DECUS
PROGRAM LIBRARY

DECUS NO.	8-595
TITLE	UPDATE, A PROGRAM TO MAKE CORRECTIONS TO A FILE CONTAINING RECORDS OF VARIABLE LENGTH
AUTHOR	Floor Anthoni and Hans Mees
COMPANY	Medical Biological Laboratory TNO Rijswijk, The Netherlands
DATE	January 5, 1973
SOURCE LANGUAGE	PAL-8

PROGRAM NAME	: UPDATE, a program to make corrections to a file containing records of variable length.
SOURCE LANGUAGE	: PAL8
AUTHORS	: Floor Anthoni and Hans Mees, Medical Biological Laboratory TNO, 139, Lange Kleiweg, Rijswijk 2100, The Netherlands.
ABSTRACT	: Designed to facilitate the management of data such as card-indexes on computer mass-storage media, UPDATE provides the user a simple, yet powerful means to correct such files by the method of string-replacement. The 4K program, expanded with the capability of doing PS8 input/output, can easily be adapted to other operating environments.
LIMITATIONS	: The described version needs the PS8 operating system. Records are limited to approximately 700 (10) characters. The "commercial AT" sign cannot be used in the records.
PROGRAM SIZE	: 4K in Field Ø and an additional 10 pages in Field 1.
EQUIPMENT AND SOFTWARE REQUIRED	: 8K PDP8 running under the OS8 or PS8 operating system.

CONTENTS PAGE

1. Introduction

A card-index of all members of a particular organisation is a typical example of a small data-"bank". The content of such a databank is not constant, with respect to time: some members leave the organisation, and their entry (their card) has to be deleted from the databank. In other cases new members have to be entered into the card-index, or data of existent members have to be changed. These changes are normally done by re-typing the respective card, including all the changes.

In the shape of a card-index, such data-management can be done easily and effectively.

Problems arise, however, if one tries to do something with the data: prepare a member's list, type mailing labels, telephone lists, or make selections according to some predetermined properties.

These operations -called "DATA-MANAGEMENT" - consume a terrific amount of man-power (girl-power), and are the direct reason for entering the data into the computer.

Once transferred into computer format, the data bank is not easily accessible as before, unless programs could be developed to facilitate the management of computer legible data.

The program UPDATE is designed to give the secretary an easy means to correct her data files.

The program has a "hole", called the USER MODULE where an assembly-written module can use internal routines to perform the various ways of automatic data processing, found in existent data-management systems. This makes UPDATE useful for a variety of jobs.

2. UPDATE from the operator's point of view

2.1 Applicability

In Chapter 1 the idea of data-management was illustrated by an example of a card-index of all members of a particular organisation with respect to names, addresses, telephone numbers etc. A complex entry would also contain: date of birth, salary group, family data etc. The applicability of "computerisation" of data extends, in general, to all data files that produce a considerable amount of type-out, or consume a substantial amount of man-power for automatic type of operations. Examples of data, suitable for UPDATE, follow below:

a) Litterature retrieval data
b) Descriptive data of thousands of color slides, films etc.
c) Vendors of equipment data
d) Scientific data files for computer processing
e) Program development
f) Program modules:filing and retrieval.

2.2 The variable length record

A "record" is a group of data that belongs together. The name, of a person, his telephone number, his address are data that belong together. The telephone number of someone else, for example, does not belong here. The length of one record, expressed in the number of characters, is in general not the same for all records. This is meant by the term "variable length record".

The use of fixed length records, where the data elements have to appear at certain, pre-determined positions has the disadvantage, that the fixed length will in general be chosen to accommodate for the longest possible record. A file containing fixed length records, will in general be substantially longer than a file composed of variable length records. A great advantage of fixed length records is that, when stored on a random-access mass storage device, they can be randomly accessed in a fast way. When stored on a sequential access mass storage medium (magnetic tape), this advantage no longer applies. The program UPDATE was designed for a sequential access system with records of variable length. An example of a variable length record is presented in figure 1. The records n need to be separated by a special marker. This record marker is the @ sign in figure 1.

```
@0002
ALPHEN,P.J.VAN
A:HOOGOVENS;2H-15;IJMUIDEN
T:02510-94893
I:PROCES AUTOMATISERING,
K:PROCESS,AUTOMATION,PDP8
@0003
ANTHONI,IR.J.F.
A:MED. BIOL. LAB. TNO;LANGE KLEIWEG 139  P.B.45;RIJSWIJK (ZH)
T:015-20330-272 (468)
I:DATA-ACQUISITIE VAN LANGZAME SIGNALEN. BIO-MEDISCHE TOEPASSINGEN.
SYSTEEM-PROGRAMMEREN.
K:PDP8,8I,12K,1970,DISK,DECTAPE,DATA-ACQ,BIOMED,PS8,FOCAL,PHYSIOLOGY
```

Figure 1.

Another example of a variable length record is the listing of UPDATE itself. By inserting the record marker @ at convenient places, the program source has been made to resemble a file containing variable length records.

Note that the @ sign was conveniently chosen because of its infrequent use in records.

2.3 Correction commands. Preparing a correction tape

There are two classes of commands:

1) commands that manipulate whole records. These are necessary to insert (I), delete (D), append (A) or change (C) a record.

