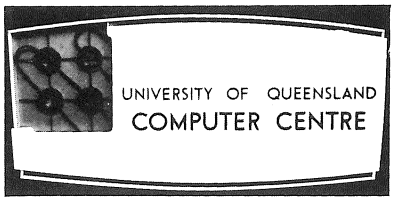


COMPUTER CENTRE

BULLETIN

Vol. 4, No. 6.
7 June 1971

Editor:
Mrs Sarah Barry

NOTICE TO COMPUTER CENTRE CLIENTS

BUILDING CHANGES TO THE CENTRE

A number of building modifications have begun in the Computer Centre area and it is expected that the modifications will take about six weeks to complete.

The modifications will give more space in the machine room which is presently overcrowded, particularly in the maintenance area behind the installation, and allow a number of activities in the machine room to be repositioned; for example, the card reader and line printer will be placed near a new counter/window for receipt and dispatch of computing work which should streamline the batch operation.

The clients' keypunches will be relocated firstly to a temporary location in the room opposite the Secretary's Office (in the corridor running to the west off the vestibule) and later to a room opening directly from the vestibule area. Also, a work counter will be placed in the vestibule for the convenience of clients.

The main building change will be the erection of a new interior wall parallel to the existing north-south glass partition separating the vestibule from the machine room, but six feet further to the west. This wall will have two counter/windows similar to tellers' windows; one for receipt and dispatch as mentioned above, the other for general and accounts enquiries.

While the building changes are in progress, the input trays and output racks will be placed in the vestibule against the western wall and enquiries will be shifted to the Secretary's Office in the corridor.

There should be no disruption of service during the building changes, but there may be minor inconveniences due to temporary relocation of activities; the Centre regrets any inconvenience which may be caused to clients.

RELEASE OF NEW FACILITIES

A new operating system has been implemented during May. This released the LIMIT command and incorporated a few minor changes to the system.

1. LIMIT

The general format for this command is:

```
                JOB
LIMIT     (PHASE)   {COST=}cost
                TASK
```

LIMIT can be abbreviated to LIM; JOB, PHASE and TASK to J, PH and T respectively; and COST can be abbreviated to C. The default option is TASK.

This command has been implemented to give users greater flexibility in controlling their expenditure within a job.

(a) Changing the job cost limit

The JOB option allows the cost limit for the complete job to be reset to the cost specified in the LIMIT command.

(b) Setting sub-limits within a job

The PHASE option sets a limit for a phase of the job, where a phase consists of a task or a number of tasks. The TASK option sets a limit for an individual task, where a task consists of the processing initiated by a single command. Each new phase or task limit will reset the previous limit.

example:

```
.LIMIT(PHASE) C=4.<cr>
-
-
-
.LIM (PH) 2.ØØ<cr>        ; resets previous limit
                            whether a task or phase
                            limit.
```

(c) Clearing sub-limits

The cost limit set for a phase or a task can be cleared by another phase or task limit command with a cost of zero.

example:

```
.LIMIT (T) 5Ø<cr>
-
-
-
.LIM Ø<cr>                ; clears task or phase limit
                            depending on which is
                            presently applicable.
```

2. ERROR MESSAGES

The following error messages have been added to the system.

(i) PHASE COST LIMIT EXCEEDED

The current task is stopped and no further tasks will be successful until the limit is cleared.

(ii) TASK COST LIMIT EXCEEDED

The task is stopped and the limit cleared.

The following system error messages have been altered.

(i) COST LIMIT EXCEEDED PLEASE LOGOUT

now reads as JOB COST LIMIT EXCEEDED. There is no need to logout now since the LIMIT command can specify a new limit for the job.

(ii) FILE LIMIT EXCEEDED

now reads as JOB FILE LIMIT EXCEEDED.

3. ERRORS IN COMMAND DECODER

The following errors have been corrected in this version of the operating system.

(i) Automatic compilation sometimes failed to select the most recent file for execution

(ii) A garbled error message was given if an illegal processor-program name was found.

PLOTTER AVAILABLE TO BATCH USERS

The digital plotter has now been made available to batch users with the release of a modified version of the Batch system.

