DYNAMIC STORAGE ALLOCATION SYSTEM

CERN LIBRARIES, GENEVA

CM-P00059730

USER GUIDE
AND
REFERENCE MANUAL
TABLE OF CONTENTS

1. INTRODUCTION 1
2. DESCRIPTION OF THE DYNAMIC STRUCTURE 2
3. USAGE OF THE SYSTEM 4
4. USAGE OF ZBOOK TOGETHER WITH HBOOK 10
5. ERROR MESSAGES 12
6. HOW TO ATTACH THE SYSTEM AT CERN 13
7. EXAMPLE \hspace{1cm} GENERATION OF ARTIFICIAL TRACKS 14
The implementation of large programs requires an optimal management of the memory in order to minimize both the space allocation and the execution time. The realization of these two objectives, which are sometimes contradictory, has led us to write a memory management system, operating on a dynamic structure defined as a region of memory in which the user can create data blocks of variable lengths. The main characteristics of the system are the following:

- possibility of using several dynamic structures which can be located anywhere within labelled or unlabelled common blocks or within dimensioned arrays
- identification of data blocks inside a structure by mnemonics
- possibility of equivalencing the first block of the structure with temporary data (working space). This allows direct addressing by mnemonics which is simpler and faster than an indexed addressing
- compatibility with any other package or system
- easy utilization
- the package is written in FORTRAN and its length is small (a few hundred words on the CDC 7600)
- possibility of using dynamic memory extension by means of a dialogue with the HBOOK package (F)

(*) HBOOK CERN - DATA HANDLING DIVISION
DD/77/9 July 1977
2. \textbf{DESCRIPTION OF THE DYNAMIC STRUCTURE}

The overall layout and the detailed organization of the data blocks in the dynamic structure are given in figure 1 and 2. During the execution of a program the blocks which are no longer used can be deleted, thus releasing space for the creation of new blocks.

In each subprogram referencing one or several of the data blocks in the structure, the user has to provide the following set of Fortran statements (for example it may conveniently be a COMDECK with UPDATE or a CDE with PATCHY)

\begin{verbatim}
COMMON/NAME/A(1),ID1,ID2, ...,IDN,WS(5000)
DIMENSION IA(1000)
EQUIVALENCE (A(1),IA(1))
\end{verbatim}

This can be schematized as follows

\begin{center}
\begin{tabular}{cccccc}
A(1) & ID1 & \ldots & \ldots & IDK & WS(1) \\
1 & 1 & 1 & \ldots & 1 & 1 \\
1 & 1 & 1 & \ldots & 1 & 1 \\
\hline
\hline
Identifiers & Working space & Data blocks \\
\hline
I & II & III \\
\end{tabular}
\end{center}

\textbf{Figure 1}

- IA or A is the name of the structure
- ID1, \ldots, IDN are the identifiers of the N data blocks. The values of ID1,\ldots, IDN are the pointers to the corresponding blocks, which are computed by the system.
- The working space is a special purpose block which can be created with varying length, but which is located at a fixed address starting immediately after the table of identifiers. This allows one to equivalence the working space with local variables or arrays.
- The starting addresses of the data blocks of variable lengths are stored in the corresponding identifiers (pointers). These blocks are stored in the order of their creation and not necessarily in the order of the identifier table.
The detailed organization of the structure is the following:

- **A** = the region of storage for all blocks in this structure
- **N** = length of the structure
- **NB** = max. number of data blocks
- **NW** = length of working space

<table>
<thead>
<tr>
<th>STORAGE LOCATION</th>
<th>CONTENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A(1)</strong></td>
<td>length of structure (=N)</td>
</tr>
<tr>
<td><strong>A(2) = ID1</strong></td>
<td>Pointer to data block identified by ID1</td>
</tr>
<tr>
<td><strong>NB words</strong></td>
<td></td>
</tr>
<tr>
<td><strong>A(NB+1) = IDNB</strong></td>
<td>Pointer to data block identified by IDNB</td>
</tr>
<tr>
<td><strong>NW words</strong></td>
<td></td>
</tr>
<tr>
<td><strong>A(NB+2)</strong></td>
<td>Special purpose block with fixed first address</td>
</tr>
<tr>
<td><strong>A(NB+NW+2)</strong></td>
<td>Data block created first</td>
</tr>
<tr>
<td><strong>A(......)</strong></td>
<td>Data block created last</td>
</tr>
<tr>
<td><strong>A(last)</strong></td>
<td>Unused space</td>
</tr>
<tr>
<td><strong>A(last+1)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>A(N-4)</strong></td>
<td>Logical unit number for output</td>
</tr>
<tr>
<td><strong>A(N-3)</strong></td>
<td>Maximum number of block (=NB)</td>
</tr>
<tr>
<td><strong>A(N-2)</strong></td>
<td>Length of working space (=NW)</td>
</tr>
<tr>
<td><strong>A(N-1)</strong></td>
<td>Number of unused locations</td>
</tr>
<tr>
<td><strong>A(N)</strong></td>
<td>Pointer to first unused location (=last+1)</td>
</tr>
</tbody>
</table>