2) commands that manipulate characters or strings of characters within a record, in order to append change insert or delete characters. For this purpose a different approach was made, in order to facilitate the editing process for the secretary: the above mentioned functions are all embedded in the string replacement command.

   For example *QICK = QUICK* would be necessary to insert the character "U". UPDATE interprets this statement as follows: it searches for the first occurrence of the string QICK, deletes that string, and inserts the new string QUICK. These commands enable us to correct nearly all possible errors in the file. The string replacement commands always occur together with the C (change) command. This informs UPDATE that string replacement statements follow. These statements are

delimited by a string delimiter character (* in our examples).

The program UPDATE must in some way recognise on which record a certain command has to be effectuated. This is obtained by automatically numbering all records sequentially. UPDATE in fact, inserts a four digit number behind every record marker (@) that it encounters. By doing so, UPDATE adds a property of its own to all records in the file, and is therefore unsensitive to the contents of each record. By referring to this internal record number one can direct the command to be effectuated on that single record. For example: ØØØ6D would delete record number ØØØ6. Strings of corrections themselves, resemble the structure of a variable-length record very much. Therefore the correction strings are separated by the same record marker (@) as used in the file.

@ØØØ6D
@ØØ14A

THIS IS AN EXAMPLE

@ØØ15C*QICK=QUICK*

illustrates the remarkable resemblance to the file of variable length records.

The internal record number can be deleted from or created for all files by a program control option (see chapter 2.4). The record marker can also be deleted at will. This was necessary to remove all properties of UPDATE from the file.

NOTE that each INSERT, DELETE, or APPEND causes a complete renumbering of all subsequent records. During the whole run, however, the "old" numbers still apply. A new listing is required to obtain the correct, new internal record numbers.

Let us cycle, for example, through a complete creation and correction of a file. All typing work can be done at an off-line teletype with low speed punch.

1) Turn on the punch and teletype (LOCAL) and produce some 60 centimeters of blank tape by pressing the key "HERE IS".

2) Now the file is entered:

THIS IS AN EXAMPLE (LF) @ THE QICK BROWN FOS JUMPS OVER THE LAZY DOC @ THIS IS A TELETYPE @XPQR@HELLO @ THIS IS THE LAST RECORD (CTRL/Z)	leading text is always ignored. first "record" second "record" last "record"

3) Produce some 30 centimeters of blank tape by pressing the "HERE IS" key, and tear off the tape.

4) The internal record numbers are generated by running the program UPDATE:

```
.R UPDATE                                          put tape in reader;
SPECIFY FILES AND OPTIONS;CORR.TAPE IN HSR.        for options ref. 2.4
*FILE.AS<PTR:/I                                    the tape reads in, and
*                                                  a file FILE.AS is pro-
.                                                  duced
```

5) This file is then listed with PIP:

```
.R PIP
*TTY:<FILE.AS/A
THIS IS AN EXAMPLE

00001
THE QICK BROWN FOS
JUMPS OVER THE LAZY
DOC
00002
THIS IS A TELETYPE
00003XPQR00004HELLO
00005
THIS IS THE LAST RECORD
*
```

The errors can be corrected by a second run. For easy reference, they are underlined. A correction tape is prepared on the off-line teletype:

6) Produce enough blank leader tape.

```
THESE ARE CORRECTIONS          (leading text is ignored)
00001C*QICK=QUICK*             (corrections, record ØØØ1)
*FOS=FOX**DOC=DOG*
(LF)                           (line feeds are ignored)
(LF)
00002C/A =A GOOD /             (insert the word GOOD)
00003D                         (record ØØØ3 is too bad)
(LF)                           (insert a CR in ØØØ4)
00004C*H=J+LF
H*                             (we want an insert here of)
00005I                         (a complete record)
THIS RECORD IS NEARLY
THE LAST ONE
```

then finish the tape with enough blank trailer tape.

7) Run UPDATE to effectuate these corrections

```
.R UPDATE                                          (enter tape in HSR)
SPECIFY FILES AND OPTIONS;CORR.TAPE IN HSR.        (for options see 2.4)
*FILE.AS<FILE.AS/C
*
```

```
.R PIP
*TTY:<FILE.AS
THIS IS AN EXAMPLE

@0001
THE QUICK BROWN FOX
JUMPS OVER THE LAZY
DOG
@0002
THIS IS A GOOD TELETYPE
@0003
HELLO
@0004
THIS RECORD IS NEARLY
THE LAST ONE
@0005
THIS IS THE LAST RECORD
*
```

A few things can be learnt from the previous example run:

1) all leading text is ignored, until the first occurrence of the record marker. This is true for both the corrections and the file.
2) The string delimiter (*) acts as a sort of switch: it alternates relevant strings and irrelevant strings: A relevant string between asterisk number 1 and 2, an irrelevant string between asterisk 2 and 3 a relevant between 3 and 4, and so on. This is the reason that the excessive carriage returns are not "seen".
3) line feeds and blank tape are wholly ignored and can be given at any time.
4) The correction strings can be chosen at will: for example @ØØØ1C*I=UI**S=X **OC=OG* has the same effect. Remember, however, that the program searches for the <u>first</u> occurrence of the string to the left of the equal sign.
5) In records ØØØ1, ØØØ2 the first character is a carriage-return. This is not compulsory, as illustrated in record ØØØ3. This also counts for the last character.
6) The carriage-return is treated as a single character, and can be inserted and deleted. Additional line feeds can be typed to clear up the text.
7) Most errors are being detected during typing. The rub-out key can be used to "rub" successive characters to the left. This key gives no echo, but UPDATE takes the appropriate action. The back-space pushbutton on the punch is not being used:
8) The rub-out key does not work past the carriage-return. This means that only the characters on the same line, up to the left hand margin can be deleted with the rub-out key. The key "←"(SHIFT/O) deletes all characters to the left by a single strike. These two correction features are effective on all input to UPDATE.

