FORCHECK

for

Windows

A Fortran Verifier and Programming Aid

version 14.6

User Guide

commandline

December 3, 2014

2

The information in this document is subject to change without previous notice and should not be taken as a commitment by Forcheck b.v. Forcheck b.v. can not assume responsibility for any errors which may appear in this document.

The software described in this document is furnished under a license and may be used, copied or disclosed only when in accordance with the terms of this license.

Website: `http://www.forcheck.nl`
Email: `info@forcheck.nl`

Contents

Chapter 1

Introduction

FORCHECK is a Fortran program development, conversion, maintenance and documentation tool. It parses Fortran programs, verifies the syntax and composes cross-reference tables. It analyzes both separate program units and the program as a whole.

1.1   What does FORCHECK do?

FORCHECK verifies the syntax by parsing the source program. This is done as precisely as possible at compile time. The full Fortran 2008 syntax (which includes the Fortran 2003, Fortran 95, Fortran 90 and FORTRAN 77 syntax) is supported. Moreover most language extensions of many compilers are accepted. As an option the syntax can be checked for strict conformance to the FORTRAN 77, the Fortran 90, the Fortran 95, the Fortran 2003, or the Fortran 2008 standard.

Cross-reference tables of all objects within program units are composed. Information and warnings concerning the usage of all objects are provided.

The reference structure (call tree) of the program can be analyzed and presented. Recursive references are traced and verified. The persistence of common-block objects and global module data is verified.

The consistency of the entire program is verified by checking the category and type of the procedures and the argument lists of all procedure references. Length, type and structure of the common blocks specified in the various program units are compared. Cross-reference tables of all procedures, common blocks, common-block objects, modules, public module data, external I/O and include files over the program are composed.

FORCHECK can emulate a specific compiler by reading a configuration file in which all types and language extensions to be supported are enumerated.

The global information of each program unit can be stored in library files which can be referenced and updated in subsequent FORCHECK runs to test program units in the context of the entire program.

1.2   Why FORCHECK?

Though your Fortran compiler verifies the syntax of the input source code, this check is in general far from complete. FORCHECK, however, performs this verification as complete as

possible at "compile-time". And, what is even more important, FORCHECK not only verifies the separate program units but also the program as a whole, beyond program unit boundaries. Procedure types, argument lists and common blocks are all verified for consistency.

FORCHECK saves you time and annoyance because errors are detected as early in the development process as possible.

As an option FORCHECK checks the conformance of your program to the FORTRAN 77 [1], the Fortran 90 [2, 3], the Fortran 95 [4], the Fortran 2003 [5], or the Fortran 2008 [6] standard. Though most compilers have an option to reveal deviations from the standard, they generally perform this in a limited way. FORCHECK, however, reveals almost all deviations which can be detected during static analysis. This is of utmost value when developing portable software.

The documentation composed by FORCHECK is compact and especially useful during program development and maintenance. The index of program units and module procedures, the reference structure (call tree) of all subprograms, the dependency tree of all modules and the cross-reference tables of procedures, common blocks, common-block objects, modules, public module data, external I/O and include files, are indispensable whenever you deal with a program consisting of more than a few program units.

FORCHECK can emulate most language extensions of many compilers. When you tell FOR-CHECK to emulate the compiler of the target system you can use it as a conversion aid.

The global information of the various program units can be stored in library files. You can verify newly developed or changed program units in the context of the entire program by specifying the library files containing the global program information without analyzing all source code anew. In this way you can develop programs in a modular way without the risk of creating inconsistencies in the subprogram interfaces.

1.3   Application Areas

FORCHECK can be used to the best advantage in the following application areas:

- · **Program development.** During program development FORCHECK signals syntax errors and presents warnings both at the program unit and program level. It will detect substantially more of the program flaws than your compiler. Subsequent compilation will raise no additional error messages any more.

- · **Program maintenance.** The optimal documentation presented, specially the table of contents, the reference structure, the module dependency tree and the cross-reference tables, will show you exactly where to find all items that will be affected when you change global items such as an argument list or a common block.

- · **Education.** In contrary to most compilers FORCHECK shows the deviations to the Fortran standard very precisely. Moreover FORCHECK shows where implicit type conversions and truncations occur. Invalid references to procedures and inconsistent common blocks are common errors which are signalled by FORCHECK.

- · **Conversion.** FORCHECK verifies, at wish, if a program is standard conforming. In that case you will have minimal problems when transferring a program to another computer system. Moreover FORCHECK can emulate most Fortran extensions of many compilers so

you can verify the portability of your program during development without moving the source code to the target system.

1.4 This manual

This reference manual does not have the intention to describe the Fortran language or the Fortran standard. A good working knowledge of the Fortran language and nomenclature is assumed.

This manual starts with a tutorial to get acquainted with FORCHECK. Then it discusses the installation and operational procedures. Thereupon a concise description of the program unit analysis, reference structure, module dependency tree and global program analysis follows. In the appendices you can find information on the supported Fortran syntax, how you can tune FORCHECK to accept the compiler extensions of your choice, and the limitations of FORCHECK. The manual concludes with a message summary with explanations, a glossary, references and an index.

Chapter 2

Installation

2.1   The distribution kit

The FORCHECK for Windows distribution kit consists of the following components:

- · The FORCHECK analyzer, the FORCHECK librarian and the FORCHECK interface builder.

- · The FORCHECK integrated development environment (IDE).

- · Forcheck interface libraries for some Fortran libraries.

- · This user- and installation guide as a PDF file.

2.2   Password protection and support

A single user FORCHECK license shall be installed and used on one machine only. If more than on user wants to use FORCHECK on one machine you can purchase a multi-user license, if you wish to install and run FORCHECK on more than one machine, you need to purchase more than one FORCHECK license or purchase a site licence. To run FORCHECK a password file is required. Technical support is provided for registered users only. You can enter a maintenance license to get email support and the right to download updates.

The FORCHECK GUI, ForcheckIDE, is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

2.3   Required configuration

To install and run FORCHECK for Windows you need at least:

- · A Personal Computer with an IA-32 processor.

- · 512 MB of available memory (RAM).

- · Monochrome or color display.

· A fixed disk with 80 MB of free space.

· MS Windows 2000 or up.

· The Forcheck distribution kit.

2.4   Installing FORCHECK

Run `forcheck-14.5-win32-x86.exe` e.g. by double-clicking on the filename. The setup program will start with a welcome screen and asks for a directory to place the FORCHECK for Windows program and ancillary files. By default this is `C:\Program Files (x86)\Forcheck\`. Subsequently the installer asks for the components to install. You can choose to install the FORCHECK analyzer, the FORCHECK IDE, or both. The additional library interfaces are useful if you develop using the cvf, Intel compiler, or the MPI library.

2.5   Password

Place the supplied password file as `fckpwd.pwd` in the share subdirectory of the installation directory.

2.6   Uninstalling

To uninstall FORCHECK for Windows select from the Start menu **Settings**, **Control Panel** and then **Add/Remove Programs**, or start `uninstall` from the Forcheck program group in Start/All Programs.

2.7   Installation directory

The installation path of FORCHECK is stored in the Windows registry, so if you want to install FORCHECK on a different location you have to uninstall FORCHECK first. If FORCHECK does not find the installation path in the registry it reads the environmental variable `FCKDIR` to find the installation directory.

FORCHECK for Windows command line can be started by double-clicking on the Forcheck Command Line icon. If you want to use FORCHECK from the command prompt in a Command window, you must specify the path of the bin subdirectory of the FORCHECK installation path, e.g. `C:\Program Files (x86)\Forcheck\bin\`, in the PATH environmental variable. In Windows 7 open the configuration screen, select system/advanced system settings. For Windows NT, 2000 or XP you right-click on My Computer and choose Properties/Environment. Under Windows 95 or 98 you must add the FORCHECK installation path to the PATH environmental in AUTOEXEC.BAT.

2.8 Scratch files

FORCHECK stores some information in scratch files. You can specify the path at which these scratch files are stored with the keyword TMP, for example:

```
TMP=C:\TMP
```

By default FORCHECK places the scratch files on the current directory.

2.9 Include files

You can instruct FORCHECK to search for include files on additional paths by specifying the keyword INCLUDE, for example:

```
INCLUDE=C:\INCL
```

More than one include path can be specified by separating the paths by a ';'. See the chapter "Operation", section "The usage of include files" for the search strategy for include files.

2.10 Adaptation to your Fortran Compiler

FORCHECK can emulate the syntax and data types of several compilers. See the Appendix "Supported Fortran syntax" for a list of all compilers and extensions supported.

By default FORCHECK for Windows emulates the Intel Fortran V11 compiler. If you use one of the other compilers you must configure FORCHECK by specifying an emulation configuration file in which the proper configuration data is stored.

You can specify the appropriate emulation configuration file with the keyword FCKCNF, for example:

```
C:\Program Files (x86)\Forcheck\share\xxx.cnf
```

in which xxx stands for the identification of the compiler to be emulated. See the appendix "Supported Fortran syntax" for a list. If your compiler is not on the list, or if you like to define your own set of language extensions, you must compose a new emulation configuration file. See appendix "Supported Fortran syntax" section "Changing the configuration file" for details.

2.11 Specifying default options

FORCHECK has a facility to specify default global options. In this way you don't have to specify the options, which you use normally, each time you run FORCHECK. You specify the default options by specifying the keyword FCKOPT, for example:

```
FCKOPT=/PW:90/PL:66/F77/DC/-SH/-SI
```

2.12   Tryout

You can verify if FORCHECK is functioning correctly by analyzing the demonstration program:

`forchk fckdem,C:\Program Files (x86)\Forcheck\share\examples\demo\fckdem.for;`

and view the list file `fckdem.lst`.

2.13   Summary of environmental variables

FCKCNF	path of the emulation configuration file to be used
FCKDIR	FORCHECK's base installation directory
FCKOPT	default options
INCLUDE	additional search paths for include files
TMP	path for scratch files

Chapter 3

Tutorial

FORCHECK has many possibilities to analyze your source code. A configuration file and numerous options are available to tune the analysis to your needs. To learn how to use FORCHECK from scratch this tutorial will guide you by taking small steps from the analysis of a single program unit up to a large project. For a complete description of the IDE and the command-line interface see the chapter "Operation". For a precise clarification of the analysis see the chapter "Analysis".

3.1 Setup

FORCHECK can handle many Fortran levels, dialects and language extensions. Before analyzing your source code you must instruct FORCHECK which Fortran language level and dialect to apply. Numerous incorrect and superfluous messages will appear otherwise. By choosing the configuration file that matches the compiler you use you define the Fortran language level, dialect and language extensions that will be accepted.

The Intel Fortran 95 compiler emulation is applied by default. This is a Fortran 95 compiler with many language extensions, so Forcheck will probably have no difficulty with analyzing your source code using this emulation. If you use a different compiler you can do the following:

When using FORCHECK in commandline mode set the environmental variable FCKCNF to the path of the configuration file of your choice, for example for the gnu Fortran compiler, gfortran:

```
SET FCKCNF=C:\Program Files (x86)\Forcheck\share\gfortran.cnf
```

Mind that specifying one of the standard-conforming options such as the /F90 option does not enable the accompanying language level (Fortran 90) but merely forces reporting any deviation from this language level standard.

3.2   Analyzing a single source file

It is advisable to start simply by analyzing a single source file. Choose a source file containing a program unit that does not use modules, or one that contains all the referenced modules. In this way you can verify the settings and experiment using some of the options. A demo project, fckdem, is supplied in the examples directory.

When using FORCHECK in command-line mode an example of the analysis of a single source file is:

```
forchk ,mysourcefile;
```

The default filename extension is .for. You may need to specify some options to indicate the source format and specify the path of the include files (if not in the path of the source files) before you get the result you expect. These options are:

/AC.   Analyze all columns of the source input records (beyond column 72 for fixed source form).

/CN:*c*.   Allow a maximum of *c* continuation lines in a statement (beyond 19 for fixed source form before Fortran 2003).

/FF.   Source code input is in free source form. This is the default for source files with a filename extension of .f90, .f95, f03, or f08.

/DF:*s*.   Define meta symbols for conditional compilation. The items in the list must be separated by a ";".

/IP:*p*.   Specify include directories. The items in the list must be separated by a ";".

These options can be specified on the commandline before the source file:
forchk [*options*],*sourcefile*;
 e.g.
 forchk /AC/CN:99/FF/DF:X86/IP:.,mysourcefile.for;
 The negative form of an option is the option preceded by NO, e.g. /NOFF for fixed form.

If the source file contains more than one program unit they are analyzed in the sequence of occurrence and a global analysis is performed in addition to the program unit analysis.

3.2.1   Suppressing messages

If you get an overwhelming number of informational messages and warnings suppress them for the time being.

/NOINF   Do not show informative messages.

/NOWA   Do not show warnings.

3.2.2   Producing a source listing with cross-references

The syntax is:

forchk *listfile,sourcefile*;

e.g.:

```
forchk mylistfile,mysourcefile.for;
```

or, combined with some options:

```
forchk mylistfile/FF/NOINF,mysourcefile.for;
```

Now you have analyzed your first source code it is time to experiment with the available options which are described in the chapter "Operation". You can tune the analysis, and the output, just try!

3.3   Analyzing more than one source file

The syntax is:
 forchk [*listfile*][*options*],*sourcefiles*;
 The source files must be separated by a "+". E.g.:
 forchk mylistfile/FF,mysourcefile1+mysourcefile2;
 The options specified before the source input files are global, they are in effect for the global analysis and operate on each source input file. If you like to specify certain options for a specific source file you can specify local options with a source filename. E.g.:
 forchk mylistfile/FF,mysourcefile1+mysourcefile2/NOFF;
 Now mysourcefile1 is supposed to be in free form and mysourcefile2 in fixed form.

3.3.1   Analyzing all source files in one or more directories

When using the commandline you can specify wild cards to analyze all source files in a directory, e.g.:

```
forchk mylistfile/NOINF,*.for/NOFF+*.f90;
```

3.3.2   Analyzing your project: using a command file

You can place the command input in a file and have FORCHECK read this command input file:

```
forchk @mycommandfile
```

The contents of the command file must look like:

[*listfile*][*global_options*]
source_input_file_1[*local options*]+
...

source_input_file_n[*local options*];

3.4   The program analysis

Analyzing the program as a whole adds an extra dimension to the analysis. When the majority of the procedures has been included in the analysis you can enable the complete option, /CO to signal unreferenced and undefined global items over the program as a whole. In that case unreferenced procedures, unreferenced common blocks, unreferenced and undefined common-block objects, unreferenced modules, unreferenced and undefined public module procedures, operators and data are flagged. See also the section "Verification of common blocks" and "Verification of modules" of the chapter "Analysis".

   If not all procedures are available you can make the interface available; see the section "Specification of procedure interfaces" of the chapter "Analysis".

3.5   The reference structure or call tree

FORCHECK can present the call tree in the listing file, or store it in xml format so you can browse and use it for further analysis or documentation. Though FORCHECK does not display needless repetitions the call tree may take wallpaper proportions. You can, however, specify one or more root nodes from where you would like to see the call tree.

/SRS   Show the reference structure.

/SRS:*r*   Show the reference structure for the roots specified. The specified roots must be separated by a ",", or a ":".

If the /AR option is in effect FORCHECK also analyses the tree. Now procedures that are referenced recursively but are not declared as such, or declared recursively but not referenced recursively are spotted. Unsaved common blocks and module variables which are not specified in the root of the referencing program units are reported. From Fortran 2008 on saving is the default and most compilers will store those objects statically. However, in earlier levels of the standard it is not standard conforming and a potential risk when porting the program to another platform.

3.6   The module dependency tree

FORCHECK can present the dependencies of modules as a tree. You can also specify specific modules for which you want to see the dependencies.

/SMD   Show the dependency tree of all modules.

/SMD:*m* Show the dependency tree for the modules specified. The specified modules must be separated by a ",", or a ":".

3.7   Using library files

The purpose of library files and how to use them is explained in the chapter "Operation", section "FORCHECK library files". The knowledge is needed to understand the next sections, so you are invited to make this detour now.

3.8   Using modules

When importing modules by the USE statement FORCHECK has to import the public items of the module to analyze the code. So the imported module has to be analyzed before analyzing the importing code. FORCHECK stores the public module information in a library file for later reference. If the modules are located in front of the importing program unit or if they are in separate files and you analyze all files in one run, this works fine without noticing. In other cases you must analyze the referenced modules first and store the result in a FORCHECK library. When analyzing the source code which references these modules you specify this library file as a reference library.

3.8.1   Using third-party libraries

When referencing third-party modules, e.g. supplied by the compiler vendor, FORCHECK needs the interfaces to perform the analysis. FORCHECK cannot read the .mod files as supplied by the vendor because they are proprietary binary files. If the source code with the interfaces is supplied by the vendor you can generate a FORCHECK library file containing the interfaces. See the section "FORCHECK library files" of the chapter "Operation" on how to generate the library file. If the interface is not supplied in source code, you can compose it from the documentation as described in the chapter "Analysis", section "Specification of procedure interfaces".

3.9   Portability and conformance to standards

To verify if a program is portable you can instruct FORCHECK to verify if it is standard conforming. See the next subsection "Standard conformance". To make your program suitable for the next Fortran level you can let FORCHECK flag the presence of obsolescent syntax (/OB option). It is also possible to instruct FORCHECK to accept only those language extensions of a compiler that are available in another Fortran language level. This is elucidated in the subsection "Compiler emulation".

3.9.1   Standard conformance

For optimal portability the program should be standard conforming. FORCHECK verifies standard conformance very precisely when you specify the /ST option. When this option is applied FORCHECK validates the syntax for conformance to the Fortran standard of the level that is in effect (as determined by the compiler emulation chosen). All nonstandard syntax will be flagged.

3.9.2   Compiler emulation

By choosing the appropriate configuration file the correct language level is chosen and the supported language extensions are enabled. If you want only those language extensions to be accepted that are in the next Fortran level, you can specify one of the specific conformance options. E.g. if you apply gfortran.cnf and allow all extensions which are in the Fortran 2003 standard you specify the /F03 option.

3.9.3   Setting your own, or company standard

You also can create a specific configuration file in which you enable those specific language extensions which are supported by all of the platforms you use and which are acceptable for good programming practice. You now can verify if all programs conform to this requirement.

3.9.4   Cross-platform development

FORCHECK can also be used for cross-platform development. By specifying the compiler emulation file of the target platform FORCHECK will analyze the program as if you were operating on that target. Problems might arise when include files are being used which are not available, or have filenames that are not acceptable on the host. See the next subsection. It could also be necessary to create interfaces for system calls that are not known on the host. Please contact the FORCHECK developers for advise and assistance. We like to help you and are interested to learn from your experiences.

3.9.5   Using include files

The syntax for the INCLUDE line or include preprocessor directive can vary with the platform for which the program has been developed. FORCHECK can handle most dialects. However, if you analyze the source on e.g. a Windows platform and the source is for unix or VMS, it could be difficult to place the include files in the correct directories.

3.9.6   Multi-platform development

If your code is standard conforming you will have minimal problems to port the program to the various platforms. You can also create a configuration file in which the language extensions which are available on all the platforms you support are enabled. The code which is specific for some of the platforms can be selected using cpp preprocessing, which is supported by FORCHECK.

   Some types can be different on the various platforms. In that case you have to analyze the code for each platform applying the specific configuration files in which the different types are listed. FORCHECK presents a warning if you use the implicit type in one instance and the explicit type in another, e.g. when associating arguments, because that is a portability risk.

Chapter 4

Operation

4.1 Using FORCHECK in command mode

The FORCHECK analyzer can be started from a command window or by double clicking on the **FORCHECK-Command Line** icon. Command input can be entered in three ways: interactive entry, command line entry, and response file entry.

4.1.1 Using Forcheck in interactive mode

If you type the `FORCHK` command without any argument, FORCHECK prompts for each argument: a listing filename, source input filenames, and library files.

By default no listing file will be generated. In this case all messages will be sent to your screen. This is suitable for a quick check of one or more program units. The default file name extension is `.lst`. If you want to create a listing on your screen, type `CON`. For a listing on your printer, type `PRN`, `LPT1`, `LPT2`, or `LPT3`.

Next you are prompted to specify the source input files to be analyzed. You can specify as many source input files as you like, separated by a "+" sign. If you end the command line with a plus sign, FORCHECK prompts for more input files. Every file specification may include a device name and directory. The source input file specifications may contain wild-cards ("*"), you can for example specify `*.FOR` to analyze all Fortran files on the current sub-directory.

After having specified all source input files, FORCHECK prompts for library files. The default is that you don't specify a library file. In that case FORCHECK will store all inter-program unit information in a scratch file, which will be deleted when FORCHECK has completed. You can, however, save this inter-program unit information by specifying a library file. In subsequent FORCHECK runs you can update this library file and reference or include the program units of one or more library files. See the section "FORCHECK library files" for detailed information.

The library files specified must be separated by a "+". When you specify a "+" as the last character on the line FORCHECK prompts for more library files. You can use wild cards ("*") to reference a group of library files.

4.1.2   Command line Entry

With command line entry you enter the `FORCHK` command along with its arguments, without waiting to be prompted.

You must separate each complete argument from the next with a comma ("`,`"). Apart from this, what you enter is the same as with interactive entry. The command syntax is therefore the following:

FORCHK [*listing_file*][*global_options*]`,`[*input_file*[*local_options*]
[+*input_file*[*local_options*]]...], [*lib_file*[*lib_option*][+*lib_file*[*lib_option*]]...]

See the section "Using FORCHECK in interactive mode" for a full description of these arguments.

To accept the default for an argument, just enter the separating comma. You can terminate a partially specified command line by a semicolon ("`;`"). The unspecified arguments all assume the default values.

When you enter an incomplete list of arguments, and no terminating semicolon has been specified, FORCHECK prompts you for the next remaining arguments. See the section "Using FORCHECK in interactive mode" for a full description of FORCHECK prompts.

Examples:

```
FORCHK  ,PROG;
```

This is a simple and quick way to check the syntax of the (sub)program `PROG.FOR`. No listing file will be generated, all messages will be sent to the screen.

```
FORCHK  /F77,SUBR1+SUBR2,PROGLIB/CR
```

The source files `SUBR1.FOR` and `SUBR2.FOR` are analyzed and compared for conformance with the FORTRAN 77 standard. No listing file will be generated. The global information will be stored in the newly created library `PROGLIB.FLB.` The reference structure and the interfaces between the program units will be checked.

```
FORCHK  PRN/CO/SC/SM,,PROGLIB
```

A cross-reference listing of all global information of the entire program (as stored in the library file) and cross-reference listings of all common-block objects and all public module variables will be produced and printed. Unreferenced procedures, unreferenced common blocks, undefined and unreferenced common-block objects, unreferenced modules, unreferenced and undefined public module variables will be listed.

4.1.3   Response file entry

Response files can be used to supply FORCHECK with the appropriate command input. In that case type

FORCHK @*response file*

to start FORCHECK. The command input is now read from the response file instead of from your keyboard. When in interactive mode you can continue command input from a response file by typing @response file as the response to a prompt.

4.2 Options

Options can be set for an entire project and for each individual file. File options overrule the project options. We distinguish five categories of options: options to tune the program-unit analysis, options to tune the global analysis, listing options, library options and miscellaneous options. Library options can only specified for library files so they do not show up in the project and source file options windows. In the following list the mnemonics between parentheses are the corresponding command line options.

4.2.1 Program-unit analysis options

/AC
Analyze all columns of the source input records. If negated and the -ff option is not in effect, only columns 1 to 72 (after expansion of tabs) will be analyzed. See also the sections "Interpretation of source code records" and "Lay-out of source code listing" of the chapter "Analysis". Default: /NOAC.

/AQI
Use the interface of the previously analyzed subprogram with an implicit interface, if present, to verify the references during subprogram analysis. If negated the actual argument lists of the references in the various subprograms will only be verified during global program analysis. You need to specify this option if you analyse an unrelated set of program units, or if you have modified interfaces and have not yet updated the FORCHECK libraries containing the interfaces. Default: /NOAQI.

/CN:c
Allow for a maximum of the specified number of continuation lines. If not specified, the maximum is that as specified in the configuration file. Default: /NOCN.

/CPP
For files with a filename extension of .FPP the default is /CPP. For all other files the default is /NOCPP.

/DC
Present a warning for all variables that are not explicitly declared in a type statement. This is equivalent to specifying the IMPLICIT NONE statement in the source code as supported by Fortran 90 and as a language extension of many FORTRAN 77 compilers. Default: /NODC.

/DE

For compilers which support D_lines, lines with D or d in the first column of the source input records will be processed. They will be treated as comment lines otherwise. For emulation of the Microsoft Fortran compiler, you can specify one character to control the lines you wish to include in the processing: `/DE:character`. If the debug character is not specified, "D" is assumed. All other lines with an alphabetic character in the first column will be treated as comment. Default: `/NODE`.

`/DP`

Map all default reals to double precision and double precision to REAL(16). Map all default complex objects to double complex and all double complex to COMPLEX(16). See also `/R8`. Default: `/NODP`.

`/EX`

Flag all external referenced procedures which have not been declared external. Default: `/NOEX`.

The next five options are to verify for Fortran conformance for a specific language level. They are provided to verify for upwards or downwards conformance relative to the language level of the current analysis. In general, however, you should use the standard conformance option which verifies conformance for the language level used for the analysis.

`/F77`

Check the syntax for conformance with the FORTRAN 77 standard. All nonstandard syntax will be flagged. Note that this option does not enable FORTRAN 77 syntax by itself. To enable FORTRAN 77 syntax a configuration file of a FORTRAN 77 compiler must be selected. Default: `/NOF77`.

`/F90`

Check the syntax for conformance with the Fortran 90 standard. All nonstandard syntax will be flagged. Note that this option does not enable Fortran 90 syntax by itself. To enable Fortran 90 syntax a configuration file of a Fortran 90 compiler must be selected. Default: `/NOF90`.