**Figure 2**
3. USAGE OF THE SYSTEM

The operations described below allow the creation and the management of a dynamic structure with various data blocks. In the following it is assumed that the dynamic structure is implemented in a region of storage starting at A(1).

```
+-----------------+----------------+----------------+
| 1               | 1              | 1              |
|                 | 1              |                |
| 1               | 1              |                |
| +-----------------+----------------+----------------+
| 1               | 1              | 1              |
| 1               | 1              | 1              |
| 1               | 1              | 1              |
| 1               | 1              | 1              |
| 1               | 1              |                |
| 1               |                |                |
| I = 1           |                |                |
| A(1) ID1        | IDN1 WS ID3 ID4| ID1            |
| 1               |                |                |
| +-----------------+----------------+----------------+
| Identifiers      | Data blocks    | Control words  |
```

Figure 3

When the system is initialized (ZINIT), all pointers are set to zero. At the creation of a data block (ZBOOK,ZBOOK0), the system sets the value of the corresponding identifier to the relative address given to that block in the structure (fig. 3). One can then store the data in that block by a statement of the form (for example for the block ID4)

\[
A(ID4 + I) = \text{VALUE}
\]

where \( I \) represents the relative address of the data value within the block, i.e. \( A(ID4 + I) \) is the \( I \)th data word of the data block identified by ID4.

Similarly, one can retrieve the value of the \( J \)th word of the data block by the statement

\[
\text{VALUE} = A(ID4 + J)
\]

```
+-----------------+----------------+----------------+
| 1               | 1              | 1              |
| 1               | 1              | 1              |
+-----------------+----------------+----------------+
| CALL ZINIT(A,WS,N) | 1              | 1              |
+-----------------+----------------+----------------+
```

This is the initialisation subroutine. A and WS have been defined in figure 1. \( N \) is the total number of words of the structure.

The logical unit number for printing and error messages is set to the value pertaining to the computer used. It can be changed by the user by setting \( IA(N-4) \) to the desired value after initialization by ZINIT (see figure 2).

The number of words NW for the working space is set to 0.
3. **USAGE OF THE SYSTEM**

+----------------------------------------+
| 1  | CALL ZWORK(A,NW)                     |
|    | 1                                   |
+----------------------------------------+

This subroutine reserves NW words for the working space. The first usable word of this working space is WS(1).

The working space length can be modified at any time of the execution of a program via the call to ZWORK. Calling ZWORK does not alter the contents of the words remaining in the working space.

In general, it is very convenient to equivalence local arrays of a subroutine with the working space provided its length has been set sufficiently high to accomodate the local arrays.

+----------------------------------------+
| 1  | CALL ZBOOK(A,ID,L)                   |
|    | 1                                   |
+----------------------------------------+

The subroutine ZBOOK creates a data block

ID is the identifier of the block. Its name must be one of those defined in the identifier table. On output its value represents the relative address of the block in the dynamic structure.

L is the number of words reserved for the user in the block

For example, in order to set the first word of the block to the value VALUE, one can write

\[ A(ID + 1) = VALUE \]

**Remarks**

- Before the call to ZBOOK, ID = 0 (preset by ZINIT). After the call, ID will be the pointer to the block, unless insufficient space is available in the dynamic structure, in which case ID remains zero.

- In fact, a call to ZBOOK reserves L+2 words, L words for the user and two words for the system (the first and the last of the block) as shown below

\[
\begin{align*}
IA(ID) &= L+2 & \text{total length of the block} \\
IA(ID+1) &= \quad & \text{L words available for the user} \\
\vdots \\
IA(ID+L) &= \quad & \\
IA(ID+L+1) &= \quad & \text{serial number of the identifier ID} \\
& & \text{in the identifier table}
\end{align*}
\]

+----------------------------------------+
| 1  | CALL ZRESET(A,ID)                   |
|    | 1                                   |
+----------------------------------------+

This call resets to zero the contents of the user area in the block ID.
CALL ZBOOK0(A,ID,L)

The subroutine ZBOOK0 has the same effect as ZBOOK but in addition it resets the user area to zero. The two statements

CALL ZBOOK(A,ID,L)
CALL ZRESET(A,ID)

are equivalent to the call to ZBOOK0.