Plotting via Batch is similar to plotting via remote terminals; plotter subroutines called by the user's program produce plotter output as a named file on disk. On completion of the program, this file is then transmitted to the plotter by a PLOT command referring to the file. The attached figure shows a sample deck for using the plotter via Batch. The PLOT command card must follow the completion of the user's program and must be followed by a file separator card.

Users should note the following arrangements for this initial release of the plotter via Batch.

(i) Plotter output is not left in the output shelves, but should be collected from the Operations Supervisor.

(ii) No charges will be made for the use of the plotter via Batch until further notice.

(iii) Plotter output from this version of Batch is not automatically identified by the user's project number. Users are therefore requested to plot their own project number as follows:

Include the following two statements immediately after the CALL to the PLOTI subroutine (which initializes the plotter file on disk).

```
CALL SYMBOL (Ø.5,6.Ø,Ø.5,'P-nnn',27Ø.Ø,5)
CALL PLOT(1.Ø,Ø.Ø,-3)
```

where 'nnn' is the user's three digit project number.

If, for example, the user's project number is 362, then

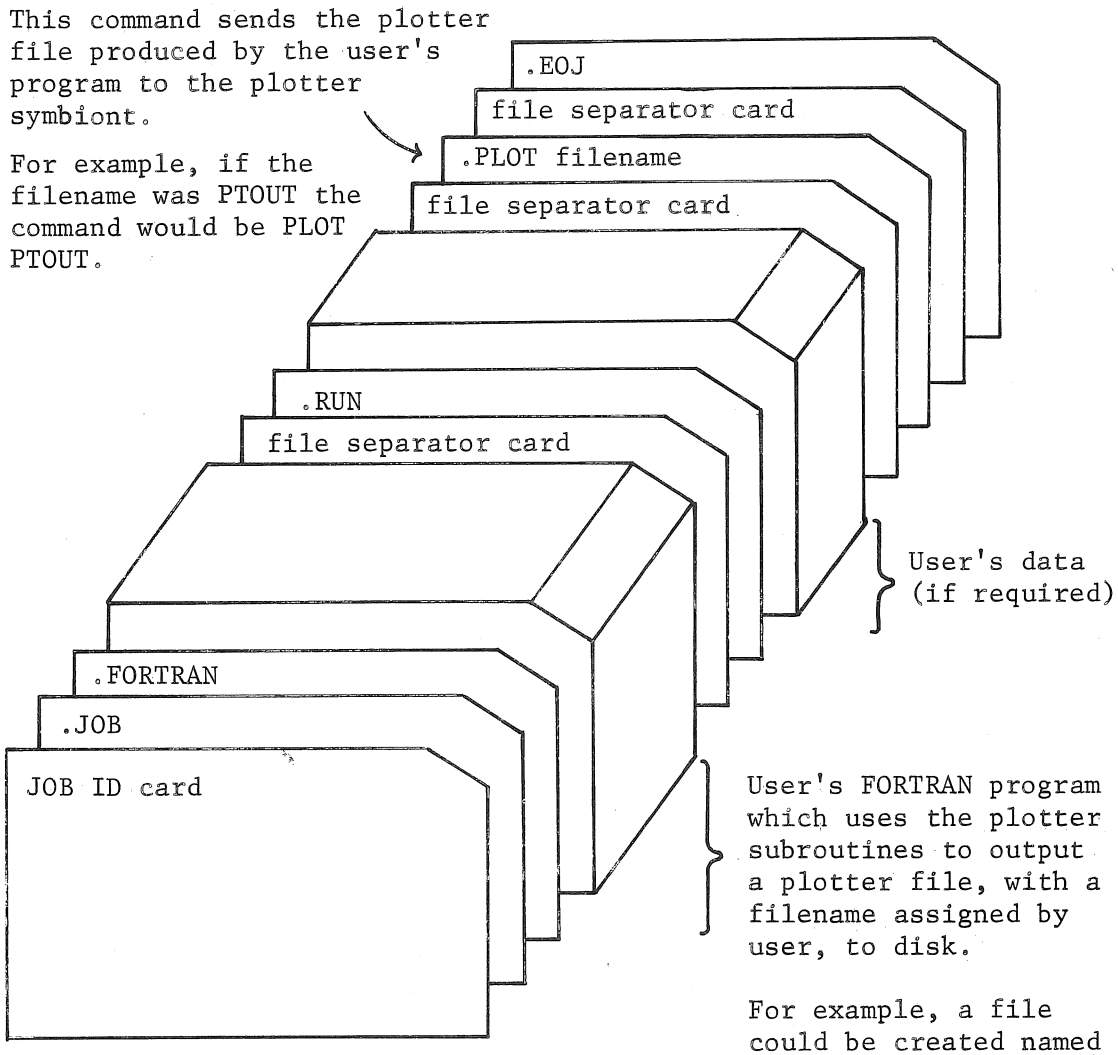
```
CALL SYMBOL(Ø.5,6.Ø,Ø.5,'P-362',27Ø.Ø,5)
```

