, any additional INC commands have no effect. A buffer containing just one character can be used as a numerical variable. nSBP addresses it and INC increments it. nMFC can be used to address a one-dimensional array. The predicates pi, pe, pf, and pl can be used to locate entries in a table, where each entry is followed by a number of occurrences. The same predicates can be used to compare number values. A number is initialized to zero by performing INS∝$. Alpha is ASCII character 000. A number is initialized to one by performing INS β$. Beta is ASCII character 001.

The NUM command converts a one byte character which has been used as a number into a three digit number for the purpose of display. Before executing NUM, two spaces must be inserted preceding the character to be converted. This is to make room for the other two digits of the three digit number. After the execution of NUM, the BP points to the first digit of the number.

4.5 The insert command in the program mode

INS_____$ INSert the given argument string into the buffer at the BP in the program mode

In the immediate mode the argument string for an insert was terminated with the ESC key. In the program mode it is terminated with the dollar sign. In this way the terminating sign can be entered in the text buffer.

4.6 Key assignments for new commands

The predicates are all entered as two lower case characters, a p followed by the other letter of the predicate, as shown in section 4.1. The EXC command is located in the block of sixteen keys. IMM is assigned to the i key, MON to the m key, and NUM to the n key. INC is assigned to the caret (^) key. See section 5.3 for the location of these keys.

5. REFERENCE

5.1 List of all commands and special characters

nk indirect addressing - the n digits following the Sem pointer are read as the argument

n represents that the command may have a numeric argument. Maximum value is 255. Negative numbers not allowed. For move, delete, and [illegible] commands, a zero argument is [illegible] for others, one is the minimum value.

⌊___⌋ represents an argument string of any length

BP abbreviation for Buffer Pointer

nCPC CoPy the n Characters following Aux to the BP (BP updated)

nCPL CoPy the n Lines following Aux to the BP (BP updated)

nDBC Delete Back n Characters preceding BP

nDBL Delete Back n Lines preceding BP

nDFC Delete Forward n Characters following BP

nDFL Delete Forward n Lines following BP

nEXC EXeCute the program located in buffer n; if n equals zero,

~~0EXC~~ ~~argument zero~~ indicates execute the program which begins at the BP

IMM call the IMMediate mode processor from a program

nINC INCrement the byte at the BP. number of which BP points to the least significant digit

INS⌊___⌋ESC INSert the given argument string into the buffer at the BP in the immediate mode

INS⌊___⌋$ INSert the given argument string into the buffer at the BP in the program mode

nMBC Move the BP Back n Characters

nMBL Move the BP Back n Lines

nMFC Move the BP Forward n Characters

nMFL Move the BP Forward n Lines

MON MONitor the current buffer

~~NUM~~ ~~Convert~~ the byte at BP to a three digit decimal NUMber

SAX Set the AuXiliary pointer to the BP

nSBP	Set the Buffer Pointer to the beginning of buffer n
SSV	Set the SaVe pointer to the BP
nSVC	SaVe n Characters following BP to the save pointer (save updated)
nSVL	SaVe n Lines following BP to save pointer (save updated)
TRA	TRAde the aux pointer and the BP
cr	cassette read
cv	cassette view
cw	cassette write
pa	Does the aux pointer equal the BP?
npe	Are the n characters following aux pointer equal to the n characters following the BP?
npf	Are the n lines following the aux pointer equal to the n lines following the BP?
pi___$	is the string following BP equal to the argument string?
pl	Is the line following the aux pointer lexically greater than the line following the BP?
pm	Is there more left in the current buffer?
nps___$	search for the nth occurrence of the argument string
(	enter a procedure
)	leave a procedure
:	jump to beginning of current procedure and execute again
;	jump to end of current procedure, fall through right parentheses
?	on true, fall through; on false, jump to semicolon
#	terminate the current EXC or IMM command
"	balanced double quotes delimit a comment
√	form feed for cassette write--fill to end of block with iota