CALL ZLEFT(A,NLEFT)

By this call the user can enquire about the amount of storage space unused so far. The system returns in NLEFT the number of words still available for the creation of new blocks.

In fact if ZDROP was never called NLEFT = IA(N-1).

CALL ZPUSH(A,ID,NPUSH)

This call modifies the previous length of the block ID by NPUSH words. NPUSH may be positive or negative for increasing or decreasing the block length. In addition, this call has the effect of causing a re-adjustment of the pointers and a shift (by NPUSH words) of the positions of the data blocks created after the block ID.

ZPUSH does not modify the user contents of the data block (except for any words released when NPUSH is negative).

Important remarks

If insufficient space is available for increasing the block, ZPUSH will give an error message and then return. The program should not assume that the block was extended, but should check the amount of available space prior to the ZPUSH call.

This can be done in one of two ways

i) a call to ZLEFT and a check that NLEFT is greater than NPUSH
ii) a direct check that IA(N-1) (see figure 2) is bigger than NPUSH, if ZDROP was never called.
CALL ZDELETE(A,ID)

This call causes the deletion of the block ID and triggers a squeeze of all the remaining blocks in the structure in order to merge the unused space of the deleted block into a single area of adjacent locations at the end of the dynamic structure.

The identifier ID is set to zero.

CALL ZDROP(A,ID)

This call instructs the system to mark the block ID, so that the corresponding space can be freed for later use. The actual deletion of the block will take place only when the space is needed, (for example for the creation of a new block).

The identifier ID is left unchanged by ZDROP, but is set to zero when the block is actually deleted.

After ZDROP the first word of the block IA(ID) is negative and its absolute value is still the length of the block.

Only if during the creation or extension of a block no more space is available then the garbage collector of ZBOOK will delete all dropped blocks from the dynamic area. This procedure is faster than individual calls to ZDELETE. When the blocks dropped are effectively deleted the corresponding pointers are reset to 0.

CALL ZAGAIN(A,ID)

In many applications the situation arises where some data blocks have to be created only once during execution, whereas some other blocks need to be created repetitively, each time starting a sequence of operations for the treatment of new data. The user can best take advantage of this situation by organizing his program so that the long lived data blocks are created first, which has the effect of locating these blocks in the lowest part of the dynamic structure. The short lived data blocks created afterwards can be deleted by a single call to ZAGAIN.

ID is the identifier of the first short lived data block. This call has the effect of deleting the block ID and all the blocks created afterwards, thus allowing a fresh start of the sequence of operations on new data.

REMARK

The action of ZAGAIN is equivalent to a set of calls to ZDELETE (one for each block to be deleted), but it is much faster since it amounts merely to resetting to zero the pointers of the blocks to be deleted.
3. USAGE OF THE SYSTEM

CALL ZPRINT(A,ID,LABEL,FORMAT,L1,M)

Causes the printing of the contents of the block ID.
- LABEL up to 20 characters of title
- FORMAT one of the following forms
  1HI - the contents of the block are printed with the format 10 I10, the floating point numbers being automatically converted by the system to integers before printing
  1HF - the format 10F10.3 is used and integers are converted to floating point numbers
  1HE - the format 5E15.7 is used and conversion of integers is performed
  1HB - this stands for one of the following formats, depending on the computer used -
    5020 for CDC machines, or when the octal representation is used
    1028 for IBM machines or any computer using the hexadecimal representation
  1HA - one of the following alphanumeric formats
    5A10 or 20A4 depending on the machine used
- L1 relative address of the first location in block ID to be printed
- M number of locations to be printed

Remarks
A) the logical unit for output is initialized in ZINIT and can be changed at any time (IA(N-4))
B) subroutine arguments can be omitted on some computers. In this case an argument may only be omitted if all the succeeding arguments are also omitted.
- If M is missing or zero, the block ID is printed from L1 to its end
- If L1 is missing or zero all words of the block are printed
- If FORMAT is missing or zero 1HF is assumed
- If LABEL is missing or zero no title is printed
- If LABEL is not missing or zero the last character must be a dollar ($)
- If ID is missing the full structure will be printed
- If ID is zero, L1 and M not missing M words of the structure are printed starting at the address L1.
3. USAGE OF THE SYSTEM

CALL ZSTORE(A,ID,LUNIT,LABEL)

This call has the effect of storing the contents of the block ID on the logical file LUNIT. LABEL is an identifier of the block ID which will be used in order to retrieve the corresponding information with ZFETCH.