will plot the identification P-362 on the user's output.

The following CALL to PLOT simply resets the origin of the plot one inch beyond this identification.

The revised Batch, currently being written, will automatically identify output.

Sample Batch Deck for Using the Plotter



FORTRAN IV ERROR

The error reported in the Bulletin, vol. 4 p. 31 on restoration of variables in a calling sequence appears to occur with DOUBLE PRECISION variables as well as COMPLEX variables. The solution is to put both DOUBLE PRECISION and COMPLEX variables at the beginning of any calling sequence.

BATCH ERROR

Since question marks have special significance to Batch, users are advised to use this character with caution in their programs. Some users have reported NO EXECUTION messages with error-free programs. This can usually be traced back to the use of '?' in their programs.

A collection of all the Batch, FORTRAN compiler and operating system errors that still exist has been culled from past Bulletins and Newsletters. It will be placed at the front of the blue binder containing past Bulletins, which is presently kept on the desk in the foyer of the Computer Centre. This has been done to aid users in tracking down causes for compilations and execution errors. The collection will be kept as up to date as possible.

CHESS

CHESS was obtained as an unsupported demonstration program in core image form and originally with word of mouth instructions only. Subsequently, the following brief instructions were located and they are listed in case any users may be interested in using CHESS. It is worth noting that a considerable amount of core and computations is required by CHESS and hence it will not be cheap to use.

1. To get CHESS from the system type

   .CHESS<cr>

2. Commands to CHESS Program

   Four commands exist for instructing the computer which side(s) it is to play. They are:

PS	Play self	Machine plays for both sides.
PW	Play white	Machine plays white; moves for black are typed in.
PB	Play black	Machine plays black; moves for white are typed in.
PN	Play neither	Machine plays for neither side; moves for both may be typed in.

If the machine is playing the side whose turn it is to move, it will automatically proceed and make its move. If not, machine may be instructed to make move anyway by typing

M

If it is desired to take back the last move made, the command is

U

This automatically enters 'play neither' mode as described above. The U command may be used repeatedly.

The machine may be instructed to produce various forms of printout as follows:

BD Produces a diagram of current board position.

PG Produces score of current game.

3. Typing in Moves

The machine accepts input in a form very similar to standard chess notation. The major differences are summarized.

(a) Check is not announced to the computer.

(b) The character 'X' or '*' is used to denote captures, '-' or space to denote moves.

(c) En passant captures are denoted by PXG (where G stands for ghost which is located one square behind pawn which advanced two ranks).

(d) Promotion is only to Queen and is not announced.

(e) For ambiguous moves the following options may be used to specify the move:

(i) Specifying the piece as K or Q (e.g. KN-B3).

(ii) Giving original square of piece moving (e.g. N(KN1)-B3).

and so on as in chess notation the only restriction being that each syllable must be a piece name or a complete square name, i.e. N/1-B3 is illegal.

The characters '(',')', '-' and space are identical in effect when used in a move.

If you type in an ambiguous move, the computer will tell you so, then retype the move. If you type an impossible move the computer will also tell you and will be ready to accept the corrected move. If the move is illegal because you are in check at the end of it, the computer will type illegal and be ready to accept the corrected move. If you type a legal move which was not the one you wanted, do the following:

(i) Type bell to suspend computation on unwanted move (computer will respond with QUIT).