`/F95`

Check the syntax for conformance with the Fortran 95 standard. All nonstandard syntax will be flagged. Note that this option does not enable Fortran 95 syntax by itself. To enable Fortran 95 syntax a configuration file of a Fortran 95 compiler must be selected. Default: `/NOF95`.

`/F03`

Check the syntax for conformance with the Fortran 2003 standard. All nonstandard syntax will be flagged. Note that this option does not enable Fortran 2003 syntax by itself. To enable Fortran 2003 syntax a configuration file of a Fortran 2003 compiler must be selected. Default: `/NOF03`.

`/F08`

Check the syntax for conformance with the Fortran 2008 standard. All nonstandard syntax will be flagged. Note that this option does not enable Fortran 2008 syntax by itself. To enable

Fortran 2008 syntax a configuration file of a Fortran 2008 compiler must be selected. Default: `/NOF08`.

`/FF`
The source file is in free source form. The specific form of the free source form input depends on the compiler emulation chosen. For files with a filename extension of .F90, .F95, .F03, F2003, F08, or .F2008 the default is `/FF`. For all other files the default is `/NOFF`.

`/I2`
Default integers occupy 2 bytes by default. The length of logicals will depend on the compiler emulated.

`/I4`
Default integers and logicals occupy 4 bytes by default.

`/I8`
Default integers and logicals occupy 8 bytes by default.

`/INTENT`
Flag dummy arguments for which no INTENT attribute has been specified. Default: `/NOINTENT`.

`/INTR`
Flag referenced intrinsic procedures which have not been declared intrinsic. Default: `/NOINTR`.

`/OB`
Flag syntax which is marked as obsolescent in the Fortran standard which is in effect. Default: `/NOOB`.

`/R8`
Map all default reals to double precision. Map all default complex objects to double complex. See also /DP. Default: `/NOR8`.

`/RE`
Relax type checking on integers, logicals and Holleriths. Mixing of integers and logicals in logical and relational expressions will be accepted. Hollerith constants may be used in relational expressions. Default: `/NORE`.

`/SAVE`
Save all variables by default. Default: `/NOSAVE`.

`/SF`
Flag referenced specific intrinsic procedures. Default: `/NOSF`.

`/ST`
Check the syntax for conformance with the Fortran standard of the level that is in effect. All

nonstandard syntax will be flagged. Default: `/NOST`.

4.2.2 Global analysis options

`/CO`
The complete program is analyzed and FORCHECK will flag unreferenced procedures, unreferenced and undefined common blocks, unreferenced and undefined common-block objects, unreferenced modules, unreferenced and undefined public module variables, unreferenced public module constants and unreferenced public module derived types. If the `/AR` option and the `/RI` option are also in effect the call tree will be traversed to detect unsaved common blocks and modules with unsaved public data which are not specified in the root of referencing program units. See also the sections "Analysis of the reference structure", "Verification of common blocks" and "Verification of modules" of the chapter "Analysis". Default: `/NOCO`.

`/AP`
Analyse the the global program. If this option is not in effect, only the individual program units are analyzed. See the section "Global program analysis". Default: `/AP`.

`/AR`
Analyze the reference structure (call tree). See also the section "Analyzing the reference structure" of the chapter "Analysis". Default: `/AR`.

4.2.3 Listing options

`/PL:`*p*
Place a maximum of p lines on a page, p >= 20. By default the IDE automatically takes the value from the page setup characteristics. Default for the command line version: `/PL:62`.

`/PW:`*w*
Place a maximum of w characters on a line, 60 <= w <= 255. By default the IDE automatically takes the value from the page setup characteristics. Default for the command line version: `/PW:100`.

`/RSF:`*file*
Specify the name of a file in which the reference structure will be stored in XML format. If no filename is specified the filename is fckrs.xml. See also the section "Reference structure" of the chapter "Analysis". This is a command line option only. Default: `/NORSF`

`/MDF:`*file*
Specify the name of a file in which the module dependencies will be stored in XML format. If no filename is specified the filename is fckmd.xml. See also the section "Module dependencies" of the chapter "Analysis". This is a command line option only. Default: `/NOMDF`

`/SB`

Generate listings and cross-reference tables of individual program units. The display of source lines can be suppressed by disabling the /SS option. See the section "Program-unit cross references" of the chapter "Analysis". Default: /SB

/SS
List source code. To list source code the /SB option must be in effect also. See the section "Program-unit cross references" of the chapter "Analysis". Default: /SS.

/SH
List lines included from include files. The Program-units cross-reference and the List source line options must also be in effect to list included lines. Default: /SH

/SI
Include unreferenced constants, namelist groups and procedures, declared in include files or modules, unreferenced common-block objects and unreferenced imported module variables in the program-unit cross-references. Default: /SI.

/SP
Show cross-reference listings of the program. See also the section "Global program cross references" of the chapter "Analysis". Default: /SP.

/SRS
Show the complete reference structure of the referenced procedures. See also the section "Reference structure" of the chapter "Analysis". Default: /SRS.

/SRS:r
Show the reference structure for the roots specified. The specified roots must be separated by a ";". Default: /SRS.

/SC
Show cross-reference listings of common-block objects. See also the section "Cross reference of common-block objects" of the chapter "Analysis". Default: /NOSC.

/SC:c
Show cross-reference listings of common-block objects of specified common blocks. The specified common blocks must be separated by a ";". Default: /NOSC.

/SMT
Show cross-reference listings of public module derived types. See also the section "Cross reference of public module derived types" of the chapter "Analysis". Default: /NOSMT

/SMT:m
Show cross-reference listings of public module derived types of specified modules. The specified modules must be separated by a ";". Default: /NOSMT.

`/SMV`

Show cross-reference listings of public module data. See also the section "Cross reference of public module data" of the chapter "Analysis". Default: `/NOSMV`

`/SMV:`m

Show cross-reference listings of public module data of specified modules. The specified modules must be separated by a "`;`". Default: `/NOSMV`.

`/SMD`

Show the dependencies of modules. Default: `/NOSMD`

`/SMD:`m

Show the dependencies of specified modules. The specified modules must be separated by a "`;`". Default: `/NOSMD`.

4.2.4   Library options

`/CR`

Create a new library and insert the analyzed program units in this library. This is a command line option only. Default: `/NOCR`.

`/IL`

Include all program units from the library in the analysis. Default: `/NOIL`.

`/IL:`s

Include specified program units from the library file in the analysis. Default: `/NOIL`.

`/UP`

Update the specified library with the analyzed program units. Default: `/NOUP`

4.2.5   Miscellaneous options

`/BA`

Exit if errors occur during command input and suppress the "Press enter to continue" prompt at exit. When FORCHECK is started from a batch file, this option suppresses interactive handling of command input errors and exit. This is a command line option only. Default: `/NOBA`.

`/ID:`d

Generate a file with all referenced include files. Default: `/NOID`.

`/INF`

Display and count informative messages. Default: `/INF`.

`/WA`

Display and count warnings. Default: `/WA`.

`/MK:`*m*
Append a makefile with dependencies and rules for the source files. The leading part of the file is generated by the IDE. Default: `/NOMK`.

`/RI`
Flag less robust and less portable code at the cost of more informative messages. Do not limit the number of messages for a statement or argument list. This option is useful when developing new code and to improve the quality of existing code. Do not use this option when analysing a project for the first time. Default: `/NORI`.

`/RP:`*r*
Generate a report file r. The default file name extension is .rep. If `/RP` is specified without a filename, the name of the report file will be FCK.REP. This is a command line option only. Default: `/NORP`.

`/TR`
FORCHECK will check whether names are unique after truncation to 6 characters. Default: `/NOTR`.

`/LG`
Show defines and undefines of meta variables. Default: `/NOLG`.

`/IP:`*p*
Specify directories to search for include files. Default: `/NOIP`.

`/DF:`*s*
Define meta symbols for conditional compilation. Default: `/NODF`.

4.2.6   Specifying options in command mode

When an option is specified with the listing file the option is global and is therefore in effect for the entire analysis, except for those input files for which the negated options have been specified. When you specify an option with an input or library file, the option is local, so it will only be in effect for that file. A local option overrules the global option temporary.

The `/AP`, `/AR`, `/CO`, `/DF`, `/ID`, `/IP`, `/MDF`, `/MK`, `/LG`, `/RP`, `/RSF`, `/SC`, `/SMD`, `/SMV`, `/SP`, `/SRS` options are always global and can only be specified with, or at the position of, the listing file. Library file options can only be specified with library files.

The options `/PL`, `/PW`, `/SB`, `/SC`, `/SH`, `/SI`, `/SMD`, `/SMV`, `/SP`, `/SRS`, `/SS` have only effect if a listing file has been specified.

Include directories, common blocks, modules, roots, symbols to be defined and program units in an option argument list must be separated by a "`;`".

An option can be negated by "`NO`", or by a "`-`" sign, for example `/NOF77`, or `/-F77`.

Defaults:

```
/NOAC/AP/NOAQI/AR/NOBA/CN:19/NOCO/NOCR/NODC/NODE/NODF/NODP/NOEX/NOF77/NOF9
/NOFF/NOHE/I4/NOID/NOIL/INF/NOINTENT/NOINTR/NOIP/NOMDF/NOMK/NOLG/NOOB
/PL:62/PW:100/NOR8/NORE/NORI/NORP/NORSF/NOSAVE/SB/NOSC/SH/SI/SMD/NOSMV/SP
/SR/SS/NOTR/NOUP/WA
```

For files with a filename extension of F90, F95, F03, or F08 the default source form is freeform (/FF).

On the page headers of the listing the specified nondefault analysis options will be shown.

You can set the default options using the IDE or by editing the file FORCHK.CFG in the FORCHECK application data directory (the subdirectory forcheck of the APPDATA directory) and specifying these options in the FCKOPT= line. For example:
FCKOPT=/PL:66/PW:100/F77

4.3   Example of FORCHECK command input

Example of FORCHECK command input:


```
C:>FORCHK


    F O R C H E C K  V14.0.0
Copyright (c) 1984-2009  Forcheck b.v. All rights reserved



   -- INT  compiler emulation

listing file [.LST] with global options: FILOUT/F77/NOSH
input file(s) [.FOR] with local options: TSTPRG+SUBR1+SUBR2/NOF77/AC+
input file(s) [.FOR] with local options: \SOURCE\SUBR*/NOSB
library file(s) [.FLB] with options:
```

In this example FORCHECK analyzes the program consisting of the files TSTPRG.FOR, SUBR1.FOR, SUBR2.FOR, and all .FOR files from the directory \SOURCE with names beginning with SUBR. Nonstandard FORTRAN 77 syntax is flagged. Listings with cross references are generated for all input files but \SOURCE\SUBR*.FOR. The statements of include files are not listed. All columns of the input records of the file SUBR2.FOR are analyzed. The listings and cross references are stored in the file FILOUT.LST. No library files will be saved or referenced.

4.4   Exit status

The FORCHECK analyzer exits with a specified exit status which can be tested in for example a batch job.

Exit status:

   0 no informative, warning, overflow, or error messages presented
   2 informative, but no warning, overflow, or error messages presented
   4 warning, but no overflow, or error messages presented
   6 table overflow, but no error messages presented
   8 error messages presented
 16 fatal error occurred

To prevent closing of the window before you could have examined the results FORCHECK prompts to press enter to continue at exit. You can suppress this prompt by specifying the `/BA` option.

4.5 Aborting FORCHECK

You can abort the command line version of FORCHECK by pressing `Ctrl` and `C` key simultaneously. All files will be closed and the scratch files deleted before FORCHECK exits.

     Only in the case of a power fail or when you reboot your system during the operation of FORCHECK you may find some temporary files in the directory denoted by `TMP=` in the file `FORCHK.CFG` or, if not specified, in the current directory.

4.6 The usage of include files

When FORCHECK encounters an include line or compiler directive it tries to open and read the include file specified. When an absolute path has been specified, for example `C:\SRC\FILE.INC`, or `\PROJECT\INCFIL.INC` it opens this file.

     When no device or directory or a relative path has been specified FORCHECK first tries to find that file relative to the directory of the source file in which the include directive has been specified. Then FORCHECK tries to open the include file relative to the current directory. After that it uses the directories as specified by the `/IP:`*p* option. When not found it uses the include paths as specified under Options/Preferences.

     You can set the default include paths using the IDE (choose **Setup/Directories**) or by editing the file `FORCHK.CFG` in the FORCHECK installation directory and specifying the include paths in the `INCLUDE=` line. You have to separate multiple include paths by a ";".

     The default file extension for include files depends on the compiler emulation chosen. See the sections on compiler emulations and supported Fortran syntax for more information.

4.7 FORCHECK library files

FORCHECK stores the global information of all program units in a FORCHECK library file. You can save this file for later reference. The first time you specify a library file it has to be created using the `/CR` option. If global program analysis is in effect (this is the default) all information from the library file is included in the global analysis.

New, or modified program units can now be analyzed and their global information stored or replaced in the library. You now specify the library file with the /UP option. If global program analysis is in effect, all information from the library will again be included in the global analysis.

When the global information of the program units of a program has been stored in one or more libraries in this way, you can analyze the program units in the context of the entire program by referring to these libraries. Now all implicit interfaces are known to FORCHECK and all references of subprograms can be verified. FORCHECK scans the libraries in the specified order and includes all referenced program units found in the global analysis. Each individual library is searched recursively until no references are resolved any more.

You can force FORCHECK to include all or only specific program units from a library in the analysis.

When you specify no source input files but library files only FORCHECK will perform a global program analysis, and presents the reference structure and program cross references when asked for. All information contained in the first library file will be included in the analysis by default. The other libraries are searched for referenced program units as explained before.

When you want to create a library file you specify the /CR option. The library file will be created and the global information of the analyzed program units will be stored in this library file. For example:

```
FORCHK ,TEST.FOR,TESTLIB.FLB/CR
```

will analyze the source file TEST.FOR and place the global information in the newly created library file TESTLIB.FLB.

Now new, or modified program units can be analyzed and their global information stored or replaced in this library file by specifying the library file with the /UP option. For example:

```
FORCHK ,TEST.FOR,TESTLIB.FLB/UP
```

will analyze the source file TEST.FOR and replace the global information in the library file TESTLIB.FLB.

Now you can analyze new or changed program units in the context of the entire program by referring to previously created libraries. If no library options has been specified, FORCHECK references the specified libraries only. For example:

```
FORCHK ,TEST1.FOR,TESTLIB.FLB
```

will analyze the source file TEST1.FOR and verify the procedure references, common blocks etc. of all references which reside in the library file TESTLIB.FLB.
By specifying the /IL option you can force FORCHECK to include all or specific program units from a library in the analysis. For example:

```
FORCHK ,TEST1.FOR,TESTLIB1.FLB/IL:SUB1:SUB2
```

will analyze the source file TEST1.FOR and verify the procedure references of the program units SUB1 and SUB2 which reside in the library file TESTLIB.FLB.

In the next two examples we analyse library files only:

```
FORCHK ,,PROJECTLIB+PLOTLIB
```

will analyze the program consisting of all program units contained in the library file `PROJECTLIB.FLB` and all references found in the library file `PLOTLIB.FLB`.

```
FORCHK ,,PROJECTLIB+PLOTLIB/IL
```

will analyze the program consisting of all program units contained in the library files `PROJECTLIB.FLB` and `PLOTLIB.FLB`.

You can delete, compress and list the information of program units in the library file using the utility `FCKLIB`. See the section "Maintaining library files".

4.8   The usage of modules

When FORCHECK encounters a USE statement it must have the public information of the module at hand. So FORCHECK needs to analyze the referenced modules before the reference is encountered. Therefore FORCHECK analyzes the input files first for "USE dependencies" and determines the order to analyze the input files.

The public information of analyzed modules is stored in the specified create or update library. If no create or update library has been specified this information is stored in a tempory library file. See the section "FORCHECK library files" for information on how to use library files.

You could also analyze modules first and store the public information in one or more libraries. When analyzing the referencing program units you must specify these libraries.

4.9   Maintaining library files

You can list and remove program units contained in a FORCHECK library file and can compress it.

When FORCHECK replaces the information of program units it actually stores the new information at the end of the library file and updates the index. When you remove the information of program units from the library file the librarian only removes the index entry from the library file. To retain the free space from the library file you have to compress it.

Also when you add the information of more and more program units the index of the library file becomes scattered and the global program unit analysis will take more time. Compressing the library file makes the index contiguous again.

4.9.1   Maintaining library files in command mode

`FCKLIB` is a utility to maintain FORCHECK library files. You can list and remove program units and compress the library.

`FCKLIB` is run by typing the `FCKLIB` command, with a library file name and options.

The `FCKLIB` command line has the following form:

`FCKLIB` *library-file/options*

where *options* indicates the actions to be performed.

In interactive mode, you can enter the library file specification as a respond to the system prompt:

library file:

The default file name extension is `.FLB`.

The following options can be specified:

`/BA`   Exit if errors during command input. When FORCHECK is started from a batch file, this option suppresses interactive handling of command input errors.

`/HE`   Present help information on screen.

`/CM`   Compress the library.

`/LI`   List the program units contained in the library. The output is send to the screen.

`/LI:l`   List the program units contained in the library. The output is placed in the file l.

`/RM:s`   Remove one or more program units from the library.

When you specify the /RM option in interactive mode, `FCKLIB` prompts for a program unit to be deleted. New prompts appear until you hit just `<enter>` in response to the prompt.

Examples:

`FCKLIB TSTLIB/RM:SUB`

This command will remove the program unit `SUB` from the FORCHECK library file `TSTLIB`.

`FCKLIB TSTLIB/RM:SUB1:SUB2`

This command will remove the program units `SUB1` and `SUB2` from the FORCHECK library file `TSTLIB`.

`FCKLIB TSTLIB/CM`

This command will create a new, compressed, library `TSTLIB.FLB` out of the existing library `TSTLIB.FLB`.

You can combine the /RM and /CM options in one command:

```
FCKLIB TSTLIB/RM:SUB/CM
```

When you enter the `/RM` and the `/CM` options in one command `FCKLIB` first will remove the information on the specified program units from the library file and then will compress it.

4.10   The usage of language extensions

FORCHECK can analyze programs written in FORTRAN 66, FORTRAN 77, Fortran 90, Fortran 95, Fortran 2003 and Fortran 2008. Moreover FORCHECK supports many language extensions of the various compilers. When using language extensions, however, a program can become less portable. FORCHECK can be used to verify portability and to assist in converting Fortran programs from one platform to another.

When specifying the `/ST` option FORCHECK flags all deviations from the Fortran standard of the level that is in effect, e.g. Fortran 77 when a Fortran 77 compiler emulation has been chosen. If the program is standard conforming, you will have minimal problems when converting the program to platforms which support the same or higher level of the Fortran standard. The `/OB` option can be used to flag syntax which is marked as obsolescent in the Fortran 90 and up standard. The `/RI` option additionally flags less portable code.

The Fortran level, the types and language extensions of a compiler to be emulated are defined in a configuration file. If you want FORCHECK to emulate a different compiler than the default compiler on your system you must do the following.

When using the commandline set the environmental variable `FCKCNF`, for example:

```
SET FCKCNF=C:\FORCHK\g95.cnf
```
When you start the analysis you will now get the message:

```
FCK - g95 compiler emulation
```

In the appendix "Supported Fortran syntax" of the user guide the supported compilers are listed. For each of the supported compilers a compiler emulation file is supplied.   The filenames of the compiler emulation files have the extension `.cnf`.

When operating in command line mode the default file name extensions (suffixes) of source and include files are extracted from the compiler emulation file which can be adapted by the user. See the table with supported language extensions in the appendix "Supported Fortran syntax".

In the appendix "Supported Fortran syntax" the supported language extensions are listed. When you want to enable different language extensions than the default you have to make a copy of the appropriate compiler emulation file and delete or add lines for the specific language extensions. You can find the numbers of these extensions in the appendix "Supported Fortran syntax".

You also can verify if the Fortran syntax extensions of the emulated compiler are accepted by a higher Fortran level. E.g. when specifying the `/F03` option FORCHECK flags all deviations from the Fortran 2003 standard.

4.10.1   Compiler emulation and include files

When you analyze a Fortran source program on a host computer the INCLUDE lines must be processed by FORCHECK and the include files must be opened and read on the host system. Therefore FORCHECK will not check the syntax of the filename specified in the INCLUDE line for conformance to the syntax of the emulated compiler, but allows for the various syntaxes. So, for example, the VAX Fortran syntax INCLUDE '(INCL1)/NOLIST' and INCLUDE 'MODEL:INC1' will be accepted on all systems. You cannot, however, use, for example, the syntax INCLUDE '[USER.PROJ]INCLIB(INCL1)' on non-VMS systems because on non-VMS systems FORCHECK cannot open a member of an include library file. The VMS symbolic path (like MODEL: in the example) is stripped by FORCHECK to allow the file to be found on non VMS systems.

Mind that when emulating a certain compiler, the default file name extension (suffix) of include files is adapted to conform to the defaults of the system and compiler chosen, as described in the previous section.

4.11   Generating Fortran 90 interfaces

The supplied utility INTERF takes a FORCHECK library file as input and produces a Fortran 90 module with an interface body for each of the subprograms in the library file. The output is in Fortran 90 free source form.

This can be useful when converting from FORTRAN 77 to Fortran 90 and to examine the properties of the subprograms as they are contained in the library file. By specifying the module in the program units which reference these subprograms the interfaces of the subprograms become explicit and both the compiler and FORCHECK can verify the references while compiling or analyzing the program unit.

4.11.1   Operation of INTERF from the commandline

The command line has the following form:

INTERF libraryfile[options]

where libraryfile is the name of the FORCHECK library file in which the information of the program units is stored.

Outputfile is the name of the file in which the generated module with the interfaces will be stored. The default suffix is .f.
The following options can be specified:

/BA   Exit if errors during command input. When FORCHECK is started from a batch file, this option suppresses interactive handling of command input errors.

/HE   Present help information on screen.

/LI:l   Place the module with the interfaces in the file l.

4.12 Storing the Reference structure and dependency of modules

Beside presenting the reference structure (call tree) and the dependency of modules in the listing file FORCHECK can store the reference structure and the module dependencies in XML format in separate output files by specifying a filename in the IDE or enabling the reference structure and module dependencies file options. See also the subsection "Reference structure in XML format" and "Module dependencies in XML format" of the chapter "Analysis".

4.13 Messages

We distinguish three kinds of messages, viz. operational messages, analysis messages and system messages.

4.13.1 Operational messages

Operational messages are generated when a problem occurs during the operation of FORCHECK. They are of the form FCK-- ...

For example:

```
FCK-- open error on input or include file
```

For many operational messages an i/o status code is presented. This code is system dependent, and is provided for debugging purposes only. When reporting problems to the FORCHECK support team, please specify the message and the i/o status code. Operational messages are sent to the report file and to your screen or log file.

Analysis messages and system messages are sent to the report file and to the listing file if specified, or to your screen or log file otherwise. The next sections apply to analysis messages and system messages respectively.

4.13.2 Analysis messages

Those analysis messages flagged with an 'I' are informative, with a 'W' are warnings, those flagged with an 'E' are errors.

Informative messages hold no conflicts with the Fortran standard. Warnings indicate the usage of extensions to the standard. Error messages will arise when the Fortran standard has been violated.

The distinction between warnings and error messages, however, is not principal. In general we can say that warnings indicate constructions which, if accepted by your compiler, impose no risk to the proper execution of the program, while errors indicate constructions which may influence the proper execution.

All analysis messages have a number. In the manual appendix "Message summary" you will find a list of all messages with explanation for those messages which are not self-explanatory. During program unit analysis the analysis message is followed by the source line number and

file name to be able to locate the source of the problem in the source file easily. To use this feature you should, however, not change the method of line or statement numbering as described in the section "Line or statement numbering".

The following remarks can be made on the presentation of analysis messages:

· Only the first 6 analysis messages in a statement are presented, unless the /RI option has been specified.

· Only the first 6 problems encountered in an argument list or common block are presented, unless the /RI option has been specified.

4.13.3   System messages

When a problem arises in FORCHECK itself (like overflow of a buffer), a system message in capitals between parentheses will show, for example:

```
** [ 5 O] (TOO MANY PROGRAM UNITS, REMAINDER NOT PROCESSED).
```

A system message is flagged with an O (overflow) or an E (error). Analysis will proceed after an overflow message, the analysis, however, is no longer complete. A system error is usually fatal.

4.13.4   Redefinition and suppression of messages

This section describes how to redefine the severity level flag of FORCHECK's diagnostic messages. To suppress them temporary see the next section.

When using the commandline interface you can redefine or suppress diagnostic messages by extending the configuration file used and add records (using an editor) consisting of the number of the message to be redefined along with the severity level flag that you want FORCHECK to present. The lines with the messages to be redefined must be placed in the section "[MESSAGES]".

The numbers and default severity level flags of the messages can be found in the appendix "message summary". If you specify a level flag ' ' (blank) then the message will be suppressed fully, and will not be counted either. For Example:

```
335 'I'
53 ' '
```

These compiler emulation file records specify that the diagnostic message "type conflict" now will be presented as Informative message and "tab(s) used" will neither be presented nor counted.

To present specific messages only you can suppress all diagnostic messages by placing the following line in this section:
```
suppress='all'
```
and subsequently list all messages that must be presented with its severity level.

To activate this compiler emulation file see the section "The usage of language extensions".

You can also concatenate a system configuration file with a private option file. To do so, you place the lines with the redefinitions of messages in a separate file, and redefine FCKCNF

with the list of the two configuration files to be concatenated, for example:

```
SET FCKCNF=C:\FORCHK\G95.CNF;FCKOPTNS.CNF
```

In which `FCKOPTNS.CNF` is your options file.

4.13.5 Temporary suppression of messages

To suppress analysis messages temporary you can insert FORCHECK directives in your source code. First you have to define the mnemonic of the directive of your choice, beginning with an '!'. You specify this directive string on the "compiler directive" line of the "Various" section of the configuration file to use. For example:

```
'!DEC$' '!fck'                              'compiler directive strings'
```

To define '!fck' as directive in addition to the '!DEC$' compiler directive.

