**SPSRB Common Standards Working Group**

**Standards, Guidelines and Recommendations for**

**Writing Fortran 77 Code**

**Ken Jensen**

**Raytheon Information Systems**



**Version 2.0**

**September, 2010**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**VERSION NUMBER IDENTIFIER**

The document version number which also corresponds to the Document Control Number (DCN) identifies whether the document is a working copy, final, revision, or update, defined as follows:

**Working Copy or Draft:** a document not yet finalized or ready for distribution; sometimes called a draft. Use 0.1A, 0.1B, etc. for unpublished documents.

**Final Copy:** the first definitive edition of the document after it passes through the drafting stage. This first edition is always identified as Version 1.0.

**Revision:** an edition with minor changes from the previous edition, defined as changes affecting less than one-third of the pages in the document. The version numbers for revisions 1.1 through 1.9, 2.1 through 2.9, and so forth. After nine revisions, any other changes to the document are considered an update. A revision in draft, i.e. before being re-baselined should be numbered as 1.1A, 1.1B, etc.

**Update:** an edition with major changes from the previous edition, defined as changes affecting more than one-third of the pages in the document. The version number for an update is always a whole number (Version 2.0, 3.0, 4.0, and so forth).

**DOCUMENT HISTORY**

**DOCUMENT REVISION LOG**

The Document Revision Log identifies the series of revisions to this document since the baseline release. Please refer to the above page for version number information.

|  |  |  |  |
| --- | --- | --- | --- |
| **DOCUMENT TITLE: Standards, Guidelines and Recommendations for**  **Writing Fortran 77 Code** | | | |
| **DOCUMENT CHANGE HISTORY** | | | |
| **Revision No.** | **Date** | **Revision Originator Project Group** | **CCR Approval # and Date** |
| 1.0 | April 2009 | Initial Release by CSWG | SPSRB approved  April2009 |
| 2.0 | September 2010 | Update by CSWG | Approved September 2010 |
| 2.1 | May 2011 | Minor Updates by Zhaohui Cheng  and Kathryn Shontz | Approved June 2011 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**LIST OF CHANGES**

Significant alterations made to this document are annotated in the List of Changes table.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **DOCUMENT TITLE: Standards, Guidelines and Recommendations for**  **Writing Fortran 77 Code** | | | | | |
| **LIST OF CHANGE-AFFECTED PAGES/SECTIONS/APPENDICES** | | | | | |
| **Version Number** | **Date** | **Changed By** | **Page** | **Section** | **Description of Change(s)** |
| 2.0 | September 2010 | CSWG | All | All | Major layout and formatting update; sections in the document were rearranged and formatted to be consistent with all other CSWG documentation. |
| 2.0 | September 2010 | CSWG | 10 | 3.1 | Changed the use of global variables guideline into a recommendation. |
| 2.0 | September 2010 | CSWG | 15 | 4.1 | The use of an ampersand (&) was changed to that of a plus sign (+) for how to correctly continue a line in F77. |
| 2.1 | May 2011 | Shontz | 1 | -- | Altered cover page date to reflect actual deployment time |
| 2.1 | May 2011 | Chen | 3, 4 | -- | Altered the “Document Title” to accurately reflect the title of the standards document. |
| 2.1 | May 2011 | Chen | 5 | -- | Edited the table of contents for a few errors |
| 2.1 | May 2011 | Chen | 10 | 3.1 | Added “and END DO” to the third standard in this section. |
| 2.1 | May 2011 | Chen | 15 | 3.1.8 | Moved last two standards to different sections (3.1.3 and 3.1.5) to be consistent with the content of these standards |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**TABLE OF CONTENTS**