9) The string delimiter (*) can be changed to any character. UPDATE takes the first character behind the "@", at the 6th position from the record marker, as the string delimiter. The same string delimiter must be used throughout the correction record.

10) Correction commands appear in ascending numerical order by record number. This is necessary because of the nature of a sequential access file. UPDATE can therefore not "step back".

11) A record is always preceded by the record marker (@) and terminated by the record marker, or the end-of-file or end-of-tape.

12) A single @ sign cannot be corrected or deleted.

13) The HEADER (leading text) cannot be corrected.

14) Corrections to a specific record may be given in one single correction record or more records, provided that the same record number is referenced.
for example: @ØØØ1C*QICK=QUICK**FOS=FOX*
is equal to: @ØØØ1C*QICK=QUICK*
@ØØØ1C*FOS =FOX*

15) Corrections can be made to corrections.
for example:
@ØØØ1C*QICK=QK* (this is obviously not intended)
@ØØØ1C*QK=QUICK*
The second statement corrects the previous one.

In order to facilitate the replacement of long strings, like a complete line, two string replacement operators can be used: "<" and ">".
The "<" operator deletes all characters to the left of the matched string, including the matched string. For example the correction string
@ØØØ2C*TYPE<THAT IS MY KEYBOARD*
Converts the string "THIS IS A TELETYPE" into the string "THAT IS MY KEYBOARD"

The ">" operator works in much the same way. It deletes, however, all characters to the right of the matched string, including it. It also deletes the carriage-return. The correction string
@ØØØ2C*THIS>*
deletes the whole line in record ØØØ2.
Note that the use of these two string operators can save a lot of typing.

2.4 Option commands

Option commands are issued to the command-decoder at the time the sources and destination should be specified:

```
°R UPDATE
* DESTINATION.AS<SOURCE 1.AS,SOURCE2.AS,'''/O/P/T/I/O/N/S=nnnn
```

These options are intended to modify certain functions in the program. The options, available in the standard version of UPDATE, follow below:

/E echoes all input to UPDATE on the teletype. This is useful for debugging purposes.

/F ignores the formfeed character and deletes it during output.

/T replaces the tabulation character by an appropriate number of spaces. The tabulation interval in 8 characters.

/C The correction part of UPDATE is used. This option automatically sets /I and /N for the purpose of re-numbering the records. If /C is not specified, a "short program" will be taken. This frees the papertape reader to become a normal PS8 input device again.

/I an internal record number of 4 digits is generated (inserted) behind every record marker.

/N The internal record numbers, or any 4 characters behind the record marker are deleted.

/Z The record markers (@) are deleted.

/U Gives control to the user module before output is being done to the destination.

=nnnn Label format option. UPDATE groups the record lines into one label. If the number of lines is less than nnnn, additional carriage-return/line-feeds will be generated. If the number of lines is greater than nnnn, a second label will be formatted, and additional labels, until the remaining lines fit into the last label. Note that octal numbers are accepted only!

/DIGIT Determines the number of empty lines between each label. When more than one digit is entered, the largest one will be obeyed.

/L Accepts the correction tape from the low speed reader instead of the high speed reader. There is, however, no echo.

NOTE: UPDATE is capable of formatting lines into LABELS. This is useful when type-out on self-sticking labels is preferred, for mailing purposes. The options =nnnn and /DIGIT give the user the flexibility to adapt the type-out to his specific needs. The use of pinfeed paper transport facilities on both printer and labels is a prerequisite.

In practice, not all characters of the ASCII set are being treated alike. Some characters have a special function, inherited by both the properties of the teletype, and DEC's software. A summary of special characters follows below, and the way UPDATE has been made to handle them.

LINEFEED - all line feeds are ignored on all input to UPDATE. They are, however, generated for each carriage return during output.
BLANK TAPE - ignored completely
FORMFEED - ignored only if /F specified. after each FORMFEED, 1Ø blanks will be put out.

PARITY - The most significant bit of all incoming characters (the parity bit) will be set on input and output.

TABULATOR - If the tabulation character is followed by a rub-out (EDITOR convention), the rub-out will be deleted on input. It will however, be re-generated during output. If /T was specified, no rub-outs will be put out, and tabs are being converted to spaces A TAB without a rub-out will also be treated correctly.
When trying to rub a TAB in off-line correction tape preparation one has to strike the RUB OUT key twice.

SHIFT/0("←") - used for off-line tape preparation to delete all characters of the current line. This is valid for all UPDATE input.

RECORD MARKER- The @ sign is used for this purpose. Any other character can be used by changing location "CAT" and "LINEL+12" in the program. The @ acts as string terminator during input. The record marker cannot be rubbed out! This also counts for each carriage return during input.

"=", "<", ">"- used in string replacement statements. See section 2.3.