5.2 The three letter mnemonic commands in the alphabetical order of their print characters

Each of the command keys generates a single lower case letter when pressed. Thus, for example, the actual code for the sort program given in section 6.1.1 takes only 34 characters of memory space:

```
"sort 2" (2b(o2b(pa?e;pl?e:;vtg)mpm?:;)2b)#
```

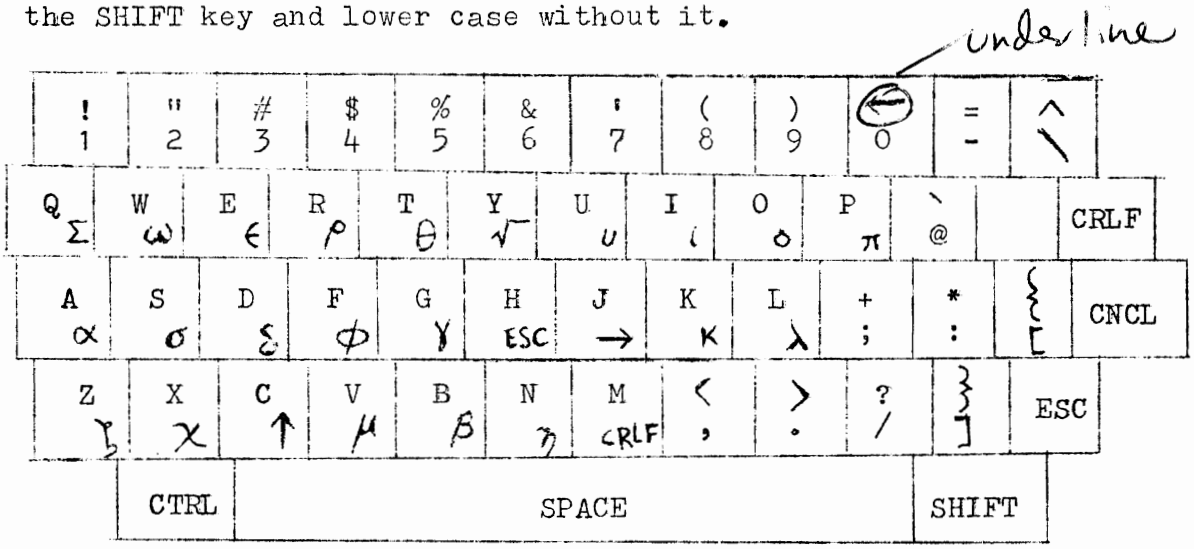
The commands are here listed in the alphabetical order of their print characters in order to facilitate the translation of a program in the text buffer back to mnemonic codes.

a	SSV	o	SAX
b	SBP	p	predicates
c	cassette commands	q	SVL
d	MFC	r	MBL
e	MFL	s	MBC
f	DFC	t	TRA
g	DFL	u	~~(unused)~~
h	CPC	v	CPL
i	IMM	w	INS
j	SVC	x	EXC
k	~~(unused)~~ indirect addressing of argument	y	DBC
l	(unused)	z	DBL
m	MON	^	INC
n	~~NUM~~ (unused)		

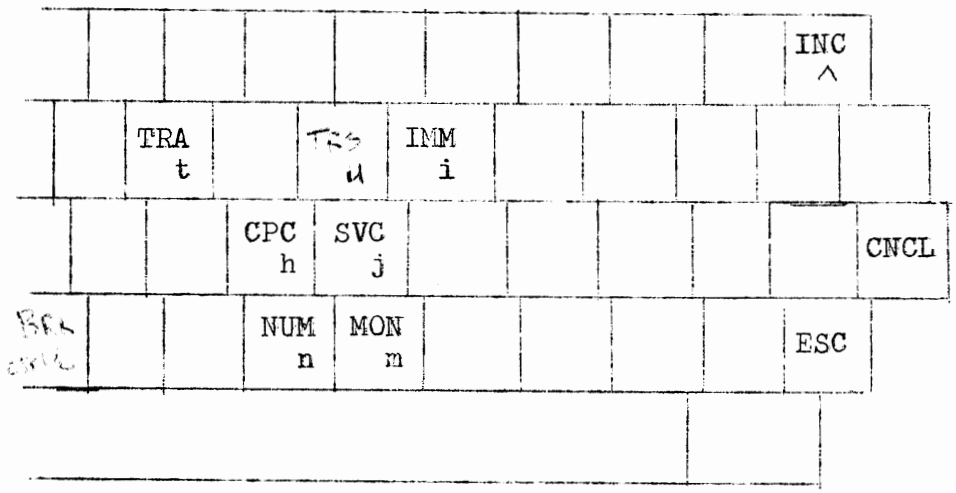
5.3 The keyboard--character set and command assignments