(ii) Type U to take back unwanted move.

(iii) Type in desired move.

(iv) Type PW or PB as appropriate so that the computer will resume play.

Moves and commands are followed with a carriage return. Moves may also be followed by a tab. The computer types out '←' to acknowledge accepting a command.

4. Setting Computer Lookahead Parameters

Three basic parameters can be set from the Teletype and these affect the speed and strength of play.

SETW Takes a list of numeric arguments which specify how many moves wide the computer is to look. The first number is effective at play one, the second at play two and so on, the last being effective at all deeper plays.

SETD Sets basic depth of search in analysis of moves.

SETF Specifies maximum number of feedovers allowed along any line in analysis. (Feedover is when computer decides whether the position is worthy of deeper analysis than that specified by SETD above).

In tournament play (where a 2 2/5 minute per move average is required) common setting is

```
SETW 15  15  9  9  7
SETD  4
SETF  2
```

The computer will average about thirty seconds a move at settings of

```
SETW  6
SETD  4
SETF  2
```

this is the setting when the program is initially loaded. For blitz play the following settings are good:

```
SETW  6
SETD  2
SETF 11
```

The current setting may be typed out by typing the command as above, followed by '='. For example, SETW=.

At the end of the game, the command

RESET

reinitializes the game to the starting position, resets the clocks, and reverts to 'play neither' mode.

If a typing error is made part way through a command or move, it may be erased by using RUBOUT. Then retype the command or move in question.

A DYNAMIC STORAGE MANAGEMENT PROCEDURE

W.J. Whiten

Mr. Whiten is a Research Officer in the Department of Mining and Metallurgical Engineering. He has contributed previously to the Bulletin with an article on 'The Simulation of Mineral Treatment Processes'.

INTRODUCTION

A method is described for dynamically allocating and de-allocating arbitrarily sized blocks of consecutive words which will be referred to as vectors. No restraint is placed on the order of allocations and de-allocations. The aim was to provide a general purpose system and hence simplifications based on special properties, which are often possible, were not used. However this system is capable of providing a basis for the support of systems which take advantage of special properties.

This method differs from Ross (1967) in being designed for a small computer where it is not practical to provide a large number of options and from Knuth (1968) in being able to handle both requests for exact amounts of storage and not placing a lower limit on the size of vectors in the system.

DESIGN CRITERIA

The method was designed to satisfy the following criteria:

(a) The storage allocated must be provided as one continuous vector.

(b) Requests for storage allocation and de-allocation may be made in any order.

(c) Both large and small vectors are to be handled. The case of a large number of small vectors can occur.

(d) Storage is to be requested by words and the number of words is to be saved for future reference.

(e) Allocated vectors cannot be moved.

(f) Storage fragmentation (the reduction of unused storage to small pieces) is to be minimised. Ways of reducing storage fragmentation in particular cases are required.

(g) Efficiency with regard to speed and storage overhead is required.

It is clear that any system with the above characteristics can be made to fail, i.e. reach a condition where a request for storage cannot be satisfied. The aim was to implement a system that could handle most cases efficiently and with some assistance from the programmer would be able to handle the more difficult.

From the design criteria the following decisions were made:

1. The maintenance of any form of segregation by size was expected to increase the complexity of the system and very likely have other undesirable properties. Hence identical treatment was required for all vector sizes.

2. A word containing the length of the vector and some control information was required to be associated with each vector. Each vector was lengthened so that a control word called the header word, could be placed at the front of the vector. The implicit location of this header word meant that no additional storage was required for its access and linking to its vector.

3. The allocation and de-allocation of vectors should as far as possible take a constant time, i.e. not be affected by the number of vectors currently in the system.

4. No vector should be allocated with free storage on both sides of it, i.e. de-allocated vectors needed to be recombined before storage is reused.

5. Storage fragmentation could be reduced by keeping recently allocated vectors as close together as possible. This may allow larger free regions to develop in the other parts of the available storage.

