PUT AND GET

The relation of pieces of data, called field values, to each other can be expressed as a tree-like structure.* Each node of the tree may contain one or more field values. The major file in the system is the Active Patient Record, and all further examples will refer to this file, which is stored on the Fastrand. The structure of the first few branches of the tree (nearest the trunk) is reflected by a set of drum pointers which point from node to node.

At this level no use is made of proximity of data to reflect logical connections of data. The interpretation of these higher levels of the tree are performed by the I-O processor. After a certain level, however, the I-O processor locates the data connected to a given node called an item and puts it all in core. It is then up to the user program to trace the tree out to the leaves to locate a given field value. For example, if the third administration of the fourth drug of a certain patient were needed, the I-O processor, directed by the user program, would find the patient (first level) and the particular drug item (second level) and then read into core the collection of administrations (the item). The program then must interpret this remaining level left in the tree to find the fourth administration.

THE COMMANDS

The user has available a command which will get the information from an item already in core.

* The most general structure, a connected graph, is not considered here.

This "GET" command accepts as input:

1. The core location of the item.
2. The core location of the map which describes the format of the item (i.e., describes each of the fields whose values may be in the item).
3. The number of the particular field described.
4. The subscripts (or branch number) needed to trace down the tree to find the value. (If the field is unique to the item, the subscript is unnecessary.)

The output of the command is a core pointer to the field value within the item and the value's length.

There is a complementary command to put a field value into the item. This "PUT" command demands, in addition to the same four input parameters, the pointer to the information to be "PUT". The third, and last, command in this micro-file structure begins the item and is executed before the first "PUT". "BGI" needs only the location of an item buffer and the map as input parameters.

THE MAP

Since PUT and GET commands simply interpret the map, or format description, of the item and either file or unfile accordingly, the specification of the map is basic to the structure of the item. Although the map in core is actually a complex configuration of bits, the existence of "map macros" permit the user programmer to define maps in the Midas macro-assembly language without knowledge of the exact bit format of the map. Following are the macro commands used in specifying a map.

1. OVERHEAD q
   This macro has one argument, q , which is the length in words of the overhead in the item. It must be the first entry in a map.

2. TREE
   This macro has no arguments. It is used once per branch at the top level of the item, immediately following the overhead macro.

3. V
   This is a macro of no arguments which indicates a field whose value may vary in length.

4. F t,w,b
   This is a macro of three arguments. It indicates a field whose value is of fixed length.
   t is the transform code, which is currently unused.
   w is the length in words.
   b is the residual length in bits (must be a multiple of 6)

5. LAST
   This is a macro of no arguments which indicates the beginning of the last (lowest) level of a tree.

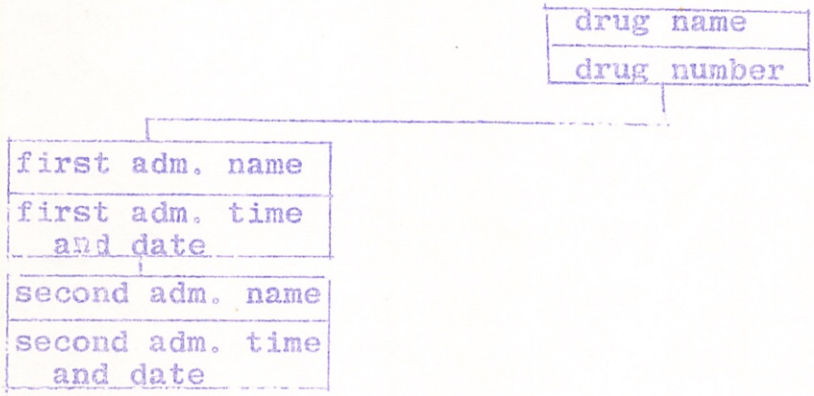
6. LEVEL
   This macro of no arguments indicates the beginning of the anything-but-the-last level of a tree.

7. END
   This macro of no arguments indicates the end of the map.

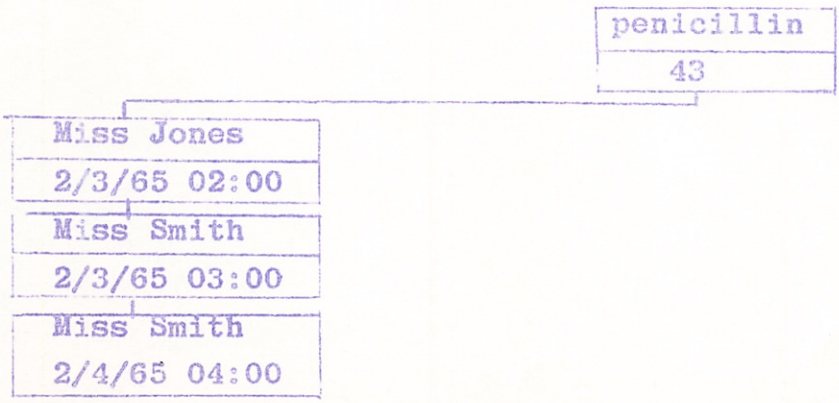
The details of the actual bit configuration of the map are in Appendix A. For example, an APR drug item consisting of the drug text name, and 30. bit drug number, the time and date of administration and the text name of the administering nurse would be as follows:

```
MAP                     OVERHEAD 7
                        TREE
DRGNAM,                 V
DRGNUM,                 F  0,1,14
                        LAST
ADMNAM,                 V
ADMTD,                  F  0,2,0
                        END
```