Now you can use this directive to disable and enable FORCHECK diagnostic messages in the source code. You can either suppress messages in a block of code or in a single statement. To suppress messages in a block of code add a line with the directive followed by a list of the message numbers which you want to suppress, each message number preceded by a minus sign. To enable messages again, add a line with the directive followed by a list of message numbers, each preceded by a plus sign. You can add online comment after the list of messages. For example:

```
      CHARACTER*120 CH1, CH2
      DATA CH1,CH2/2*' '/
!fck -313 -384     !suppress "possibly no value assigned" and "truncation"
      CH1 = '123'
      CH2 = 'ab'
!fck +313 +384
```

To suppress messages for a single, compound, or line with a list of statements only, add the directive with the list of messages you want to suppress, each preceded by a minus sign, after the first line of the statement. For example:

```
      CHARACTER CH*120
      DATA CH/' '/
      IF (.TRUE.) CH = '123'               !fck -384 -314
```

4.13.6 Reporting messages

During subprogram analysis a message is presented in the listing file after the relevant source code statement. In the report file, or if no listing file has been requested the message is generally preceded by the source code statement. You can suppress the source code statement in the report file by specifying:

```
      source_stm = 'no'
```

in the [VARIOUS] section of the configuration file. You also can suppress only the line or statement number of this source code statement:

```
    source_linstm_number = 'no'
```
When presenting a message FORCHECK adds a line with the filename and line number. The format of this line can be specified, e.g.:
```
    file_line_format = '("(file:  ",a,", line:  ",i0,")")'
```
The output of the filename and line can me made gnu-conforming by specifying:
```
    file_line_format = '(a,":",i0,":")'
```
If you replace the i0 edit descriptor by an x the line number will be suppressed.

4.14   Tuning the output

The output options as decribed in the section 'Options' determine which parts of the analysis are displayed in the listing file. Moreover using the miscellanious options you can specify if you want to create a report file and if you want to present the internal table usage.

Beside using these command line options you can specify what information is sent to stdout, is stored in the listing file, and in the report file. You can do this by setting keywords in the [OUTPUT] section of the configuration file. In the following table the keywords that can be applied are listed with their meaning and default value. Acceptable keyword values are 'TRUE' and 'FALSE'.

STDOUT_MSGSUM	send message summary to stdout	true
STDOUT_METRICS	send metrics to stdout	false
STDOUT_USAGE	send internal table usage to stdout	false
LISTING_MSGSUM	display message summary in listing file	true
LISTING_METRICS	display metrics in listing file	true
LISTING_USAGE	display internal table usage in listing file	false
REPORT_MSGSUM	store message summary in report file	true
REPORT_METRICS	store metrics in report file	true
REPORT_USAGE	store internal table usage in report file	true

For example, if you want to see the message summary on your screen and the metrics not, you specify the following lines in the [OUTPUT] section of the configuration file:

```
STDOUT_MSGSUM = 'TRUE'
STDOUT_METRICS = 'FALSE'
```

Note that the keyword value has to be placed within apostrophes. You can concatenate a system configuration file with a private option file as described in the section 'Redefinition and suppression of messages'.

4.15   Line or statement numbering

By default FORCHECK numbers each source input line sequentially. Lines in include files are numbered in an hierarchical way. Line numbering starts anew for each source input file. In this way you can use your editor to locate the lines of interest in the easiest way.

However, you can instruct FORCHECK to number lines or statements in a different way. To do so, you can place `count_mode` option lines in the [VARIOUS] section of the configuration file. The lines to be added have the form `count_mode = 'mode'`, in which mode can be:

`line`	number source input lines
`statement`	number statements
`new_in_sub`	start numbering anew for each subprogram
`new_in_file`	start numbering anew for each source input file
`new_in_include`	apply hierarchical numbering for included lines c.q. statements
`continue_in_include`	proceed numbering sequentially for included lines c.q. statements

For example, if you want statement numbering, beginning from 1 in each subprogram and proceed statement numbering sequentially in included lines, you specify the following lines in the [VARIOUS] section of the configuration file:

```
count_mode = 'statement'
count_mode = 'new_in_sub'
count_mode = 'continue_in_include'
```

Note that the mode keyword has to be placed within apostrophes. You can concatenate a system configuration file with a private option file as described in the section 'Redefinition and suppression of messages'.

4.16   Date and time format

By default FORCHECK presents the date and time according to the ISO standard. You can change this by adding a `date_format` or `time_format` option line to the configuration file in the [VARIOUS] section of the configuration file. The lines to be added have the form `date_format = 'format'` and `time_format = 'format'`, in which format is a template for the presentation of the date and time respectively.

In the template for the date the day must be specified by dd, the month by mm or mmm (which causes a three letter mnemonic of the month to be displayed), the year by yy or yyyy. The year, month and day codes must be separated by a character of your own choice which will be used as separator in the actual presentation.

In the template for the time the hours must be specified by hh or h (which causes hours below 10 to be displayed with one digit), the minutes by mm, and the seconds by ss. The hour, minutes and seconds codes must be separated by a character of your own choice which will be used as separator in the actual presentation.

e.g.:

```
date_format = 'yyyy-mm-dd'
date_format = 'mmm-dd-yy'
date_format = 'dd/mm/yyyy'
time_format = 'hh:mm:ss'
```

```
time_format = 'h:mm:ss'
```

If you use an x as the format character, the date and/or time will be suppressed in the listings. This can be usefull if you want to compare listings of different FORCHECK runs.
e.g.:
```
date_format = 'xx xx xx'
time_format = 'xx xx xx'
```

When using the IDE you can choose the format by selecting
Options/Preferences/Date/time Format.

Chapter 5

Analysis

In this chapter we describe concisely what FORCHECK actually does and what the generated output exactly means. The analysis is carried out in three stages: the analysis of the separate program units, the analysis of the reference structure and dependency of modules, and the analysis of the integral program. Command-line options determine which of the analysis stages are activated. Beside specifying options you can specify language extensions and analysis options in the configuration file which is in effect.

5.1   Program unit analysis

5.1.1   Interpretation of source code records

If you specify the /FF option FORCHECK reads the source input in free form, as supported by the compiler emulation chosen. If you specify the /ST, /F90, /F95, /F03, or /F08 option as well, FORCHECK reads the source input according to the Fortran 90 and up free-source form standard.

Tabs are expanded to blanks before the statement is processed. In fixed source form source lines are extended with blanks or truncated in the following way. If a source line, after expansion of tabs, consists of less than 72 characters, it will be extended with blanks to 72 characters. This is significant for character and Hollerith constants. Any characters beyond column 72 are ignored, unless the /AC option is in effect.

Lower case characters are converted to upper case before interpretation, except within character and Hollerith constants. If your compiler does not accept lower case characters, tabs, or form feeds or when you specify one of the Fortran standard conformance options one message only will be given for each subprogram to inform you that you used lower case characters, tabs, or form-feeds respectively. So not every lower case character, tab, or form feed will be flagged separately. Also if you use include files, only one warning for each subprogram will be presented.

5.1.2   Lay-out of source code listing

A source code listing is generated if a listing file has been requested, and both the /SB and the /SS options are in effect. To make clear which part of fixed source records is being ignored,

the source record past column 72 of non-comment records is printed at column 83 and higher. Comment records, however, are printed as they are. If the /AC or the /FF option is enabled, all records are printed as they are.

Source input lines or statements are numbered as described in the section "Line or statement numbering" of the chapter "Operation". If the /SH option is specified, input records which are read from an include file are presented with hierarchical line numbers.

The pages on the listing file are numbered. When you use FORCHECK's library facility a hierarchical page numbering system is provided. In that case FORCHECK maintains a library version number which is updated each time you insert or replace program units in the library. The page numbers printed on the listing present the library version number and the page sequence number as "version.page".

5.1.3   Syntax analysis

FORCHECK verifies the syntax of each program unit. If the /ST option is in effect the the syntax will be verified for conformance to the Fortran standard of the level that is currently in effect. If the /F77, /F90, /F95, /F03, or /F08 option is in effect, the syntax will be verified for conformance to the FORTRAN 77, Fortran 90, Fortran 95, Fortran 2003 or Fortran 2008 standard respectively, as close as possible during static analysis. For Fortran 90 and up all constraints, as specified in the standard, are verified. By specifying the /OB option FORCHECK signals all obsolescent features as specified in the Fortran standard which is in effect.

You can also instruct FORCHECK to accept certain vendor specific Fortran language extensions. The appendix "Supported Fortran syntax" describes all language extensions supported. By default FORCHECK accepts common extensions of the default compiler of the system on which FORCHECK operates. To emulate a different compiler or to enable a different set of language extensions, see the section "The usage of language extensions" of the chapter "Operation".

Beside performing a lexical analysis and parsing the syntax, FORCHECK performs a semantic analysis. FORCHECK presents a message if a variable is referenced without being defined. Unless the /RI option has been enabled this is limited to statements which are certainly executed sequentially.

Loop structures, IF-THEN-ELSE blocks and CASE constructs are verified. Because of this, extended DO loops (though this is a language extension of some compilers) will always be flagged as an error by FORCHECK.

5.1.4   Type verification

As part of the syntax analysis FORCHECK detects type conflicts. In general the typing rules are applied more strictly than most compilers do. Type checking is relaxed for type less data and if the /RE option has been enabled. FORCHECK signals implicit type conversions if they impose a certain risk:

· A character datum that is converted to a shorter type, or an integer that is converted to a shorter integer ("truncation").

· A real or complex expression that is converted to a type of less precision.

· A complex expression that is converted to a real.

· A real expression that is converted to a complex.

· A literal constant that is specified in a type with less precision than that of the target. This check is relaxed for the value zero.

If you specify the /RI option any implicit type conversion will be flagged. Moreover padding of character variables with blanks will be flagged unless the right hand side of the assignment statement is a character constant with zero length or consists of blanks only.

5.1.5   Local verification of argument lists

Within a program unit the argument list of each reference of a procedure is compared with the declared interface if the interface is explicit. If the interface is implicit, FORCHECK tries to locate the interface in the temporary and specified library files. If the interface is not found the argument list is compared with that of the first reference.

The number of arguments, data types and data-type kind and length must correspond. When an argument is a scalar at one reference, the argument cannot be an array name at another reference. In that case the message "array versus scalar conflict" will be presented.

An array element as actual argument is compatible with both an array name and a scalar. In that case the first occurrence, other than an array element, determines the expected argument type of the referenced procedure. If array shapes differ and the /RI option is in effect you are informed.

For argument lists of dummy functions and subroutines all these checks are relaxed and only informative messages will be presented.

Only the explicit interface specified or the first argument list of an implicit interface, augmented with type information as described, of each reference will be stored to be used in the global program analysis.

5.1.6   Verification of procedure entries

FORCHECK verifies the dummy (formal) argument list of each individual ENTRY statement of a procedure. Unreferenced dummy arguments are flagged. If a dummy procedure name is used after an ENTRY statement, it must be present in the argument list of that ENTRY statement. Arguments that specify the dimension of adjustable arrays must be present in each ENTRY argument list in which the name of the adjustable array occurs. After each ENTRY statement FORCHECK will detect variables which are referenced before they are defined, as long as the statements are executed sequentially or if the /RI option has been enabled.

If the /RI option is in effect FORCHECK informs you if the "entry blocks" are not disjoint, that is to say if paths from one ENTRY statement and another coincide. This is relaxed for an ENTRY statement which follows the specification statements immediately.

5.1.7   Fortran intrinsic procedures

For each reference of a Fortran intrinsic generic function, FORCHECK generates a specific function according to the data type and data-type kind and length of the arguments. The name of the generated specific function is inserted in the cross-reference table of referenced procedures.

FORCHECK does not need to recognize all specific functions of every compiler because you should use preferably the appropriate generic function. Only for type conversion of actual arguments you may need specific functions, which are supplied.

FORCHECK can flag each intrinsic function which has not been declared intrinsic by specifying the /INTR option. By specifying the /SF option you can flag each specific intrinsic function used.

5.1.8   Function procedure

If the function performs external i/o, (de)allocates memory, contains a STOP or PAUSE statement, modifies any argument, common-block object, or saved item and the /RI option has been enabled, the function is flagged as "impure".

5.1.9   Program-unit cross references

Program-unit and procedure cross references are generated if a listing file has been requested and the /SB option is in effect. If no program-unit cross references are being generated, all diagnostic messages are sent the report file and to your screen or log file. An "..." after a list of line or statement numbers in a cross-reference table indicates that there are more references to that item than could be presented.

The cross-reference table of each module- and internal procedure is presented straight after its source code listing. The cross-reference tables of the program unit are presented after all module- and internal procedures.

Variables in a statement context (data-implied-do-variables, ac-implied-do-variables, forall-indices, and statement-function-dummy-arguments) are not included in the cross-reference lists. The cross-reference tables of module- and internal procedures contain locally declared objects and use-associated objects from locally referenced modules only. Host-associated objects are listed in the host program unit cross-reference tables.

Subprogram entries

The cross-reference table of entries displays the following information:

- · The name of the program unit or procedure entry.

- · The program unit or procedure entry type.

- · The type of the result.

- · The nondefault type kind and length of the result.

- · The rank of an array valued result.

- · The number of dummy arguments.

- · The line or statement numbers of all occurrences of the name of the entry.
  The line or statement number at which the entry is defined is flagged with a "#".

Program unit and procedure types:

B BLOCK DATA program unit

F function

M module

P main program

S subroutine

Subcodes:

M module

N interface

R recursive

T internal

Intrinsic types of function entries, named constants, variables, and referenced functions:

C complex

CH character

R real

I integer

L logical

N numeric (integer, real, or complex)

? typeless

Labels

The cross-reference table of labels displays all labels,the label type, and the line or statement number of all occurrences. The line or statement number at which the label is defined is flagged with a "#".
Label types:

F format

L DO loop

For labels, other than `DO` loop or `FORMAT` statements, the label type field is left blank.

Derived types

The cross-reference table of derived types displays the following information:

· the name of the derived type.

· the type length: the number of bytes a scalar instance of this type will occupy.

· the line or statement numbers of all occurrences of the name of the derived type.
The line or statement number at which the type is defined is flagged with a "#".

Unreferenced derived types, which are not specified in an include file or a referenced module, are listed. These derived types are not used and can therefore be removed from the program unit without affecting the operation of the program.

Constants

The cross-reference table of named constants displays the following information:

· The name of the constant.

· The type: see entries.

· The nondefault type kind and length.

· The rank of array valued constants.

· The size the constant occupies.

· The line or statement numbers of all occurrences of the name of the constant.
The line or statement number at which the constant is defined is flagged with a "#".

Only when the /SI option is in effect, all unreferenced constants which have been specified in an include file or module, are listed.
For types of named constants see the section on entries.
Unreferenced constants are listed, except those which are defined in an include file or referenced module. These constants are not used and can therefore be removed from the program unit without affecting the operation of the program.
To get an idea of its size FORCHECK presents the total size of the referenced named constants.

Variables

The cross-reference table of variables displays the following information:

· The name of the variable.

· The type: see entries.

· The nondeafult type kind and length.

· The rank of arrays.

· The size the variable occupies.

· The operation codes.

· The line or statement numbers of all occurrences of the name of the variable.
  The line or statement numbers at which the variable is (re)defined are flagged with a "#".

The kind of usage of variables and procedures is presented as a set of operation codes with the listed meaning. Only one set of operation codes is presented for each variable. The set of operation codes presented is the or-ed set of operation codes on all array elements, structure components, or character positions of a variable. The operation codes of the various array elements, components, or character elements cannot be viewed separately.
operation codes:

A "defined" by means of

- an assignment statement
- an actual argument associated with an INTENT(OUT) dummy argument
- a statement function definition statement
- an ASSIGN statement
- "associated variable" in DEFINE FILE or OPEN
- "IOSTAT=" in an IO statement
- an INQUIRE statement

C in COMMON
D initialized in a DATA or explicit type statement
I input by means of

- READ, or ACCEPT
- list in DECODE
- conversion buffer in ENCODE
- internal file in a READ

L DO variable, or FORALL index
O output by means of

- WRITE, TYPE, PRINT
- list in ENCODE
- buffer in DECODE
- internal file in a WRITE

P dummy argument
Q in EQUIVALENCE
R referenced, for example by means of:

- an expression
- an argument of an intrinsic procedure
- an argument of a statement function
- an actual argument associated with an INTENT(IN) dummy argument

S actual argument associated with a dummy argument with unknown intent or INTENT(INOUT).

An "*" after C, or Q denotes that the name is not referenced (used) and therefore is dummy. When variables are specified in an EQUIVALENCE statement, the operation codes are presented for each variable name separately. However, when a variable is in a common block, all objects

specified in the equivalence lists concerned, are in common and a "C" will be presented for all these objects. An "*" after this C indicates that none of the objects in the equivalence lists, containing this variable, have been used.

Only when the /SI is in effect, common-block objects, and module data that are not referenced, are included in the cross-reference listing. Referenced but undefined variables are flagged. Unreferenced variables are flagged, except those which are in common or in a module. They are not used and can therefore be removed from the subprogram without affecting the operation of the program.

To get an idea of its size FORCHECK presents the total size of the used local variables. Use associated, allocatable and automatic objects are not included. Variables with the POINTER attribute account for the size of a pointer only.

Structures and records

Structures and records are a Fortran language extension as offered by some compiler vendors. The cross-reference table of records displays the following information:

- · The name of the record.

- · The name of its structure.

- · The length of the structure: the number of bytes a record occupies.

- · The rank for arrays of records.

- · The operation codes.

- · The line or statement numbers of all occurrences of the name of the record.
  The line or statement numbers at which the record is (re)defined are flagged with a "#".

The kind of usage of records is presented as an operation code as described for variables. As for arrays, only one operation code is presented for each record or array of records. This is the or-ed operation code of all the operations on the various fields of the record and the various array elements of an array of records.

Only when the /SI option is in effect, common-block objects, and module records that are not referenced, are included in the cross-reference listing. Unreferenced records, which are not in common or in a module, are listed. Unreferenced structures, which are not specified in an include file or module, are also listed. They are not used and can therefore be removed from the subprogram without affecting the operation of the program.

Namelist groups

The cross-reference table of namelist groups displays the following information:

- · The name of the namelist group.

- · The line or statement numbers of all occurrences of the name of the namelist group.
  The line or statement number at which the namelist group is defined is flagged with a "#".

Only when the /SI option is in effect, unreferenced namelist groups, which have been specified in an include file or module, are listed.

Referenced procedures

The cross-reference table of referenced procedures displays the following information:

- · The name of the procedure.

- · The type: see entries.

- · The nondefault type kind and length of a function.

- · The rank of array valued functions.

- · The operation codes.

- · The line or statement numbers of all occurrences of the name of the procedure.

Procedure types:

E external procedure, unknown whether subroutine or function
F function
S subroutine
P procedure

Subcodes:

D dummy
E elemental
G generic
I intrinsic
M module
N interface
n abstract interface
P pure
p pointer
R recursive
S statement
T internal

For the type of functions see the section on entries. Only when the `/SI` option is in effect, unreferenced procedures which have been specified in an include file or module, are listed.

When flagged as unreferenced the external declaration can be removed from the subprogram, except when it declares a block data subprogram to be included by the linker.

Operators

The cross-reference table of operators displays the following information:

- · The name of the operator.

- · The line or statement numbers of all occurrences of the operator.

When flagged as unreferenced the definition of the operator can be removed from the subprogram.

Common blocks

The cross-reference table of common blocks displays the following information:

- · The name of the common block.

- · The type.

- · The size of the common block.

- · The operation codes.
  The or-ed operation code of all objects in each common block is presented.

- · The line or statement numbers of all occurrences of the name of the common block.

Common-block types:

CH character
N   numeric

If both character and numeric variables are stored in a common block the type will be left blank.

The size of the common block is presented in bytes. If the name table is full, or if the common block has too many objects to check, or if an array is too long, the size cannot be determined and will be left blank.

When none of the objects of a common block have been used, the common block will be flagged as unreferenced unless is has been specified in an include file or a referenced module. When flagged as unreferenced the common block declaration can be removed from the subprogram, except when this subprogram is the root of those subprograms which use this common block and the common-block does not have the SAVE attribute in each of the occurrences. In that case the declaration may be necessary to save the data and the linker may need it to build correct overlay structures.

External files

The usage of external files is shown as a list of unit-identifiers with access types and operation codes. The unit-identifier is the name or expression as specified in the I/O statement.

The value of the unit-identifier is not known to FORCHECK. Therefore I/O references may be placed incorrectly together or separately. By using consistent names for all unit-identifiers throughout the program the I/O reference tables, however, will be of utmost value.
type of I/O:

D direct access
Q sequential access
S stream access
F formatted
U unformatted

When the access type or format type is unknown to FORCHECK, the access type field or format type field will be left blank.
I/O operation codes:

```
A auxiliary: REWIND, BACKSPACE, ENDFILE, DELETE, UNLOCK, or LOCKING
C CLOSE
F FIND
I INQUIRE
O OPEN, or DEFINE FILE
R READ, or ACCEPT
W WRITE, REWRITE, PRINT, or TYPE
```

Include files

Include files which contain only definitions of constants, variables, and common blocks which are not referenced outside the include file are marked as unreferenced except in the specification part of a module. Then the INCLUDE line can be removed from this program unit, except when common blocks, which are in the root of those subprograms which use these common blocks and do not have the SAVE attribute, have been declared in the include file concerned. In that case the declaration may be necessary to save the data and for your linker to build correct overlay structures.

5.2 Reference structure (Call tree)

The reference structure (call tree) is analysed if the /AR option is in effect. The reference structure is presented in the listing file if a listing file has been requested and the /SRS option is in effect. The reference structure is stored in XML format in the reference-structure file if the /RSF:*file* option has been specified.

5.2.1 Analysis of the reference structure

If the /AR option and the /RI is in effect the call tree will be traversed to detect unsaved common blocks and modules with unsaved public data which are not specified in the root of referencing program units.

Recursive references are traced, also if one of the entries of a procedure in the chain is being referenced. If recursive reference is not supported, or the procedures in the chain are not specified RECURSIVE, these procedures are flagged. Moreover, if the /CO option has been specified and a procedure is specified RECURSIVE but is not recursively referenced, it is flagged.

5.2.2 Display of the reference structure

All referenced procedures are presented in a call tree. For each program unit or procedure each referenced procedure is presented only once and in order of occurrence in the source code. The reference structure is static only and does not show the actual sequence of calls during program execution. Module procedures are "qualified" with the name of the module from which they are referred. Renamed procedures are presented by their "use" name.

The lines are being numbered and when a sub tree has already been presented, a reference is made to the line at which the sub tree was presented, for example:

```
1     PROGRAM
2            SUBR1
3                  SUB2
4                            FUN1
5                            FUN2
6                                        FUN21
3            SUBR2
4                  SUB2 > 3
```

For the reference structure all entries of a procedure are equivalent, so if an entry with its call
tree has been presented, all next entries referenced will refer to this sub tree.

   Unreferenced entries with their call tree are presented as separate sub trees and are num-
bered in a hierarchical way, for example:

```
1     PROGRAM
2            SUBR1
3            SUBR2

1.1   MAIN2
1.2          SUBR3
1.3          SUBR4
```

When long names are being used and the nesting is too deep for the reference structure to fit
on the page, the tree is continued as a separate sub tree and a reference is made to the line at
which the continued tree starts, for example:

```
1     PROGRAM_LONG_NAME
2            SUBROUTINE1_LONG_NAME
3                  SUBROUTINE11_LONG_NAME
4                        SUBROUTINE111_LONG_NAME > 1.1
5            SUBROUTINE2_LONG_NAME

1.1  >
1.2   SUBROUTINE1111_LONG_NAME
```

   When a procedure has more references than FORCHECK can store in its tables a message
will be printed and the remaining referenced procedures with its references will be printed in
separate sub trees.

5.2.3   Display of sub trees of the reference structure

One or more separate sub trees can be displayed by specifying the roots of the sub trees as the
root list in the /SRS:*r* option. Now the referenced procedure tree is displayed down from the
procedures specified only.

5.2.4   Reference structure in XML format

The reference structure is stored in XML format in the reference-structure file together with its data type definition (dtd). Reference is made to the XSL-stylesheet file _fck_tree.xsl which must be in the working directory. With a suitable browser you can browse through the reference structure. Suitable browsers are the one integerated in the FORCHECK IDE, Mozilla Firefox, Microsoft Internet Explorer, Opera and Apple Safari. You can also transform the XML file to an HTML file, using for example the Unicorn Enterprises SA XSLT processor
(http://www.unicorn-enterprises.com/products_uxt.html)
the HTML file can then be explored using your internet browser. Because the data are stored in xml format you also can write your own programs to analyse and visualize the reference structure.

5.3   Display of module dependencies

The dependencies of modules is presented in in the listing file as a tree view if the /SMDis in effect. The dependencies of modules is stored in XML format in the dependencies of modules file if the /MDF:*file* option has been specified.

5.3.1   Display of dependencies for specific modules

The dependencies of specific modules can be displayed by specifying these modules the root list in the /SMD:*m* option. Now the module dependencies tree is displayed down from the modules specified only.

5.3.2   Display of module dependencies in XML format

The module dependencies are stored in XML format in the module dependencies file together with its data type definition (dtd). See the section "Reference structure in XML format" for information how to use this file.

5.4   Global program analysis

Global program analysis is carried out if the /AP is in effect.

5.4.1   Verification of procedure references

FORCHECK checks the type of all references, the type, the type length, the rank and shape of referenced functions. Conflicts of user procedure names with intrinsic procedures are detected. When the /CO has been enabled, unreferenced procedures will be listed.