PTP CHARACTER SET

For the alphabetic keys, the character in the lower right hand corner is the character printed when the CTRL key is held down at the same time. For all the other keys, the lower character is the unshifted one, and the upper character is the one obtained when the SHIFT key is held down at the same time. For the alphabetic keys, upper case characters are obtained with the SHIFT key and lower case without it.



PTP COMMAND ASSIGNMENTS



MFC	MFL	DFC	DFL
MBC	MBL	DBC	DBL
SVL	CPL	SSV	SAX
INS	EXC	SBP	

6. PROGRAMMING EXAMPLES

6.0 Introduction

As an example of PTP programming, two packages of programs which have been used for processing linguistic data are here reproduced and explained. The first package of programs is used for vocabulary file maintainence, the second for phonemic analysis. In addition to the programs given here, program packages have been written which count correspondence sets for the comparative method, count the occurrences of phonemes and words in free text for the sake of developing literacy materials, and which extract and merge standard word lists for the sake of dialect and language comparison. Work has also begun on a phrase structure grammar parser.

The program descriptions which follow are organized as follows. First, there is a paragraph which gives the intent of the program, describes the input and output of the program, describes how the program works, or suggests how its results can be applied. Second, the program listing is given. The spaces and CRLF's in the program listings are strictly for the purpose of making them easier to read. In a text buffer programs are stored in compact format with no spaces between commands or CRLF's and indentations. Where a space is actually part of the program, that is, it is a character of an argument string for INS or a predicate, it is represented as an underline (_). Where a CRLF occurs in an argument string, it is represented by a cent sign (¢). Finally, a structured program description is given if the first paragraph does not give adequate description and if the procedures in the program are not similar to routines described in detail in an earlier program. In the structured descriptions, the PTP control structure is copied from the program listing and filled in with prose statements which give the intent of the commands and predicates. It is assumed that all programs reside in buffer one. Each program title tells the buffers which the program uses.

6.1 Vocabulary file maintainence package

6.1.1 Vocabulary file inversion

This package of programs is concerned with maintaining a a vocabulary file. A vocabulary file consists of a list of vernacular words along with their glosses, or meanings, in the language of the investigator. The following standard format is used for the entries. Each entry ends with a CRLF:

$$\text{word:gloss}_1/\text{gloss}_2/\ \ldots\ /\text{gloss}_n$$

Inverting the file means reversing it into the order of gloss first, vernacular word second. This is done in two steps. The first procedure in the program reverses the order of the word and its glosses:

$$gloss_1/gloss_2/ \ . \ . \ . \ /gloss_n:word$$

The second procedure makes a separate entry for each gloss:

$$gloss_1:word$$
$$gloss_2:word$$
$$. \ . \ .$$
$$gloss_n:word$$

```
"invert vocabulary file in 2"
     (2SBP (SAX ps:$? DBC INS¢$ ps¢$ DBC INS:$ CPL TRA DFL MFL:;)
      2SBP (ps/$? DBC SSV ps:$ MBC SVL MBL MON:;)
      2SBP)#
```