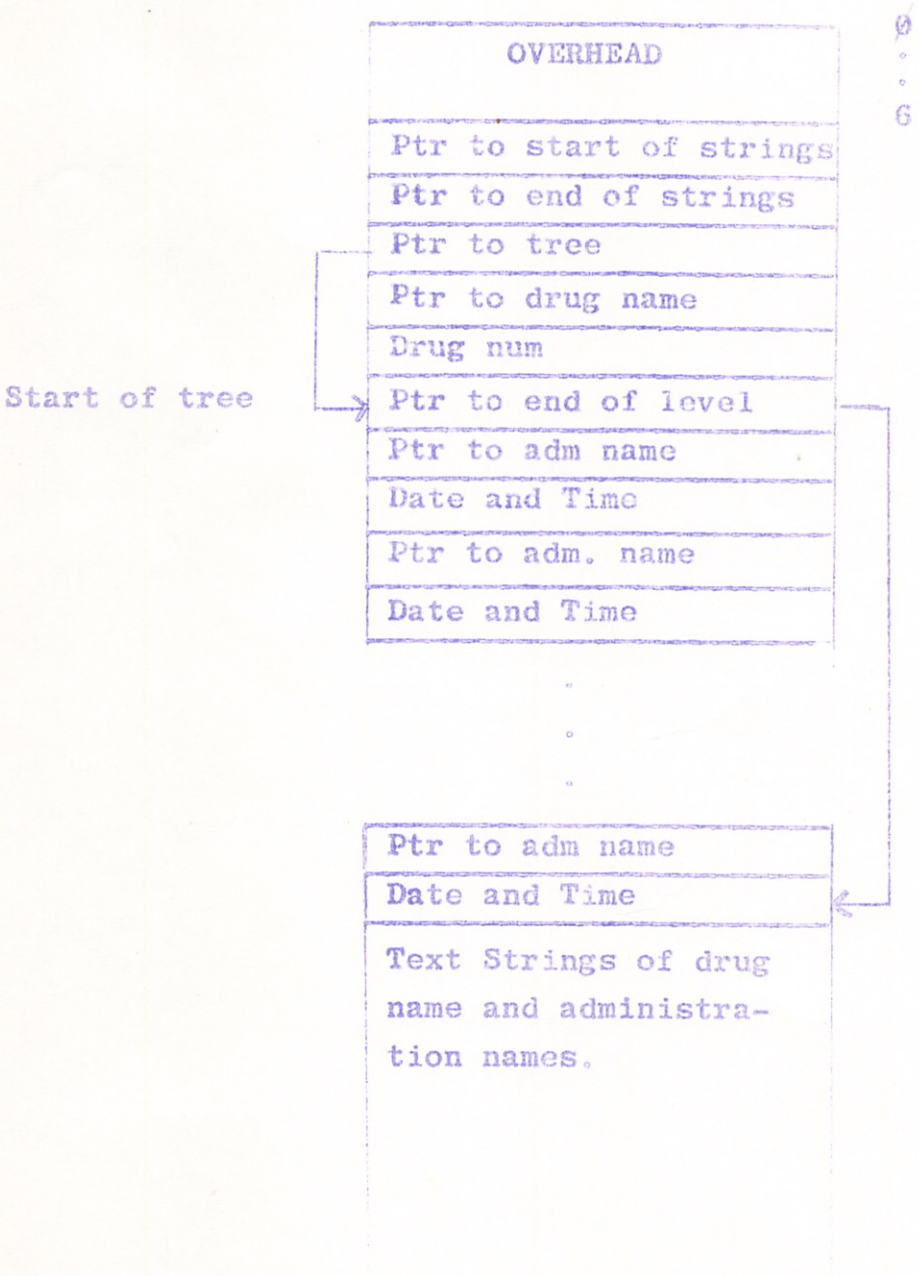
The structure of the data described by this map is as follows:



An example of a penicillin item having the structure described by the above map and tree diagram is:



The actual item which would be constructed (using "BGI" and then a series of "PUTS"'s) would look like:



The details of the structure of a "PUT" and "GET" item are found in Appendix B.

The distinction between the two tree diagrams is analogous to the distinction between a map and an item of the structure specified by a map. The first tree diagram, like a map, is the format or logical relation between fields. The second tree diagram, like the item, is a collection of data in a particular relationship.

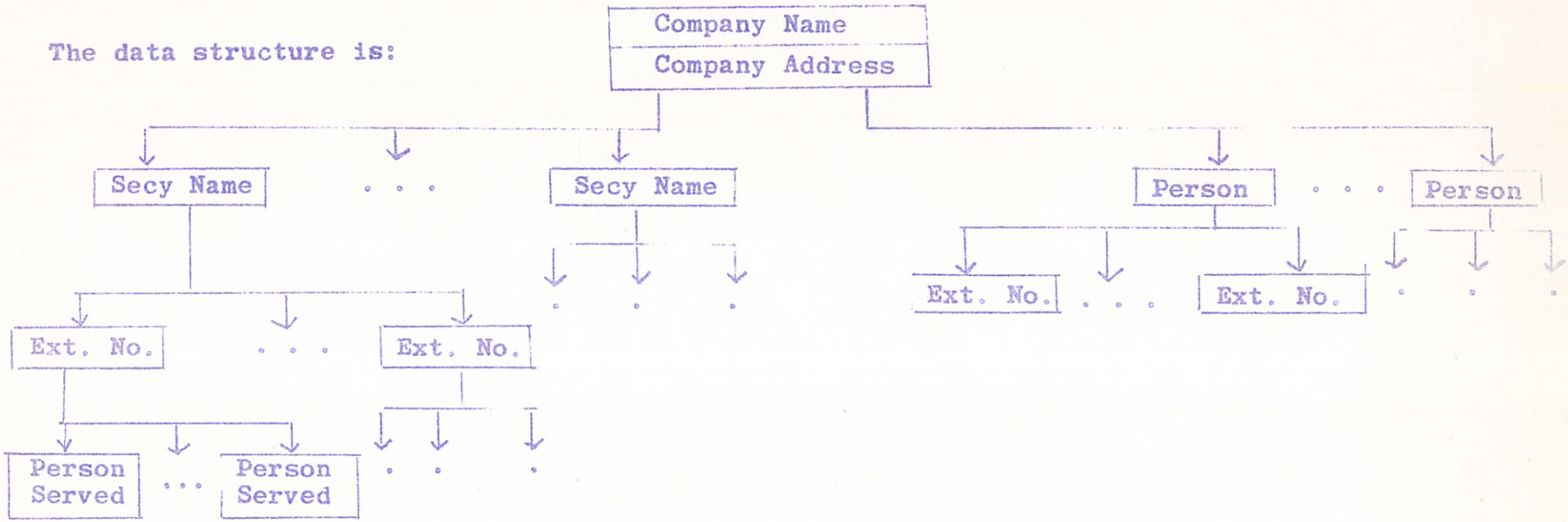
One other more complex example should illustrate the full generality of the structures specified by a map.