6. Storage fragmentation could be reduced by allowing the programmer, if he desires, to use several storage blocks (which may if desired be nested). Hence each storage block required all the data for its operation associated with it.

7. The de-allocation of a vector needs to be independent of the storage block which contains the vector. Thus the de-allocation routine will require only the address of the vector to be de-allocated. This eliminated the need to save (implicitly) the address of the storage block associated with each vector.

8. A free region of one word can occur and hence accounting for free regions must allow for this case.

9. A linked list of free storage would require searching operations to satisfy 5 and also free storage would be divided into two types (on the list or too short). Hence it was decided not to link free regions but to search memory moving away from the most recently allocated region.

DETAILED DESIGN

The final design is:

Both allocated vectors and free regions (free vectors) are started by a header word which contains:

1. A free-vector indicator which is set if the vector is a free region.

2. A free-before indicator which is set if the preceding vector is free.

3. The number of words in the vector including the header word.

In addition to the header at the beginning, the free vector contains a copy of the header in the last free word. However in free vectors longer than one word the last free word sometimes has the free before bit set.

The storage block contains two control words as follows:

1. At the beginning of the block a pointer word gives the location immediately after the last vector to be allocated.

2. The end of the storage block is marked by a header word with the free vector bit not set and a zero word count. The free-before indicator is used as in the other header words.

The rest of the storage block is always filled with vectors. No two free vectors can be adjacent as they are combined as they occur. Figure 1 shows the typical contents of a storage block.

STORAGE ALLOCATION

This routine requires the address of the storage block to be used and the number of words required.

The word at the beginning of the storage block gives the location immediately after the last allocated vector. This location may, due to subsequent de-allocations, not be the start of a vector. If the free before bit is set in this location the preceding word contains a decrement to update the word at the beginning of the storage block. A location with the free before bit not set marks the end of this update chain and is always the start of a vector.

This vector is tested to determine if it is free and large enough, if it is not the program indexes forward over the vectors until such a free vector is found. Should the end of the storage block be encountered for the first time the search continues from the start of the storage block. If the end of the storage block is found a second time the allocation procedure has failed.

When a suitable free vector is found the requested vector is taken from the left portion and the remainder if any is made into a free vector. An exact fit causes the free-before bit of the following vector to be cleared.

Finally the pointer in the first word of the storage block is set to the vector immediately following the one just allocated.

STORAGE DE-ALLOCATION

The de-allocation routine requires only the address of the vector to be de-allocated. This routine is responsible for combining adjacent free vectors as they form. In the case where the vector given by the pointer word at the start of the storage block and the vector before it are combined enough information must be left for the start of the new vector to be found later by the allocation routine. The procedure for de-allocation is:

1. If the vector before is free combine the two vectors.

2. Set the free-vector bit in the header word and copy the header word into the last word of the new free vector. Set the free-before bit in the header of the following vector.

3. If the vector following is free combine the two vectors to give a free vector with copies of the header word at both ends. Then set the free-before bit in the last word of this free vector. (This is necessary if the second vector is both referenced by the pointer at the beginning of the storage block and it is one word in length).

SUMMARY OF STORAGE ROUTINES

The effect of the allocate and de-allocate routines is:

1. All adjacent free regions are combined as they are formed.

2. The search for sufficient free memory starts at the boundary of the most recently allocated storage and the least recently searched storage. Advantage is taken of any adjacent de-allocations and then if necessary a search is made moving into the least recently searched storage.

DISCUSSION

Of the original seven design criteria the first five are satisfied and so we now examine the remaining qualitative criteria.

Storage Fragmentation: As the search start, and hence the allocations, tends to move cyclically around the available memory the normal case is for the least recently searched storage to be also the least recently used for allocations and hence most likely to be free, i.e. the tendency is for new allocations to be made immediately to the right of the previous allocation

hence keeping recently allocated vectors in the same region. Should the performance be unsatisfactory the programmer may be able to get better performance by dividing his storage requests among several storage blocks. The programmer is able to predict his needs and hence may be able to take advantage of these predictions to get better storage organisation.