6.1.2 Sort

This sort program is used to sort a vocabulary file into alphabetical order of the vernacular words, or to sort an inverted file into alphabetical order of the glosses. This sort program is an insertion sort. At any time the first part of the file is in sorted order and the remainder is unsorted. The first item in the unsorted section is always the next item to be sorted into order. It is inserted into the sorted section in its proper place and then deleted from the unsorted section. The sorted section thus grows one item at a time as the unsorted section diminishes. Finally all items are sorted.

```
"sort file in 2"
     (2SBP (SAX 2SBP
               (pa? MFL; pl? MFL:; CPL TRA DFL)
           MON pm?:;)
      2SBP)#
```

```
(Set BP to beginning of file.
      (Set Aux to first item in unsorted section.
       Set BP to beginning of sorted section.
         (Has BP reached first item of the unsorted section?
          Yes. The first item in unsorted section is at its
            proper place at the end of the sorted section.
            Move BP forward a line and leave item there;
          No. Is item being sorted greater than entry at BP?
            Yes. Move BP forward to next entry:;
            No. Proper place has been found.  Copy item to
              location at BP. Delete it from unsorted section)
       Display next item to be sorted. More in buffer?:;))
```

6.1.3 Merge sort

The merge sort is used to merge a sorted list of additional vocabulary items in buffer 3 into the sorted main vocabulary file in buffer 2.

```
"merge sort 3 into 2"
     (3SBP SAX 2SBP
        (MON pl? MFL:; CPL TRA DFL pm? TRA:;)
      2SBP)#


     (Initialize Aux and BP.
        (Display current position in the main file.
         Is entry at Aux in 3 greater than the one at BP in 2?*
           Yes. Move to the next entry in buffer 2:;
           No. Proper place for item in 3 has been found.  Copy
             it into 2 at BP.  Delete item in 3. More in 3?:;))
```

*N.B. This predicate will not result in an infinite loop when the BP is at end of buffer 2 and more items remain in buffer 3. The buffer boundary character is lexically greater than any character in the PTP character set. Thus the predicate will always return false in this situation.

6.1.4 Interactive entry and filing of new vocabulary items

This program is used to add new vocabulary items into a sorted vocabulary file. The main file is in buffer 2 and new items are entered through buffer 3. When ready to receive a new item a prompt message is printed in buffer three and the immediate mode processor is called. After inserting the new entry into buffer 3, the operator types # to return to the program. If he enters "done" the program terminates; otherwise, the new entry is inserted into the file and the process is repeated.

```
"enter new items interactively from 3 to 2"
     (3SBP INSEnter new item:¢$ MBL
        (IMM MBL pidone$?;SAX 2SBP
           (pl? MFL MON:; CPL TRA DFL MON:)
      2SBP)#
```

6.2 Phonemic analysis package

6.2.1 Preliminary

The four major programs of this package compile a table of frequency counts in buffer 4. Before the final output is given, the table is sorted into alphabetical order. Rather than repeating the sort code in each program, it is placed at the beginning of the program buffer and executed as a subroutine by using the 1EXC command.

```
"sort 4"
    (4SBP (SAX 4SBP (pa? MFL; pl? MFL:; CPL TRA DFL) MON pm?:;)4SBP)#
```

We begin with a vocabulary file (as in section 6.1) in buffer 2. First the complete file is copied into buffer 3.

```
"copy 2 to 3"
     (3SBP SSV 2SBP (pm? SVL MFL MON:;) 3SBP)#
```

Only the vernacular words are analyzed. Thus all the glosses must be deleted. The result of the following program is a list of vernacular words in buffer 3.

```
"delete glosses in 3"
     (3SBP (ps:$? DBC DFL INS¢$:;) 3SBP)#
```

Any digraphs in the original data must be converted to a unique single character for the purpose of the analysis programs which follow. Using the immediate mode, replace the "--" in the next program with the digraph and the "-" with the unique single character to which it will be converted. The program will convert every instance of the digraph.

```
"systematic change of digraphs in 3, replace -- and -"
    (3SBP (ps--$? 2DBC INS-$:;) 3SBP)#
```

