
George C. Driscoll

Follow this and additional works at: http://preserve.lehigh.edu/engr-civil-environmental-fritz-lab-reports

Recommended Citation
http://preserve.lehigh.edu/engr-civil-environmental-fritz-lab-reports/2130

This Technical Report is brought to you for free and open access by the Civil and Environmental Engineering at Lehigh Preserve. It has been accepted for inclusion in Fritz Laboratory Reports by an authorized administrator of Lehigh Preserve. For more information, please contact preserve@lehigh.edu.
DYNAMIC STORAGE ALLOCATION

by

George C. Driscoll
Professor of Civil Engineering
Associate Director,
Fritz Engineering Laboratory
Lehigh University
ABSTRACT

A FORTRAN programming feature for the management of storage for dimensioned variables is described. Use of Dynamic Storage Allocation allows space requirements to be calculated and arrays to be packed in the least amount of space depending on the size of the current problem being run. It is especially valuable for programs which can be used for both very small and very large problems.

The paper describes the concepts, special program features used, planning necessary, how-to-do-it and writing the program. A sample program is written.

Additional hints, tips, and optional advanced features can be used by the advanced programmer.

The method is simple to program for anyone familiar with use of subroutines. It has been used to improve many programs limited by storage space or by changing patterns of storage space.
TABLE OF CONTENTS

ABSTRACT
Advertisement
1. Introduction
2. The Concept
3. Special Program Features Used
4. Your Planning Needed
5. How To Do It
6. Writing the Program
7. Sample Program
   (1) Save Parameter Lists for Array Names and Dimensions
   (2) Integer Arrays
   (3) Storage of Several Arrays in One Array
   (4) Reuse of Storage Space for Another Array
   (5) Obtaining Information About Field Length and Time Usage
   (6) Dynamic Storage on Other Computers
9. Acknowledgement
1. Advertisement

- Are you fed up with OFF hours processing of your large computer programs?
- Won't the Computing Center authorize you a large enough CM to get your work done?
- Do you detest waiting for output from your large FORTRAN program while dozens of small jobs pour off the line printer?
- Have you tried DYNAMIC STORAGE ALLOCATION??

* * *

1. Introduction

Dynamic Storage Allocation is a program feature easily adopted in many FORTRAN programs to permit the exact amount of storage needed for arrays to be used for each given problem -- and no excess storage. It is especially powerful when used in programs which are designed for large professional or commercial applications but are used on smaller problems for training purposes. Programs with a large amount of storage in dimensioned arrays are a candidate for this program feature. It has been demonstrated that some programs which formerly compiled and ran at well over 100K can run at 20K and less for many of their useful runs.

2. The Concept:

The main program is converted into a subroutine with variable dimensions for its arrays. All other subroutines use variable dimensions for arrays.
A small controlling program reads some integer variables defining the sizes of the arrays for the current problem being run. It calculates the required sizes of all arrays and determines first word addresses allowing all arrays to be packed into one single array in blank common. It then requests the exact amount of needed computer space by a call to a system routine followed by a call with a dummy parameter list to the main subroutine.

3. Special Program Features Used

Array names of variably dimensioned arrays and integer variable names of the variable dimensions must appear as formal parameters in the parameter list of any subroutine which processes the arrays.

A dummy array name is declared as blank common with a dimension of one in the controlling program. (COMMON A(1))

A CDC system routine LOCF is used to determine the first word address of that dummy array for purposes of calculating required field length.

After the sizes of all arrays are calculated and summed, a system routine REQMEM is called to provide the field length required.

Optionally a system routine JOBTD may be used to obtain additional information about actual field length used and about times at certain milestones in program execution.
4. Your Planning Needed

To plan for dynamic storage allocation, the user should make a systematic list of all the array names to be used in the program. Beside these array names, place the variable names of the corresponding integer dimensions of the arrays. These integer dimensions arise from a limited number of "how many" quantities involved in most problems. They include: how many items, how many lengths, how many time intervals, how many force quantities, how many complete sets of known quantities for a given problem. The integer quantities are often evident from the terminal index of DO Loops used for reading, printing, or calculating.