5.4.2   Verification of argument lists

The argument lists of each procedure reference is compared with the dummy (formal) argument list of the analyzed procedure. When the referenced procedure has not been analyzed, the

argument lists will be compared with that of the interface definition provided, or with that of the first reference. Verification is done as specified in the section "Program unit analysis".

Arguments are compared for type, and type parameters. If the /RI option has been enabled and the rank or shape of array arguments differ, you are informed. If a dummy array argument is longer than the actual an error is presented.

If an actual argument is a constant, expression, active DO variable, an active FORALL index or if a variable is specified more than once in an actual argument list, then it is invalid to (re)define the dummy argument in the procedure. In that case the message "invalid assignment" will be given with the reason. This check will only be performed one reference level deep.

If the assigned dummy argument appears in more than one argument list of the entries of a procedure, this verification is only carried out, as long as the entries are disjoint.

If a dummy argument is not defined, or referenced before defined, the corresponding actual argument must be defined before each reference. Because FORCHECK's limited path-flow analysis, referenced-before-defined of dummy arguments will only be flagged as long as statements are guaranteed to be executed sequentially, or if the /RI option is in effect.

When the actual argument is a literal constant without a kind parameter or a constant expression of primaries without a kind parameter the type length is supposed to be the default type length of the type of the constant or constant expression.

5.4.3 Verification of common blocks

The type, size and list of objects of common blocks are compared with the occurrence in the main program, if present, or with the first occurrence otherwise. The size of the largest occurrence of the common block is presented in the cross-reference table. An occurrence of a common block with a different list of objects will be flagged with the message "inconsistent list of objects". If the /RI option has been enabled each inconsistent object will be flagged separately. An object could differ in type, type parameters, array length, array rank, or shape.

When the /CO option is in effect and all occurrences of a common block are identical, common-block objects which are not referenced, not defined, not associated, or not defined before referenced will be listed. If the /RI option has been enabled each common-block object which is only conditionally defined before referenced is listed also.

When a common block has been specified in an include file, it should be included from the same include file at all instances. If that is not the case an informational message will be presented.

If the /AR option is also in effect the call tree will traversed to detect unsaved common blocks which are not specified in the root of referencing program units. See also the section "Analysis of the reference structure".

5.4.4 Verification of modules

When the /CO option is in effect each module which is analyzed but not referenced is reported. All public module variables which are not referenced, not defined, not allocated or not associated will be listed. All public constants and public derived types which are not referenced are listed

If the /AR option is also in effect the call tree will be traversed to detect modules with unsaved public data which are not referenced in the root of referencing program units. See also the sections "Analysis of the reference structure".

5.4.5   Global program cross references

Global program cross references are generated if a listing file has been requested and the /SP option is in effect. If no global program cross references are presented, all diagnostic messages are send to your screen or the log file. An "..." after a list of names in a cross-reference table indicates that there are more references to that item than presented.

Module procedures are "qualified" with the name of the module from which they are referenced. Renamed procedures are presented by their "use" name.

Program units and procedures analyzed

In this table all program units and module procedures which have been analyzed are listed with the page number of the listing and the filename in which the program unit or module procedure resides. When you did not ask for a listing of a specific program unit its page number will be left blank.

When you use FORCHECK's library facility then a hierarchical page number system will be applied. The library maintains a version number for each program unit which has been stored and for which a listing has been made. This program unit version number becomes the library version number at the moment you insert or replace the program unit. The library version number will increase at each FORCHECK run in which you update the library. In the table of analyzed program units and procedures the version number and page number are shown as "version.page".

Referenced procedures not analyzed

All referenced procedure entries which were not analyzed are listed here. Because a program often references external procedures of which no Fortran source is available to include in the FORCHECK analysis (for example system library routines), no separate messages will be presented for these "undefined references". To make the analysis more complete see the section "Specification of procedure interfaces".

Cross reference of program units and procedures

All names of the program, modules, block data program units, external and module procedures are listed with their type and number of arguments. For functions the type with nondefault kind and length will also be presented. For each procedure all program units and procedures which reference that procedure are shown.
Program unit and procedure types:

```
B  BLOCK DATA program unit
E  external, unknown whether subroutine or function
F  function
M  module
```

```
P  main program
S  subroutine
```

Subcodes:

```
E  elemental
M  module
N  interface
P  pure
R  recursive
```

Intrinsic types of functions and function entries:

```
C   complex
CH  character
I   integer
L   logical
R   real
?   typeless
```

The total size of the local data of all program units and procedures is presented. Allocatable and automatic objects are not included.

Cross reference of common blocks

All common blocks referenced in the program are listed with all subprograms in which the common blocks have been specified. A "#" in front of a subprogram name indicates that the common block is modified directly in that program unit or procedure. Mind that if a common-block object is used as an actual argument of a procedure reference, a modification of the common block in that procedure will not be indicated.

   The type of the data in each common block and the common-block size in bytes are presented. When the common block has been saved this will be indicated.

Common-block types:

```
CH  character
N   numeric
```

When types have been mixed the common-block type will be left blank.

   The size of the common block is presented in bytes. When the name table is full, or the common block has too many objects to check, or when an array or record is too long, the size cannot be determined and will be left blank. The largest size of all occurrences of the common block is presented
The total size all common blocks will occupy is presented.

Cross reference of external files

All external files used in the program are shown as a list of unit-identifiers with all subprograms in which the external files are referenced. The types and operation codes are presented.

The unit-identifier is the name or expression as specified in the I/O statement. Because the value of the unit-identifier is not known to FORCHECK I/O references may be placed incorrectly together or separately. By using consistent names for all unit-identifiers throughout the program the I/O reference tables, however, will be of utmost value.

Type of I/O:

- `D` direct access
- `F` formatted
- `S` sequential access
- `U` unformatted

When the access or format type is unknown to FORCHECK the access or format type will be left blank.

I/O operation codes:

- `A` auxiliary: `REWIND, BACKSPACE, ENDFILE, DELETE, UNLOCK, LOCKING`
- `C` `CLOSE`
- `F` `FIND`
- `I` `INQUIRE`
- `O` `OPEN`, or `DEFINE FILE`
- `R` `READ`, or `ACCEPT`
- `W` `WRITE, REWRITE, PRINT`, or `TYPE`

Cross reference of modules

For each module all subprograms which reference that module are presented.
Module type;

- `I` module nature is intrinsic
- `N` module nature is non-intrinsic
- `S` submodule

Cross reference of include files

For each include file all program units which contain that include file are presented.

5.4.6 Cross references of common-block objects

Cross references of common-block objects are displayed if a listing file has been requested and the /SC option is in effect.

All objects of each common block for which a cross-reference table is requested are listed with all subprograms in which the common-block object is used. A "#" in front of a subprogram name indicates that the common-block object is (re)defined directly in that subprogram. Mind that if a common-block object is used as an actual argument in a subprogram, the common-block object may be (re)defined indirectly.

A cross-reference of common-block objects is only meaningful if the lists of objects at the various occurrences of that common block are identical.

If a common-block object is defined and referenced in a single subprogram only, the object could be replaced by a local variable, or record.

Because the amount of information can be huge if you have many common blocks with many objects, FORCHECK's internal tables can easily become full. In that case you have to split up the process in several runs in which you request the cross references of the objects of a limited number of common blocks at a time. The optimal procedure is to compose a FORCHECK library file first and to analyze this library file repeatedly.

5.4.7   Cross references of public module derived types

Cross references of public module derived types are displayed if a listing file has been requested and the /SMT is in effect.

All public derived types of each module for which a cross-reference table is requested are listed with all subprograms in which the derived type is used. If a derived type is used in one or more module procedures of the module in which the derived type is used, the module name is listed instead of the these individual module procedures.

Because the amount of information can be huge if you have many modules with many public derived types, FORCHECK's internal tables can easily become full. In that case you have to split up the process in several runs in which you request the cross references of the derived types of a limited number of modules at a time. The optimal procedure is to compose a FORCHECK library file first and to analyze this library file repeatedly.

5.4.8   Cross references of public module data

Cross references of public module data are displayed if a listing file has been requested and the /SMV is in effect.

All public constants and variables of each module for which a cross-reference table is requested are listed with all subprograms in which the module constant or variable is used. If a module constant or variable is used in one or more module procedures of the module in which the constant or variable is specified, the module name is listed instead of the these individual module procedures.

A "#" in front of a subprogram name indicates that the variable is (re)defined directly in that subprogram. Mind that if a variable is used as an actual argument in a subprogram, the variable may be (re)defined indirectly.

Because the amount of information can be huge if you have many modules with many public variables, FORCHECK's internal tables can easily become full. In that case you have to split up the process in several runs in which you request the cross references of the variables of a limited number of modules at a time. The optimal procedure is to compose a FORCHECK library file first and to analyze this library file repeatedly.

5.5 Specification of procedure interfaces

You can make the analysis more complete by defining the interface for all procedures which have not been included in the analysis, such as system procedures and third party procedure packages. There are two ways to specify procedure interfaces, namely applying the traditional FORTRAN 77 syntax or using the Fortran 90/95 syntax features.

5.5.1 Using FORTRAN 77 syntax

You can use FORTRAN 77 syntax to specify a procedure interface by constructing a template for the procedure. Just specify the appropriate procedure statement (`FUNCTION` or `SUBROUTINE`) with the dummy argument list, a type specification statement for the result in case of a `FUNCTION` procedure and a type specification for each of the dummy arguments. If an argument is an input argument, reference it, if it is an output argument provide an assignment statement to define it, and if it is an input/output argument reference it first and define it later on. Conclude the template procedure with an `END` statement. For example:

```
FUNCTION MYFUN(ARG1)
REAL MYFUN, ARG1
MYFUN=ARG1
END
```

Include the templates in the FORCHECK analysis by specifying them as an input source file or place them in a FORCHECK library file.

5.5.2 Using Fortran 90 syntax

Fortran 90 and up provide the appropriate syntax to specify a procedure interface. You create a module and define an interface block. In this interface block you create one or more interface bodies to define the interfaces of procedures. Each interface body should consist of the appropriate procedure statement (`FUNCTION` or `SUBROUTINE`) with the dummy argument list, a type specification statement for the result in case of a `FUNCTION` procedure and a type specification for each of the dummy arguments. If an argument is an input argument, supply the `INTENT(IN)` attribute, if it is an output argument supply the `INTENT(OUT)` attribute, and if it is an input/output argument supply the `INTENT(INOUT)` attribute, which is the default. For optional arguments specify the `OPTIONAL` attribute. Conclude the interface body with an `END FUNCTION` or `END SUBROUTINE` statement. For example:

```
MODULE PLOTLIB
  INTERFACE
    FUNCTION MYFUN(ARG1, ARG2)
      REAL MYFUN
      REAL, INTENT(IN) :: ARG1
      REAL, INTENT(IN), OPTIONAL :: ARG2
    END FUNCTION MYFUN
  END INTERFACE
END MODULE PLOTLIB
```

Include this module in the FORCHECK analysis by specifying it as an input source file or place it in a FORCHECK library file.

When using Fortran 90 or up you include the procedure interface in the program-unit analysis by referring the module which defines the interface. You do this with the USE statement, for example:

```
USE PLOTLIB
```

Even if you are still restricted to use FORTRAN 77 you can apply the Fortran 90 way for the FORCHECK analysis! Just enable Fortran 90 or up syntax in the FORCHECK configuration file to analyze the interface modules and enable extension 217, modules, for the analysis of the other program units. Place the USE statement in an INCLUDE file which you conditionally use for the FORCHECK analysis. For compilation you replace this INCLUDE file by one with an EXTERNAL statement specifying the procedure.

You can use the supplied utility INTERF to generate a module with interface bodies from a FORCHECK library file. See the chapter "Operation".

5.5.3   Using FORCHECK attributes

To define the interface for C, or system procedures, FORCHECK has the possibility to specify additional attributes for the procedure and dummy arguments. For the global program analysis they can be specified in an external template procedure. For the program-unit analysis you can specify procedure attributes in an EXTERNAL statement which could be placed in an INCLUDE file which you conditionally use for the FORCHECK analysis. For both the program-unit analysis and the global program analysis you can specify the attributes in an interface body in a module.

These attributes have the form [attribute-list] in which attribute-list is a comma separated list of attributes. You have to enable the [] type attribute extension, nr 69, in your configuration file to use this facility.

The following attributes can be specified for dummy arguments:

· OMITTABLE
   By specifying the OMITTABLE attribute for a dummy argument of a procedure template you can tell FORCHECK to allow the actual argument to be left empty.

· PLURI
   By specifying the PLURI attribute for a dummy argument of a procedure template you can tell FORCHECK not to verify the argument.

· %VAL
   By specifying the %VAL attribute for a dummy argument you specify that actual arguments have to be passed by value using the %VAL built-in function (VMS). An example of the specification of the %VAL attribute is:
   SUBROUTINE SUB(ARG1[%VAL]).

The following attributes can be specified for external procedure names:

· INQUIRY
   By specifying the INQUIRY attribute for a procedure template FORCHECK can indicate

that the arguments do not have to be defined or associated. For example:
```
REAL FUNCTION FUN[INQUIRY](Arg1)
```
And within a subprogram: EXTERNAL FUN[INQUIRY].

· PLURI
By specifying the PLURI attribute for a procedure interface you can tell FORCHECK not to verify the number of arguments and the argument lists, for example:
```
REAL FUNCTION FUN[PLURI](Arg1,Arg2)
```
And within a subprogram: EXTERNAL FUN[PLURI].

· SUBREF
By specifying the SUBREF attribute for a procedure template you can allow a CALL to a function procedure, for example:
```
REAL FUNCTION FUN[SUBREF](Arg1,Arg2).
```
And within a subprogram: EXTERNAL FUN[SUBREF].

· VARYING
By specifying the VARYING attribute for a procedure template FORCHECK can allow a varying number of arguments. For example:
```
REAL FUNCTION FUN[VARYING](Arg1,Arg2)
```
And within a subprogram: EXTERNAL FUN[VARYING].

5.6 Metrics

If a listing file has been requested a table will be presented with some metrics of each program unit and procedure. This table shows the number of (non-comment) source lines, (non-blank) comment lines, statements and maximum construct nesting. The number of source lines, comment lines, and statements are split up into a total as read in, and the number not read from include files.

In the totals the lines and statements of the include files are counted only once for each include file.

The program metrics shows the number of program units, (sub)modules, subprograms, module procedures, internal procedures and source files analyzed.

5.7 Final report

After completing the analysis, a final report will be presented with a message summary.

The message summary lists all messages presented and the number of messages in each category. It will be stored in the report file and, when the listing device is not your screen, it will both be included in the listing file and presented on your screen.

If the /LG option has been enabled, the usage of FORCHECK's internal tables will also be displayed.

Appendix A

Supported Fortran syntax

Forcheck supports the full Fortran 2008 syntax, which includes Fortran 2003, Fortran 95, Fortran 90 and Fortran 77. Moreover FORCHECK supports many of the language extensions of various compilers. Not all the vendor specific Fortran language extensions which FORCHECK can support for a compiler being emulated are enabled by default. The reason is that some of the language extensions are only provided to be compatible with earlier versions of that compiler or now have standard Fortran equivalents which you can use preferably. Moreover some of the extensions make a program less secure, for example less strict type checking, so enabling these extensions will weaken the possibilities of FORCHECK to detect programming flaws. FORCHECK has, by default, enabled only those Fortran language extensions which:

- · Are generally accepted and have no standard Fortran equivalent, or are present in a more recent Fortran standard,

- · Impose no risk and can be easily converted to standard Fortran,

- · Improve the readability or the maintainability.

In the table in Section A.3 on page 71–79 the language extensions, relative to Fortran 77, which are supported by FORCHECK are listed. In the table in Section A.4 on page 81–91 the language extensions, relative to Fortran 90 and Fortran 95, which are supported by FORCHECK are listed.

In the tables you can see which extensions are supported by FORCHECK and the various compilers. A "+" denotes an extension which is by default enabled by FORCHECK if the compiler emulation concerned has been chosen. A "o" denotes an extension which is by default not enabled. A "@" means the support of that particular extension is explained in the text.

You can enable or disable each of the listed extensions by editing the appropriate configuration file. For Fortran 90, Fortran 95, Fortran 2003, or Fortran 2008 compilers you can use the respective default configuration file as a template. See the section "Changing the configuration file".

A.1   Compilers supported

Configuration files for the following Fortran 77 compilers are supplied. In the first column the filename of the configuration file is listed (without the filename extension). The second column

presents the mnemonic used in the table of Fortran extensions.

Configuration file	Mnemonic	Compiler name
`absoftf77.cnf`	AB	Absoft FORTRAN 77 V4.3
`cyber.cnf`	CBR	Control Data Cyber NOS/VE Fortran Version 1, level 1.6, PRS level 700
`cd4000.cnf`	CD4	Control Data 4000 Fortran
`convex.cnf`	CVX	Convex Fortran, Version 6.0
`crayf77.cnf`	CF77	Cray Fortran 77, V4
`decvms.cnf`	DAV	DEC Equipment FORTRAN for Open VMS Alpha
`decux.cnf`	DEC	Digital Equipment FORTRAN for Ultrix and DIGITAL UNIX
`domain.cnf`		Apollo/Domain Fortran, SR 10
`vax.cnf`	VAX	Digital Equipment VAX Fortran, Version 5.0 and
		VAX Fortran-HPO, Version 1.0
`f2c.cnf`	F2C	F2c Fortran 77
`g77.cnf`	F77	GNU Fortran 77
`hp77.cnf`		HP Fortran 77 for series 800
`hp9000.cnf`	HP9	HP-UX FORTRAN/9000 for series 300/400/700 and 800
`ibmvs2.cnf`	VS2	IBM VS Fortran, Version 2, Release 2.5
`laheyf77.cnf`	LH	Lahey F77L, V5.00 and F77L-EM32 V5.00
`msf5.cnf`	MS5	Microsoft Fortran, V5.1 and Microsoft Fortran PowerStation, V 1.0
`ndp.cnf`	NDP	NDP Fortran, Release 2.0
`pdp11.cnf`	PDP	DEC Equipment PDP-11 Fortran-77, Version 5.0
`prime.cnf`	PR	Prime Fortran-77, T1.0-21.0
`prospero.cnf`	PF	Prospero Fortran, V2.12
`rm.cnf`	RM	Ryan-McFarland RM/Fortran, V1.00 and IBM Professional Fortran, V1.23
`rm2.cnf`	RM2	Ryan-McFarland RM/Fortran, V2.40
`sgif77.cnf`	SGI	Silicon Graphics MIPSpro Fortran 77, Version 3.4.1
`sunf77.cnf`	SUN	Sun Fortran 77
`ftn77.cnf`	FTN	Salford FTN77, V3.62
`unisys.cnf`	UNI	Unisys 1100 Fortran-77, L10
`watcom.cnf`	WAT	WATCOM Fortran 77 V11.0

Not all of the compilers are listed in the table. The DEC FORTRAN for AXP/VMS (DAV) extensions are equivalent to those of DEC; only the default file name extensions differ. For the Digital Research compiler a configuration file with the supported types is supplied and the `%INCLUDE` directive is supported. When you want FORCHECK to accept the Digital Research compiler extensions you have to adapt the configuration file.

Configuration files for the following Fortran 90, Fortran 95, Fortran-2003 and Fortran 2008 compilers are supplied:

Configuration file	Mnemonic	Compiler name
absoftf95.cnf	AB95	Absoft FORTRAN 95 V6
crayf90.cnf	CF90	Cray Fortran 90, V2
crayf03.cnf	Cray	Cray Fortran, V7
cvf.cnf	CVF	Compaq Visual Fortran V6.6
decf90.cnf		DEC Fortran 90
decf95.cnf	DEC	DEC Fortran 95
fujitsu.cnf	FUJ	Fujitsu Fortran 90
gfortran.cnf	gfort	GNU Fortran 95
g95.cnf	g95	Open source Fortran 95 based on GNU
hpf95.cnf	HP95	Fortran for HP-UX
intel7.cnf		Intel Visual Fortran V7.0
intel9.cnf		Intel Visual Fortran V9.0
intel0.cnf		Intel Visual Fortran V10.0
intel11.cnf		Intel Visual Fortran V11.0
intel12.cnf		Intel Visual Fortran V12.0
intel13.cnf	INT	Intel Visual Fortran V13.0
ibmxlf.cnf	XLF	IBM AIX XL Fortran
laheyf90.cnf	LF90	Lahey Fortran 90
laheyf95.cnf	LF95	Lahey Fortran 95
msfps.cnf	MSF	Microsoft Fortran PowerStation V4.0
nagf90.cnf		NagWare f90 Compiler
nagfor.cnf	NAG	NagWare f95 Compiler
nasf95.cnf		NASoftware Fortran Plus Compiler
pgif90.cnf		The Portland Group Fortran 90 Compiler
pgif95.cnf	PGI	The Portland Group Fortran 95 Compiler
pathscale.cnf	PATH	PathScale EKOPath Compiler
pgif03.cnf		The Portland Group Fortran 2003 Compiler
ftn90.cnf		Salford FTN90
ftn95.cnf	FTN	Silverfrost FTN95
sgif90.cnf		Silicon Graphics MIPSpro Fortran 90, Version 7.3
sgif95.cnf	SG95	Silicon Graphics MIPSpro Fortran 95
sunf90.cnf	SF90	Sun Fortran 90
sunf95.cnf	SF95	Sun Fortran 95

The Fortran 90/95 extensions marked in the column F2003 of the table are included in the Fortran 2003 standard. The Fortran 90/95 extensions marked in the column F2008 of the table are included in the Fortran 2008 standard.

A.2   General language extensions supported

· Tab formatting is supported when fixed form source is enabled. If the first column of a fixed form input record consists of a tab succeeded by a digit as continuation character, then the continuation character will be located at column 6 and the next characters from column 7 on. If this tab is not followed by a digit the next characters are placed from

column 7 on. Subsequent tabs, or tabs in columns past the continuation field are expanded to blanks to columns 9, 17, 25, etc. before processing the statement.

This is different from the way some compilers will treat tabs. Some compilers consider tabs after column 6 as one blank character or discard tabs at these positions. Because of this difference FORCHECK may locate characters past column 72, discarding them, while the compiler will not.

This way has been chosen because an expansion of tabs will generally be used when source code is transformed to standard Fortran 77, or when sending your program to a different computer system. Moreover the compiler will probably expand tabs in the source listing. In the FORCHECK way you can see which characters will be interpreted by any compiler and which may not.

· Though some compilers accept longer source records (e.g. in free form), the maximum record size FORCHECK can read is 512 characters, after expansion of tabs and of cpp macros.

· Though some compilers support an unlimited number of continuation lines FORCHECK can read up to 999 continuation lines.

· `LOGICAL*1` data are treated as logicals. `BYTE` data as integers.

· The nonstandard form of the `PARAMETER` statement (without parentheses) is not equivalent to the standard Fortran `PARAMETER` statement. In the nonstandard form the type of the named constant takes the type of the literal constant, which may be different from that of the implicit or specified type of the name using the Fortran 77 syntax.

· Though a specific compiler may support longer names, FORCHECK supports names of up to 64 characters only.

· Some compilers support directives which are identified by a key in the first columns followed by a keyword. These compiler directive strings can be specified in the configuration file. Some of these directives will not only be accepted, but also interpreted by FORCHECK: see the notes on each specific compiler emulation.

· Some compilers support directives using keywords in column 7-72. Detection of these keywords can be enabled if the keyword is present in the tables of Fortran language extensions.

· FORCHECK can handle cpp preprocessor directives. cpp preprocessing is enabled by enabling extension 7 in the configuration file. You can also enable or disable cpp preprocessing using the enable cpp command line option or by setting this option in the IDE. Parameterized macro expansion is supported with some limitations. The macro must be on a single line and variadic macros are not supported. Macro expansion must be used with great care because it can cause significant characters be placed beyond character position 72 in fixed source format and change character constants. If a file includes another file with the Fortran INCLUDE statement, the included file is not preprocessed. Files included using the cpp directive #include are preprocessed.

The usage of language extensions will be flagged when the `/ST`, the `/F77`, the `/F90`, the `/F95`, the `/F03`, or the `/F08` option has been specified. By specifying the `/OB` option all language features which are marked as obsolescent in the Fortran standard which is in effect will be flagged.