This map describes the structure of a phone system--it contains company name, company address, and two trees. The first tree contains a list of secretary names, phone extensions handled by that secretary, and names of people served by each extension. The second tree contains a list of names of people, and the phone extensions where that person might be reached.

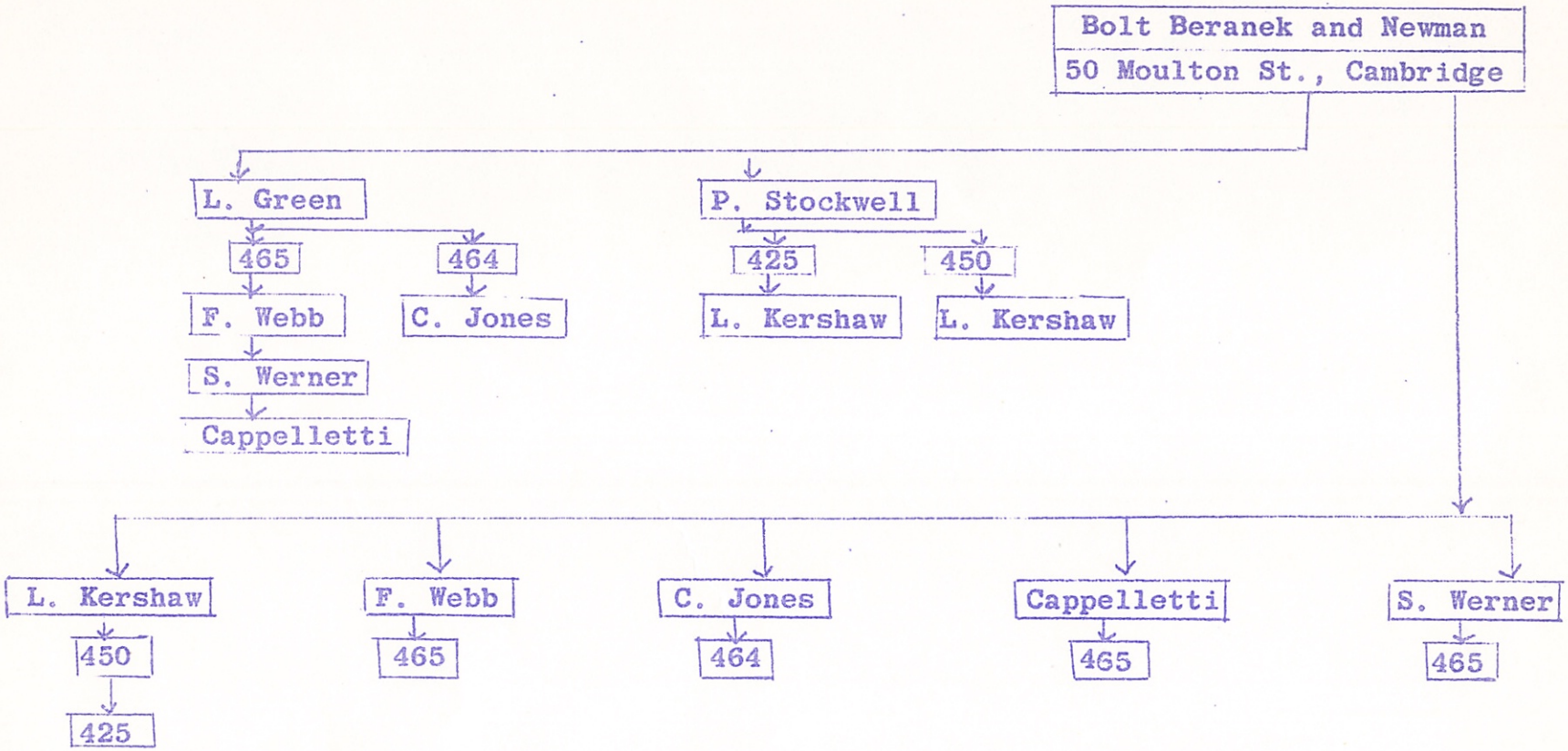
The map is:

```
        MAP2,           OVERHEAD 7
                        TREE
                        TREE
        COMNAM,         V
        COMADD,         V
                        LEVEL
        SECNAM,         V
                        LEVEL
        EXTNM1,         F  0,1,0
                        LAST
        NAMSRV,         V
                        LEVEL
        NAM,            V
                        LAST
        EXTNM2,         F  0,1,0
                        END
```