CTRL/Z - This character is the general PS8 file terminator. It also terminates the last record as if it were a record marker. The high speed reader routine possesses a facility to detect the end-of-tape situation and therefore no CTRL/Z is needed to terminate the corrections tape. When using the low speed reader, the last character has to be a CTRL/Z! The CTRL/Z is typed by holding down the CONTROL key and striking the Z key.

CTRL/C - is the break character to stop program execution and return to the monitor immediately. This character is obeyed only when issued from the keyboard. The program writes the current record away, then exits to the monitor. The program is not restartable.

2.5 Error messages

In the eyes of the secretary, it is very important that the corrections and additions to the database are executed in such a way that erroneous data never occur. This can be met by thorough error checking in the program. In most cases, the correction command will not be executed, and in all cases an error message results. The general format of the error message is:

ERROR MESSAGE AT RRRR FFFF NNNN

where the error message is in self-explaining English. See Appendix 3 for a complete summary.. The three numbers refer to the internal record numbers at the time that the error occurred.

RRRR is the number of the correction "record".
FFFF is the old number of the FILE record.
NNNN is the new number of the FILE record.

The new number NNNN enables the operator to repeat the corrections that could not be executed. He must therefore prepare a new correction tape, and refer to the new internal record numbers. The corrections can then be effectuated in a second run. By doing so, one is able to prevent the complete typeout of the file in between.

A second type of error message results from inadvertent use of PS8 files or devices. These are the USER ERRORS in Appendix 3.

2.6 Loading, Running and Saving

The program UPDATE is supplied in two binary tapes, UPDATE.BN and PS8IO.BN. The tape PS8IO.BN contains two modules that make PS8 input/output programming resemble an ordinary teletype. Complete documentation is available through DECUS 8-472. The modules have been assembled according to the core allocation map in Appendix 4. The tape UPDATE.BN contains all UPDATE CODE. To load and save the program, proceed as follows:

```
. R     ABSLDR
* PTR:, PTR: $ (alt mode)
* SAVE  SYS: UPDATE.SV
```

The tapes can be loaded in any order. An example of a complete run has been given in section 2.3.

Section 3.5 illustrates the use of a USER MODULE, and the way a USER MODULE can be loaded. The automatic concept, used in the modules PS8IN and PS8OUT prevents the program from being RESTARTABLE. All operations in UPDATE eventually result in a return to monitor. One destination (output) file and up to 9 source (input) files can be specified to the command decoder.

3. Program description

This chapter describes the internal structure and the working of the program UPDATE. A complete description of the program would be an impossible task.

However, the design philosophy will be described in the sections of this chapter. The reader is referred to the accompanying documents in the appendices for more detail. These are:

*1) The program LISTING, produced by the PAL8 assembler. It has CROSS REFERENCE tables in the rear.

** 2) FLOW-CHARTS of the most important program parts.

3) PROGRAM HIERARCHY TABLE. This table illustrates the hierarchy of the program's internal subroutines. The reader can use this sheet to trace which subroutine calls which other subroutine.

** 4) A BITMAP visualises all locations in core that have been used by UPDATE. The user can easily detect small holes in the program for small changes.

5) The CORE ALLOCATION MAP symbolically pictures the location of buffers, working-areas, PS8 handlers, etc. in core.

6) The reader is also referred to chapter 4 where most internal subroutines are discussed in more detail.

3.1 The main program

The main program of UPDATE has a simple structure. It starts printing a message, and reads and writes all preceding "Header" text without processing. This action calls the command decoder through PS8IN, where up to 9 sources, one destination and several options may be entered. These options are then decoded into specific page zero locations: OPTA through OPTZ, LINENO and NOPRINT. If option C has been set, the main program will be started. This main program reads a correction record and a file record. It compares the record numbers, and spools more file records if the record-number of the correction record is greater than the record number of the file record.

* Offered Separately

** Included with listing package

In the case of a match, the next character is read from the correct-Insertion record, and used to branch the program to 4 parts: INSERT, Append, Change, Delete. Program flow can easily be understood from the flowcharts, in the case of APPEND, DELETE and INSERT. The CHANGE part, however, is more complex. The next character is taken as the string delimiter, and is being deposited in the list CHLIST at the location CHTERM. The "match-string" is then isolated. This string always occurs between the string delimiter, and the = or < or > sign. The File record is then searched from the beginning each time, for a match with this string and if found, it will be deleted from the file record in the characteristic way of the = and < and > operators. The string, following the operator (= <>) is then inserted at that point.

Then the next replacement operation is executed. When all commands contained in a correction record have been executed, the program reads the next correction record. If the internal correction record number still agrees with the internal file record number, then another set of correction operations will be executed on the same record. Before a record is being written to the output, the option U is tested, and if set, control is given to the USER MODULE. In absence of such a module, a dummy subroutine returns control to the main program. The general output subroutine FORMAT takes care of the execution of the options F, I, N, T, Z, DIGIT and =nnnn.

3.2 The spool program

In the case the option C has not been set, a very short "spool" program is effectuated. This program reads records from the PS8 sources, and writes them to the PS8 destination. By doing so, the papertape reader can be used as an input device.

All options remain effective in this spool program.