Storage-overhead: One word of control information is required for each vector. This word contains two indicators plus the length of the vector. In many cases the vector length is required by the user and hence may save a storage word elsewhere. The storage block uses only two control words. Both the allocate and de-allocate routines are quite short. The loss of storage due to fragmentation is dealt with above.

Execution time: The de-allocation routine takes less than a fixed maximum time. The allocation includes two searches: Firstly for the start of a vector which at worst contains one step for each de-allocate since the proceeding allocate. Secondly for a free region which is large enough. As this search is always into the least recently searched and usually least recently allocated region the average length is normally short. In some cases it may be possible to shorten the searches for free vectors by defining additional storage blocks within the original block.

The two special cases of last allocated first de-allocated (working storage for subroutines) and first allocated first de-allocated (delay line) are both handled so that a free vector of adequate size is readily available and unnecessary storage fragmentation does not occur. In both these cases the time taken is proportional to the number of allocations and de-allocations. A valid comparison with other methods of storage management can be made only for particular cases - usually requiring stimulation using the storage requirements of the particular case.

The main disadvantage of the system is the proximity of critical control information to the data. Should any of the vector headers be over-written by data the system cannot recover. This may increase the difficulty of debugging programs using this system.

CONCLUSION

This method of storage management was developed and implemented as part of a continuing project to add dynamic data structures to a fortran system operating on a relatively small computer. Considerable effort has been put into making this method as simple as possible. This has made implementation and testing much easier. The use of very few alternative paths through the program means that is soon thoroughly checked and then considerable confidence can be placed in the correct operation of the storage management procedures.

REFERENCES

ROSS, D.T. The AED Free Storage Package. Communications of the ACM, 10, 8(August 1967), 481-492.

KNUTH, D.E. The Art of Computer Programming, Volume 1. Addison Wesley, Reading, Massachusetts, 1968. 435-455.