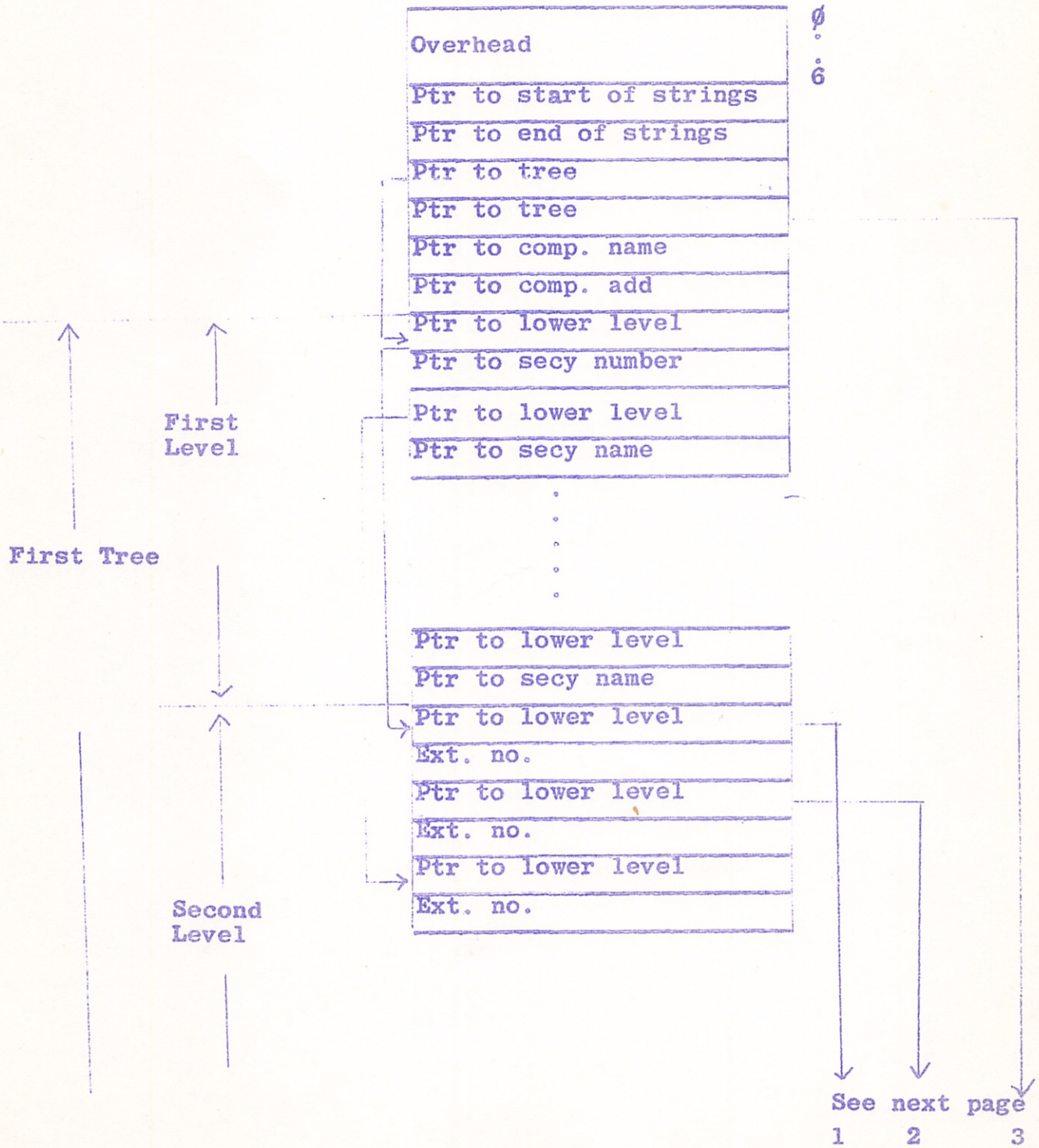
The data structure is:

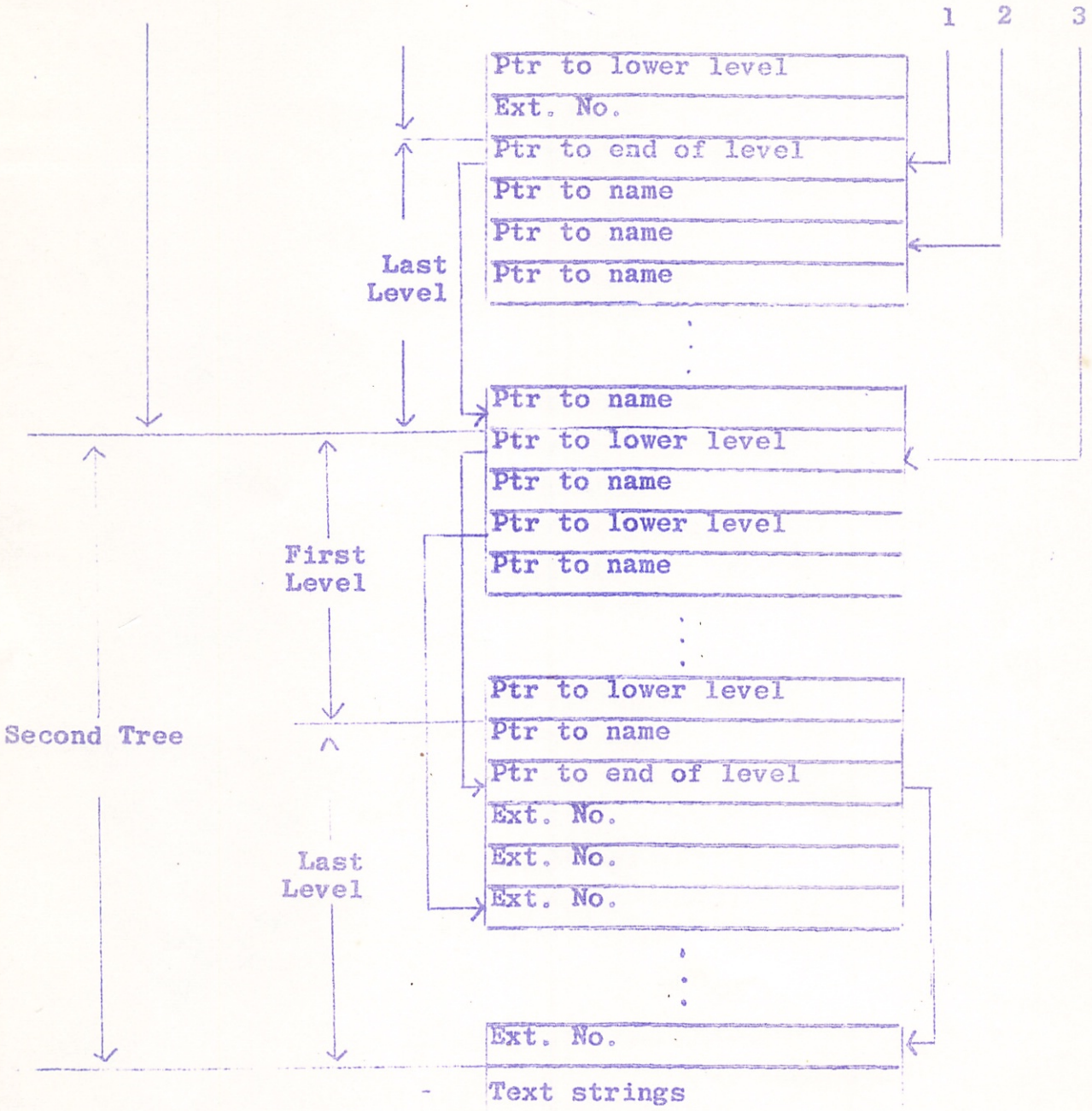


An example item is:



The item would look like:




1
2
3
Ptr to lower level
Ext. No.
Ptr to end of level
Ptr to name
Ptr to name
Ptr to name
Last Level
Ptr to name
Ptr to lower level
Ptr to name
Ptr to lower level
Ptr to name
First Level
Ptr to lower level
Ptr to name
Ptr to end of level
Ext. No.
Ext. No.
Ext. No.
Second Tree
Last Level
Ext. No.
Text strings

THE MAP MACROS:

Any map must correspond to the following format:

First: The macro OVERHEAD (must be present)
The maximum overhead count is 15_{10} or 17_8.

Second: The macro TREE--once for each tree in the item--all occurrences of TREE must come after OVERHEAD and before any other map macros.

Third: The variable fields not in any tree.

Fourth: The fixed length fields not in any tree.
The variable fields must be before the fixed fields.

Fifth: The trees--

General format:

The macros LEVEL or LAST followed by whatever variable and fixed fields (in that order) are on this level of the tree.

The first group of LEVEL's (up to and including the first LAST and its associated fields) are the first tree, the second group the second tree, etc. The macro LAST must appear the same number of times as the macro TREE did.

Sixth: The macro END (must be present)

Between the macro OVERHEAD and the macro END, no lines of coding should appear except map macros, and address tags. Any address tags used within a map will be defined relative to the beginning of the map. That is, address tags within a map will be equal to the field number of the fields they reference, as used by "PUT" and "GET".

Note: All the macros except LAST and END produce a field definition. TREE and LEVEL produce pointer fields while OVERHEAD produces word Ø of the map, plus field maps for the pointer to start of variable strings and the pointer to end of item. An address tag on the OVERHEAD will be defined as the core origin of the map.

The maximum possible length of a fixed length field is 15_{10} words plus 17_{10} bits.

THE IOT'S:

There are 3 IOT's used to manipulate items. The first of these is "GET". "GET" is used to retrieve a pointer to a field value in an item. "GET" is called with a pointer to a five-word parameter block in the AC. The parameters are:

First word: Address of a 3-word block of error returns. These words contain addresses to which GET will transfer. The errors are field number too high, subscript error, and attempt to retrieve a pointer, rather than information.

Second word: Field number of field to be retrieved -- address of first word of field map relative to beginning of item map.

Third word: Core address of item.

Fourth word: Core address of map.

Fifth word: Pointer to subscripts area.
The subscripts area contains one word for each level of the tree above the level of the field we are GETing. This word is the number of the branch we are interested in at that level. (Numbering starts with 0).

Sixth word: Core address of buffer area (see appendix C)

It is possible to get a subscript error on any level. GET returns in the AC a pointer to the information sought, and in the IO the 3rd word of the field map of this field--i.e., the sign bit specifies fixed length or variable (whether the pointer is a bit pointer or a byte pointer), the next 8 bits are the transform bits, the next 4 the word length, and the last 5 the bit length (these lengths are always 1 and 0, respectively, for variable information).

On error returns, for a field number error, the AC is the complement of the number of words beyond the end of the map.

On the subscript error, the AC is Ø if the subscript is exactly one too high, and positive if the subscript is more than one too high.

On the pointer error, a pointer to the pointer is returned anyway.

Typical "GET" calling sequence:

```
            LAW A
            GET = IOT I 2300
            .
            .
            .
A,          ERR
            ADMNAM
            ITM
            MAP
            SBS
            .
            .
            .
            .
ERR,        Ø                   /don't expect -CRASH-
            DONE
            Ø
            .
            .
            .
            .
SBS,        3                   /want fourth administration
                                  (subscripts start with Ø)
            .
            .
            .
DONE,       HALT                /finished
```

The next IOT is "BGI" - "BGI" sets up an "empty" item at a specified location in core--all bits in each fixed length field are set, and all pointers point to empty trees. "BGI" is called with the map address in the AC and the empty address in the IO, and is a one return IOT.

Typical calling sequence:

```
            LAW MAP
            LIO  (ITM)
            BGI = IOT I 2200
```

The third and last IOT is "PUT". "PUT" places information in the appropriate place in the item. "PUT", like "GET", is called with a pointer to a 5-word parameter block in the AC. Those are:

First word: A pointer to the information to be "PUT". (Byte pointer for variable information, bit pointer for fixed length) pointer is indirectable.

Second through Fifth words: As in "GET".

"PUT" has no error return but is an illegal IOT if any errors occur.

"PUT" first does a "GET" to find a pointer to the field, then places the information at the indicated spot. If "PUT" receives the "GET" error "subscript-one-too-high", the appropriate tree is expanded by one group, and the information is "PUT". Any other "GET" error, except a pointer error, causes a crash. A "GET" pointer error is ignored. "PUT" will not allow you to "PUT" a field more than one level below the lowest level that has previously been "PUT", and will tolerate the "GET" subscript error only on the level of the field to be "PUT". Any other situation will cause a crash.

Typical "PUT" calling sequence:

```
            LAW  B
            PUT = IOT I 2301
            .
            .
            .
B,          INFO
            ADMNAM
            ITM
            MAP
            SBS
            .
            .
            .
SBS,        3
            .
            .
            .
INFO,       TEXT/MISS JONES#/
```

APPENDIX A

Map Structure:

A map consists of a number of individual field maps.
Each field uses 3 words in the map.

Field structure:

Word Ø -

Bit Ø - Ø = primary - i.e. does not occur in any tree; that is, occurs only once in each item.
1 = secondary - i.e. may occur more than once in each item; that is, a repeated field.