6.2.2 Count phone occurrences

The data is in buffer 3; the table of occurrence counts is tabulated in buffer 4. Each entry in the table is two characters long. The first character is the phone being counted; the second character is the count of occurrences. The final output is an alphabetized list of the phones and their number of occurrences:

```
a 234
b 057
d 061
e 174
etc.
```

```
"phone occurrences in 3, table in 4"
    (4SBP (pm? DFL:;)
     3SBP (SAX MON 4SBP
           (pe?MFC INC; 2MFC pm?:; CPC INS$)
          TRA MFC pm?:;)
     4SBP (pi¢$? DFC INS#$; 2MFC:)
     4SBP (pm? MFC INS___$ NUM 3MFC INS¢$:;)
     1EXC)#


    (     (Clear buffer 4 to make room for table.)
     Set BP to beginning of data in buffer 3.
          (Set Aux to current character in data, BP to table.
             (Is this the current character?
                Yes, move to the count character and increment;
                No, move to new entry.  More in table?
                   Yes, repeat:;
                   No, make a new entry.  Copy character and
                     initialize count to one.)
           Return to data and advance to next character. More data?:;)
          (Change CRLF in table to #--a printable character.)
          (Format the table and change counts to decimal numbers.)
     Sort the table into alphabetical order of the phones.)
```

6.2.3 Count phone co-occurrences

This program differs from the previous one only in that the occurrence of sequences of two characters is counted rather than the occurrence of single characters. Each table entry has three characters. The first two are the two co-occurring phones and the third is the count. The output is an alphabetical list of phone co-occurrences and their frequencies:

```
a-# 023
a-b 017
a-d 021
a-e 052
etc.
```

Using the output of the phone occurrences program, the rows and colums of a phone co-occurrence matrix can be labelled beforehand and the results of this program copied directly into the matrix. When the matrix is filled in, the complementary distribution of allophones will become apparent.

```
"phone co-occurrences in 3, table in 4"
     (4SBP (pm? DFL:;)
      3SBP INS¢$ MBC
           (SAX MON 4SBP
              (2pc? 3MFC INC; 3MFC pm?:; 2CPC INS$)
            TRA 2MFC pm? MBC:;)
      3SBP DFC 4SBP
            (pm? (pi¢$? DFC INS#$; MFC)
                 (pi¢$? DFC INS#$; MFC) MFC:;)
      4SBP (pm? MFC INS-$ MFC INS__$ NUM 3MFC INS¢$:;)
      1EXC)#
```

6.2.4 Tabulate word shapes

The program first translates every word in buffer 3 to a sequence of C's and V's. The translation is done via a translation table stored in the program buffer. Each entry in the table consists of two characters, a character from the phonetic alphabet used followed by the character it is to be translated into. In general, vowels will be translated into V and consonants into C. If the linguist suspects that some classes of phones, like nasals or fricatives, have special distribution restrictions, then they can be singled out as a class by translating them to unique characters, say N or F. Special characters such as CRLF, stress mark, hyphen, and space should be translated back to themselves. If translation fails, that is, if a character cannot be found in the table, the program prints an error message and returns to immediate mode. The operator can

examine the data to find the error and either change the data or the table. Giving the # command will return to program execution and the translation will be attempted again. After all the words are translated, they are syllabified according to a set of string transformations. The translation tables and syllabification rules in the following program are the ones which were used for the Biliau language.