1. **Introduction ………………………………………………….**
   1. Programming Standards and Guideline Definitions …………….......
   2. Key Word Definitions ……………………………………………….
   3. Reference Documents ….....…………………………………………
2. **Language Features …………………………………..............**
   1. Features ……………………………………………………………..
   2. Error Trapping ….………………………...…………………….…...
      1. Example 1: Sample Introduction Program …………………………...
   3. Subroutine Control ………………………………………………….
3. **Statements …..……………………………………………….** 
   1. Basic Statement Guidance …………………………...……………...
      1. GO TO Statements ……………………………………………..…….
      2. IMPLICIT NONE Statements ………………………………………..
      3. DO Statements ……………………………………………………….
         1. Example 2: DO Statement …………………………………………..
      4. FORMAT Statements ………………………………………………..
         1. Example 3: FORMAT Statement ……………………………………
      5. IF Statements …………………………………………………………
         1. Example 4: IF Statement ……………………………………………..
         2. Example 5: Arithmetic IF Statement …………………………………
         3. Example 6: ELSE Statement …………………………………………
         4. Example 7: ELSE IF Statement ………………………………………
         5. Example 8: Computed GO TO Statement ……………………………
      6. SAVE Statements …………………………………………………….
      7. EQUIVALENCE Statements ………………………………………...
   2. Statement Numbers ………………………………………………….
4. **Readability and Documentation ……………………………**
   1. Readability …………………………...……………………………..
   2. Documentation ….………………………...…………………….…..
5. **Libraries and Memory …………………………………........**
   1. Common Libraries …………………………...……………………..
   2. Memory Allocation ….………………………...……………………..

**6**

7

8

8

**9**

9

9

10

10

**10**

10

11

11

11

11

12

12

12

12

12

13

13

13

14

14

14

**15**

15

15

**15**

15

16

# **1. INTRODUCTION**

The National Environmental Satellite Data Distribution Service (NESDIS) develops and implements algorithms that transform environmental satellite images of the Earth into meaningful environmental data which are then employed in a full-time operational setting. In the past, software developed within NESDIS was created by the differing entities throughout the service, each creating code to fulfill various research, operational and archival needs. This meant software was written in various programming languages and idiosyncratic styles, moreover suffering from a lack of coordinating documentation in most cases. The resulting software is consequently often costly to maintain as the source code may have been mislaid, the code may be difficult to read and understand, documentation may be inadequate, or the original developers may no longer be able to maintain their code.

The purpose of developing common software programming standards is to reduce the cost of the software lifecycle and streamline the algorithm implementation process. This follows a trajectory from initial research and software development to operational use and finally through to divestiture and retirement where costs accumulate throughout the lifecycle. Implementation of these Satellite Products and Services Review Board (SPSRB) approved coding standards will shift costs away from operations and maintenance as the problems are resolved upstream. Promoting the accountability of the developers and scientists to create standardized software programs will benefit NESDIS as a whole. Higher front-end expenditure will be repaid in the form of lower operational and maintenance costs over subsequent years. It is intended that the implementation expenses of the common software standards will be funded through the Office of Systems Development (OSD) Product System Development and Implementation (PSDI) process, and must be included in relevant budgets and projects plans when applying for PSDI funds.

Having common programming standards used by all SPSRB stakeholders will aid in cross-organization communication and implementation of codes. It will also produce a software catalog that:

* Is robust
* Is readily portable (platform independent)
* Is modular and reusable
* Is inexpensive to implement and maintain operationally
* Is written in a widely used and supported language
* Has a common look and structure
* Adheres to best programming practices
* Is well documented
* Is easily readable and understandable
* Behaves in a standard manner (exception handling, file input/output)
* Uses common shared libraries

Specifically, the objective of this document is to provide standards for Fortran 77 code that is intended for operational use through the SPSRB process. The intended users of this document are programmers, testers, and reviewers of Fortran 77 code that will be used to implement an algorithm that creates an operational product from remote sensing satellite data. To achieve the objective, this document shall:

• Establish Fortran 77 programming standards, drawn from international standards

• Provide Fortran 77 programming guidelines for SPSRB stakeholders

• Provide examples of good Fortran 77 programming practices

The document is based on the recognition that many Fortran programmers and potential reviewers are accustomed to the Fortran 77 version, and much of the legacy Fortran code is in the Fortran 77 style. While it is recommended that Fortran programming be in accordance with Fortran 90 or later standards when practical, it is not required. Fortran 77 code, when written according to its standards, is compatible with newer Fortran compilers.