Making this systematic list of arrays and dimensions will require that the programmer understand the problem better. Once the list is complete, it is a routine procedure to set up the program for dynamic storage allocation.

5. How To Do It

(1) First, make a list of all the array names and variable dimensions in the main subroutine. These array names should be listed in the sequence in which they appear in the parameter list of the main subroutine. Remember, the array name and the integer variable names of all variably dimensioned arrays must be formal parameters. Usually, the array names in the main subroutine will be the problem-oriented variable names familiar to the programmer.
(2) Second, next to the list of array names and dimensions, make a list of the First Word Addresses (FWA) which will correspond to each array in the calling program. For instance, the First Word Addresses could be A(I1), A(I2), A(I3),..., etc.

An example of the two lists is given below:

<table>
<thead>
<tr>
<th>Arrays in Main Subroutine</th>
<th>First Word Addresses in Calling Program</th>
<th>Calling Program</th>
<th>Subscript Storage Calculation</th>
</tr>
</thead>
<tbody>
<tr>
<td>AA(L,M)</td>
<td>A(I1)</td>
<td>I1</td>
<td>1</td>
</tr>
<tr>
<td>BB(M,N)</td>
<td>A(I2)</td>
<td>I2</td>
<td>I1 + L*M</td>
</tr>
<tr>
<td>CC(L,N)</td>
<td>A(I3)</td>
<td>I3</td>
<td>I2 + M*N</td>
</tr>
<tr>
<td>...</td>
<td>....</td>
<td>I4</td>
<td>I3 + L*N</td>
</tr>
<tr>
<td>...</td>
<td>....</td>
<td>I24</td>
<td>I23 + ...</td>
</tr>
<tr>
<td>XX(M,M)</td>
<td>A(I24)</td>
<td>ISTOR</td>
<td>I24 + M*M</td>
</tr>
</tbody>
</table>

(3) Once the two lists are established, it is time to calculate the subscripts and storage for each array and record them beside the First Word Addresses as shown in the table above.

Usually I1 is set equal to 1 because we want the first word of the first array to appear as the first word of BLANK COMMON. Each subsequent subscript Ixx is calculated as equal to the subscript of the first word of the preceding array plus the product of the integer variable dimensions of the preceding array.

(4) After the FWA of the last array is calculated, the total storage for arrays is calculated as ISTOR which is equal to the subscript of the last array plus the size of the last array. (Actually this definition of
ISTOR results in the FWA subscript of an array which might follow the last one.

(5) The size of field length to be requested will be determined from the address of the first word of blank common plus the amount of storage required. The FWA of BLANK COMMON can be determined through use of the system function LOCF as in IADDR = LOCF(A)

Then the amount of field length to request can be calculated as

\[ IRFL = IADDR + ISTOR - 1 \]

All of the information necessary for organization of the program is now ready, and writing the main program can begin.

6. Writing the Program

(1) It is presumed that the principal subroutine with its parameter list and variable dimensions has already been written. Only the small calling program remains to be written.

(2) The blank common declaration COMMON A(1) should precede all executable statements.

(3) Read statements and print statements for the list of integer variable problem parameters should come next.

(4) The statement defining the first word address of blank common IADDR must appear before the final calculation of the amount of field length to request.
(5) Calculation of the FWA subscripts of all arrays can use FORTRAN statements identical to the "Subscript/Storage Calculation" of Table 1.

(6) The calculation of IRFL, the field length request can then be made. IADDR, ISTOR, and IRFL should be printed. They give useful information in case of a subsequent bombout. They should be printed in octal. ISTOR might also be printed as a decimal integer.

(7) A call to the CDC System routine REQMEM will direct the computer to assign the correct field length for the current problem.

CALL REQMEM (IRFL)

(8) Now you are ready to call the main subroutine with a list of First Word Addresses of each of the arrays in the proper sequence, i.e.:

CALL MAIN(A(I1), A(I2), A(I3),..., A(I24), L, M, N)

(9) Last you will either CALL EXIT, STOP, or provide a GO TO to the beginning of the program to read input for a new problem. In the latter case, a test should be provided to determine if there is no new problem.