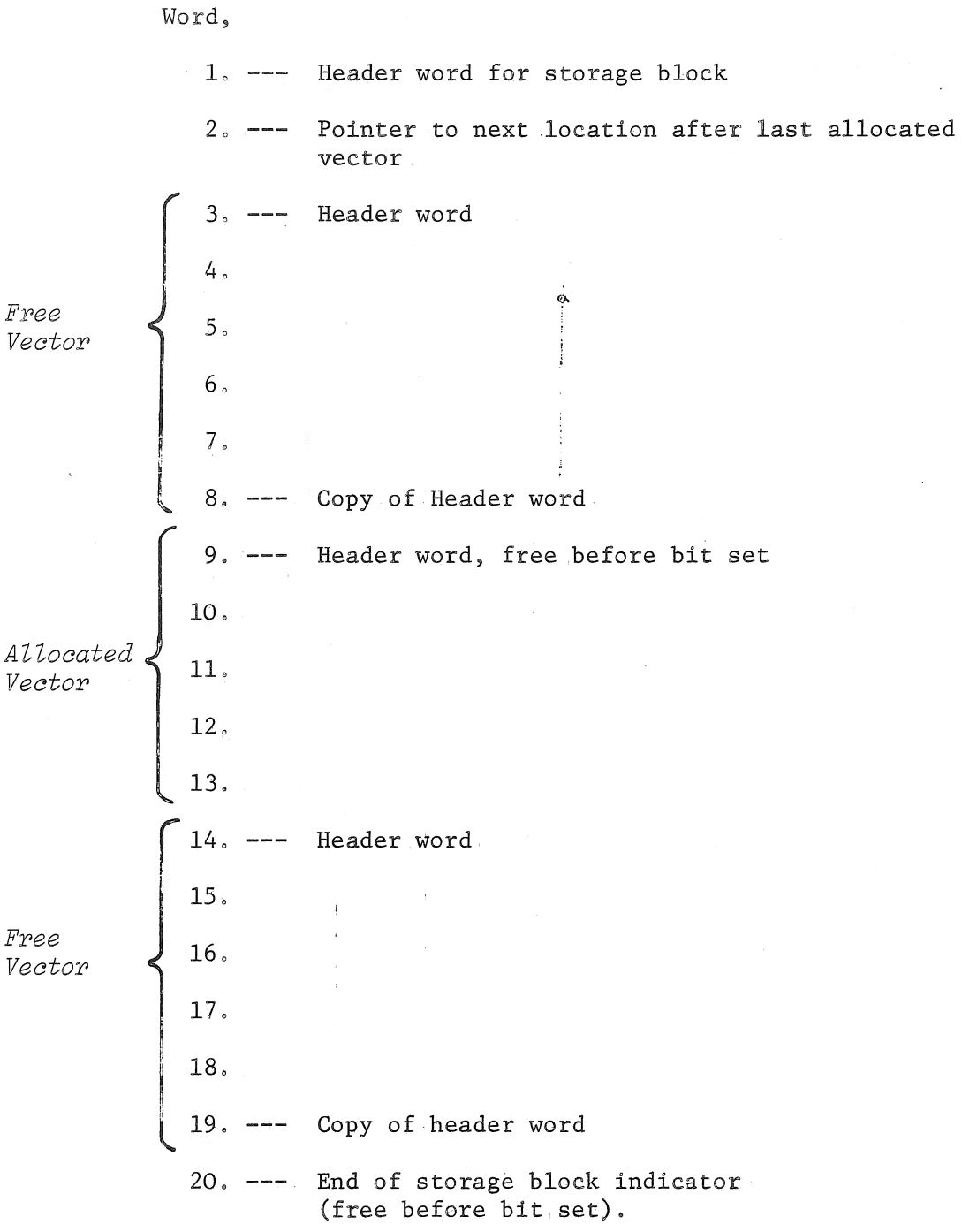


Figure 1. Typical Contents of a Storage Block

AUSTRALIAN COMPUTER SOCIETY (ACS) OVERSEAS VISITORS PROGRAMME

The first distinguished visitor to be brought to Australian under the ACS Overseas Visitors Programme will be Professor M.V. Wilkes, Professor of Computer Technology and Head of the Computer Laboratory in the University of Cambridge, England. Professor Wilkes was responsible for the construction of EDSAC 1, which was working early in May 1949. He was the first President of the British Computer Society and helped to found the International Federation of Information Processing Societies. With two colleagues, Professor Wilkes published the first book on computer programming: this appeared in 1951.

Professor Wilkes will deliver a public lecture and conduct a one-day seminar this month.

LECTURE - Tuesday 22 June 8.00 p.m. The lecture will be entitled "Twenty years of computer progress" and is designed to appeal to a general audience. Hawken Auditorium, The Insitute of Engineers, 447 Upper Edward Street.

SEMINAR - Wednesday 23 June 9.30 a.m. to 4.30 p.m. (Location to be announced)

The general topic of the seminar will be "The development of timesharing (or multiple-access) computer systems". Professor Wilkes writes, "the seminar is intended as a professional development seminar, and is designed to help people concerned in one way or other with the operation of large-scale computer systems to upgrade their technical knowledge, particularly on the software side. I have in mind the requirements of systems programmers and people who aspire to become systems programmers, and also those who have some knowledge of programming and of the operation of large systems, and who wish to obtain an insight into the working of operating systems. I also have in mind managers of multiple-access computer facilities who wish to develop their technical knowledge, and to understand the management aids that advanced systems are beginning to provide. I shall assume that those attending the seminars have had some experience of computers and have a thorough understanding of the principles of programming. Documentation for the seminar would be provided by my book on "Time-sharing computer systems" (Macdonald 1968).

There will be a fee of $1 for the lecture which includes supper and a nominal contribution to assist the Programme's expenses. For the seminar, the fee will be $11 for ACS members and $16 for others, including a buffet lunch and drinks at the conclusion of the day. Persons wishing to attend the seminar are requested to complete an application form available from the Hon. Secretary, ACS (Qld. Branch), G.P.O. Box 1484, BRISBANE, 4001.

A second distinguished visitor will be Professor B.A. Galler, Professor of Computer and Communication Sciences and Mathematics, University of Michigan who will visit Brisbane on 29 July 1971. Details will be announced in the next Bulletin.