3.3 Input-Output switchability

An important feature, employed in UPDATE is the switchability of inputs and outputs. All input routines have been written such that they exhibit the same properties. Input routines have two subroutine returns: one signaling the end-of-tape or end-of-file situation, the other is the normal return. After return, the fetched character will be in location CHAR. This concept could be realised because of the two modules PS8IN and PS8OUT that make PS8 behave like a Teletype.

The output routines have been designed in almost the same way, however, with only one subroutine return. The software switches are the locations INDEV and OUTDEV in page Ø, that contain a pointer to one of the input routines FREAD (from PS8), HREAD (from high speed reader) and KEYBD (from keyboard), or one of the output routines OUTPS8 (to PS8), PRINT (to teleprinter) or DUMOUT (which causes no output at all). The DUMOUT dummy-routine is used to obscure the echo of the input routine LINE, in the case no echo is preferred.

The user can easily apply his own input and output routines.

3.4 PS8IN and PS8OUT

The two subroutines PS8IN and PS8OUT were designed to make PS8 look to the programmer as a normal teletype. As PS8 is block-oriented with file structured devices, some complex operations have to be performed automatically. These can be illustrated by describing the actions for the first character, and for the last character (CTRL/Z).

A JMS to PS8IN for the first time, calls the command decoder in core. It then loads the input device handler in FIELD Ø, and opens the first input file. The first block is read into an appointed buffer in FIELD 1, and the first byte is unpacked from it. A subroutine return is then taken with the byte in the AC.

Successive calls to PS8IN deliver succcssive bytes. If the first file is exhausted, the module opens the next one, loading device handlers if necessary. The CTRL/Z character, appearing at the rear of each input file is ignored, unless it is detected to be the last character of the last input file. The CTRL/Z character thus designates the absolute end of all input files. The first character delivered to PS8OUT calls the output handler, and opens the output file. Successive characters are packed into the assigned output buffer in FIELD 1. Full buffers are being written to the output file.

A CTRL/Z character, sent to PS8OUT causes the following actions: The CTRL/Z character is packed into the buffer, and the remaining space of the output buffer is "padded" with zeroes. The buffer is then written to the output file, and the file is closed. A direct jump to the MONITOR concludes all actions.

The two routines are not restartable. Configuration parameters with respect to size and location of the buffers, size and location of the input- and output handler, and default extension can be set at assembly time.

The modules have proved to be resistant to all possible errors in PS8 input/output programming, and produce relevant USER ERROR messages. These have been listed in Appendix 3. A complete documentation can be ordered from the DECUS library (catalogue number 8-472).

3.5 The user module

The main design philosophy of UPDATE was: flexibility. This is reflected in an important feature of UPDATE: a 4 page "hole", free for additional code to change interral functions, or for other purposes. The hole can be overlaid by a BINARY or IMAGE format file. This extends the use of UPDATE to those applications that have not been foreseen by the authors. The USER MODULE is discussed in more detail in chapter 4.

4. Writing your own USER MODULE

The contents of this chapter are intended to instruct the user in detail how he can design his USER MODULE by making effective use of internal routines within UPDATE.

The chapter contains, however, useful information for those users that want to make some minor modifications to the program. For a survey of all useful internal subroutines see Appendix 4.

4.1 Adaptability of UPDATE

Much effort has been put in designing UPDATE as flexible as could be possible. The authors are aware, however, that the ultimate can only be reached by a thorough re-design of UPDATE for which they lack time. The adaptability is expressed in the following features:

1) The presence of a regular 4 page program hole for a USER MODULE.
2) All lists in UPDATE are open-ended, accessible for future expansion.
3) Switchability of inputs and outputs.
4) A simple string structure: one byte per location; no packed bytes in the working areas.
5) Working areas cannot easily be re-dimensioned during program execution, but can be re-defined at assembly time!
6) Program subroutines can all be used by a USER MODULE.
7) A strong hierarchical structure prevents ambiguous situations.
8) All possible options have been decoded in page zero, ready to be used by a USER MODULE.
9) The program is originally designed for 4K operation. All that exceeds 4K is the code and buffer areas, necessary to accommodate for PS8 input and output facilities. The program can be adapted for 4K operation under a different operating system.
10) String manipulation and "garbage collection" is straightforward and can easily be understood.
11) A great part of FIELD 1 is available to the USER.
12) The number of digits in the internal record number can be changed by changing location ZNUM.
13) The inter-record marker @ can be changed by changing locations CAT and LINEL+12.
14) Locations Ø to 7 have not been used. This is particularly useful for debugging purposes.
15) Users can use several page Ø locations for their modules (17Ø-177).
16) No special programming "tricks" have been used in order to save space in core. The code is straight forward and easily understandable.

4.2 Input/Output Routines

All input routines were designed to exhibit the same properties. Standard input routines (HSREAD, PS8IN) are thereto adapted by higher level subroutines.

These routines (FREAD, HREAD, KEYBD) read one character at a time. This character is deposited in location CHAR. The parity bit is ignored and forced on. When detecting the end-of-file or end-of-tape a record marker is transferred into CHAR, and a normal return results. The next attempt to read a character, results in an EOT return. The subroutine LINE calls the input routines through the indirect software switch INDEV. It processes the linefeed, rub-out, shift-O, and formfeed characters, and loads the characters into a designated core area. It produces an echo for each character, back-slashes for rub-outs, etcetera to the output device, pointed to by OUTDEV. A carriage-return or a record marker terminates a line.