7. Sample Program

The sample program below follows each of the steps described in the previous section. The principal subroutine BOSS calls subroutines from the matrix package FLMXPKR or FLMXPKF, to perform a routine matrix multiplication. Two matrices AA and BB are input and then printed. Their product is calculated and stored in the array CC. Then CC is printed.
The SCOPE Control Card Sequence is as follows.

JOBCARD
ATTACH (A, FLMXPKR)
RUNT.
LDSET(LIB=A)
LDØ.
7-8-9
FORTRAN DECK
7-8-9
Data deck with Integers
L,M,N and
Arrays AA and BB
6-7-8-9
PROGRAM MATRIX(INPUT, OUTPUT)

PROGRAM TO DEMONSTRATE THE OPERATION OF DYNAMIC STORAGE ALLOCATION

PLACEMENT ARRAY A IN BLANK COMMON
READ 1000, L, M, N
PRINT 2000, L, M, N
IADDR = LCCF(A)

FIND INITIAL ADDRESS OF FIRST WORD OF BLANK COMMON
I1 = 1

DEFINE FIRST WORD ADDRESS OF FIRST ARRAY.
I2 = I1 + L*N
CALCULATE STORAGE OF FIRST ARRAY AND FWA OF SECOND ARRAY
I3 = I2 + M*N
ISTOR = I3 + L*N

REPEAT FOR ALL SUCCEEDING ARRAYS IN SAME SEQUENCE AS PARAMETER
LIST OF PRINCIPAL SUBROUTINE (BOSS)
LAST CALCULATED AMOUNT IS TOTAL ARRAY STORAGE

IRFL = IADDR - 1 + ISTOR
CALCULATE REQUEST FOR FIELD LENGTH
PRINT 3000, IADDR, ISTOR, IRFL
CALL RECHEM(IRFL)

CALL RECHEM TO REQUEST FIELD LENGTH
CALL ECSS(A(I1), A(I2), A(I3), L, M, N)
CALL PRINCIPAL SUBROUTINE WITH A(IN) TYPE PARAMETERS REPRESENTING
EACH ARRAY IN PRINCIPAL SUBROUTINE PARAMETER LIST
CALL EXIT

1000 FORMAT (I5)
2000 FORMAT (I5, 7X, 9X, 1HL, 9X, 1HM, 9X, 1HN, 7X, I10)
3000 FORMAT (I5, 7X, 0E, 1HE, 2X, 34H REQUEST FOR FIELD LENGTH, 7X)
END

SUBROUTINE BOSS(AA, BB, CC, L, M, N)
REAL AA(L, M), BB(M, N), CC(L, N)
CALL ROCOGL(AA, L, M)
TTL = 6H MATRIX
CALL OUTG(AA, L, M, TTL, 2MAA)
CALL OUTG(BB, M, N, TTL, 2MHB)
CALL MULT(AA, BB, CC, L, M, N)
CALL OUTG(CC, L, N, TTL, 2MCC)
RETURN
END
LOAD MAP - MATRIX

FWA OF THE LOAD    111
LWA+1 OF THE LOAD   10502
TRANSFER ADDRESS -- MATRIX  112

PROGRAM AND BLOCK ASSIGNMENTS.