A.3 Table with Fortran 77 language extensions

no.		PDP	VAX	VS2	UNI	CBR	PR	CF77	CVX
	maxima in lay-out:								
	max. number of characters per line	88	132	80	80	80	80	96	
	max. number of continuation lines	99	@	99	@	19	@	99	19
	max. length of names	6	31	31	6	7	32	31	@
	max. length of subprogram names	6	31	7	6	7	32	31	@
	max. length of common-block names	6	31	7	6	7	32	31	@
	type length modifiers:								
	`INTEGER *1`								+
	`INTEGER *2`	+	+	+		+	+	@	+
	`INTEGER *4`	+	+	+	+	+	+	@	+
	`INTEGER *8`					+		@	+
	`REAL *4`	+	+	+	+		+	@	+
	`REAL *8`	+	+	+	+	+	+	@	+
	`REAL *16`		+	+		+	+	@	
	`COMPLEX *8`	+	+	+	+		+	@	+
	`COMPLEX *16`		+	+	+	+	+	@	+
	`COMPLEX *32`			+				@	
	`LOGICAL *1`	+	+	+			+		+
	`LOGICAL *2`	+	+				+	@	+
	`LOGICAL *4`	+	+	+	+		+	@	+
	`LOGICAL *8`							@	+
	maximum length of type `CHARACTER`**:**								
	`CHARACTER *255`	+							
	`CHARACTER *511`			@	+				
	`CHARACTER *16384`							+	
	`CHARACTER *32767`			@			+		
	`CHARACTER *65280`								
	`CHARACTER *65535`		+						+
	`CHARACTER *2147483647`					+			
	default source file name extension	FTN	FOR				F77	f	f
	default include file name extension	FTN	FOR						
	include list option delimiter	/	/						
	compiler directive string			@	#		$		#
	free form continuation character			–					
	free form 1st column comment char.			"					
	lay-out:								
1	lower case characters	+	+	+	+	+	+	+	+
2	debug lines (D)	+	+						+
3	debug lines (A-Z)								
4	tabs	+	+					+	+
5	formfeeds	+	+						+
6	in-line comment after !	+	+	+		+		+	+
7	cpp preprocessor directives								
8	in-line comment after @				+				
10	statement separator ;								+
11	any character allowed as continuation character		+						
	names:								
13	names with $		+	+	+		+	+	+
14	names with _		+	+			+	+	+
15	names beginning with $			+					
16	built-in functions beginning with %		+						+
17	names with @								
18	names beginning with _								

no.	SGI	SUN	HP9	DEC	CD4	RM	RM2	MS5	LH	PF	NDP	FTN	WAT	AB	F2C
	132	132	80	132	132	80	80	80	80	80	@	80		132	@
	99	@	99	19	19	@	@	@	@	19	99	19	61	99	@
	32	32	@	31	32	31	31	31	31	31	31	32	32	31	@
	32	32	@	31	32	8	8	31	31	31	31	32	32	31	@
	32	32	@	31	32	8	8	31	31	31	31	32	32	31	@
	+			+	+				+	+	+	+	+	+	+
	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
															+
	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
	@	+	+		@										
	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
	+	+	+	+	+	+	+	+	+		+	+	+	+	+
		+													
	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
	+	+	+	+	+			+			+	+	+	+	+
	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
						+	+								
		+	+					+	+	+			+		
	+			+	+						+		+		
														+	+
	f	f	f	f	f	FOR	FOR	FOR	FOR FOR	FOR	f	FOR	FOR FOR		f
	$	#	$		$			$	%					$	$
								–	*					&	&
								"						!	!
1	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
2	+		+	+	+		+				+		+	+	
3								+							
4	+	+	+	+	+		+	+	+	+	+	+	+	+	+
5	+	+	+	+	+			+	+		+		+	+	+
6	+	+	+	+	+			+	+	+	+	+	+	+	+
7															
8						+									
10														+	
11				+											
13	+	+	+	+	+	+	+	+	+		+	+	+	+	
14	+	+	+	+	+			+	+	+	+	+	+	+	+
15								+	+		+		+		
16	+		+	+	+									+	
17												+			
18	+				+								+		+

no.		PDP	VAX	VS2	UNI	CBR	PR	CF77	CVX
	constants:								
20	character constants between ""					+		+	
21	REAL*16 with Q-exponent		+	+			+		
22	named constants in complex constants	+	+			+			+
23	Hollerith	+	+	+		+	+	+	+
24	B'xxx', B"xxx" binary								
25	O'xxx', O"xxx" octal					+	+	+	
26	X'xxx', X"xxx" hexadecimal							+	
27	Z'xxx', Z"xxx" hexadecimal					+	+	+	
28	'xxx'B, "xxx"B binary								
29	'xxx'O, "xxx"O octal	+	+						+
30	'xxx'X, "xxx"X hexadecimal	+	+						+
31	'xxx'Z, "xxx"Z hexadecimal								
32	Oxxx octal	o	o		+				
33	Zxxx hexadecimal	o	o	+					
34	xxxB octal							+	
35	[-]:xxx hexadecimal						+		
36	"xxx octal	+	+						+
37	$ xxx hexadecimal								
38	[radix]#value								
39	radix 50	+	+						
40	C-string: 'xxx'C*								
41	Length modifier suffix (B,S,L)								
42	C-string: \ editing*								
	specification statements:								
43	ALLOCATABLE								
44	STATIC								
45	[DE]ALLOCATE, deferred dim.spec.								
46	AUTOMATIC								
47	BOOLEAN					+			
48	BYTE	+	+			+			
49	C EXTERNAL								
50	DOUBLE COMPLEX								
51	IMPLICIT NONE		+	+		+		+	+
52	IMPLICIT UNDEFINED								
53	IMPLICIT AUTOMATIC/STATIC								
54	OPTIONAL, INTENT								
55	integer (Cray) POINTER							+	
56	LC, BC, HC, MS, MSC EXTERNAL								
57	NAMELIST		+	+		+	+	+	+
58	F90 extended NAMELIST features								
59	STRUCTURE, RECORD		+						+
60	F90 derived type								
61	VIRTUAL	+	o						
62	VOLATILE		+						
63	F90 POINTER, TARGET								
64	DEFINE				+				
65	automatic arrays							+	
66	DLL_IMPORT, DLL_EXPORT								
67	C_EXTERNAL, Salford STDCALL								
68	specif. functions in specif. expressions								
69	[..] type attributes								
70	/../ init. of var. in type spec.stmnt.		+	+	+		+		+
107	F90 init. of var. in type spec.stmnt.								
71	length modifier after dimension								
72	PARAMETER symbol=constant	+	+		+		+		+

*If extensions 40 and 42 are both enabled backslash editing is only applied for 'xxx'C-strings.

no.	SGI	SUN	HP9	DEC	CD4	RM	RM2	MS5	LH	PF	NDP	FTN	WAT	AB	F2C
20	+	+	+	+	+			+	+		+			+	+
21	+														
22	+		+	+											+
23	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
24	+	+		+	+						+	+		+	+
25	+	+	+	+	+						+	+		+	+
26	+	+		+	+										+
27	+	+	+	+	+		+		+		+	+		+	+
28	+			+	+										+
29	+		+	+	+								+	+	+
30	+		+	+	+								+	+	+
31	+			+	+										+
32						+									
33													+		
34			+												
35															
36															
37	+				+					+					
38								+							+
39											+				
40								+					+		
41												+			
42	+	@		+	+			+			@				+
43			+						+						
44	+	+	+	+	+						+				+
45			+					+	+				+		
46	+	+	+	+	+			+			+				+
47															
48	+	+	+	+	+			+			+				+
49							+				+				
50	+	+	+	+	+			+	+		+		+	+	+
51	+	+	+	+	+		+	+	+	+	+	+	+	+	+
52	+	+			+						+				+
53	+	+			+						+				
54															
55	+	+		+	+									+	
56									+						
57	+	+	+	+	+	+	+	+	+		+	+	+	+	+
58															
59	+	+	+	+	+			+			+		+	+	
60															
61	o		o	o	o						o			o	
62	+		+	+	+						+			+	
63															
64															
65			+												
66															
67												+			
68															
69								+							
70	+	+	+	+	+	+	+	+	+			+	+	+	
107															
71									o						
72	+		+	+	+	+						+			

no.		PDP	VAX	VS2	UNI	CBR	PR	CF77	CVX
73	/../ initialization of structure components		+						+
74	intrinsic functions in PARAMETER		+		+	+			
75	intrinsic functions in dimension spec.					+		+	
76	DATA statements mixed with spec. stmnts.	o	o	o				o	o
77	IMPLICIT mixed with specification stmnts.			o					
78	F90 KIND and Character selectors								
79	F90 attributes and entity oriented decl.								
80	F90 specification expressions								
81	Record fields and records in DATA								
82	Subobject of constant in DATA								
83	intrinsic functions in DATA								
84	pointers can be initialized in a DATA stmnt								
	subprograms:								
215	INTERFACE TO								
216	RECURSIVE							+	
217	MODULE								
220	argument list in PROGRAM statement							+	
221	[type[*len]] FUNCTION name	+	+				+		+
222	[type] FUNCTION name [*len]	o	o	+	+	+			o
223	F90 interface block								
224	F90 internal subprograms								
225	Unisys internal subprograms				+				
226	array valued functions								
227	END INTERFACE name								
228	STDCALL*								
229	recursive subprograms							o	
	commons:								
85	initialization of blank COMMON	o	o			o	o	o	o
86	differing lengths for a named COMMON	o	o		o		o	o	
87	initialization of COMMON not in BLOCK DATA	o	o			o	o	o	o
88	mixing of numeric and character in COMMON		o	o		o	o	o	
	executable statements:								
93	WHERE								
94	FORALL								
95	EXIT, CYCLE								
96	DO [label] [WHILE] .. ENDDO		+	+			+	@	+
97	SELECT CASE								
98	debug packet statements			+	+				
100	named constructs								
101	Watcom constructs								
102	REMOTE BLOCK, EXECUTE								
	general syntax:								
109	[...] array constructor								
110	.XOR. exclusive or as .NEQV.	o	o			o	o	o	o
111	alternate relational operators <, ==, etc.			+					
112	alternate return label &label		o						o
113	alternate return label $label						o		
114	RETURN in main as STOP			o					
115	null-arguments	+	+						

*When extension 67 is enabled, the Salford variant of STDCALL is accepted.

no.	SGI	SUN	HP9	DEC	CD4	RM	RM2	MS5	LH	PF	NDP	FTN	WAT	AB	F2C
73	+	+		+	+						+				
74	+	+	+	+	+	+									+
75															
76	o		o	o	o	o									
77															
78	+														
79															
80															
81													+		
82															
83															
84															
215								+							
216									+						
217															
220			+												@
221	+	+	+	+			+	+	+	+	+	+	+	+	+
222	o	o	o	o		+							o	o	
223															
224															
225															
226															
227															
228															
229	+	+	+	+	+										
85			o	o		o	o								o
86	o			o	o	o	o								
87		o	o	o		o							o	o	o
88	o	o	o	o	o	o	o	o		o	o	o	o	o	o
93															
94															
95			+					+	+				+	+	
96	+	+	+	+	+			+	+		+	+	+	+	+
97			+					+	+				+	+	
98						+									
100			+						+						
101													+		
102													+		
109															
110	o		o	o	o			o		o	o		o	o	
111							+		+					+	+
112	o	o	o	o	o	o									
113						o									
114			o			o	o	o							
115		+	+	+											

no.		PDP	VAX	VS2	UNI	CBR	PR	CF77	CVX
116	array expressions, but no dummy or alloc.								
117	F90 array expressions and sections							+	
118	constant arrays,constructors and substr.								
119	END program unit [name]								
120	keyword actual arguments								
121	zero sized data objects								
	type checking:								
125	mixing of DP and COMPLEX in expressions	+	+	+	+	+		+	+
126	string argument compatible with Hollerith	o	o		o	o			
127	strings can be assigned to INT/REAL/LOG	o	o						
128	strings can be ass. to BYTE and LOGICAL*1	o	o						
129	boz constants can be used in expressions	+	+			+	+	+	
130	boz constants in PARAMETER statement	+	+						
131	equivalence of numeric and character	o	o	o		o	o	o	
132	real array indices and substring expressions	o	o	o		o			
133	i and l const. comp. with shorter dummy	+	+						
	I/O statements:								
140	ACCEPT, TYPE	+	+						+
141	INPUT								
142	ENCODE, DECODE	o	o					o	o
143	FIND, DEFINE FILE	o	o						
144	direct access (lun'record)	o	o				o		
145	READ, PRINT, INPUT without format								
146	READ(KEY=) REWRITE, DELETE		+	+					
147	LOCKING								
148	UNLOCK		+						
	I/O:								
155	NUM= in READ			+					
156	list directed on internal file	+	+	+					
157	F90 nonadvancing I/O								
158	Formatted derived type I/O								
	OPEN/CLOSE/INQUIRE **specifiers:**								
165	RECL= for sequential files	+	+		+	+	+		+
166	RECL= not required if STATUS='OLD	+	+		+				
	format specifiers and edit descriptors:								
175	noncharacter array name allowed	o	o	o	o	o			o
176	variable length fields <...>	+	+						
177	aEw.dDe double precision exponent			+					
178	aQw[.d] quadruple precision mantissa			+			+		
179	aOw[.m] octal edit descriptor	+	+			+	+	+	+
180	aZw hex edit descriptor			+					
181	aZw[.m] hexadecimal edit descriptor	+	+		+	+	+	+	+
182	aR[w] char edit descriptor							o	
183	\ edit descriptor								
184	Q edit descriptor	+	+						+
185	$ edit descriptor	+	+					+	+
186	aBw[.m] binary edit descriptor							+	
189	Zero field width in edit descriptor								
	compiler directives:								
200	INCLUDE	+	+	+	+		+	+	+
201	OPTIONS		+						+
203	OPTION [N]BREAK								
204	EJECT		+			+			
205	[NO]LIST						+		
206	COMPILER(...					+			

no.	SGI	SUN	HP9	DEC	CD4	RM	RM2	MS5	LH	PF	NDP	FTN	WAT	AB	F2C
116								+							
117			+												
118															
119															
120															
121															
125	+		+	+		+	+	+			+	+	+	+	+
126			o	o			o				o	o		o	o
127	o		o	o	o							o		o	o
128	o		o	o	o									o	
129	+		+	+	+	+	+					+			+
130														+	+
131		o	o	o		o	o	o		o	o	o	o	o	o
132	o		o	o	o	o		o							
133				+											o
140	+	+	+	+	+						+			+	
141									+						
142	o	o	o	o	o	o					o			o	
143	o			o	o	o									
144		o				o									
145									o						
146	+			+	+										
147								+							
148	+			+	+										
155															
156	+	+	+	+	+						+	+	+		+
157															
158															
165				+			+		+			+	+		
166							+					+			
175		o	o	o		o	o	o				o	o		
176	+		+	+	+							+			
177						+							+		
178															
179	+	+	+	+	+	+			+		+	+		+	+
180								+	+		+		+		
181	+	+	+	+	+	+	+			+		+		+	+
182			+												
183							+	+	+	+		+	+	+	
184	+	+	+	+	+				+		+			+	
185	+	+	+	+	+			+	+		+	+	+	+	+
186					+							+		+	
189															
200	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
201	+	+		+	+						+	+			
203									+						
204						+									
205															
206															

A.4 Table with Fortran 90/95/2003/2008 language extensions

no.		F2003	F2008	Cray	NAG	XLF	DEC	FTN95	LF95	MSF
	maxima in lay-out:									
	max. number of characters per line	132	132	132	132	132	132	132	132	132
	max. number of cont. lines (fixed)	255	255	@	255	255	99	19	19	99
	max. number of cont. lines (free)	255	255	@	255	255	99	39	39	99
	max. length of names	63	63	63		250	31	63	240	31
	max. length of subprogram names	63	63	63		250	31	63	240	31
	max. length of common-block names	63	63	63		250	31	63	240	31
	type length modifiers:									
	INTEGER *1			+	+	+	+	+	+	+
	INTEGER *2			+	+	+	+	+	+	+
	INTEGER *4			+	+	+	+	+	+	+
	INTEGER *8			+	+	+	+	+	+	
	REAL *4			+	+	+	+	+	+	+
	REAL *8			+	+	+	+	+	+	+
	REAL *10							+		
	REAL *16			+	+	+	+		+	+
	COMPLEX *8			+	+	+	+	+	+	+
	COMPLEX *16			+	+	+	+	+	+	+
	COMPLEX *20							+		
	COMPLEX *32			+	+	+	+		+	
	LOGICAL *1			+	+	+	+	+	+	+
	LOGICAL *2			+	+	+	+	+	+	+
	LOGICAL *4			+	+	+	+	+	+	+
	LOGICAL *8			+	+	+	+		+	
	maximum length of type CHARACTER:									
	CHARACTER *255									
	CHARACTER *511					@				
	CHARACTER *16384									
	CHARACTER *32767				+	@		+		+
	CHARACTER *65280						+			
	CHARACTER *65535									
	CHARACTER *2147483647			+					+	
	default source file name extension			f		f				FOR
	default include file name extension									
	include list option delimiter									
	compiler directive string									$
	free form continuation character									
	free form 1st column comment char.									
	lay-out:									
2	debug lines (D)						+	+		
3	debug lines (A-Z)									+
4	tabs			+	+	+	+	+	+	+
5	formfeeds							+	+	+
7	cpp preprocessor directives						+		+	
8	in-line comment after @									
9	in-line comment {...}									
11	any character allowed as continuation character							+		
12	line may start with ;		+							
	names:								+	
13	names with $			+		+	+		+	+
15	names beginning with $					+				+
16	built-in functions beginning with %			+		+	+			+
17	names with @							+	+	
18	names beginning with _					+	+			

no.	FUJ	SG95	SF95	HP95	INT	CVF	AB95	gfort	g95	PATH	PGI
	255	132	132	254	132	132	132	132	132	132	132
	@	99	99	255	511	511	99	@	@	255	@
	@	99	99	255	511	511	99	@	@	255	@
	31	32	31	255	255	63	31			63	31
	31	32	31	255	255	63	31			63	31
	31	32	31	255	255	63	31			63	31
	+	+	+	+	+	+	+	+	+	+	+
	+	+	+	+	+	+	+	+	+	+	+
	+	+	+	+	+	+	+	+	+	+	+
	+	+		+	+	+		+	+	+	+
	+	+	+	+	+	+	+	+	+	+	+
	+	+	+	+	+	+	+	+	+	+	+
			+		+			+	+	+	
	+	+		+	+						
	+	+	+	+	+	+	+	+	+	+	+
	+	+	+	+	+	+	+	+	+	+	+
			+		+			+	+	+	
	+	+			+						
	+	+	+	+	+	+	+	+	+	+	+
	+	+	+	+	+	+	+	+	+	+	+
	+	+	+	+	+	+	+	+	+	+	+
	+	+		+	+	+			+	+	+
	+		+	+	+	+					
		+									
							+	+	+	+	+
	f	f	f	f				f	f	f	f
				$	!DIR$	$	$	#	#	#	#
	-										
	"										
2				+	+	+	+		+		+
3											
4	+	+	+	+	+	+	+	+	+	+	+
5		+	+	+	+	+	+		+	+	+
7					+			+	+	+	+
8											
9											
11					+	+					
12								+		+	
13	+			+	+	+	+	+	+		+
15	+				+	+	+				
16		+		+	+	+	+		+	+	+
17											
18											

no.		F2003	F2008	Cray	NAG	XLF	DEC	FTN95	LF95	MSF
	constants:									
21	REAL*16 with Q-exponent					+				
22	named constants in complex constants	+	+	+	+	+	+	+	+	
23	Hollerith			+		+	+			+
26	X'xxx', X"xxx" hex			+		+	+			
28	'xxx'B, "xxx"B binary					+	+			
29	'xxx'O, "xxx"O octal			+		+	+			
30	'xxx'X, "xxx"X hex			+		+	+			
31	'xxx'Z, "xxx"Z hex					+	+			
32	0xxx octal									
33	Zxxx hex					+				
34	xxxB octal			+						
35	[-]:xxx hex									
36	"xxx octal									
37	$ xxx hex									
38	[radix]#value									+
39	radix 50									
40	C-string: 'xxx'C*									+
41	Length modifier suffix (B,S,I)									
42	C-string: \ editing*					+				+
	specification statements:									
44	STATIC					+	+			+
46	AUTOMATIC					+	+			+
47	BOOLEAN									
48	BYTE					+	+		+	+
49	C EXTERNAL									
50	DOUBLE COMPLEX			+	+	+	+	+	+	+
52	IMPLICIT UNDEFINED					+				
53	IMPLICIT AUTOMATIC/STATIC					+				
55	integer (Cray) POINTER			+		+	+			+
56	LC, BC, HC, MS, MSC EXTERNAL									
59	STRUCTURE, RECORD						+			+
61	VIRTUAL					o	o			
62	VOLATILE	+	+	+	+	+	+		+	
64	DEFINE									
66	DLL_IMPORT, DLL_EXPORT								+	
67	C_EXTERNAL, Salford STDCALL							+		
68	specif.functions in specif.expressions	+	+							
69	[..] type attributes									
70	init. of var. in type spec.stmnt /../					+	+		+	+
71	length modifier after dimension									
72	PARAMETER symbol=constant						+			
73	/../ initialization of structure components						+			
108	F95 initialization of structure components	+	+			+	+	+	+	
77	IMPLICIT mixed with specification stmnts.									
81	Record fields and records in DATA									
82	Subobjects of constants in DATA	+	+			+				
83	intrinsic functions in DATA	+	+			+				
84	pointers can be initialized in a DATA stmnt.	+	+			+				
231	procedure pointers	+	+	+	+	+				
239	PROTECTED	+	+	+	+	+				
240	C-binding and enumerators	+	+	+	+	+				
241	VALUE	+	+	+	+	+				
242	VALUE for arrays		+							

*Extensions 40 and 42 are mutually dependent. If both are enabled backslash editing is only applied for 'xxx'C-string

no.	FUJ	SG95	SF95	HP95	INT	CVF	AB95	gfort	g95	PATH	PGI
21		+			+			+	+	+	
22				+	+			+	+	+	
23	+	+	+	+	+	+	+	+	+	+	+
26	+		+					+	+		
28					+			+			
29	+			+	+			+			+
30	+			+	+	+		+			+
31						+		+			
32											
33											
34			+								
35											
36											
37											
38					+	+					
39											
40				o	+	+	+				
41											
42	+			o	+	+	+	+	+		+
44				+	+	+	+				
46				+	+	+	+				
47											
48				+	+	+	+		+		+
49											
50				+	+	+	+	+	+		+
52											
53	+										
55		+	+	+	+	+	+	+		+	+
56	+										
59				+	+	+	+				+
61				o	o						
62				+	+	+	+	+	+	+	+
64											
66											
67											
68	+	+	+	+	+		+	+	+	+	+
69											
70				+	+	+	+	+			+
71	+										
72					+	+	+				
73				+	+	+					+
108		+	+		+		+	+	+		+
77	+										
81											
82	+	+	+	+	+		+	+	+	+	+
83	+	+	+	+	+		+	+	+	+	+
84	+	+	+	+	+		+	+	+	+	+
231				+	+			+	+		
239					+		+	+			
240				+	+			+	+	+	+
241					+			+	+	+	
242											

no.		F2003	F2008	Cray	NAG	XLF	DEC	FTN95	LF95	MSF
243	type parameter enquiry	+	+	+	+	+				
244	TS 29113, further interop of Fortran with C	+	+	+	+	+				
245	allocatable structure components (TR 15581)	+	+	+	+	+				
247	access spec. of components	+	+	+	+	+				
249	procedure components	+	+	+	+	+				
250	type bound procedures	+	+	+	+	+				
251	IMPORT statement	+	+	+	+	+				
252	pointer INTENT attribute	+	+	+	+	+				
257	renaming of operators in USE statement	+	+	+	+	+				
259	allocatable scalars	+	+	+	+	+				
260	deferred character length	+	+	+	+	+				
261	F2003 specification and initialization expressions	+	+	+	+	+				
262	PROCEDURE	+	+	+	+	+				
263	mixing of subroutines and functions in generic									
264	allocatable dummy arguments (TR 15581)	+	+	+	+	+				
265	CONTIGUOUS attribute		+							
266	implied-shape array		+							
267	initialization of pointer with target		+							
268	maximum rank 15		+							
269	:: after PROCEDURE allowed		+							
270	type extension	+	+	+	+	+				
271	parameterized derived type	+	+	+	+	+				
272	deferred binding and abstract type	+	+	+	+	+				
273	polymorphic entities, CLASS statement	+	+	+	+	+				
274	TYPE statement for intrinsic type		+							
275	empty type-bound-procedure-part		+							
276	list of type-bound-procedures		+							
277	omitting an all. component in a structure con-structor		+							
	program units, subprograms, interfaces:									
214	IMPURE		+							
215	INTERFACE TO									+
218	PURE	+	+							
219	ELEMENTAL	+	+							
220	argument list in PROGRAM statement									
221	[type[*len]] FUNCTION name			+		+	+			+
222	[type] FUNCTION name [*len]					o	o			
225	Unisys internal subprograms									
227	END INTERFACE name				+	+				
228	STDCALL									
229	recursive reference of all procedures allowed									
230	intrinsic modules: USE, [NON_]INTRINSIC ::	+	+	+	+	+				
232	SUBMODULE		+							
233	ABSTRACT INTERFACE	+	+	+	+	+				
234	Data in main or module are saved implicitly		+							
235	allocatable function result (TR 15581)	+	+	+	+	+				
236	defining interface of containing procedure	+	+	+	+	+				
237	empty contains section			+	+					
238	END statement for internal and module procedure		+							
	commons:									
85	initialization of blank COMMON						o			
86	differing lengths for a named COMMON	+	+	o		o	o			
87	initialization of COMMON not in BLOCK DATA			o			o			

no.	FUJ	SG95	SF95	HP95	INT	CVF	AB95	gfort	g95	PATH	PGI
243					+						
244					+			+			
245					+			+	+	+	+
247					+			+			
249					+			+			
250					+			+			
251				+	+			+	+	+	
252					+			+	+	+	
257					+			+	+		
259					+			+			
260					+			+			
261								+	+		
262					+			+			
263					+	+					
264					+			+	+		+
265								+			
266								+			
267								+			
268								+			
269					+			+			
270					+			+			
271											
272					+			+	+		
273					+			+			
274								+			
275											
276								+			
277											
214								+			
215						+					
218	+	+	+	+	+		+	+	+	+	+
219	+	+	+	+	+		+	+	+	+	+
220											
221	+	+		+	+	+	+	+	+		+
222				o							+
225											
227	+	+	+	+	+		+	+	+	+	+
228							+				
229											
230				+	+			+	+		+
232											
233					+			+			
234											
235				+	+			+	+		+
236					+	+		+	+		
236									+		
237								+			
238								+			
85											
86									o		
87				o							