The next higher level subroutine GETREC, reads several lines that constitute a complete record. The two subroutines on top of GETREC, INHSR and INFIL set parameters to read a complete record into its specific working area in core. Note that the end-of-file return is transferred all the way up, in order to control the flow of the main program on the highest level of hierarchy.

The same sort of organisation rules the output, with the difference that the subroutine FORMAT processes and transmits all output on a character-by character basis to the output routines.

All output routines have a normal return only. These routines are OUTPS8, PRINT and DUMOUT. Note that OUTPS8 is a FIELD Ø resident routine that calls the FIELD 1 resident output module PS8OUT.

4.3 String manipulation routines

Characters are stored one byte per 12 bit location. This makes string manipulation easy to understand, and relatively fast. The basic search routine is COMPAR. a routine, able to compare two strings A and B with a pre-determined length. The routine has 3 returns: for A>B, A=B and A<B. It uses two auto-index pointers in page zero: COMPTR for string A and MSPTR for string B. The number of characters is entered in the AC.

The higher level SEARCH routine uses COMPAR. The string in core is searched for a match with a string B of predetermined length. The search starts with the character, pointed to by AXINØ, and proceeds until SCHEND (search end) is reached or a match was found. The pointer AXINØ travels along with the search, and points to the beginning of the equivalent of string B in the case a match was found. This fact is used by the DELETE and INSERT operators. The deletion of for example 4 characters is performed by moving all following characters four positions forward. This is the simplest way to perform some sort of "garbage collection". The pointer FILEND, pointing to the last character of the string in the FILE working area, is corrected. The pointer SCHEND, however, is not corrected!

The pointer AXINØ and the parameter COUNT 1 determine the DELETE action.

The INSERT function inserts a string at the point AXINØ, by moving all following characters backward to create a hole for the string to be inserted. The hole is then filled by the insertion string. Pointer FILEND is corrected for the new situation. Before an INSERT is attempted, the available room in the working area is tested. If there is not enough room to perform the INSERT operation, an error message "RECORD TOO BIG" will be printed while no INSERT is performed.

Note that the pointer SCHEND is not affected!

4.4 Miscellaneous routines

The ERROR MESSAGE routine prints an error message at the console Teletype. In addition it prints the internal record numbers of the current correction record, of the current file record and of the current file record as it is written to the output. If the internal record number makes no sense, the four characters ???? will be printed instead. For this purpose the internal record number itself is replaced by ???? in its working area.

The BRANCH routine can be used by user modules to branch the program flow through the list look-up technique.

The INCREMENT routine can be used to count certain events. The INCREMENT routine treats an ASCII string of digits as a decimal number, and increments that number. The number of digits is determined by the ACCUMULATOR. The first argument points to the most significant digit of the string. An overflow causes return to monitor without any message.

The MOVE routine is one of the most frequently used routines! it is able to move a block of core memory to a different part within 4K. It takes care for the direction of the MOVE in order that the block does not overwrite itself.

The tape "UPDATE SYMBOLS" contains all the necessary definitions to design a user module. When assembled in front of the module assembly source, and preceded and followed by the XLIST pseudo-operator it will not be listed in the ultimate listing.

APPENDIX 1 SUMMARY OF CORRECTION COMMANDS

General syntax examples:

```
@ØØ61I  THIS IS A NEW RECORD
@ØØ85D
@ØØ94C * CORCT=CORRECT **NW = NEW **A:>*
       |  └── can be any character.  However, use it throughout the
       |      same record.
       |
       A   APPEND a record behind the one, numbered here.
       I   INSERT this record before the one, numbered here.
       D   DELETE this record.
       C   CHANGE this record ──┐
                                │
                                │
       The CHANGE command expects string replacement statements
       of the following type:
```

LEFT =	RIGHT	Search the string 'LEFT' from the beginning of the record. Delete string 'LEFT'. Insert string 'RIGHT' at this point.
LEFT	< RIGHT	Search the string 'LEFT' from the beginning of the record. Delete string 'LEFT' and all the preceding characters until the left hand margin (previous C.R.). Insert string 'RIGHT' at this point.
LEFT	> RIGHT	Search the string 'LEFT' from the beginning of the record. Delete string 'LEFT' and all the following characters up to and including the next carriage return character. Insert the string 'RIGHT' at this point.

APPENDIX 2 SUMMARY OF PROGRAM CONTROL OPTIONS

Options to be specified to the command decoder.

/C - The CORRECTION part of the program is used. The tape in the high speed reader is considered to be a correction tape. The records are all re-numbered. /C automatically sets /I and /N.

/E - ECHO all input from high speed reader and PS8 on the teletype. Useful for debugging.

/F - The FORMFEED character is being ignored during input.

/I - INSERT an INTERNAL RECORD NUMBER behind each occurring record marker (@).

/L - The correction tape is read from the LOW SPEED READER (Teletype).

/N - NO NUMBER ! Deletes the internal record number or any four first characters of each record. The use of /I and /N together re-numbers the records.

/T - Replaces TABULATION characters by spaces during output.

/U - Activates the USER MODULE.

/Z - ZEROES all record markers (@).

/digit - Determines the number of empty lines between labels. When typing more than one digit option, the largest one will be obeyed.