After all the words are translated, the frequency of occurrence of word shapes is tabulated in buffer 4. This table is a line oriented table, rather than character oriented, because it is not known beforehand how many characters an entry will take up. Each entry thus has two lines. The first is the word shape; the second is just one character and is the count for that word shape. The output is an alphabetized list of the word shapes that occur and their number of occurrences:

```
CV 023
CV.CV 135
CV.CVC 178
CVC 079
etc.
```

The output is used to study the distribution of syllable types within the word and the distribution of stress with respect to syllable types and position in the word.

```
"word shapes in 3, count in 4"
     (2SBP SAX 1SBP ps"table"$ MFC
          (TRA MON pm? TRA
             (pe? MFC TRA CPC DFC TRA 2MFC 3MBL;
                  2MFC pi#$? 5SBP INSFAILURE$ IMM:;:):;)
      3SBP (psVCV$? 2MBC INS.$:;)
      3SBP (psVCC$? MBC INS.$:;)
      3SBP (psVV$? MBC INS.$:;)
      3SBP (SAX MCN 4SBP
             (pf? MFL INC; 2MFL)pm?:; CPL INSβ¢$)
           TRA MFL pm?:;)
      4SBP (pm? MFL DBC INS___$ NUM MFL:;)
      1EXC)#

"table"¢
¢¢__--aVeViVoVuVAVpCtCkCbCdCgCfCsCzCmCnCηCrClCwCyC#
```

```
(Find the translation table.
     (Go to the data buffer.  Is there more data? Yes,
      set Aux on data character and BP on table.
         (Is this the character in the table?
             Yes, translate the character and move BP back
               to beginning of table;
             No, advance to next entry.  End of table?
                 Yes, translation fails.  Give message and
                    go to immediate mode; return when corrected:;
                 No, repeat:)
      Character was advanced during the translation routine:;)
     (Syllabify, VCV →V.CV)
     (Syllabify, VCC →VC.C)
     (Syllabify, VV →V.V)
     (Count occurrences of word shapes in buffer 4)
     (Format the table and convert counts to decimal numbers)
 Sort the table)
```

6.2.5 Tabulate syllable shapes

The file of word shapes remaining in buffer 3 after execution of the previous program is used to count the occurrence of syllable shapes. First the CRLF's separating words are replaced by the string "#.#"; the # represents a word boundary, and the period represents a syllable division. The program searches for the periods to delimit the beginning and end of syllables. By useing #, a distinction is maintained between initial syllables (eg. #CVC), medial syllables (eg. CVC), and final syllables (eg. CVC#). Counting, output formatting, and sorting routines are as in the previous program.

```
"syllable shapes in 3, count in 4"
     (3SBP INS#$
          (ps¢$? DBC INS#.#$:;)
      3SBP (SAX ps.$? MBC INS¢$ 4SBP
              (pf? MFL 2INC; 2MFL pm?:; CPL INS β¢$)
           TRA ps¢$ DBC MFC:;)
      3SBP DFC
          (ps#.#? 3DBC INS¢$:;)
      4SBP (pm? MFL DBC INS___$ NUM MFL:;)
      1EXC)#
```

6.2.6 Extract a file of specific examples

When studying the output of the occurrence and co-occurrence counts, the linguist may find something which looks suspicious or needs further checking. He would thus like to examine all the words involved. This program will search the whole buffer and save every word which has an occurrence of the desired phone or sequence of phones. To use the program he must first replace the "-" with the character or sequence of characters to be searched for.

```
"extract occurrence and co-occurrence examples from 3 into 6"
   (6SBP SSV 3SBP
      (ps-$? MBL SVL MON MFL:;)
    6SBP)#
```

The above program cannot be used for the word and syllable shapes because the original word data is no longer in buffer 3, only the CV shapes of the words. To find examples of a word or syllable shape, replace the "--" in the following program with the desired shape. Buffer 3 is then searched to find instances of the argument string. Whenever an instance is found, the corresponding line in buffer 2 (the original data including glosses) is saved into buffer 6. The program works by searching buffer 3 one line at a time. Every time the search moves to the next line in buffer 3, the Aux pointer in buffer 2 is advanced to the next line. Thus the two pointers are always on the corresponding lines of the two buffers.

```
"extract word or syllable shape examples in 3, save from 2 to 6"
   (6SBP SSV 2SBP SAX 3SBP
      (pm? (pi¢$?     TRA MFL TRA;)
           (pi--$? TRA MON SVL TRA;)
       MFC:;)
    6SBP)#
```