Bit 1 - Ø = pointer
1 = information

On secondary only -

Bit 2 - Ø = not last level
1 = last level

Bits 3-9 - # of words in group

Bits 1Ø-17 - ptr to next higher level (relative to start of map - points to field map of first field in next higher level)

Word 1 -

Bits Ø 4 - bit position — Primary: Relative to beginning of info. in item.

Bits 11-17 - word position — Sec: Relative to beginning of group.

Word 2 - Bit Ø - Ø - variable (points to strings)
1 - fixed length

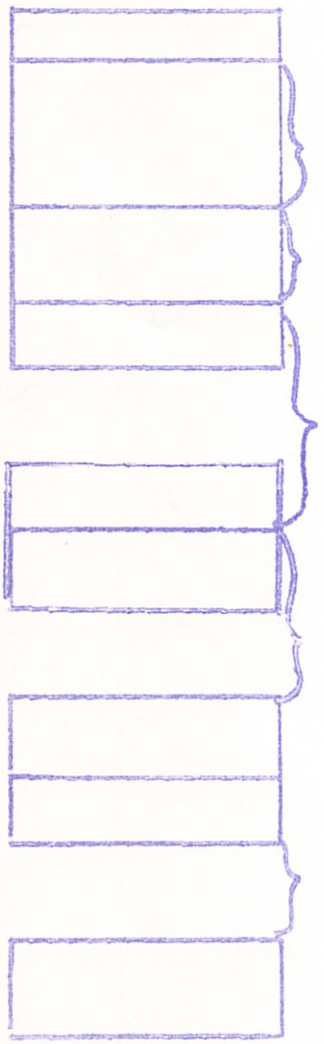
Bits 1-8 - transform bits (unspecified)

Bits 9-12 - length of info-words

Bits 13-17 - length of info-bits over last full word.

Variable fields must have bit position Ø, length 1 word.
Pointers must be fixed length -- 1-word, bit position Ø.

Map structure -



Word Ø
- Bits Ø-7 - # words in map.
- Bits 8-13 - length of primary info in item
- Bits 14-17 - overhead count

1st Field - ptr to string beg. (Primary pointer - fixed length - word Ø)

2nd Field - ptr to end of strings (Primary pointer-fixed length - word 1)

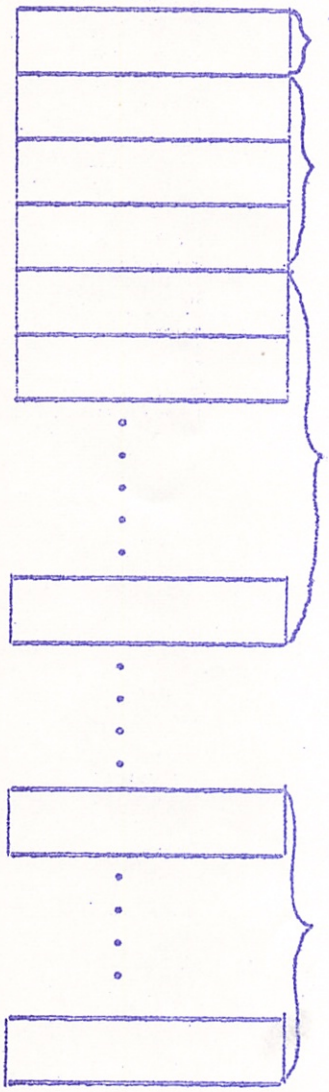
Tree beg. field maps

Primary pointers - fixed length
Word 2,...

Primary information field maps
(Fixed and Variable)

Trees

Tree map structure -



Ptr to next level field map
Sec. ptr fixed length -word Ø

Sec. information field maps

1st level

2nd level (similar to 1st level)

Sec. information field maps

last level

Note: Last level has no ptr to next level. However first word in level must be ptr to end of level, thus all word positions in last level of tree must be incremented by 1 to account for this.