APPENDIX A. SUPPORTED FORTRAN SYNTAX

no.		F2003	F2008	Cray	NAG	XLF	DEC	FTN95	LF95	MSF
	executable statements:									
94	FORALL				+	+	+	+	+	
98	debug packet statements									
99	SELECT TYPE construct	+	+		+	+				
101	Watcom constructs									
102	REMOTE BLOCK, EXECUTE									
105	ASSOCIATE	+	+	+	+	+				
106	ERRMSG= in (DE)ALLOCATE	+	+	+	+	+				
299	EXEC									
300	bounds/remapping in pointer assignment	+	+	+	+	+				
301	transfering an allocation; typed allocation	+	+	+	+	+				
302	SOURCE= specifier on ALLOCATE	+	+	+	+	+				
303	MOLD= on ALLOCATE		+							
304	copy bounds and values from SOURCE and MOLD		+							
305	DO [label][,] [CONCURRENT] .. ENDDO		+							
306	FORALL index kind specification		+							
307	BLOCK construct		+							
308	EXIT any construct		+							
	general syntax:									
309	STOP and ERROR STOP with constant expression		+							
310	F2003 array constructor enhancements	+	+	+	+	+				
311	co-array		+							
312	real and imag part-ref		+							
320	internal procedure as actual argument		+							
321	unallocated actual argument allowed for optional dummy		+	+						
322	target actual arg. assoc. with dummy pointer		+							
323	reference of pointer function		+							
109	F2003 array constructor syntax: [..]	+	+	+	+	+		+		
110	.XOR. exclusive or as .NEQV.			o		o	o			o
112	alternate return label &label						o			
113	alternate return label $label									
114	RETURN in main as STOP						o			o
115	null-arguments						+			
124	F2003 structure constructors: comp. keywords	+	+	+	+	+				
	type checking:									
126	string argument compatible with Hollerith			o			o			
127	strings can be assigned to INT/REAL/LOG			o			o			
128	strings can be ass. to BYTE and LOGICAL*1						o			
129	typeless (BOZ) can be used in expressions			+	+	+	+	+		
131	equivalence of numeric and character			o			o	o		o
132	real array indices and substring expressions						o	o		o
133	i and l const. comp. with shorter dummy						+			
134	passing character scalar actual to dummy array	+	+	+	+	+				
135	BOZ constants in CMPLX, DBLE, INT, REAL	+	+	+	+	+				
136	intrinsic assignment of characters of different kinds	+	+							

no.	FUJ	SG95	SF95	HP95	INT	CVF	AB95	gfort	g95	PATH	PGI
94	+	+	+	+	+	+	+	+	+	+	+
98											
99					+			+			
101											
102											
105					+			+			
106					+			+			
299											
300								+	+		
301					+			+			
302					+			+			
303								+			
304											
305								+			
306											
307								+			
308								+			
309								+			
310					+			+	+		
311								+			
312											
320					+			+			
321								+			
322											
323											
109				+	+			+	+	+	
110				o	o	o					
112				o							
113											
114											
115											
124					+			+	+		
126											
127											
128											
129				+	+	+		+	+		+
131				o	o	o					
132											
133											+
134											
135				+	+			+	+		
136								+			

no.		F2003	F2008	Cray	NAG	XLF	DEC	FTN95	LF95	MSF
	I/O statements:									
140	ACCEPT, TYPE statement						+			
141	INPUT statement									
142	ENCODE, DECODE statement			o			o			
143	FIND, DEFINE FILE statement						o			
144	direct access (lun'record)									
145	READ, PRINT, INPUT without format									
146	READ(KEY=) REWRITE, DELETE						+			
147	LOCKING statement									
148	UNLOCK statement						+			
149	FLUSH statement	+	+	+	+	+				
155	NUM= in READ					+				
156	list directed on internal file				+	+				
157	non-advancing i/o				+	+				
158	formatted derived type i/o				+	+				
159	asynchronous i/o	+	+	+	+	+				
160	stream access i/o	+	+	+	+	+				
161	temporary i/o mode	+	+	+	+	+				
162	IOMSG= specifier in all i/o statements	+	+	+	+	+				
163	namelist i/o on internal file	+	+	+	+	+				
164	recursive i/o	+	+	+	+	+				
	OPEN/CLOSE/INQUIRE **specifiers:**									
165	RECL= for sequential files				+	+	+			
166	RECL= not required if STATUS='OLD									
	format specifiers and edit descriptors:									
175	noncharacter array name allowed			o			o			o
176	variable length fields <...>					+	+			+
177	aEw.dDe double precision exponent					+				
178	aQw[.d] quadruple precision mantissa					+				
182	aR[w] char edit descriptor				+					
183	\ edit descriptor									+
184	Q edit descriptor					+	+		+	+
185	$ edit descriptor			+		+	+		+	+
189	zero field width in edit descriptors				+	+				
190	derived type (DT) edit descriptor	+	+	+	+	+				
191	RU .. round edit descriptors	+	+	+	+	+				
192	DC, DP decimal edit descriptors	+	+	+	+	+				
193	comma after P optiona, if followed by repeat	+	+	+	+	+				
194	g0 edit descriptor		+							
195	unlimited repeat of format list		+							
	compiler directives:									
201	OPTIONS statement						+			
203	OPTION BREAK statement									
204	EJECT statement					+				
205	[NO]LIST compiler directive									+
206	COMPILER(...									

no.	FUJ	SG95	SF95	HP95	INT	CVF	AB95	gfort	g95	PATH	PGI
140				+	+	+	+				+
141											
142				o	o	o					
143	o			o							
144	o										
145											
146					+	+					
147											
148					+	+					
149					+			+	+	+	
155	+										
156					+						
157					+						
158											
159					+			+	+		
160					+			+	+		
161					+			+	+		
162				+	+			+	+		
163					+			+			
164					+			+	+	+	
165											
166											
175				o							
176					+	+					
177				+		+					
178	+			+							
182				+	+						
183					+	+			+		
184				+	+	+	+				+
185	+			+	+	+	+	+	+		+
189	+	+	+	+	+		+	+	+	+	+
190											
191				+	+			+			
192				+	+			+	+		
193				+	+			+	+	+	
194								+			
195								+			
201				+	+	+					+
203											
204											
205											
206											

In the following sections some of the extensions are elucidated and the limitations of the FOR-CHECK implementation of these extensions described.

A.5   Absoft Fortran 77 extensions

· FORCHECK folds all input to uppercase. The Absoft compiler supports folding to upper-case, to lowercase or treat input case sensitive.

· Absoft has compiler options to specify the kind of free form source code. FORCHECK also supports various kinds of free form input but you have to specify this in the configuration file. Default is the Fortran 90 format.

· Absoft has a compiler option to support C-string backslash editing. For FORCHECK you have to enable extension 42 in the configuration file.

· Absoft has compiler options to support conditional compilation lines beginning with 'D', 'd', 'X', or 'x'. In the supplied configuration file for FORCHECK only conditional lines beginning with 'D', or 'd' are enabled. To accept also lines beginning with 'X', or 'x' you must enable extension 3 in the configuration file which, however, accepts conditional lines beginning with any letter.

· FORCHECK supports DO WHILE .. ENDDO, but not WHILE .. ENDDO.

· FORCHECK does not support the following keywords: GLOBAL, INLINE, VALUE, GLOBAL DEFINE, REPEAT.

A.6   Apollo/Domain Fortran extensions

· The Apollo/Domain compiler can read source records up to 1023 characters in free-form mode, FORCHECK reads a maximum of 256 characters only.

· The number of continuation lines is unlimited for the Apollo/Domain compiler, FOR-CHECK can read a maximum of 999 continuation lines only.

· The Apollo/Domain SR10 Fortran compiler accept names up to 4096 significant charac-ters, FORCHECK considers only the first 64 characters as significant.

· The Apollo/Domain compiler accepts by default in-line comment between curly brack-ets (`{ }`). FORCHECK no longer supports this form of comment. For the Apollo/Domain compiler you can specify the in-line comment character using the `–inline` option. In FOR-CHECK you can enable the exclamation mark as the start of in-line comment by enabling extension 6 in the configuration file.

· Apollo Domain Fortran supports C-string backslash editing when the `–uc` compiler option has been enabled. FORCHECK supports backslash editing if extension 42 has been enabled in the configuration file.

· The `INCLUDE` line and the compiler directives `%include`, `%eject`, `%list`, `%nolist` are supported.

· Conditional source input lines can be specified starting with "`D`", or "`Debug`".

A.7   Compaq Fortran extensions

Compaq Visiual Fortran, formaly Digital Visual Fortran, supports most DEC Fortran and Microsoft Fortran Powerstation Fortran extensions.

· The compiler directive `OPTIONS` will be recognized but the specified qualifiers will have no effect.

· cpp preprocessing is supported with limitations.

· The keyword `VIRTUAL` is supported but the limitations in usage will not be checked.

· DEC FORTRAN 4+ synonyms for Fortran 77 keywords in `OPEN` and `CLOSE` are supported, but are flagged.

· Type attributes are skipped, except for `ALLOCATABLE`, which is processed to allow for allocatable arrays. The limitations and consistency in usage of the attributes are not verified.

A.8   Control Data 4000 Fortran extensions

· The Control Data 4000 Fortran compiler accepts the extended precision `REAL*16` type specification. These data will occupy 16 bytes, but will be interpreted as double precision (`REAL*8`).

A.9   Convex Fortran extensions

· The `VAX-FORTRAN` extensions are enabled by default (Convex Fortran `-vfc` option). To disable these options, or enable the Sun Fortran extensions, adapt the configuration file.

· The `INCLUDE` line and the `#include` preprocessor directive are supported.

· The `OPTIONS` directive is accepted, but the options specified will have no effect.

· The `C$DIR` compiler directive is treated as comment.

· Though Convex Fortran accepts names up to 42 characters FORCHECK considers only the first 32 characters as significant.

A.10   Cray Fortran 77 extensions

· The Cray Fortran 77 compiler accepts type specifications with length modifiers but interprets `INTEGER*2`, `*4`, `*8` as 64 bit integers, `LOGICAL*2`, `*4`, `*8` as 64 bit logicals, `REAL*4`, `*8` as 64 bit reals. They occupy a full 64 bit word. `REAL*16`, `COMPLEX*8` and `COMPLEX*16` data occupy two words (128 bits).

· The `CDIR$` directives are treated as comment and have no effect.

· Though Fortran 90 and FORCHECK do, Cray Fortran 77 does not allow an `ENDDO` statement to be labeled.

· Cray Fortran 77 allows recursion in subprograms either by using the prefix RECURSIVE in the subprogram header or by specifying the recursive option in the command line when compiling. In FORCHECK the RECURSIVE prefix is accepted for Cray Fortran 77 (extension 216). Recursive reference without the RECURSIVE prefix can be enabled by specifying extension 229 in the configuration file.

A.11 Cyber NOS/VE Fortran extensions

· The compiler directives are treated as comment. You can use the NOS/VE source code utility to prepare the source code to be analyzed by FORCHECK.

A.12 DEC PDP-11 Fortran-77 extensions

· DEC PDP-11 Fortran-77 does not support the full language, but an extended subset. FORCHECK does not signal the usage of unsupported full language Fortran 77 features, but optionally signals extensions to the full standard.

· The keyword `VIRTUAL` is supported but the limitations in usage will not be checked.

· DEC FORTRAN 4+ synonyms for Fortran 77 keywords in `OPEN` and `CLOSE` are supported, but are flagged.

A.13 DEC FORTRAN and VAX Fortran extensions

· VAX Fortran accepts more than 19 continuation lines as long as the records fit in the statement buffer. Only when the statement buffer becomes full you have to specify the `/CONTINUATIONS` qualifier. FORCHECK accepts a maximum of 999 continuation lines for the VAX Fortran emulation. You can specify the `/CONTINUATIONS` qualifier to change this number, or use the `/F77` qualifier to allow 19 continuation lines only.

· The compiler directive `OPTIONS` will be recognized but the specified qualifiers will have no effect.

· The keyword `VIRTUAL` is supported but the limitations in usage will not be checked.

· DEC FORTRAN 4+ synonyms for Fortran 77 keywords in `OPEN` and `CLOSE` are supported, but are flagged.

A.14 Digital Research Fortran-77 extensions

· The `%INCLUDE` compiler directive is supported.

A.15   F2c Fortran 77 extensions

· The tab is supported but does not imply the analysis of characters beyond column 72.

· By default f2c Fortran supports C-string backslash editing. This can be disabled using
the compiler option −!bs. FORCHECK supports backslash editing if extension 42 has
been enabled in the configuration file, which is the default for the f2c Fortran 77 compiler
emulation.

A.16   GNU Fortran 77 extensions

The GNU Fortran 77 compiler has many options to enable or disable certain language exten-
sions. The configuration file supplied should therefore be considered as a skeleton. You can
easily adapt this configuration file to your needs when using certain optional extensions, when
migrating to Fortran 90. The compiler is now succeeded by gfortran.

· cpp preprocessing is supported with limitations.

· The tab is supported but does not imply the analysis of characters beyond column 72.

· The length of symbolic names is unlimited in GNU Fortran 77. Forcheck considers only
the first 64 characters as significant.

· By default GNU Fortran supports C-string backslash editing. This can be disabled using the
compiler option −!bs. FORCHECK supports backslash editing if extension 42 has been
enabled in the configuration file, which is the default for the GNU Fortran 77 compiler
emulation.

· GNU Fortran accepts a statement label after a statement separator (;). FORCHECK does
not support this feature.

· GNU Fortran accepts continuation lines of INCLUDE directives and more than one INCLUDE
directive can be placed on a single line using statement separators (;). FORCHECK does
not support these extensions.

A.17   HP-UX FORTRAN/9000 and HP Fortran 77 extensions

There are minor differences between the HP-UX FORTRAN/9000 compiler of the HP 9000/300
and 9000/700 series and the HP Fortran 77 compiler of the HP 9000/800 series.

· Though the HP Fortran compilers accept names up to 255 significant characters, FOR-
CHECK considers only the first 64 characters as significant.

· HP compilers interpret a ! as end of line comment when in column 1 or in column 7 to
72. FORCHECK interprets a ! in all columns but column 6 as end of line comment (as in
Fortran 90). In FORCHECK the ˆL character is always processed as a formfeed. In HP-UX
FORTRAN/9000 ˆL is only accepted when found in column 1 of an input record.

· The INCLUDE line and the $include compiler directive are both supported.

- All compiler directives are accepted. Some of them are processed and have the expected effect, such as $LIST, $PAGE, $ANSI. Others have no effect on the Forcheck analysis, such as $ALIAS, $INLINE etc.

- cpp preprocessing is supported with limitations.

A.18 IBM AIX XL FORTRAN extensions

- The XL Fortran compiler has no limit on the length of source records in free-form mode, Forcheck only reads a maximum of 256 characters.

- Though the XL compiler accepts tabs, a tab before a continuation character is not supported. FORCHECK accepts a tab before a continuation character.

- The XL Fortran Fortran compiler accepts names up to 250 significant characters, FORCHECK considers only the first 64 characters as significant.

- By default in FORCHECK the maximum length for type character is set to 32767 for the XL compiler emulation. The default for the XL Fortran Fortran compiler, however, is 500. A larger length for type character for the XL Fortran compiler is allowed by specifying the CHARLEN(len) compiler option or the qcharlen=num command line flag. You also can adapt the FORCHECK configuration file used to have FORCHECK flag the usage of character lengths larger than 500.

- The free form source syntax is not fully supported. A continuation character in front of the on-line comment character (!) is not always detected.

- cpp preprocessing is supported with limitations.

- The PROCESS directive will be accepted, but the compiler options specified have no effect.

- The INCLUDE line is supported, but not conditional.

A.19 IBM VS Fortran V2 extensions

- In FORCHECK the maximum length for type character is set by default to 32767 for the VS Fortran emulation. The default for the VS Fortran compiler, however, is 500. A larger length for type character for the VS Fortran compiler is allowed when specifying the CHARLEN(len) compiler option. You also can adapt the FORCHECK configuration file used to have FORCHECK flag the usage of character lengths larger than 500.

- The free form source syntax is not fully supported. A continuation character in front of the on-line comment character (!) is not always detected.

- The PROCESS directive will be accepted, but the compiler options specified have no effect.

- The INCLUDE line is supported, but not conditional.

· `DEBUG` packets are supported, but with restrictions. Within debug packets all variables are supposed to have the implicit type, and no array-element references are allowed. Moreover, invalid transfer of control from and into debug packets will not be signaled.

· Asynchronous I/O and double byte characters are not supported.

A.20   Intel Fortran extensions

· cpp preprocessing is supported with limitations (fpp).

· The compiler directive `OPTIONS` will be recognized but the specified options will have no effect.

· The keyword `VIRTUAL` is supported but the limitations in usage will not be checked.

· DEC FORTRAN 4+ synonyms for Fortran 77 keywords in `OPEN` and `CLOSE` are supported, but are flagged.

· Type attributes are skipped, except for `ALLOCATABLE`, which is processed to allow for allocatable arrays. The limitations and consistency in usage of the attributes are not verified.

A.21   Lahey F77L Fortran-77 extensions

· The number of continuation lines is unlimited for the Lahey compilers. FORCHECK can read a maximum of 999 continuation lines.

A.22   Microsoft Fortran extensions

The syntax extensions listed apply for both Microsoft Fortran V5.1 and Microsoft Fortran PowerStation V1.0

· The compiler directives are supported.

· Type attributes are skipped, except for `ALLOCATABLE`, which is processed to allow for allocatable arrays. The limitations and consistency in usage of the attributes are not verified.

· Array expressions are supported, but user supplied functions cannot produce an array result.

Most extensions of Microsoft Fortran PowerStation V4.0 are supported. However, only simple logical expressions (name oper const) in the if and elsif directives are supported.

A.23   NDP Fortran extensions

· The NDP compiler can read source records up to 132 characters in fixed-form mode and 13200 in free-form mode, FORCHECK only reads a maximum of 256 characters.

· NDP Fortran supports C-string backslash editing if the compiler option `-f6` is specified. FORCHECK can support backslash editing by enabling extension number 42 in the configuration file.

A.24   Prime Fortran-77 extensions

· In-line comment between /* and */ is not supported anymore.

· The maximum number of continuation lines allowed depends for Prime Fortran on how many language elements each line contains. FORCHECK allows 19 continuation lines by default.

· Both the `INCLUDE` line and the `$INSERT` directive are supported.

· The B-field edit descriptor is not supported.

· The `SHORTCALL` statement is not supported.

· The `FULL LIST` compiler directive is not supported.

A.25   Salford Fortran extensions

Most FTN77/386 extensions are supported but a number of the newer FTN extensions are not.

· Though the maximum number of continuation lines supported is 19 for fixed format and 39 for free format, the FTN compilers allow more continuation lines depending of the length of the lines. FTN95 allows 19 in fixed format, 39 in free format and 99 in free format in .NET configuration or if the Fortran 2003 switch /F03 has been specified.

· The compiler directive `OPTIONS` will be recognized but the specified qualifiers will have no effect.

· Internal procedures are not supported.

· INTERRUPT SUBROUTINE, SPECIAL SUBROUTINE and SPECIAL ENTRY are not supported.

· Conditional compilation (CIF, CELSE, CENDIF) is not supported.

· The % prefix to denote an address in a DATA statement is not supported.

· Business editing is not supported.

A.26   Silicon Graphics MIPSpro Fortran 77 extensions

· cpp preprocessing is supported with limitations.

· The SGI Fortran 77 compiler accepts the extended precision `REAL*16` type specification. These data will occupy 16 bytes, but will be interpreted as double precision (`REAL*8`).

· By default the SGI Fortran 77 compiler supports C-string backslash editing. This can be disabled using the compiler option `-backslash`. FORCHECK supports backslash editing if extension 42 has been enabled in the configuration file, which is the default for the SGI compiler emulation.

· SGI Fortran 77 supports recursive subprogram references when the -automatic compiler option is specified during compilation. In FORCHECK extension 229 is enabled in the compiler emulation file to allow for recursion.

A.27   Sun Fortran 77 extensions

· cpp preprocessing is supported with limitations.

· The tab is supported but does not imply the analysis of characters beyond column 72.

· The default maximum number of continuation lines for the Sun compiler is 19. This maximum can be increased using the `-Nln` option. FORCHECK also allows a maximum of 19 continuation lines by default. FORCHECKś maximum can be increased up to 999, using the `-cont n` option.

· By default Sun Fortran supports C-string backslash editing. This can be disabled using the compiler option `-xl`. FORCHECK supports backslash editing if extension 42 has been enabled in the configuration file, which is the default for the SUN compiler emulation.

A.28   Unisys 1100 Fortran-77 extensions

· Records beginning with "#" or "@" are skipped.

· Though the number of continuation lines is unlimited for the Unisys Fortran compiler FORCHECK can read a maximum of 999 continuation lines.

· `DEBUG` packets are supported with the restrictions as described for IBM VS Fortran.

A.29   Watcom Fortran 77 extensions

· The Watcom compiler interprets a ! as end of line comment in any column. FORCHECK interprets a ! in column 6 as a continuation character (as in Fortran 90).

· FORCHECK does not support the Watcom `*$include` compiler directive.

A.30   Changing the configuration file

The configuration file is composed of the following sections.

Sections of the configuration file

- · GENERAL

- · EXTENSIONS

- · INTRINSICS

- · OCI

- · MESSAGES

- · VARIOUS

The sections are identified by a header with the section name within brackets. In the folowing sections each configuration file section is described. Lines beginning with "!" are treated as comment. To enable a specific configuration file, see the section "The usage of language extensions" of the chapter "Operation".

A.30.1   GENERAL

Mnemonic of the emulated compiler, Fortran conformance level

The first line specifies the lowest FORCHECK version number which can read this configuration file. The next line "Mnemonic of the emulated compiler, Fortran conformance level" specifies the following:

1. Mnemonic of the emulated compiler. This is a eight character string which will be presented at program startup and in the headers of the list file. It has no effect on the analysis.

2. Fortran conformance level. This is a three character string and can be: "F77", "F90", "F95", "F03", or "F08". All extensions are relative to the language level specified and all syntax of this language level will be enabled.

Type information

The next subsection "Type information" specifies the types and kinds supported, and the limits of the types.

1. Number of bits, difference between ABS(min) and max value of default integer.

2. Number of bits for an address as used for integer POINTER (extension 55).

3. Number of bits for the various integer types.

4. Number of significant binary digits of reals.

5. Decimal exponent range of reals.

6. Maximum exponent of reals.

7. Minimum exponent of reals.

8. Minimum real which is not zero.

9. The maximum length of character constants and variables.

10. Type mnemonics.

11. Default byte-lengths of the various types

12. Byte-lengths with short-length option enabled.

13. Byte-lengths with short-length option disabled.

14. Supported types

15. Supported types for generic procedures.

16. Table of available kinds and byte lengths (5 lines).

Miscellanious

The next subsection "Miscellanious" is composed of the following lines:

1. Default file name extensions: source, include. List-option delimiter for `INCLUDE` line.

2. Maximum number of continuation lines in fixed source form, and free source form, 0: unlimited, so accept the maximum FORCHECK can handle.

3. Maximum length of identifiers: local names, entry names, common-block names, 0: unlimited, so accept the maximum FORCHECK can handle.

4. Compiler directive strings. Two strings can be specified with a maximum length of 10 characters each. For cpp preprocessing one of these strings must be '#'.

5. Free-form continuation characters. The first character specified is the character which indicates the current line will be continued. Except in character context, if the last nonblank character before a ! is this character, the line will be continued. The second character is a character which can be used to indicate a continuation line.

6. First column free-form comment characters. Two characters can be specified which indicate for free-form input a comment line when placed in the first column.

A.30.2   EXTENSIONS

In this section you can include the numbers of the syntax extensions you like to enable. Each number must be specified on a single record, optionally followed by comment within apostrophes.

A.30.3 INTRINSICS

FORCHECK recognizes all standard Fortran intrinsic procedures. Moreover the additional intrinsic procedures as specified in the configuration file will be recognized. You can modify the configuration file and remove, add, or change the nonstandard intrinsic procedures to be recognized. Not all *specific* names of each generic procedure are specified in the various configuration files, because in general there is no need to use these names.

FORCHECKcan accept added intrinsic functions which are standardized in a higher Fortran standard level than the Fortran conformance level as specified in this configuration file without reporting. You can group the added intrinsic functions for each language level. Each group must have one of the following headers:
!Fortran 90 additions
!Fortran 95 additions
!Fortran 2003 additions
!Fortran 2008 additions
The nonstandard compiler specific additions must be in a group with the following header:
!Nonstandard additions

If you specify e.g. /F03 only the intrinsic functions which are not in the Fortran 2003 standard are reported.

In the next paragraphs we describe the way intrinsic procedures can be specified in the configuration file. The properties of intrinsic procedures are very divers and hard to specify in a general way, covering all implementations. Moreover the various Fortran language reference manuals describe the intrinsic functions each in their own way from which it is often hard to discover the system behind the generation of specific functions from generic functions. Therefore it is not an easy task to specify additional intrinsic procedures in the configuration file. However, if you follow the rules described below and use the configuration files supplied as examples you will be able to fulfill the job.

In the record "allowed type lengths for generic procedures" of the configuration file you can specify which argument type lengths will be accepted by a generic function to generate a specific function. To allow the BYTE type as argument, specify it as INTEGER*1.

Each specific intrinsic procedure is specified by a header record and a record for each of its arguments. The list of intrinsic procedures is delimited by a record with a zero.

The header record is composed of the following fields:

1. Generic procedure name, string.

   If blank, the procedure is specific only. If non-blank, and if the procedure does not exist already, it is added to the list of generic procedures.

2. Specific procedure name, string.

   If the specific procedure name already exists, the specific procedure specified overrules the existing one. Otherwise the specific procedure name is added to the list of specific procedures.

   If the generic procedure name is non-blank, the procedure is added to the chain of specific procedures which can be generated from the generic procedure.

If the specific name is left blank the generic name is used as the specific name.

Specific procedures must have different names if they can be generated from a single generic procedure and have different resulting types or type lengths.

If the intrinsic procedure is a subroutine, the procedure type must be specified as 'S', the type length and rank are not relevant and can be set to zero.