<table>
<thead>
<tr>
<th>BLOCK</th>
<th>ADDRESS</th>
<th>LENGTH</th>
<th>FILE</th>
<th>DATE</th>
<th>PROCSSR</th>
</tr>
</thead>
<tbody>
<tr>
<td>MATRIX</td>
<td>111</td>
<td>2221</td>
<td>LGO</td>
<td>12/04/78</td>
<td>RUNT F</td>
</tr>
<tr>
<td>BOSS</td>
<td>2332</td>
<td>70</td>
<td>LGO</td>
<td>12/04/78</td>
<td>RUNT F</td>
</tr>
<tr>
<td>/IYENGAR/</td>
<td>2422</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOLT</td>
<td>2440</td>
<td>52</td>
<td>UL-A</td>
<td>06/14/73</td>
<td>RUNT 2</td>
</tr>
<tr>
<td>OUTE</td>
<td>2512</td>
<td>233</td>
<td>UL-A</td>
<td>06/14/73</td>
<td>RUNT 2</td>
</tr>
<tr>
<td>RDOCOLG</td>
<td>2745</td>
<td>27</td>
<td>UL-A</td>
<td>06/14/73</td>
<td>RUNT 2</td>
</tr>
<tr>
<td>LCF</td>
<td>2777</td>
<td>3</td>
<td>SL-RUNP3</td>
<td>03/14/75</td>
<td>COMPASS</td>
</tr>
<tr>
<td>REQMEM</td>
<td>3012</td>
<td>1064</td>
<td>SL-RUNP3</td>
<td>02/12/76</td>
<td>COMPASS</td>
</tr>
<tr>
<td>KRAKER</td>
<td>4076</td>
<td>1115</td>
<td>SL-RUNP3</td>
<td>01/08/76</td>
<td>COMPASS</td>
</tr>
<tr>
<td>SYSTEM</td>
<td>5213</td>
<td>17</td>
<td>SL-RUNP3</td>
<td>01/08/76</td>
<td>COMPASS</td>
</tr>
<tr>
<td>GETBA</td>
<td>5232</td>
<td>121</td>
<td>SL-RUNP3</td>
<td>02/08/76</td>
<td>COMPASS</td>
</tr>
<tr>
<td>INPUTC</td>
<td>5353</td>
<td>72</td>
<td>SL-RUNP3</td>
<td>01/08/76</td>
<td>COMPASS</td>
</tr>
<tr>
<td>OUTPTC</td>
<td>5445</td>
<td>1475</td>
<td>SL-RUNP3</td>
<td>01/08/76</td>
<td>COMPASS</td>
</tr>
<tr>
<td>SIOS</td>
<td>7142</td>
<td>1300</td>
<td>SL-RUNP3</td>
<td>04/22/77</td>
<td>COMPASS</td>
</tr>
<tr>
<td>KODER</td>
<td>10442</td>
<td>37</td>
<td>SL-NUCLEUS</td>
<td>07/02/76</td>
<td>COMPASS</td>
</tr>
</tbody>
</table>
//                      | 10501   | 1      |        |            |         |

.532 CP SECONDS  21700B CM STORAGE USED
8. Hints, Tips, and Optional Advanced Features

(1) Save Parameter Lists for Array Names and Dimensions

Since subroutine parameter lists are limited to a maximum number of terms, it is best to reserve those parameter lists for only names of arrays which can vary in size and for their variable dimension names. Labelled common blocks can be used optionally to transmit unsubscripted variables, integer variables which serve as logical input/output unit numbers, and also arrays which absolutely never change dimension.

The combination of variable dimension statements for only the parameters used in a subroutine and labelled common statements used selectively is very effective in cutting down on the mass of unnecessary declarative statements typical in many subroutines.

(2) Integer Arrays

An integer array declared and dimensioned in a subroutine with a proper integer array name as a formal parameter is permissible even though it matches a REAL actual parameter in the calling program. As long as the integer values are created and processed in subroutines using integer type names and arithmetic, the contents of storage will be in correct form. Remember that the calling program only defined first word address and the amount of storage space.

(3) Storage of Several Arrays in One Array

In programs with many arrays, straight forward application of the process described can cause the limit for number of parameters in a subroutine to be exceeded. To get around this, several arrays related to a given procedure might be stored successively under one first word address.
in the calling program and then broken into their separate identities in deeper levels of subroutines. Suppose the three arrays and SUBROUTINE BOSS are only a small part of the storage and processing of another SUBROUTINE MAIN.