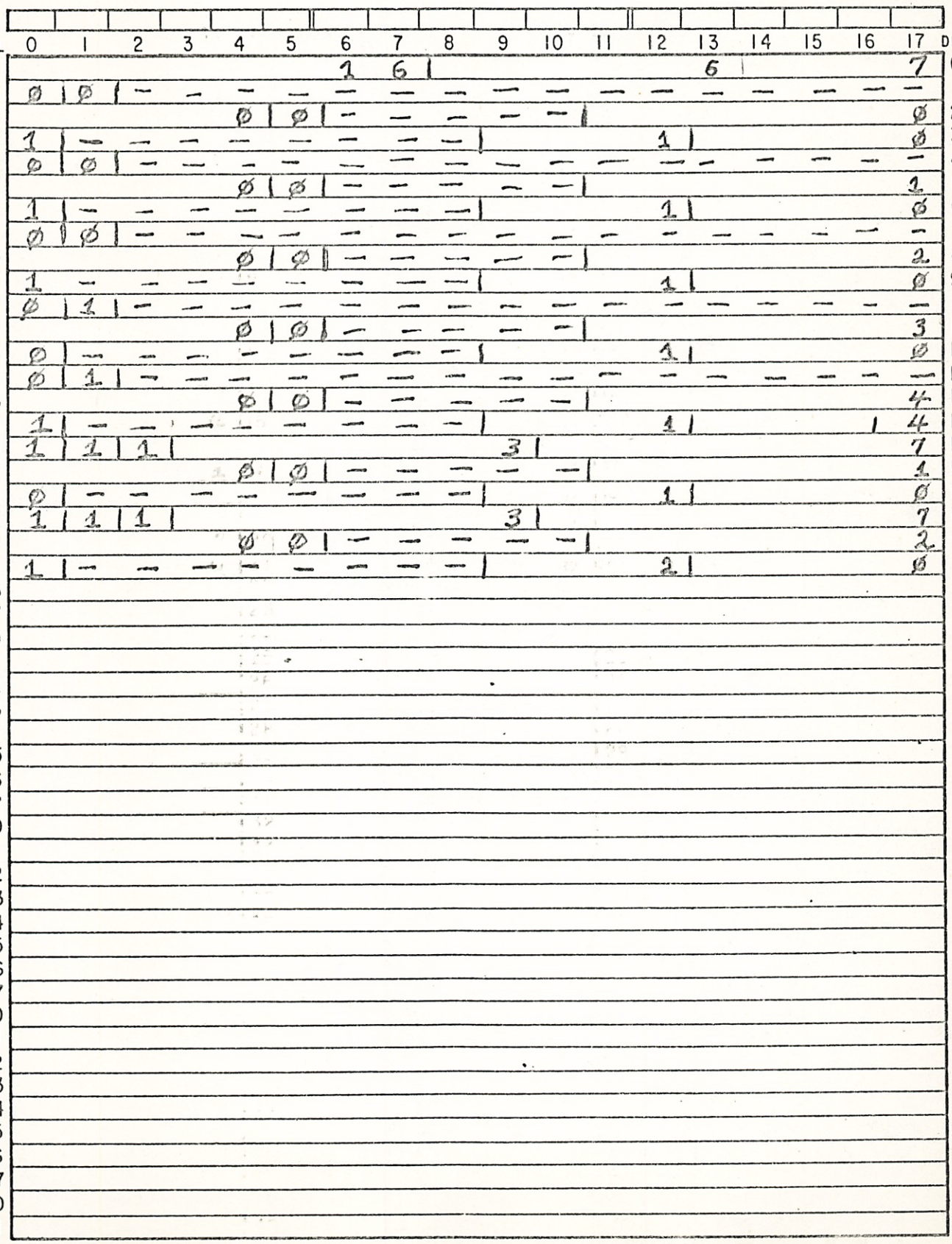
PUT AND GET
PAGE 20

Page_____

Title Example Map

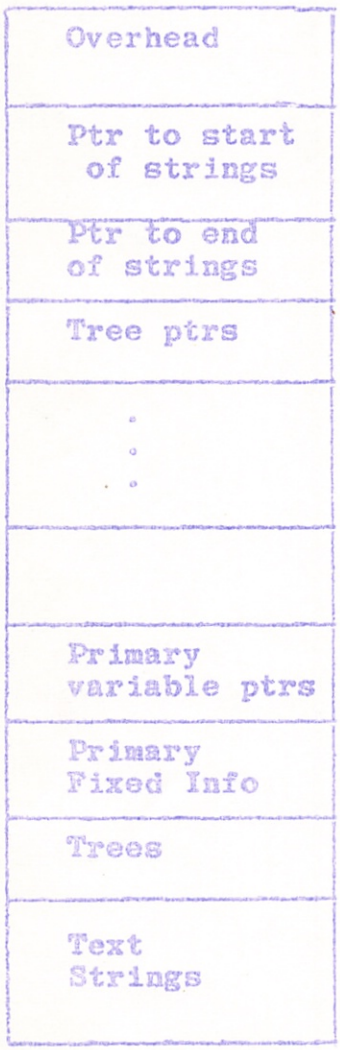
Date_____

By_____



APPENDIX B

General Structure of PUT and GET Item --



Relative to .

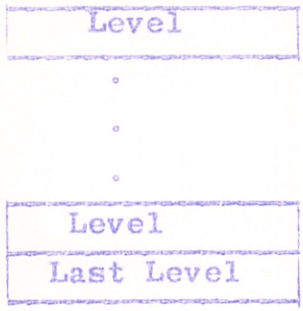
Byte pointer to first free byte -
Relative to start of strings

Ptrs to beginning of trees --
Relative to .