LABEL can be of any type (integer, floating point, alphanumeric ....).

CALL ZFETCH(A,ID,LUNIT,LABEL)

This call is the reverse of ZSTORE. It causes the retrieval of the block labelled LABEL from logical file LUNIT and creates the block of identifier ID in order to store the corresponding information.

Remark
It is not necessary to rewind the file LUNIT before this call.

CALL ZDEBUG(A,LUN,IERROR,LABEL)

It may happen that the user overwrites the contents of some important parts of the structure A. It is sometimes difficult to discover the source of the error. If one suspects something wrong this routine helps the user, telling him where the error is located:

- LUN is the logical unit into which the debugging information has to be written
- IERROR is a flag = 0 if no errors are detected (≠ 0 otherwise)
- LABEL is a hollerith string (up to 6 characters) used to label the printed error message

This routine can be called at various places in the debugging phase of a program. When everything seems correct the flag IERROR is always 0 and no printout is generated on the file LUN.

If something illogical appears in the pointers and the organisation of the memory, several messages will be written on the file LUN starting with ( i.e. LABEL = 5HSUBR1 )

**** ZDEBUG IS CALLED AT SUBR1 ****
and then the values of pointers with the localisation of the wrong one.

If an error is detected subsequent calls to ZDEBUG will be ignored.
As mentioned in the introduction, the simultaneous usage of the packages ZBOOK and HBOOK offers additional possibilities such as dynamic extension of memory space and automatic checking of available space. In order to fully exploit these advantages, an additional set of subroutines is available for the user. These can only be used with a dynamic structure located in the blank common, since this is the storage place normally used by HBOOK for its data. The sketch below shows the two data storage areas in the blank common.

//A(1)

--- Zbook area --- Hbook area

The boundary between the two areas can be moved around quite easily with the utility routines described below.

CALL YINIT(A,WS,N)

The parameters have the same meaning as in ZINIT. This call causes two calls, one to the HBOOK subroutine HISTGO, i.e.

CALL HISTGO(N)

which in fact defines the first word usable by HBOOK as

A(N+1)

and the second to ZINIT, i.e.

CALL ZINIT(A,WS,N)

which reserves N words for the storage of the ZBOOK data.

On the CDC machines, automatic memory extension takes place if N is bigger than the actual length of the blank common.
These routines have the same set of parameters as the routines ZBOOK, ZBOOKO, ZPUSH, ZWORK, ZFETCH described earlier. In addition, when insufficient space is available for the reservation or extension of a block an automatic shift of the boundary between the two storage areas is performed by the system.

This is the communication subroutine between HBOOK and ZBOOK.

NMOVE (positive or negative) is the number of words that have to be added (subtracted) to the dynamic structure via a call to HISTGO.
When an illegal request is detected, a message is printed with the corresponding error number:

100 - not enough space left for booking or extending a block. This error suppresses the actions of the calls to ZBOOK, ZBOOKO, ZPUSH or ZWORK. The control of the remaining space can be done with a call to ZLEFT prior to any reservation or extension of a block.

200 - attempt to create a block with an already existing identifier. The creation is suppressed. One can check whether a data block exists or not, since a block which does not exist has its corresponding ID=0 and can thus be created.

300 - the ID parameter of a call to ZAGAIN refers to a non-existing block.
At CERN the dynamic storage allocation system is stored as a library on a 6000 permanent file and can be obtained by the CDC 7600 control cards

```
FIND,GENLIB,ID=PROGLIB.
LIBRARY,GENLIB.
```

on the 6000 the corresponding file is GENLIB6

on the IBM 370 the library is

```
CR.PUB GENLIB
```

The source of the ZBOOK system is available for various computers (CDC, IBM, UNIVAC) and can be obtained on request from the authors or the CERN PROGRAM LIBRARY.

The source can be obtained as an UPDATE OLDPL, as a Patchy PAMFILE or as Fortran image cards.

Acknowledgements

We are indebted to Mme M. Brun who has written a COMPASS version of some routines of this package.