=nnnn - Determines the number of lines contained in one label. The number is OCTAL! If the record contains more lines than a label can contain, the remainder will be put on a new label and so on.

APPENDIX 3 ERROR MESSAGES

General format:

ERROR MESSAGES AT CCCC FFFF NNNN
where CCCC = the record number of the current correction record
FFFF = " " " " " " file "
NNNN = " NEW record number of the current file record.

WRONG SEQUENCE - an attempt has been made to access a record that has already been written to the output file. Correction ignored.

ILLEGAL COMMND - The first character behind the correction record number could not be identified. Correction ignored.

ILLEGAL SYNTAX - Use of string delimiter and string operators in string replacement statement is incorrect. Part of the correction string could have been executed all right.

STRING NOT FND - The string towards the left of the string operator cannot be found. No action is being taken. The following string replacement statements may be executed all right. Or a command failed to occur behind the record number.

RECORD TOO BIG - The record did not fit into the working area, during Insertion or when generating numbers for the first time. The insertion is being executed partly.

ILLEGAL NUMBER - The record number in the correction record was not a four digit string. The correction record is being ignored.

PS8IN and PS8OUT error messages are fatal and cause a return to the monitor.

USER ERROR 1 - The output device handler could not be found; an attempt was made to load a 2-page handler into one page.

USER ERROR 2 - The output file could not be opened or closed; the device was a read-only device; No room for output; Two tentative files on this device; An illegal device name was specified.

USER ERROR 3 - No room for output.

USER ERROR 4 - Write error during output.

USER ERROR 5 - The input-device handler could not be found.

USER ERROR 6 - Read error during input.

5

APPENDIX 4 SURVEY OF ROUTINES IN UPDATE

Input/Output routines

1) Type a character on the Teletype:
TAD CHARACTER; TYPE (= JMS PRINT); RETURN (AC=Ø)

2) Type a CR LF on the Teletype:
NEWLIN (= JMS CRLFX); RETURN (AC=Ø)

3) Output a character over OUTDEV to the output device
TAD CHARACTER; JMS I OUTDEV; RETURN; AC = Ø. Note that the output routines are: OUTPS8, PRINT and DUMOUT

4) Output a CR LF over OUTDEV to the output device:
CRLF (=JMS FORCR); RETURN (AC=Ø)

5) Read a character over INDEV from the input device:
JMS I INDEV; EOT RETURN; NORMAL RETURN (AC=Ø)
Note!The input routines are KEYBD, HREAD and FREAD.
The character is in location CHAR!

6) Read a character from PS8 (OS8):
CIF 1Ø; JMS PS8IN; RETURN (character in AC) or
JMS INPS8; RETURN (character in AC)

7) Read a character from high speed reader:
JMS HSREAD; EOT RETURN; NORMAL RETURN (char. in AC)

8) Read one line of text from INDEV, and echo if
OPTE Ø . Ignores LF, RUBs following TABs, FORMFEEDs if
OPTF Ø. Back-arrow deletes the line. TAD (INPUTROUTINE;
DCA INDEV; TAD (OUTPUTROUTINE; DCA OUTDEV; JMS LINE; BUFBEGIN;
BUFEND; EOT RETURN; NORMAL RETURN
Both returns with pointer to last character in the AC.

9) Print a message on the Teletype
MESG (= JMS PRMESG); ABC, RETURN (AC=Ø)-----
ABC, TEXT /ABCDEF/

String Manipulation Routines

1) Compare two strings A and B of a pre-determined length. There are three returns.
TAD (A-1; DCA COMPTR; TAD (B-1; DCA MSPTR;
TAD (-LENGTH; COMPAR (=JMS COMP); B<A RETURN;
B>A RETURN; B=A RETURN (AC= Ø in all cases)

2) Search the working area (FILE BUFFER) from a point, given by AXINØ to an end, given by SCHEND for the occurrence of a string STRING.
TAD ENDPOINTER; DCA SCHEND; TAD BEGIN-1;
DCA AXINØ; TAD (STRING-1; DCA MSBEG; TAD (-5(length);
DCA COUNT1 ; SEARX (= JMS SEARCH); NOT FOUND RETURN;
FOUND RETURN; ----- (in both cases AC=Ø)
STRING, "H; "E; "L; "L; "O

3) Search the current record from its beginning
for the occurrence of the string ABC.
SRCHLL (=JMS SRCH); 3; ABC; NOT FOUND RETURN;
FOUND RETURN; ----- (in both cases AC=Ø)
ABC, "A; "B; "C
In case of NOT FOUND return: AXINØ points to FILEND-3.
In case of FOUND return: AXINØ points to the character, preceding the found string.

4) DELETE a string of length 4 characters starting at the location pointed to by AXINØ + 1:

```
TAD (4; CIA; DCA COUNT1 ; JMS DELETE; RETURN (AC=Ø)
```

5) DELETE the string that was currently searched for
AXINØ and COUNT1 have been set already, during SEARCH!
JMS DELETE; RETURN (AC=Ø)

6) INSERT a string in the current record, at the position, pointed to by AXINØ

```
JMS INSRT; + LENGTH; STRING; RETURN (AC=Ø)-----
STRING, "A; "B; "C -----
```

7) MOVE a STRING or a block of computer words in core.
MOVEX (=JMS MOVE); BEGIN OF BLOCK; END OF BLOCK;
DESTINATION (NEW BEGIN); RETURN (AC=Ø).