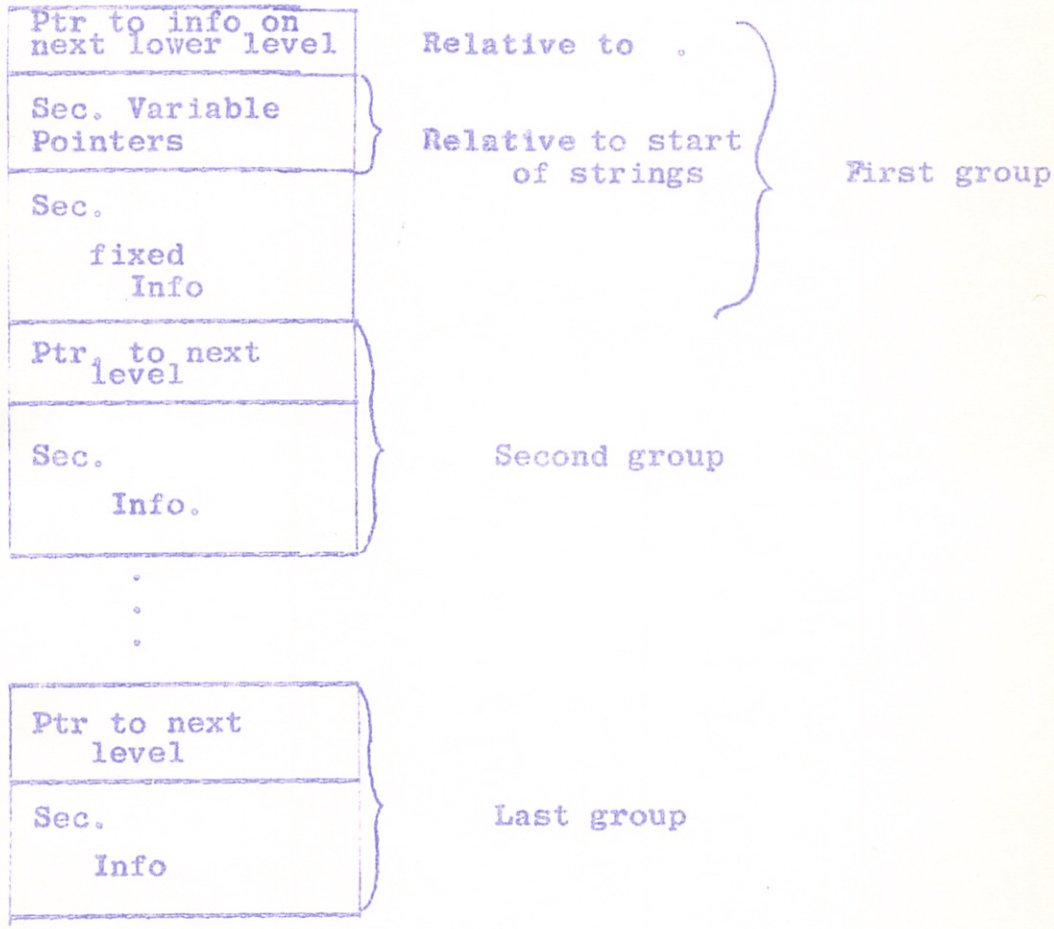
Relative to start of strings

See next page
Trees occur in order pointed to by tree pointers

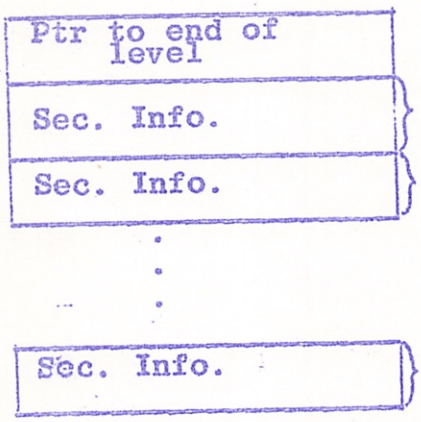
Tree structure



Level structure (all except last level of a tree)



Last level structure --



Relative to .

1st group

2nd group

last group

Page_____

Title Example Item Date_____

By_____

	TAG																		
	OCTAL	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	DECIMAL
	0																			0
	1																			1
	2																			2
OVER HEAD	3																			3
	4																			4
	5																			5
	6																			6
start of str.	7																	2	0	7
End of item	10	J	M	P														1	6	8
Tree Ptr	11																		4	9
Drgnam	12	L	A	C															0	10
Drgnum	13																		0	11
	14													4	3					12
end of str.	15																		1	13
Admnum	16	J	M	P															3	14
	17					2/	3/	65												15
Admtd	20					02:	00	P.M.												16
2nd	21	A	D	D															7	17
p:	22					2/	3/	65												18
	23					03:	00	P.M.												19
	24	L	A	C															3	20
3rd Grp.	25					2/	4/	65												21
	26					04:	00	P.M.												22
	27			Θ						P						E				23
	30			N						I						C				24
Text	31			I						L						L				25
String	32			I						N						Θ				26
	33			M						I						S				27
	34			S						△						J				28
	35			O						N						E				29
	36			S						Θ						M				30
	37			1						S						D				31
	40			△						S						M				32
	41			T						T						H				33
	42			Θ						M						1				34
	43			S						S						△				35
	44			S						M						I				36
	45			T						H						Θ				37
	46																			38
	47																			39
	50																			40
	51																			41
	52																			42
	53																			43
	54																			44
	55																			45
	56																			46
	57																			47
	60																			48
	61																			49

APPENDIX C
DRUM FEATURES

PUT and GET are also capable of working with an item directly off the Fastrand drum. The following is a more detailed description of the operation of the IOT GET which shows how to use these drum features.

GET = IOT I 2300
AC = Parameters address
Bits:
Bit 12 = 1 PUT
= 0 GET
Bit 13 = 1 Info preserved in buffer from previous CALL to PUT or
0 clean GET buffer area
Bit 14 = 1 BGI before PUTing
0 No BGI
Bits 15-16 Third of drum
Bit 17 Trap mode
Bits 15-17 are exactly as in Fastrand IOTs. Fastrand errors are treated as indicated by bit 17, but GET errors are treated as they presently are.
Bit 13 may be set only if a previous call to PUT, GET, or BGI used the same buffer area, the same item, and the same map. If any intervening calls have used the same buffer, or a portion thereof, with a different item or map, bit 13 must be clear.
Bits 12 and 14 are interpreted as follows:

Bit 12	Bit 14	
0	0	GET only
1	0	PUT only
0	1	BGI only
1	1	BGI, then PUT

Parameters:

Word 0:		GET errors add PUT info add
	1:	Field number
	2:	Item address
	3:	Map address
	4:	Subscripts area
	5:	Buffer address

The item and map addresses are communicated in the following manner:

A positive number is taken as a core address

A negative number is taken as the complement of a drum address

Both item and map must be on the same third, if both are on the Fastrand

Due to time-sharing bugs in adding words in the middle of an item, any program using the drum features should HOLD the drum address of the item before using PUT or GET. The item may be released upon returning from PUT or GET. The rewrite number of such an item must not be changed while it is held, as PUT may leave a partially modified item on the drum. Thus all programs which reference an item which may be used by the drum feature of PUT, should use HOLD before reading and HOLD must be used before rewriting such an item.

All Fastrand errors will be returned directly to the user. If bit 13 is set on the call, and the item has been rewritten since the previous call, either error 2000, rewrite number or 4000, compare error on scatter-gather may be returned to the user. Both of these errors mean the same thing, and the recommended recovery is to reexecute the PUT or GET without bit 13 set.

A PUT-GET buffer area can be established by calling the macro PUTGET, which is on the ptape. PUTGET has 3 arguments, the last one optional. The first 2 arguments are the lengths of buffers for the item and the map respectively. Either of these may be 0 if they will not be used (i.e., the item or map is in core). The third is the overhead count, which, if supplied, will mean that this buffer may be used only for items with this overhead count or less. If it is omitted, enough space will be reserved for any possible item's overhead. The minimum length of the item buffer is the length of the longest piece of information that will be expected in core at one time.

If it is desired to examine information which will not all fit into the buffer, the location of the information in the item may be found in the scatter gather table located at the beginning of the buffer. The format of this table is:

```
BUF,    BUF + n          /OVERHEAD READ AREA
        OVC + 2          /LENGTH
        I                /SKIP
        RELAD            /LENGTH
        BUF+OVC+2+n      /RD AREA
        m                /LENGTH
        -0               /TERMINATE
```

The first command pair reads the overhead of the item, plus the two words of PUT and GET overhead at the beginning.

The next is a skip to the beginning of the piece now in core.

The next pair describes the location and length of the piece now in core. If this piece is at the end of the item, the length may be longer than the actual length of information in core.