We would like to express our thanks to Miss R. Willibald and F. G. De Billo for typing this document, which was produced with the text-editing program BARBASCII.
EXAMPLE

GENERATION OF ARTIFICIAL TRACKS

In order to study the acceptance of a particle detector a program performs the following operations

- event generation
- generation of track coordinates by a step by step process
- intersections of the tracks with the chambers
- acceptance tests

All the blocks identifiers are stored in the blank common as follows

A  name of the dynamic structure
WS  working space
KEV  block containing general information relative to one event
KTITLE  block of long lived data necessary to carry out the computations
KINEM  blocks of kinematical quantities
KTRACK  blocks of track coordinates
KHITS  blocks of intersections

In addition we define

NPART  number of particles
NDET  NUMBER OF DETECTORS

PROGRAM SIMUL

C
DIMENSION A(10000), COMMON/A(1),KINEM(10),KTRACK(10),KTITLE,KEV,KHITS(10),WS(1)
EQUIVALENCE(A(1),IA(1))

C
INITIALISATION OF THE DYNAMIC STRUCTURE
CREATION OF A BLOCK FOR THE LONG LIVED DATA
(1000 WORDS REQUIRED)

C
CALL ZINIT(A,WS,10000)
CALL ZBOOK(A,KTITLE,1000)

C
FILL IN THIS BLOCK AND PRINT IT

C
READ ...(A(KTITLE+1),I=1,1000)
CALL ZPRINT(A,KTITLE,$GHTITLE$,1HF,0,0)

C
INITIALISE EVENT COUNTER AND WORKING SPACE

C
NEVENT=0
CALL ZWORK(A,100)
NEVENT=NEVENT+1

C
CREATION OF THE BLOCK KEV TO RECORD GENERAL INFORMATION ON THIS EVENT

C
CALL ZBOOKO(A,KEV,5)
IA(KEV+1)=NEVENT
EVENT GENERATION
THE KINEMATICAL QUANTITIES ARE STORED ON AN EXTERNAL
DEVICE. READ IN THE WORKING SPACE THEN CopIED IN THE
APPROPRIATE BLOCKS. WE ASSUME AN EVENT WITH 10
PARTICLES AND 10 WORDS PER PARTICLE

READ ...(WS(I),I=1,100)
NPart=10
DO 20 N=1,NPart
CALL ZBOOK(A,KINEM(N),10)
L1=10*(N-1)+1
L2=KINEM(N)+1
CALL UCOPY(WS(L1),A(L2),10)
CONTINUE

GENERATION OF TRACK COORDINATES
FOR EACH POINt SIX WORDS ARE STORED
SPACE IS RESERVED FOR TEN POINTS AT A TIME
TRACKING ENDS WHEN PARTICLE LEAVES CHAMBER
VOLUME

DO 50 N=1,NPart
CALL ZBOOK(A,KTRACK(N),60)

CHECK THAT THIS BLOCK WAS ACTUALLY BOOKED
AND PRINT ERROR MESSAGE IF NOT.

IF(KTRACK(N).NE.0) GO TO 30

PRINT ....
GO TO 10

30 KT=KTRACK(N)

INITIALISE POINT COUNTER
NPOINT=0
GET NEXT POINT

40 NPOINT=NPOINT+1

CHECK SPACE IS LEFT TO STORE THIS POINT. IF SPACE
INSUFFICIENT INCREASE LENGTH OF BLOCK

NLEFT=IA(KT)-6*NPOINT-2
IF(NLEFT.LT.6)CALL ZPUSH(A,KT,12)

STORE POINT

.......

ON LAST POINT DROP BLOCK IF KINEMATICAL QUANTITIES
NO LONGER USEFUL

CALL ZDROP(A,KINEM(N))

CONTINUE
INTERSECTIONS WITH THE CHAMBERS PROCEED AS PREVIOUSLY
FOR THE BOOKING AND THE FILLING OF THE BLOCKS.
AT THE END THE BLOCKS OF TRACK COORDINATES ARE DROPPED WHILE
THE BLOCKS OF INTERSECTIONS ARE SAVED FOR LATER USE ON
LOGICAL UNIT 'LUN' WITH LABEL 1000+BLOCK NUMBER

DO 60 N=1,10
CALL ZBOOK(A,KHITS(N),60)

CALL ZSTORE(A,LUN,KHITS(N),1000+N)

CALL ZDROP(A,KTRACK(N))

PERFORM ACCEPTANCE TESTS

IF LAST EVENT PROCESSED TERMINATE JOB
IF NOT, DELETE ALL BLOCKS EXCEPT THE LONG
LIVED DATA BLOCK CREATED FIRST AND REPEAT PROCESS

CALL ZAGAIN(A,KEV)
GO TO 10

STOP

END