PROGRAM MATRIX( )
...
ISEV=L*M + M*N + L*N
...
I2 = I1 + ISEV
....
CALL MAIN(A(I1), A(I2), ..., L,M,N,ISEV)
...
END
SUBROUTINE MAIN(AA,XX,YY,ZZ, ... , L,M,N,ISEV)
REAL AA(ISEV),XX(...),YY(...),....
J1=1
J2=J1 + L*M
J3=J2 + M*N
CALL BOSS(AA(J1),AA(J2),AA(J3),L,M,N)
....
RETURN
END
SUBROUTINE BOSS(AA,BB,CC,L,M,N)
...
END

(4) Reuse of Storage Space for Another Array

Sometimes all creation, processing, and printing of an array is completed early in a program and then its contents are no longer needed. The storage space may be reused for another array of the same size which can be created after work with the original array is completed. To accomplish this, the two arrays planned to occupy the same space can be given the same First Word Address in the calling program.
For example, suppose array \( XX \) is the same size as array \( AA \) and will only be needed when space is no longer needed for \( AA \). Then the subroutine may be called with the same FWA (Say \( A(I) \)) for both arrays.

```
PROGRAM PROG(INPUT,OUTPUT)
...
CALL MAIN(K,L,M,N, A(I1),A(I2),...,A(I1))
...
END
SUBROUTINE MAIN(K,L,M,N,AA,BB,....,XX)
DIMENSION AA(K,L),BB(L,M),...,XX(K,L)
....
END
```

(5) Obtaining Information About Field Length and Time Usage

A system routine \( JOBTD \) (standing for Job Time and Date) may be called one or more times to make available for printing various data about the date of execution, time, and current field length. This is especially useful to use immediately after a call to \( REQMEM \) to find out how much field length is actually allocated. Calls to \( JOBTD \) before and after the execution of a processing operation can be used to calculate elapsed central processor and peripheral processor times by the difference between values obtained in the two calls.

The routine \( JOBTD \) has four parameters, each of which is a dimensioned array. Its use to determine date, current field length, and elapsed CP and PP time is illustrated below. The reader is referred to the \( JOBTD \) documentation for further information. The reader is left to their own imagination to provide better output formats for identifying the data printed.
PROGRAM HATRIX (INPUT, OUTPUT)
COMMON A(1)
DIMENSION TIME(3), CLOCK(3), IAUTH(3), MISC(2)
DIMENSION TIME2(3)

... ...
CALL REQMEM(IRFL)
CALL JOBTD(TIME, CLOCK, IAUTH, MISC)
PRINT 1000, CLOCK(2), IAUTH(2)
1000 FORMAT(7X, 4H DATE, A10, 2X, 13H FIELD LENGTH, :6, 1HB)
CALL MAIN(A(I1), A(I2), ... )
CALL JOBTD(TIME2, CLOCK, IAUTH, MISC)
ETCP = TIME2(1) - TIME(1)
ETPP = TIME2(2) - TIME(2)
PRINT 2000, ETCP, ETPP
2000 FORMAT(7X, 7H CP TIME, F10.3, 2X, 7H PP TIME, F10.3)
...
END

Where:
TIME(1) = CP sec since start (F.P.)
TIME(2) = PP sec since start (F.P.)
CLOCK(2) = Roman Date (10H MM/DD/YY)
IAUTH(2) = Current Field Length (Octal)

(6) Dynamic Storage on Other Computers

Other computers may not have routines equivalent to LOCF, REQMEM, and
JOBTD, but the benefit of packing all arrays into blank common is still
available. The organized list of storage calculations should make it
possible to write a single formula in terms of integer dimensions (problem
parameters) for the total storage required.

Then only one dimension statement needs to be changed! The line
reading COMMON A(1) may be changed to COMMON A(50000) or whatever decimal
value is necessary. This is far superior to changing multi-line dimen-
sion statements in every subroutine for each new problem when variable
dimensions are not used.
9. Acknowledgement

The author is grateful for the instruction in FORTRAN and dynamic storage allocation provided by Dr. Suresh Desai and Dr. Sampath N.S. Iyengar,

Opportunities for learning and applying this concept were provided by teaching of structural and computer programming courses in Civil Engineering and by research at the Fritz Engineering Laboratory of Lehigh University. Dr. David A. Van Horn is chairman of Civil Engineering and Dr. Lynn S. Beedle is Director of Fritz Engineering Laboratory.