**1.1 Programming Standards and Guideline Definitions**

It is recognized that certain stylistic suggestions which make code easier to read (e.g. lining up attributes, or using all lower case or mixed case) are subjective and therefore should not have the same weight as techniques and practices that are known to improve code quality. For this reason, the standards within documents produced by the SPSRB Common Standards Working Group are divided into three components; Standards, Guidelines and Recommendations (SGRs):

***Standard***:Aimed at ensuring portability, readability and robustness. Compliance with this category is mandatory.

***Guideline***:Good practices. Compliance with this category is strongly encouraged. The case for deviations will need to be argued by the programmer.

***Recommendation***:Compliance with this category is optional, but is encouraged for consistency.

These three standards will thus be found in the above format throughout this document, indicating the weight of a particular standard. If possible, all standards, guidelines and recommendations should be followed when programming. Else, programmers should include these components whenever possible, keeping in mind their respective weight. Please refer to these definitions as needed.

**1.2 Key Word Definitions**

These words and phrases are vital to the comprehension of this document. These are language-specific and general terms with which a user may not be readily familiar. Listed below in alphabetical order are the terms specified by the CSWG to be helpful to users and may be referenced as necessary.

***Main Program****:* A Fortran main program begins with the reserved word PROGRAM and ends with a matching reserved word END, and consists of a sequence of executable statements and optional declarations. A program is always a separately compilable unit.

***Program Unit****:* A program unit is any of the three structures in Fortran, namely PROGRAM, SUBROUTINE, and FUNCTION.

***Subprogram****:* A Fortran subprogram begins with either the reserved word FUNCTION, SUBROUTINE, MODULE, or BLOCK DATA and ends with a matching reserved word END. It consists of a sequence of executable statements and is called by a main program or by another subprogram. A subprogram may or may not be a separately compilable unit, depending upon implementation.

**1.3 Reference Documents**

Three main documents were used in creation of this standard and can be used as additional references if any presented material is unclear.

**ISO 1539:1980** **Standard**

This is the international standard for Fortran 77 code. This is a very large document that can be used as a reference at the programmer’s discretion. Equivalent information can be found on-line at http://www.fortran.com/fortran/F77\_std/rjcnf0001.html.

Einarsson, B. and Y. Shokin**,** ***Fortran 90 for the Fortran 77 Programmer.*** Version 2.3, 1996.

This document is a useful training document for Fortran 77 programmers who are interested in programming with Fortran 90 and later versions of Fortran. This document can be found online at http://www.nsc.liu.se/~boein/f77to90/f77to90.html.

Boukabara, S.-A. and P. van Delst**. *Standards, Guidelines and Recommendations for Writing FORTRAN 95 Code.*** Version 1, 2007***.***

This documentprovides the NOAA/NESDIS common standards, guidelines and recommendations for Fortran 95 code. This document is available on the SPSRB website.

**2. Language Features**

Consistency is the key to making programs easily readable.

**2.1 Features**

***Standard***: *Only language features and capabilities that are documented or defined in the* ***ISO 1539:1980 (c.f. Section 2)*** *shall be used.*

Use the “-iso” compiler flag to ensure this.

***Standard***: *Fixed source form (Fortran 77) and free source form (Fortran 90/95) shall not be mixed within a subprogram.*

Program units written in fixed form and free form may be mixed in the same program, but each unit must be only in one form and at the compilation both forms may not be in the same source file.

***Guideline***: *Complete the code in the Fortran 77 (fixed form) style if an algorithm is re-using a substantial amount of legacy Fortran 77 code.*

If there is no re-use, new code should be re-written in free form style. If the amount of re-use is small, new code should be written in free form style and legacy code re-written in free form style. An exception to this guideline is the case where the designated programmers are not familiar with free form style and the scope of the project does not warrant the training cost. In this case, new code written in fixed form style is acceptable.

***Recommendation****: During the development and testing phases, use compiler options that provide additional checks of the code.*

**2.2 Error Trapping**

***Standard***: *Handle the end-of-file condition when reading beyond the end of a sequential or internal file.*

If an item of the form: END=label (as opposed to ERR=999 in the above example) then control is transferred to the labeled statement when the end-of-file condition is detected. The END= [keyword] may only be used in READ statements, but it can be used in the presence of both ERR= and IOSTAT= [keywords]. End-of-file detection is very useful when reading a file of unknown length.

***Standard***: *Take special care with I/O statements since these are usually affected by events beyond the control of the programmer.*

Include an item of the form ERR=label which causes control to be transferred to the statement attached to that label in the event of an error. This must, of course, be an executable statement and in the same program unit. See example below.