3. Procedure type, character.

    ' ' same as the type of the argument(s)
    '?' typeless
    'C' complex
    'CH' character
    'I' integer
    'L' logical
    'R' real
    'X' result of MATMUL
    'S' no function but subroutine

4. Procedure type kind/length, integer. Special codes:

    >0 type length
    0 same as the type kind/length of the argument(s)
    -1 default type kind/length of the function type
    -2 default type kind/length of type double precision
    -3 same as the type kind of the argument(s); half the type length of the type length of the arguments
    -4 unknown
    -5 type kind of an address (or integer POINTER)

5. Procedure rank and shape code, integer. Special codes:

    0 scalar
    1 rank 1
    2 rank 2
    -1 take shape of argument with largest rank
    -2 rank N+1
    -3 scalar or rank 1
    -4 scalar or rank N-1
    -5 rank 1 or N-1
    -6 shape of second array argument
    -7 follow the rules of matrix multiplication

In which N is the highest rank of all arguments.

6. Number of arguments, integer. Special codes:

    0 any number of arguments allowed, none argument line must follow
    -1 one or two arguments allowed, one argument line must follow
    -2 two or more arguments allowed, one argument line must follow

    -3 one or none arguments allowed, three argument lines must follow

    -4 two or three arguments allowed, two if first argument is complex; three argument lines must follow

7. Procedure name allowed as actual argument, logical.

8. Intrinsic procedure class, string:

    'A' atomic subroutine
    'E' elemental function
    'I' inquiry function
    'P' procedure (can be referenced as function or subroutine)
    'S' subroutine
    'T' transformational function

9. Compile-time inquiry, or transformational function, logical.

10. Optional comment, string.

Each record for an argument is composed of the following fields:

1. Argument name, character.

2. Argument type, character.

    ' ' any type allowed (but all arguments must have the same type)
    '?' typeless
    'C' complex
    'CH' character
    'I' integer
    'L' logical
    'N' numeric: integer, real, complex
    'R' real
    'T' derived type
    'U' intrinsic type
    'X' any type allowed, don't check

3. Argument type kind/length, integer.

    >0 type length
     0 any kind/length allowed which is allowed for the generic procedure
    -1 default kind/length of the argument type
    -2 double precision

4. Argument rank, integer. Special codes:

     0 argument must be scalar
     1 argument must be array of rank 1
     2 argument must be array of rank 2
    -1 array argument required

-2 argument can be scalar or array, even in Fortran 77

-3 argument can be scalar or array

-4 argument can be scalar or rank N-1

-5 argument can be rank 1 or 2

-6 argument must be a dummy argument

-7 argument must be the name of a variable or external procedure

-8 argument must be a pointer or pointer procedure

In which N is the highest rank of all arguments.

5. Argument must have the same type as the previous ones (if any), logical. If the resulting type kind of the intrinsic procedure depends on the type kind of this argument, this flag must be set true.

6. Argument is optional, logical.

7. Argument must be defined on entry, logical.

8. Argument will be defined, logical.

9. Optional comment, string.

A.30.4   OCI (OPEN/CLOSE/INQUIRE) specifiers

FORCHECK recognizes all standard Fortran specifiers. Moreover the additional specifiers as specified in the configuration file will be recognized. You can modify the configuration file and remove, add, or change the nonstandard specifiers to be recognized.

FORCHECKcan accept added specifiers which are standardized in a higher Fortran standard level than the Fortran conformance level as specified in this configuration file without reporting. You can group the added specifiers for each language level. Each group must have one of the following headers:

!Fortran 90 additions

!Fortran 95 additions

!Fortran 2003 additions

!Fortran 2008 additions

The nonstandard compiler specific additions must be in a group with the following header:

!Nonstandard additions

If you specify e.g. /F03 only the specifiers which are not in the Fortran 2003 standard are reported.

In the next paragraphs we describe the way specifiers can be specified in the configuration file.

Each OPEN, CLOSE or INQUIRE keyword or combination of keyword and value must be specified on a single record of the configuration file. The list is delimited by a record with a zero. Each record has the following format:

1. Keyword, string.

If a keyword starts with the characters of another keyword, the longest keyword has to be specified first, or the "=" must be included in the name of the shortest keyword. Specify a blank before the "=" to allow non-significant blanks between the keyword and the "=". If a keyword may be split up in more than one part, separated by blanks (Fortran 90 free form input), include a blank in the specification at these positions.

2. `OPEN/CLOSE/INQUIRE` indicator, character.

   'O' can be used in `OPEN` statement
   'C' can be used in `CLOSE` statement
   'I' can be used in `INQUIRE` statement

   Specify additional records with the same keyword for each statement type in which the keyword can be specified.

3. Value or value type, string.

   This field can either denote a value keyword (character constant), or the type of a variable value.

   If a value can be a value keyword, specify a value keyword in the value type field. Each keyword and value combination must be specified in a separate record. A value keyword cannot be shorter than two characters. If it has a length of two characters, it cannot end with an 'R' or an 'A'. If a value keyword starts with the characters of another value keyword, this value keyword has to be specified first. If a value keyword may be split up in more than one part, separated by blanks, include a blank in the specification at these positions. A specific value keyword can be specified for two different open keywords and one close keyword.

   If the value can be a variable, the first character of the value type field denotes the type of the value.

   'N' no value expected
   ' ' any type allowed
   'E' external expected
   'I' integer datum expected
   'K' key description expected
   'L' label or logical expected
   'C' character
   'U' unit specifier expected
   'V' scalar-default-char-variable expected

   The second character of the value type field denotes reference or assignment.

   'R' reference
   'A' assignment

   For `OPEN` and `CLOSE` 'R' is the default, for `INQUIRE` 'A' is the default. Note that the value type and reference/assignment character are to be specified in a single string field, for example 'IA' to denote an integer assignment.

4. Synonym keyword, string.

   Here you can specify for which keyword the keyword is a synonym. If the keyword is no synonym specify a blank string. If nonblank the value type field is not relevant. Synonyms will be flagged as nonstandard.

5. Standard Fortran specifier, logical.

   T The keyword is a standard Fortran specifier
   F The keyword is no standard Fortran specifier

A.30.5   MESSAGES

In the section "messages" you can redefine messages. You specify the numbers of the messages which you want to suppress or of which you want to change the severity. Each message number, followed by the severity level flag within apostrophes, must be specified on a single record. See the section "The usage of language extensions" of the chapter "Operation" for a precise description. You also can specify `suppress='all'` to suppress all diagnostic messages.

A.30.6   OUTPUT

In this section you can specify what information is sent to stdout, is stored in the listing file and in the report file. See the section "Tuning the output" of the chapter "Operations".

A.30.7   VARIOUS

In this section you can specify the count mode, the format of the message reporting and the date/time format. See the corresponding sections of the chapter "Operations".

Appendix B

Limitations

FORCHECK is a static analyzer, therefore it cannot detect any errors which manifest themselves at run time only. For example, a variable array index, or variable character substring expression which is out of bounds, cannot be detected. Likewise, the detection of operations on external files can hardly be checked without executing the program. For example a file which has not been opened before usage, or a variable logical unit not being used consistently, cannot be detected.

FORCHECK warns you, if possible, when a variable has not been defined in a program unit, when a common-block object has not been defined in the program (use the `/CO` option to enable this feature), when an allocatable variable has never been allocated , or when a pointer has never been associated to a target or procedure. However, if an object is used as an input/output actual argument FORCHECK cannot verify this. In a limited number of cases FORCHECK reports when an item has been referenced, before it was defined, allocated, or associated. However the path flow analysis to detect this is limited. As soon as a labeled executable statement has been encountered and either a forward reference to a label has been made, or we are in a construct, FORCHECK cannot signal this kind of errors any more. So avoid labels and goto's. This is another good reason to use `IF` and `SELECT CASE` constructs as much as possible! By specifying the `/RI` option FORCHECK will detect more occurrences of "referenced before defined" at the cost of more false alarms

Arrays, character variables and variables of derived type are treated as a single entity. The individual array elements, substring elements or structure components are not checked for unreferenced, undefined, or not allocated. This is not only to reduce the storage and processing time requirements, but also because most array and substring elements are referenced using variable array indices or substring values which cannot be verified statically.

Recursive I/O attempts will only be detected in a limited number of cases. FORCHECK does not compare the consistency of format strings with the actual I/O list. This is because many I/O lists have implied `DO` loops which generate a variable number of elements. Future versions of FORCHECK may check format strings as far as possible.

B.1   Configuration determined limits

The tables used in FORCHECK to store all information have limited sizes. The sizes of all internal tables will be specified in the following table.

These limits cannot be changed by the user. When a limit has been exceeded a system message will be given. Analysis will proceed, but will no longer be complete.

value	description
255	max. length of a file specification
255	max. length of an include filename
512	max. number of characters in an input record
512	max. number of characters in an output record
25	max. nesting of include files
100	max. nesting of modules
50	max. nesting of references in call-tree
200	max. number of library files
1000	max. number of (non-comment) lines in a statement
20000	max. number of characters in a statement
8000000	length of name table
20000	max. number of contexts in a program unit
100	max. nesting of structures + unions + maps
16	max. number of parameters of a derived type
10000	max. nesting of DO + IF + ELSEIF + ELSE + SELECTCASE + CASE
7	max. nesting of implied DO loops in DATA statement
30	max. nesting level in an expression
2000	max. number of objects being checked in an argument list, or equivalence list
4000	max. number of shape, bound, or vector values in an argument list, equivalence list, or common-block list
16	max. number of derived-type parameters for a derived type
20000	length of argument key list
4000	max. number of objects in a common-block list, or data list
200000	max. number of entries in the symbol table
1000	max. number of references in a cross-reference table presented
1000000	max. total number of references in the cross-reference tables
1000	max. number of non-analyzed procedures presented
100	max. number of messages that can be redefined
25	max. number of common blocks specified with the /SC:c option
25	max. number of modules specified with the /SMT:m option
25	max. number of modules specified with the /SMV:m option
25	max. number of roots specified with the /SRS:r option
25	max. number of roots specified with the /SMD:m option
20	max. number of program units specified with the /IL option
50	max. number of include directories specified with the /IP:p option
500	max. maximum number of intrinsic procedures
100	max. maximum number of OPEN/CLOSE/INQUIRE keywords
100	max. maximum number of OPEN/CLOSE/INQUIRE value keywords

Appendix C

History of changes

See the supplied file history.txt for all relevant changes that have been made to FORCHECK since the introduction of version 14.

Appendix D

Message summary

In this appendix all system and analysis messages are listed. The messages which are not self-explaining are elucidated.

1 I (MESSAGE LIMIT REACHED FOR THIS STATEMENT OR ARGUMENT LIST)

- FORCHECK displays only the first 5 messages in a statement or argument list.

2 E (OPEN ERROR ON INCLUDE FILE)

- FORCHECK cannot locate or open the include file.

3 E (INCLUDE NESTING TOO DEEP)

- The nesting of include files is larger than FORCHECK can handle.

4 O (NEXT SOURCE RECORD TOO LONG, REMAINDER NOT PROCESSED)

- The source input record is longer than FORCHECK can handle.

5 O (TOO MANY (COMMENT) RECORDS IN STATEMENT, REMAINDER NOT PROCESSED)

- The number of (comment) lines in the statement is longer than FORCHECK can handle.

6 O (STATEMENT TOO LONG, REMAINDER NOT PROCESSED)

- The number of characters in the statement is larger than FORCHECK can handle.

7 O (TOO MANY STATEMENTS, REMAINDER NOT PROCESSED)

- The number of statements in the program unit is larger than FORCHECK can handle.

8 O (NAME TOO LONG, TRUNCATED)

- The identifier is longer than FORCHECK can handle.

9 O (ARRAY TOO LONG, LENGTH NOT VERIFIED)

- The length of the array is longer than FORCHECK can handle.

10 O (CHARACTER ENTITY TOO LONG, LENGTH NOT VERIFIED)

- The character constant or type length is longer than FORCHECK can handle.

11 O (NUMBER CANNOT BE CONVERTED)

- The number concerned is too large for the system being used.
- The format of the number is not available on the system being used.

12 O (NAME TABLE FULL, REMAINDER NOT PROCESSED)

- The table with identifiers is full. When using many long names the name table can become full before the symbol table is full.

13 O (SYMBOL TABLE FULL, REMAINDER NOT PROCESSED)

- The table with information concerning named entities is full.

14 O (CONTEXT TABLE FULL)

- The number of contexts is larger than FORCHECK can handle.

15 O (NESTING TOO DEEP)

- The nesting of array subscripts, function-, and subroutine argument lists is too deep.
- The nesting of implied DO loops in a DATA statement is too deep.
- The context nesting is too deep

16 O (EXPRESSION STACK OVERFLOW)

- The expression is to complex to analyze.

17 E (EXPRESSION STACK UNDERFLOW)

- Internal FORCHECK error, please report.

18 O (CONSTRUCT STACK OVERFLOW)

- The nesting of constructs, is too deep.

19 O (DERIVED-TYPE/STRUCTURE NESTING TOO DEEP)

- The stack for nesting of derived-types and structures is full.

20 O (TOO MANY OBJECTS IN DATA STATEMENT, REMAINDER NOT VERIFIED)

21 O (TOO MANY EQUIVALENCE LISTS, REMAINDER NOT PROCESSED)

22 O (TOO MANY ARGUMENTS, REMAINDER NOT VERIFIED)

23 O (TOO MANY ARGUMENT SHAPES, REMAINDER NOT VERIFIED)

24 W (ROOT ENTRY NOT FOUND)

25 O (TOO MANY REFERENCES, REMAINDER PRINTED IN SEPARATE SUB-TREES)

26 O (TOO MANY PROGRAM UNITS, REMAINDER NOT PROCESSED)

27 O (CROSS-REFERENCE TABLE FULL, REMAINDER NOT PRESENTED)

28 O (TOO MANY COMMON-BLOCK OBJECTS TO CROSS-REFERENCE)

29 W (LIBRARY ENTRY NOT FOUND)

30 O (TOO MANY LIBRARY ENTRIES, REMAINDER NOT PROCESSED)

31 O (ARGUMENT-KEY STACK FULL, REMAINDER NOT PROCESSED)

- The stack with argument keys is full. When using many long argument keys the argument key stack can overflow before the argument stack overflows.

32 O (CONDITIONAL-COMPILATION SYMBOL TABLE FULL)

33 O (CONDITIONAL-COMPILATION NESTING TOO DEEP)

34 O (INVALID NESTING OF CONDITIONAL-COMPILATION META COMMANDS)

35 O (EXPRESSION COULD NOT BE EVALUATED)

36 O (STACK OVERFLOW WHILE PROCESSING REFERENCE STRUCTURE)

37 I (SOURCE POSSIBLY IN FREE FORM. SPECIFY THE FREE-FORM OPTION)

38 O (TOO MANY MESSAGES SUPPRESSED, REMAINDER IGNORED)

39 O (NAME AND REFERENCE DO NOT FIT ON A LINE, ENLARGE PAGE WIDTH)

40 E a ';' must not be the first nonblank character on a line

41 E invalid line

- A non-comment, non-compiler directive line with less than 6 characters has been read.

42 E first line must not be a continuation line

- The line is the first line encountered in the statement and has not a zero or blank in column 6.

43 E invalid characters in front of continuation line

- Characters have been found in column 1-5 of a fixed form continuation line.

44 W first line after an INCLUDE line must not be a continuation line

45 W too many continuation lines

- The statement has more continuation lines than the emulated compiler can handle.

- The statement has more than 19 continuation lines and the Fortran 77 standard option has been specified.

- The statement has more than 19 continuation lines and the Fortran 90 or 95 standard option has been specified and the source is in fixed form.

- The statement has more than 39 continuation lines and the Fortran 90 or 95 standard option has been specified and the source is in free form.

46 E unrecognized characters at end of statement

- After processing the statement there were characters left in the statement buffer.

47 W statement field empty, CONTINUE assumed

48 E invalid characters in label field of statement

- Only a label in column 1-5, and a zero or blank in column 6 are allowed in front of a statement.

49 W continuation character not in Fortran character set

50 W lower case character(s) used

51 W nonstandard Fortran comment used

52 W conditional compilation or D_line(s) used

53 W tab(s) used

54 W formfeed(s) used

55 W include line(s) used

56 E unbalanced delimiters

57 E invalid filename specification

58 I none of the entities, declared in the include file, is used

59 I character constant split over more than one line

- This may be non-portable.

60 W fixed source form used

61 I no statement found in program unit

- Only comment lines or non-included conditional source lines were read.

62 E continuation character missing

- In freeform input the first nonblank character of a continuation line in a character context must be an &.

63 I unrecognized characters after compiler directive

- the cpp preprocessor does not allow characters after directives without arguments.

64 W line too long

65 I   continued character constant has more than one leading blank

66 I   comment line(s) within statement

69 E   unrecognized statement

- The syntax is not recognized. This may be caused by a non- standard keyword which is not part of the supported extensions.

70 I   ambiguous statement. Type statement assumed

- A function statement must have an (empty) argument list, so this statement is treated as an explicit type statement.

71 W nonstandard Fortran statement

72 E   statement not allowed in MAIN

73 E   statement not allowed in BLOCKDATA

- In a blockdata program unit only specification statements, and no executable statements are allowed.

74 E   statement not allowed within the specification part of a (sub)module

75 E   this statement can only be used within a construct

76 E   this statement can only be used within a loop construct

77 E   statement not allowed within this context

78 E   statement out of order

79 E   type specification out of order

- The type specification must confirm the implicit type or be defined before the declaration statement where it is used.

80 W non-DATA specification statements must precede DATA statements

- In Fortran 77 any DATA statement should be placed after other specification statements.

81 E   no shape specified, or statement function out of order

- An undeclared subscripted variable or function name with arguments is used at the left side of an assignment statement.

82 E   this statement cannot have prefixes

- Only a FUNCTION or SUBROUTINE statement can have prefixes.

83 E   internal or module procedure expected

- After a CONTAINS statement at least one internal or module procedure must be specified.

84 I   no path to this statement

85 E   procedure END missing

86 E   program unit END missing

87 E   non-matching program unit or subprogram type in END

88 E   non-matching name in END

89 E   missing delimiter or separator

90 E   unmatched parentheses

91 E   missing parenthesis

92 E   ")" expected

93 E   "/" expected

94 E   syntax error

95 W   nonstandard Fortran syntax

96 W   obsolescent Fortran feature

- This syntax is marked as obsolescent in the effective Fortran standard.

97 I   PARAMETER statement within STRUCTURE

- Defined named constants are not local to the structure, so they can better be placed outside the structure definition.

98 W   deleted Fortran feature

- This syntax is marked as deleted in the effective Fortran standard.

99 W   DATA statement among executable statements

- This is marked as obsolescent in the Fortran 95 standard.

100 E   statement not allowed within a pure procedure

101 E   statement not allowed within an interface block

102 E   statement only allowed within an interface block

103 E   statement only allowed within the spec. part of a (sub)module

104 E   statement only allowed in interface block or spec. part of subprog.

105 E   statement not allowed within a BLOCK construct

106 E lexical token contains blank(s)

- In free form source form blanks in a name, literal constant, operator, or keyword are not allowed.

107 E blank required in free source form

108 I use a blank to delimit this token

- In fixed form source form of Fortran blanks are not significant but the absence of a delimiter between these lexical tokens might indicate a syntax error.

109 I lexical token contains non-significant blank(s)

- In fixed form source form blanks are not significant. However, a blank in a name, literal constant, operator, or keyword might indicate a syntax error.

110 W name or operator too long

- The name or is longer than 6 characters and the conformance to the Fortran 77 standard option has been specified.

- The name or operator is longer than 31 characters and the conformance to the Fortran 90 standard option has been specified.

- The name or operator is longer than the maximum name length supported by the emulated compiler.

111 E operator name must consist of letters only

112 W name is not unique if truncated to six characters

113 E invalid name

- The syntax of the name is in error. Invalid characters have been used in the identifier.

114 E statement label too long

- A statement label must consist of 1 to 5 digits.

115 E multiple definition of statement label, this one ignored

116 E statement label already in use

117 E statement label type conflict

- A label must either be used to identify a format statement, or a non-format statement.

118 E invalidly referenced

119 E invalid reference

120 I referenced from outside entry block

121 E statement label invalid

122 E format statement label missing

123 E undefined statement label

  - A referenced statement label has not been defined.

124 I statement label unreferenced

  - A statement label has been defined but is never referenced (used).

125 I format statement unreferenced

134 E missing apostrophe or quote

  - The closing apostrophe or quote of a character constant is missing.

135 E zero length character constant

  - In Fortran 77 a character constant must not be of zero length.

136 E invalid binary, octal or hexadecimal constant

137 E kind type parameter of real constant not allowed for this exponent

  - If the kind is specified, only E is a valid exponent letter.

138 E invalid complex constant

139 E invalid Hollerith or Radix constant

140 E missing character to escape in C-string

  - The closing apostrophe or quote of the C-string is preceded by a "\".

141 E invalid usage of named constant

  - A named constant is used in a context where a variable, or procedure name is expected.

  - In standard Fortran no named constants are allowed to define the real or imaginary part of a complex constant.

142 E real or integer constant expected

143 W character length too large

  - A character constant or variable is longer than the emulated compiler can handle.

144 E number too large

145 I implicit conversion of scalar to complex

- An integer or real value is assigned to a complex variable. The imaginary part of the complex becomes zero. If the real is zero this information is only presented if the rigorous option has been specified.

146 E unsigned nonzero integer expected

147 E unsigned integer expected

148 E positive integer expected

149 E integer too large for its kind

150 W integer larger than default

151 E invalid or unrecognized attribute

152 I PRIVATE is already the default

- PRIVATE has already been specified.

153 I PUBLIC is already the default

154 E implicit type already used; type declaration must confirm this type

155 E conflict with generic name

156 E conflict with derived-type name

157 E invalid usage of subscripts or substring

158 E already specified PUBLIC

- PUBLIC has already been specified.

- PRIVATE has been specified but PUBLIC has been specified before.

159 E name already in use

160 E invalid usage of variable

- Because of the previous context the name appeared to be a variable but is now used in a context where a procedure name is expected.

161 E scalar variable name expected

- An array element, array name, constant, external, structure, derived-type name or namelist name is not allowed in this context.

162 E named scalar expected

- No array name, array section, array element, substring, or expression is allowed in this context.

163 E no array allowed

- No array name or array section allowed.

164 E   missing array or shape specification

165 E   invalid shape specification

166 E   missing array subscripts

167 E   invalid usage of subscripts or bounds

- An array element is not allowed in this context.

- A scalar can not be subscripted or have bounds.

168 E   invalid number of subscripts or bounds

- The number of subscripts is larger than the maximum rank.

- The number of subscripts or bounds is different from the declared rank.

- The number of lower-bound expressions or bound remappings is different from the declared rank.

169 E   invalid shape bounds

- The first bound of a specified shape is higher than the second bound.

- Array must not be zero sized in this context.

170 E   shape specification out of order

- The shape must be specified before first usage.

171 E   multiple specification of shape

- The shape of the array has been declared more than once.

172 E   invalid array or coarray specification

173 E   invalid usage of assumed-size array specification

- Only dummy array arguments can be specified with an assumed-size.

- The function name of an array-valued function must not be declared assumed-size.

174 E   invalid usage of assumed-size array name

- An assumed-size array name can only be used as an actual argument in a procedure reference for which the shape is not required.

175 E   invalid usage of adjustable-array dimension

- Only dummy-array arguments can be specified with adjustable dimensions.

176 E   invalidly used in adjustable or automatic type declaration

- A variable which specifies an array dimension or character length must either be a procedure argument (with intent(in)), in common, or a global module variable.

177 E deferred- or assumed-shape array specification not allowed

178 E deferred-shape array specification required

- A POINTER or an ALLOCATABLE array must be specified as a deferred-shape array.

179 E explicit-shape array specification required

- An array valued function result, without the POINTER or ALLOCATABLE attribute, must have an explicit shape.

180 E invalid usage of automatic-array specification

- An automatic array must not appear in the specification part of a (sub)module

181 E invalid usage of assumed length

- Only a dummy argument, function result, or named constant of type character can be specified with assumed length.

- The type length of a statement-, internal-, or module function cannot be of assumed length.

- The type length of a dummy statement function argument can not be of assumed length.

- A function with pointer valued result cannot be of assumed length.

182 E invalid usage of adjustable-length specification

- Only dummy arguments or automatic objects can be specified with an adjustable length parameter.

- Statement functions and statement function arguments cannot be specified with adjustable length

183 E invalid length or kind specification, default assumed

- A kind type parameter must be a nonnegative scalar integer constant expression.

184 E multiple specification of attribute

185 E invalid combination of attributes

186 E attribute not allowed in this context

187 E invalid to (re)define type or attribute

188 E OPTIONAL and INTENT only allowed for dummy arguments

189 E already specified PRIVATE

- PUBLIC has been specified but PRIVATE has been specified before.

- PRIVATE has already been specified.

190 E type parameter not allowed for this type

191 E invalid specification of type parameters

192 E invalid usage of type parameters

193 I already specified in host context

194 W unsupported type length, default assumed

- A type length specification of this type is not supported by the emulated compiler.

195 E type length invalidly specified

- The type length cannot be specified in this context

- The emulated compiler does not support this nonstandard Fortran syntax.

196 E initialization only allowed in attributed form of type spec.

- Use '::' between statement keyword and list.

197 E a named constant cannot have the POINTER, TARGET, or BIND attribute

198 E constant expected

199 E missing parentheses

- In standard Fortran the list of a PARAMETER statement must be enclosed in parentheses. Be aware, however, that the syntax extension without parentheses provided by some compilers uses a different assumption of the type of named constant. In standard Fortran the type is the implicitly or explicitly defined type of the name. In the syntax extension the type becomes the type of the named constant.

200 E constant expression missing

- If the PARAMETER attribute has been specified, the named constant must be given a value.

201 E entity must have been declared previously

202 E multiple specification of type, this one ignored

- The entity has already been typed by an explicit type statement.

203 E name invalidly typed

- The name must not appear in an explicit type statement.

204 I implicit type already used, change sequence

- An explicit type specification confirms the implicit type of a variable that has already been used.

205 E implicit properties already used, statement out of order

- An explicit type specification defines the type of a variable that has already been used.

- An implicit statement defines the type of an entity while the implicit type of the entity has already been used.