Miscellaneous routines in Update

1) Branch routine. It tests the contents of location CHAR against a LIST, and jumps to the appropriate address.

```
TAD AC; DCA CHAR; BRANX (= JMS BRANCH); LIST-1;
NOT FOUND RETURN (AC=Ø)-------
LIST, 212; LF ADDRESS; 215; CR ADDRESS; ----; Ø
```

Note that the list must be terminated by a zero!

2) Type an error message, the current correction record number, the old file record number and the new file record number.

```
TAD (MESSAGE; ERPRNT (=JMS ERROR); RETURN (AC=Ø)-----
MESSAGE, TEXT/18(10) CHARACTERS/
```

3) Increment an ASCII string of digits as if it were one decimal number. The accu contains - number of digits.

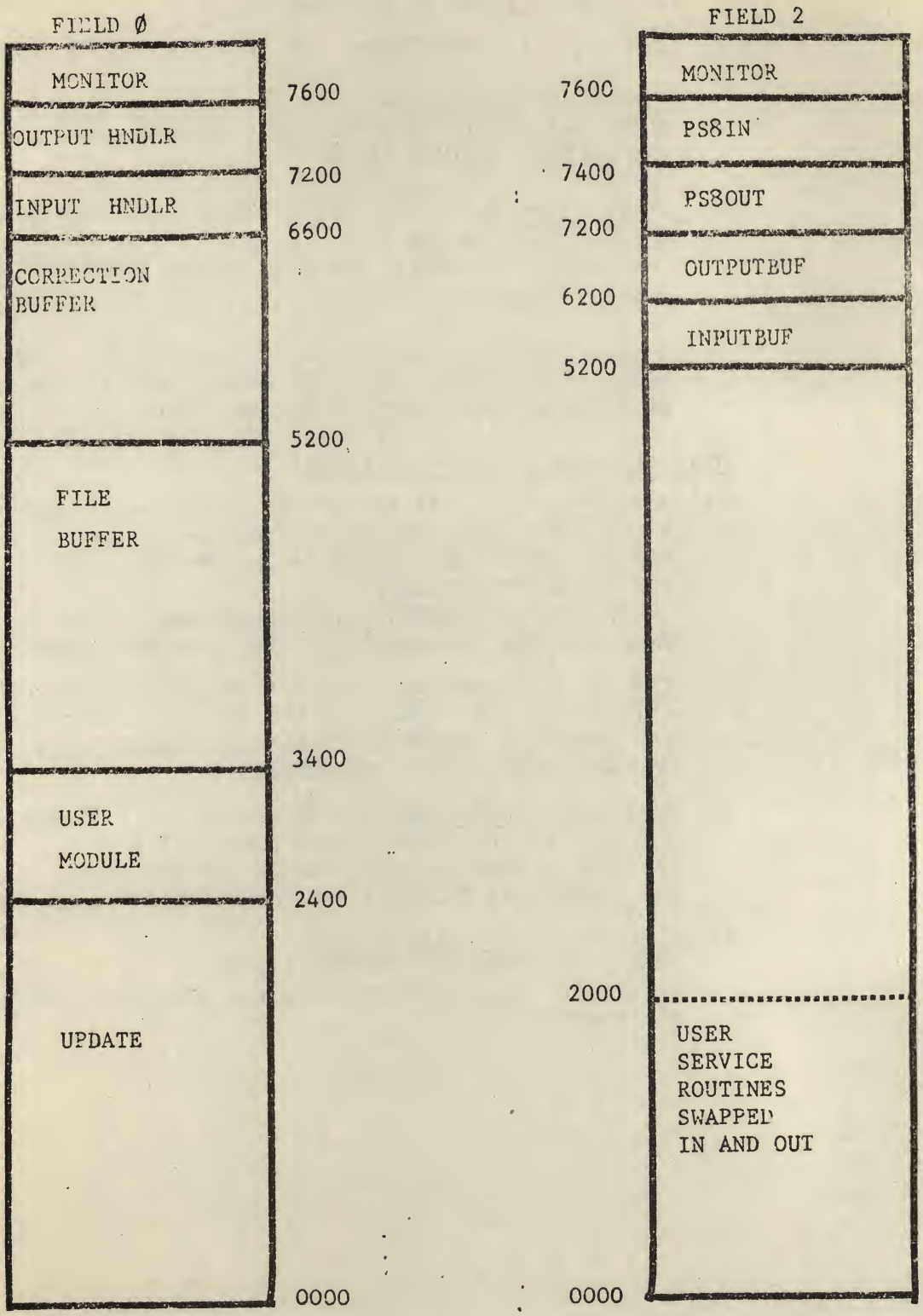
```
TAD (-4; JMS INC1Ø; NUM; RETURN (AC=Ø)-----
NUM, 26Ø; 261; 262; 263 /The number Ø123 becomes Ø124.
```

4) Test the keyboard for CTRL/C

```
TESTC (=JMS CTRLC); RETURN.(AC=Ø)
```

Detection of the CTRL/C character causes an immediate return to the MONITOR.

APPENDIX 5 - CORE ALLOCATION



APPENDIX 6 UPDATE PROGRAM HIERARCHY