### **2.2.1 Example 1:** Sample introduction to a program.

!----------------------------------------------------------------------

! Name: Noise

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*

READ(UNIT=IN, FMT=\*, ERR=999) VOLTS, AMPS

WATTS = VOLTS \* AMPS

[rest of program in here…]

STOP

999 WRITE(UNIT=\*,FMT=\*)'Error reading VOLTS or AMPS'

END

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*

**2.3 Subroutine Control**

***Standard***: D*o not use an alternate return specifier as an argument in a calling sequence in the event of an error .*

For example, do not use "CALL foo (a, b, \*999)".

**3. Statements**

This section outlines how to craft statements in Fortran 77.

**3.1 Basic Statement Guidance**

***Standard***: All variables shall be declared using a type-statement, INTEGER, REAL, DOUBLE PRECISION, COMPLEX, LOGICAL, and CHARACTER.

***Standard***: If COMMON blocks are used, the following practices should be followed:

1. All variables in a COMMON block should be named the same in every program unit that uses the common data.
2. COMMON blocks shall be declared in a separate file and copied into the source file using the INCLUDE statement.
3. Variables in the COMMON statement shall be aligned on their proper word and byte boundaries. This generally means listing the variables in descending order according to their data type size: quad-word variables first, then double-word variables, word variables, half-word variables, and finally single byte variables.

***Standard:*** *All loops shall terminate with a unique* CONTINUE or END DO *statement.*

***Recommendation***: *Use global variables or modules instead of* COMMON *blocks and* INCLUDE *files.*

### **3.1.1 GO TO Statements**

***Standard***: Every GO TO statement *shall be accompanied by comments placed near the* GO TO *statement to document the applicable constraints or commentary regarding the statement*.

***Standard****:*  GO TO *statements shall not be used to transfer control outside loops*.

***Standard****: Each* GO TO *statement shall target a unique* CONTINUE *statement.*

***Standard***: *Assigned* GO TO *statements shall not be used*.

***Guideline***: *Computed* GO TO *or* ELSE IF *statements are acceptable in lieu of a* CASE *statement.*

However, out of range conditions should be included in the statement.

***Guideline***: *The use of* GO TO *statements in new Fortran 77 code is discouraged*.

GO TO statements should be used only where required to meet specific execution time, space constraints, or to reduce unnecessary complexity.

### **3.1.2 IMPLICIT NONE Statements**

***Standard***: *All program units shall include the "*IMPLICIT NONE*" statement and should be compiled with the option so the compiler flags any variables that are not explicitly declared.*

### **3.1.3 DO Statements**

***Standard***: *The* DO *statement shall not contain any statements that change the value of the loop-controlling variable.*

***Standard***: *The last statement in a* DO *loop shall be an* END DO or Continue *statement*.

If there are a large number of statements in a DO loop, include a comment that relates the END statement to the DO statement for that loop.

***Standard***: *Indent the statements following the Fortran* DO *statement one (1) level of indentation*.

See example below.

**3.1.3.1 Example 2: DO Statement**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

DO <stmt#> ( <expression> )

<statements>

<stmt#> CONTINUE

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

***Guideline***: DO *loops should not be numbered to avoid confusion and formatting errors.*

### **3.1.4 FORMAT Statements**

***Guideline***: *Accompany each* READ *and* WRITE *statement with the corresponding* FORMAT *statement.*

A format statement must be coded only once and be placed after the first I/O statement that refers to it. See example below.

**3.1.4.1 Example 3: FORMAT Statement**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

READ (5,10) SIZE

10 FORMAT(I5)

READ (5,10) SIZE2

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

### **3.1.5 IF Statements**

***Standard***: *Enclose the condition(s) following the Fortran reserved word* IF *in parentheses*.

***Standard***: *The last statement in an* IF *loop shall be an* END IF *statement.*

If there are a large number of statements in an IF loop, include a comment that relates the END statement to the IF statement for that loop.

***Standard***: *Indent the statements following the* IF, ELSE IF, *and* ELSE *statement, one (1) level of indentation*.

***Standard***: *Format the following expressions in the manner described below:*

**3.1.5.1 Example 4: IF Statement**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

IF ( <expression> ) THEN

<statements>

END IF

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* + - 1. **Example 5: Arithmetic IF Statement**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

IF ( <expression> ) s#1, s#2, s#3

C

s#1 CONTINUE

<statements>

GO TO s#4

C

s#2 CONTINUE

<statements>

GO TO s#4

C

s#3 CONTINUE

<statements>

C

s#4 CONTINUE

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* + - 1. **Example 6: ELSE Statement**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

IF ( <expression> ) THEN

<statements>

ELSE

<statements>

END IF

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* + - 1. **Example 7: ELSE IF Statement**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

IF ( <expression> ) THEN

<statements>

ELSE IF ( <expression> ) THEN

<statements>

ENDIF

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* + - 1. **Example 8: Computed GO TO Statement**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

GO TO ( s#1, s#2, s#3 ) i

C

s#1 CONTINUE

<statements> )

GO TO s#4

C

s#2 CONTINUE

<statements> )

GO TO s#4

C

s#3 CONTINUE

<statements> )

C

s#4 CONTINUE

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

*Guideline: Use the* ELSE IF *statement in lieu of an Arithmetic* IF *statement.*

*Guideline: Use the* ELSE IF *statement in lieu of the Computed* GO TO *statement.*

### **3.1.6 SAVE Statement**

***Standard***: *All loops shall terminate with a unique* CONTINUE *statement*.

***Standard***: *Every* GO TO *statement shall be accompanied by comments placed near the* GO TO *statement to document the applicable constraints and comments placed near the statement receiving control to document the origin of the transfer of control.*

***Standard***: *To retain the value of a variable in a program unit after control is returned from that unit, use a* SAVE *attribute or* SAVE *statement.*

Do not rely on the compiler to retain the variable's value for you. By explicitly putting the variable in a SAVE statement, it will help make it clear to anyone reading the code that the value of the variable is being retained between calls to the routine.

### **3.1.7 EQUIVALENCE Statement**

***Guideline****: The use of* EQUIVALENCE *statements is discouraged*.

The TRANSFER intrinsic function should be used instead of EQUIVALENCE statements.

### **3.1.8 END Statement**

***Standard***: *An* END *statement shall be used as the last statement for all functions, subroutines, modules and programs*.

**3.2 Statement Numbers**

***Standard***: *Consistently justify statement numbers in columns two (2) through five (5) in ascending order throughout a program.*

***Guideline***: *Avoid* *referencing statement numbers in comments.*

**4. Readability and Documentation**

Herein described are formatting concerns relating to readability and documentation within Fortran 77 code. A program should have adequate formatting and commentary such that any user would be able to interpret the code correctly and with ease.

**4.1 Readability**

***Standard***: *When a single statement extends to more than one (1) line, begin each continuation line with a plus sign (+) in column six (6), and indent two (2) levels of indentation relative to the first statement line.*

***Standard***: *Begin each continuation line with a plus sign (+) in column six (6), and indent two (2) levels of indentation relative to the first statement line*.

**4.2 Documentation**

***Standard***: *Accompany each program unit* END *statement with a comment indicating the program unit name.*

***Guideline***: *Delimit comments consistently, with a 'C' in column one (1) and a blank line before the comment line.*

***Guideline***: *Precede each conditional statement (e.g.,* DO*,* IF*, Computed* GO TO*) with a block comment describing the condition being tested and the branching alternatives.*

**5. Libraries and Memory**

**5.1 Common Libraries**

***Recommendation****: Use the IMSL Fortran Numerical Library.*

This library is usually available from suppliers of Fortran compilers. The IMSL Fortran Library, a complete collection of mathematical and statistical algorithms for high performance computing applications, integrates the IMSL F90 Library with the IMSL Fortran 77 library into a single, cohesive package. The IMSL Fortran Library includes all of the algorithms from the IMSL Family of Fortran libraries for more than three decades, including the IMSL F90 Library, the IMSL Fortran 77 Library, and the IMSL parallel processing features.

**5.2 Memory Allocation**

***Guideline***: *Use dynamic allocation of memory wherever possible.*

***Guideline***: *Do not allocate memory for local variables until they are used in a subprogram, and deallocate the memory for a local variable as soon as its use in the program is finished.*