- An IMPLICIT ALL compiler directive has been specified while the implicit type of one or more entities has already been used.

- A shape specification defines the shape of a variable or function that has already be used as a scalar.

206 E invalid implicit range

- The first and second character in an IMPLICIT list must in lexicographic order.

207 E multiple implicit type declaration, this one ignored

- An implicit type has been specified more than once for one or more characters in the list.

- IMPLICIT NONE has been specified and another IMPLICIT statement has already been specified.

- IMPLICIT NONE has been specified but an implicit type has already been used.

208 W name not explicitly typed, implicit type assumed

- The entity has not been explicitly typed and:

- IMPLICIT UNDEFINED has been specified for the first character of the symbol.

- The declare option has been specified.

209 W conflict with IMPLICIT NONE specification or option

- An IMPLICIT statement has been specified while IMPLICIT NONE has been specified or enabled.

210 E SAVE has already been specified for this entity

211 E SAVE and AUTOMATIC cannot be specified both

212 E invalid to save this entity

- Only named common blocks and variables can be saved.

- There is no need to save the blank common because the common-block values in blank common do not become undefined after a RETURN or END.

- Common-block objects cannot be saved.

- Automatic and static arrays and pointees cannot be saved.

- Local variables of pure procedures must not be saved.

213 E   SAVE or BIND specified but entity not declared

- A variable or common block has been specified in a SAVE or BIND statement but has not been declared or used.

214 E   not saved

- If a common block has been specified in a SAVE statement in a subprogram, it must be specified in a SAVE statement in every subprogram in which the common block has been specified.

- If an object of a type for which component initialization is specified appears in the specification part of a (sub)module and does not have the ALLOCATABLE or POINTER attribute, the object must be saved.

- An object in an initial data target must be saved.

215 E   already specified automatic, static or allocatable

- An object must only be specified automatic, static or allocatable once.

- AUTOMATIC and STATIC cannot be specified both.

216 E   invalidly specified automatic, static or allocatable

- A dummy variable, a common-block object and a pointee must not be specified automatic, static or allocatable.

- An allocatable array must not be specified automatic or static and must not be a pointer.

- An automatic, static or allocatable object must not be equivalenced.

- A target in a pointer initialization must not be allocatable.

- An assumed-type object must not be allocatable.

217 E   conflict with program unit or ENTRY name

- The name of a constant, as defined in a PARAMETER statement must not be the same as a global name of the subprogram, such as the name of the program unit, or an entry.

- The name of a common block must not be the same as the name of a program unit or ENTRY.

218 E   conflict with common-block name

- The name of a constant, as defined in a PARAMETER statement must not be the same as the name of a common block specified in the current subprogram.

- A global name, such as the name in a PROGRAM, BLOCKDATA, SUBROUTINE, FUNC-TION or ENTRY statement, must not be the same as the name of a common block of the program.

219 E invalidly in COMMON, EQUIVALENCE, or NAMELIST

- A dummy procedure argument, automatic or allocatable variable and a pointee cannot be stored in a common block, and must not be equivalenced.

- A pointer array cannot be stored in common.

- If a compiler supports NAMELIST as a FORTRAN 77 extension, a dummy argument and a pointee can not be placed in a namelist.

- A dummy argument with non-constant bound, a variable with nonconstant character length, an automatic object, a pointer, a variable of a type that has a pointer, or allocatable variable, can not be placed in a namelist.

- An equivalence object must not have the TARGET attribute or be a pointee.

- An object, imported from a (sub)module, must not be in EQUIVALENCE or COMMON.

220 E invalid initialization of entity in DATA or type statement

- In a blockdata program unit, only common-block variables can be initialized.

- A dummy procedure argument, automatic array, allocatable variable and pointee cannot be initialized in a DATA or type statement.

- In Fortran 90 a pointer can only be initialized with a pointer assignment, ALLOCATE or NULLIFY statement. From Fortran 95 pointer initialization is supported.

- A component with the ALLOCATABLE attribute can not be initialized by default.

- A variable in a pure procedure must be initialized other than by default.

221 E more than once in BLOCKDATA

- The common block has been specified in more than one block-data program unit.

222 W mixing of character and numeric types

- In standard Fortran it is not allowed to store character and numeric data in the same common block.

223 W initialization of named COMMON should be in BLOCKDATA

- Variables in a named common block should only be initialized in a blockdata program unit.

224 W invalid initialization of variable in blank COMMON

- Variables in blank common should not be initialized.

225 E more than once in COMMON

226 I objects not in descending order of type length

- This order could cause alignment problems on some processors.

227 I extension of COMMON

- This COMMON statement extends a previously declared common block with the same name.

228 W size of common block inconsistent with first declaration

- Named common blocks must have the same length in every occurrence. The length of the common block in this occurrence is different from that as specified in the main program or as specified in the first occurrence encountered.

229 W type in COMMON inconsistent with first declaration

- Numeric and character objects must not be stored in the same common block. The type of the objects in this occurrence of the common block is different from that in main or in the first occurrence encountered.

230 W list of objects in named COMMON inconsistent with first declaration

- In this occurrence of the named common block objects with different types, type lengths, or array sizes have been stored than in the main program or in the first occurrence encountered.

231 W array bounds differ from first occurrence

232 I only specified once

- The common block has been specified in one subprogram only.

233 I common block inconsistently included from include file(s)

- The common block has been specified in an include file at one occurrence and specified directly in another occurrence.
- The same common block has been specified in different include files.

234 E invalid equivalence with object in COMMON

- If more than one of the objects in an equivalence list is in a common block, the objects cannot be equivalenced.

235 E equivalence of variable to itself

- The equivalence lists are such that you try to equivalence an object to itself.

236 E storage allocation conflict due to multiple equivalences

237 I equivalence of arrays with possibly different type lengths

- When using short integers and/or logicals, this code may be highly non-portable.

238 E invalid storage association of object with a pointer component

- A variable of a derived type with pointer components must not be used in EQUIVALENCE or COMMON.

239 E invalid extension of COMMON through EQUIVALENCE

- An object in a common block is in such a way equivalenced with an array that storage must be allocated before the start of the common block.

240 W extension of COMMON through EQUIVALENCE

- An object in a common block is in such a way equivalenced with an array that the common block has to be extended.

241 W nonstandard mixing of types in EQUIVALENCE

- Character and numeric data must not be equivalenced.

- Objects of type character must be of the same kind.

- Objects of an intrinsic, non default kind, must be of the same type and kind.

- Objects of a sequence derived type that is not a numeric sequence or character sequence type, must be of the same type and have the same type parameter values.

242 E more constants than variables

- More constants than variables have been found in this data statement list.

243 E more variables than constants

- More variable elements than constants have been found in this data statement list.

244 E more than once initialized in DATA or type statement

245 E no expression allowed

246 E invalid type or type length for an integer POINTER

247 W assumed length character functions are obsolescent

- This is marked as obsolescent in the Fortran 95 standard.

248 I object already used, change statement sequence

- An explicit specification of an attribute confirms the attribute of an object that has already been used.

249 W list of objects in blank COMMON inconsistent with first declaration

- In this occurrence of the blank common-block objects with different types, type lengths, or array sizes have been stored than in main or in the first occurrence encountered.

250 I when referencing modules implicit typing is potentially risky

- There is an increased potential for undetected errors in a scoping unit that uses both implicit typing and the USE statement because module objects can be typed differently from the implicit type.

251 E SAVE has already been specified for each entity in this scoping unit

252 E a private object must not be placed in a public namelist group

- If a namelist-group-name has the PUBLIC attribute, no object in the namelist-group-object-list shall have the PRIVATE attribute or have private components.

253 W common-block data not retained: specify in root or save it

- The common block has not been SAVEd, has not been specified in the main program or in the root procedure of the referencing program units so the data become undefined after leaving the program unit.

254 W public module data not retained: specify in root or save it

- Not all public module data has been SAVEd, the module was not referenced in the main program or in the root procedure of the referencing program units so the data become undefined after leaving the program unit.

255 E derived type or structure undefined

- A variable of derived type is declared but the derived type has not been defined.

- A record is declared but the structure has not been defined.

- A parent type name shall be the name of a previously defined extensible type.

256 E statement invalid within derived type or structure definition

- This statement is not allowed within the definition of a derived type or structure.

257 E derived type or structure name missing

- The derived type name is missing in the type declaration

- The outer structure must have a name.

258 E invalid structure nesting

259 E missing END TYPE or END STRUCTURE

260 E missing END UNION

261 E missing END MAP

262 E invalid usage of record or aggregate field name

- A record must not be specified in an EQUIVALENCE, DATA, or NAMELIST statement.

- An aggregate field name is not allowed in formatted I/O.

263 E component or field name missing

- No derived type components or structure fields have been specified.

- A structure field which is a structure must have a field name.

264 E unknown component, field name, or type parameter

- A component or type parameter has been referenced which has not been declared in the derived type.

- A record field has been referenced which has not been declared in the structure.

265 E derived type must be of sequence type

266 E derived type or components must be PRIVATE

267 E no fields specified in structure definition

268 E incorrect number of component specs in structure-constructor

269 E malformed structure component

- At most one of the parts of a structure-component can be an array.

- A part-name to the right of an array must not have the POINTER attribute.

270 E derived-type component(s) or binding(s) inaccessible

- The component(s) or binding(s) of the derived-type are declared private.

271 E derived-type is inaccessible

272 E an object with a PRIVATE type cannot be PUBLIC

273 E invalid usage of structure-component or type-parameter

- A structure-component is not allowed in an EQUIVALENCE statement.

- The left side part of a structure must be of derived type.

- A type inquiry can not be defined.

274 E initialization of component or field not allowed

- In Fortran 90 initialization of derived-type components is not supported.

275 E derived-type object must be of sequence type or have the BIND attr.

- The derived-type of an object in COMMON or EQUIVALENCE must be of sequence type or have the BIND attribute.

- The type of a dummy argument must be of sequence type or have the BIND attribute if the type is defined in the local context.

- The type of an actual argument of an external procedure must be of sequence type or have the BIND attribute

276 I derived type or structure inconsistently included from include file

- The derived type or structure has been specified in an include file at one occurrence and specified directly in another occurrence.

- The same derived type or structure has been specified in different include files.

277 E component must be allocatable

279 E invalid usage of derived-type name

280 E no type parameter

281 E unknown type-bound procedure

282 E the parent type must be extensible

283 E invalid sequence of operators

284 I not allocated

- A conditionally referenced or defined allocatable variable was not allocated.

- An INTENT(IN) argument was not allocated.

285 E scalar integer constant expression expected

286 E undefined when entered through ENTRY, specify SAVE to retain data

287 E scalar integer constant name expected

288 E scalar integer variable name expected

- An integer which is not an array element, array name, constant, external, structure, derived-type name or namelist name is expected.

289 E scalar integer variable expected

290 E constant or scalar integer variable expected

291 E unsigned nonzero integer expected

292 E expression expected

293 E constant expression expected

294 E integer expression expected

295 E   scalar integer or real variable expected

296 E   NULL() or target expected

297 E   integer, logical, or character expression expected

298 E   integer or character expression expected

299 E   logical expression expected

300 E   character constant or unsigned integer constant expected

301 E   character expression expected

302 E   character substring must not be zero sized in this context

303 E   scalar logical expression expected

304 E   scalar integer expression expected

305 E   scalar integer or real expression expected

306 E   array expected

307 E   variable not defined

- The variable is referenced but has not been defined. No value has been assigned to the variable, to the elements of the array (if the variable is an array), or to the components (if the variable is of derived type), or the fields of a record.

- The variable is specified in a (sub)module but is not saved.

308 E   no statement label assigned to this variable

- The variable has been referenced as a label but no label has been assigned to the variable.

309 I   possibly no statement label assigned to this variable

- The variable has been referenced as a label but, if statements are executed sequentially, no label has been assigned to the variable. There might be, however, a path through which the variable is assigned before referenced.

310 I   label assigned to dummy argument or variable in COMMON

- It is unsafe and not functional to use a global variable to denote a label.

311 I   both a numeric value and label assigned to this variable

- The variable is used both to denote a label and a numeric value. This is potentially unsafe.

312 E   no value assigned to this variable

- The variable is referenced but no value has been assigned to the variable, an element of the array, a component of the structure, or a field of the record.

- The variable is a dummy output argument but no value has been assigned to it.

313   I   possibly no value assigned to this variable

- The variable has been referenced in an expression but, if statements are executed sequentially, no value has been assigned to the variable. There might be, however, a path through which the variable is defined before referenced.

- A dummy argument is referenced but it is not a dummy argument in all entries through which this statement can be reached.

314   I   possible change of initial value

- A variable has been initialized in a DATA statement or explicit type specification statement and a new value has been assigned to it. For a scalar of intrinsic type this means that the initial value has been superseded permanently. For an array or a variable of derived type this means that the value of one or more elements or components might have been superseded.

315   I   redefined before referenced

- A new value was assigned to the variable before it was referenced.

- The dummy argument is apparently an output variable while the last operation on the actual argument was an assignment.

316   W   not locally defined, specify SAVE in the module to retain data

- The variable is not defined in this program unit or in the module where it is declared. It could have been defined by another program unit using the module. In that case you must save the data in the module to preserve the data. From Fortran 2008 on module data are saved by default.

317   E   entity imported from more than one module: do not use

318   E   not allocated

- An allocatable variable must be allocated before being defined or referenced.

319   W   not locally allocated, specify SAVE in the module to retain data

- An allocatable variable must be allocated before being defined or referenced. The variable is not allocated in this program unit. It is use associated but not saved. From Fortran 2008 on module data are saved by default.

320   E   pointer not associated

321   I   pointer not associated

322   I   target not associated with a pointer

323 I variable unreferenced

- A variable has been defined but is not referenced.

324 I variable unreferenced as statement label

- A label has been assigned to this variable but the variable has not been referenced as a label.

325 I input variable unreferenced

- A variable which is defined by a READ, INPUT, or DECODE statement is not referenced.

326 I entity, declared in include file, not used

- An external, namelist, or local variable has been declared in an include file but is not used in the current subprogram.

327 E subscript out of range

328 I array, array extent, or character variable is zero sized

- The array extent is zero.

- The first bound of a specified shape is higher than the second bound.

- The first substring value is higher than the second.

329 E substring expression out of range

330 E invalid substring

331 E invalid usage of substring

332 W referenced character elements defined

- In Fortran 77 none of the character positions defined may be referenced in the same statement.

333 E division by zero

334 E invalid power execution

- It is invalid to raise a negative number to a real exponent.

335 E types do not conform

336 W typeless data used in invalid context

- Octal, hexadecimal and Hollerith data should only be used in DATA or PARAMETER statements

337 I implicit conversion to shorter type

- The type length of the variable is shorter than the resulting type length of the expression.

338 I character variable padded with blanks

339 E integer overflow in expression

340 I equality or inequality comparison of floating point data

- Because of limited precision and different implementations of real and complex numbers the result of this comparison may be unpredictable.

341 I eq. or ineq. comparison of floating point data with integer

- Because of limited precision and different implementations of real and complex numbers the result of this comparison may be unpredictable.

342 I eq.or ineq. comparison of floating point data with zero constant

- Because of limited precision and different implementations of real and complex numbers the result of this comparison may be unpredictable.

343 I implicit conversion of complex to scalar

- An integer or real is assigned to a complex variable.

344 I implicit conversion of constant (expression) to higher accuracy

- In an assignment statement precision is lost if the variable is of a more accurate type than the constant or constant expression.

- In a complex constant precision is lost if one of the components is of a less accurate type than the other.

- In an expression precision is lost if a constant is specified in a less accurate type than the resulting expression.

345 I implicit conversion to less accurate type

- Precision is lost due to conversion of real to real of less precision.

346 I implicit conversion of integer to real

347 I non-optimal explicit type conversion

- If the target of an expression is of type double precision real, best is to convert the expression primaries to double precision real explicitly, e.g. by specifying the kind type parameter.

- If the target of an expression is of type double precision complex, best is to convert the expression primaries to double precision complex explicitly, e.g. by specifying the kind type parameter.

348 E invalid usage of logical operator

349 E invalid usage of relational operator

350 E invalid mixed mode expression

351 E invalid usage of operator

352 W nonstandard operator

353 E undefined operator

354 E invalid concatenation with character variable of assumed length

- In Fortran 77 concatenation with a character variable of assumed length is only allowed in a character assignment statement.

355 E array-section specification invalid for assumed-shape array

- The second subscript of a subscript triplet of an array section must not be omitted for an assumed-shape array.

356 E array section specified incorrectly

357 E no array section allowed in this context

358 E invalid stride

359 E array has invalid rank

360 E each element in an array constructor must be of the same decl. type

361 E each element in an array constructor must have the same type length

362 E vector-valued subscript not allowed in this context

363 E array does not conform to expression, other arguments or target

- The rank or shape of the argument differs from that of the other arguments of the intrinsic procedure reference.

- The rank or shape of the expression differs from that of the left-hand side of an assignment statement.

364 E arrays do not conform

- The rank or shape of the operands in an expression differ.

365 E only nonproc.pointers and allocatable variables can be (de)allocated

366 E defined assignment not allowed in this context

367 E pointer assignment expected

368 E invalid usage of pointer assignment

369 E invalid assignment to pointer

370 E   invalid target for a data pointer

- the Object must have the POINTER or TARGET attribute to be assigned to a data pointer

371 E   only pointers can be nullified

372 E   target must have the same rank as the pointer

373 E   shape of variable differs from the shape of the mask expression

374 E   assignment of array expression to scalar

375 E   integer overflow in assignment

- The right-site expression yields a value which does not fit in the left-site target.

376 W   scalar integer variable name expected

- An integer which is not an array element, array name, constant, external, structure, derived-type name or namelist name is expected.

377 W   scalar integer expression expected

378 W   pointer not locally associated, specify SAVE in the module

- A pointer must be associated before being referenced. The pointer is not associated in this program unit. It is use associated but not saved. From Fortran 2008 on module data are saved by default.

379 E   invalid operation on a non-local variable in a pure procedure

- A global variable must not be modified in a pure procedure.

- Allocation, deallocation of global variables is not allowed in a pure procedure.

- pointer operations on global variables are not allowed in a pure procedure.

380 E   shape of mask expression differs from shape of outer WHERE construct

- If a WHERE construct contains a WHERE statement, a masked ELSEWHERE statement, or another WHERE construct then each mask expression shall have the same shape.

381 E   none of the equivalenced variables of the same type defined

- The variable is referenced but the variable and none of the equivalenced variables with the same type are defined.

382 I   none of the equivalenced variables of the same type referenced

- The variable is defined but the variable and none of the equivalenced objects with the same type are referenced.

383 I   truncation of character constant (expression)

- The type length of the variable is shorter than the resulting type length of the expression.

384 I truncation of character variable (expression)

- The type length of the variable is shorter than the resulting type length of the expression.

385 E invalid usage of construct name

386 E construct name expected

387 E non-matching construct name

- The construct name does not match the name of a construct.

388 E invalid construct nesting

389 E invalid statement in logical IF

- A statement in a logical IF must be executable, but no IF, ELSEIF, ELSE, DO, or END.

390 E ENTRY within an IF construct

391 E too many ENDIF's

392 E ELSE must be between IF and ENDIF

393 E missing ENDIF('s)

394 E THEN missing

395 E invalid sequence of ELSEIF and ELSE

396 E ELSEIF, ELSE, or ENDIF at invalid DO level

397 E more than one ELSE at this IF level

398 E invalid DO loop incrementation parameter

- The incrementation parameter of an (implied) DO loop is too small.

399 E invalid implied-DO specification

400 E invalid DO-loop specification

401 E terminal statement of loop at invalid IF level

402 E invalid terminal statement of DO construct

- A DO construct must end with an executable statement, but no IF, ELSEIF, ELSE, ELSEIF, DO, STOP, RETURN, or END.

403 E invalid transfer of control into construct

- A branch is detected which transfers control into a DO, an IF, CASE, WHERE, or FORALL construct

404 E referenced from outside construct

405 E redefinition of DO variable or FORALL index within construct

- A DO variable of an active loop is modified.

- An index name of a FORALL statement is modified in the forall statement or active FORALL construct.

406 E ENTRY within DO construct

407 E terminal statement of DO construct out of order

408 E missing terminal statement of DO construct

- No definition of the label of the terminal statement of the DO loop has been found.

- END DO missing

409 E missing END LOOP or UNTIL

410 E missing END WHILE or UNTIL

411 E too many END DO's, END LOOP's, or END WHILE's

412 E terminal statement of DO construct at invalid CASE level

413 W shared DO termination

- This syntax is marked as obsolescent in Fortran 90 and up.

414 E ENTRY within REMOTE BLOCK construct

415 E too many END BLOCKS

416 E missing END BLOCK ('s)

417 E ENTRY within a CASE construct

418 E type inconsistent with SELECT CASE expression type

419 E kind inconsistent with SELECT CASE expression kind

420 E invalid range of values specified

- A range of values of type logical cannot be specified

421 E overlapping CASE range

422 E CASE statement expected after a SELECT CASE statement

423 E a CASE statement must be within a CASE construct

424 E too many END SELECT's

425 E missing END SELECT ('s)

426 E only one CASE DEFAULT statement allowed in a CASE construct

427 E CASE, or END SELECT at invalid DO level

428 E CASE, or END SELECT at invalid IF level

429 E ELSEIF, ELSE, or ENDIF at invalid CASE level

430 E invalid statement after WHERE

431 E ENTRY within WHERE construct

432 E too many END WHERE's

433 E an ELSEWHERE must be within a WHERE construct

434 E missing END WHERE('s)

435 E too many END FORALL's

436 E missing END FORALL('s)

437 E reference of FORALL index in a forall triplet specification list

438 W obsolescent terminal statement of DO loop

- In Fortran 90 and up a terminal statement of a DO loop must be an END DO or a CONTINUE statement

439 E statement invalid within ASSOCIATE construct

440 E too many END ASSOCIATES's

441 E statement invalid within SELECT TYPE construct

442 E TYPE IS, CLASS IS, or CLASS DEFAULT at invalid DO level

443 E TYPE IS, CLASS IS, or CLASS DEFAULT at invalid IF level

444 E TYPE IS, CLASS IS, or CLASS DEFAULT at invalid CASE level

445 E only one CLASS DEFAULT statement allowed in a SELECT TYPE construct

446 E missing output item list

447 E invalid input/output list

448 W "," not allowed

- After a command-info list, no comma must be used.

- In an explicit type statement a comma may only be used in a CHARACTER statement after the length specification.

449 W invalid usage of parentheses

- Redundant parentheses are not allowed in an I/O list.

450 E invalid reference of standard unit

- OPEN, CLOSE, ENCODE, DECODE, BACKSPACE, REWIND is not possible on the standard unit.

451 W list directed I/O not allowed

- List directed I/O is only allowed for sequential I/O, and not on internal files.

452 E sequential formatted access expected

- Only sequential formatted I/O is allowed for internal I/O and I/O on the standard unit.

453 E invalid reference of internal file

- Only read and write operations can be performed on an internal file.
- The unit identifier must be a character variable, but not a constant or expression.

454 I possible recursive I/O attempt

- A function in which I/O may occur is referenced in an I/O statement.

455 W unrecognized or unsupported specifier

- An unsupported, nonstandard Fortran specifier has been detected.
- The specifier is not supported for this IO statement.

456 W nonstandard Fortran specifier

- One of the standard options is specified and the specifier is not in the Fortran standard.
- The specifier is an old, obsolescent, synonym for a standard specifier.

457 E more than once specified

- The specifier has already been specified in the list.

458 E invalid usage of specifier

- POS= only allowed for an external unit that is not specified by an asterix.
- ID= only allowed in combination with PENDING=
- If NEWUNIT= specified, FILE= or STATUS= must be present.

459 E no unit specified

460 E no unit or filename specified

461 E unit and filename specified

462 E invalid or missing io-unit identifier

- A unit identifier must be an asterix (standard unit), a positive integer expression, or a character variable.

463 E missing or invalid format specifier

- A format specifier must be: a label of a format statement, an integer variable to which a label of a format statement is assigned, a character expression containing the format specification, a non-character array name (language extension).

- In Fortran 90 a namelist group name must be specified with the NML= specifier.

464 W missing delimiter in format specification

465 E statement label expected

466 E more than once in OPEN, CLOSE, or INQUIRE list

- A variable or array element, or any associated entity, must not be both referenced and defined, or defined more than once in an OPEN, CLOSE or INQUIRE statement.

467 E "FMT=" or "NML=" expected

- When in a control-info list a keyword has been used, all specifiers from there on must be specified using keywords.

468 E "END=" only allowed in a sequential READ or WAIT statement

469 W "FILE=" not allowed for a scratch file

470 W "RECL=" only allowed for a direct access file

471 E "BLANK=" only allowed for a formatted file

472 E "ADVANCE=" only allowed for external formatted sequential i/o

- The ADVANCE= specifier may be present only in a formatted sequential input/output statement with explicit format specification and with no internal file unit specifier.

473 E "EOR=" and "SIZE=" only allowed in READ with "ADVANCE=NO" or WAIT

- The EOR= and the SIZE= specifiers are only allowed in an input statement that contains the ADVANCE= specifier with the value NO.

474 W no recordsize specified

476 E must be declared EXTERNAL

- The procedure name specified in "USEROPEN=" must have been declared EXTERNAL

477 E invalid combination of specifiers

- For namelist I/O no format must be specified.

- POS= and REC= must not be specified both.

478 E invalid usage of namelist name

- A namelist specifier is only allowed in sequential read and write statements on an external file.

479 E namelist name expected

480 E namelist i/o only allowed on an external file

481 I extension of previously defined namelist

- This NAMELIST statement extends a previously declared namelist with the same name.

482 E invalid type

483 W unrecognized value

- An unsupported, nonstandard Fortran value has been detected.

484 E invalid usage of value

485 W nonstandard Fortran value

486 E invalid repeat

- A nonzero, unsigned, integer constant is required.

487 E missing repeat

- A nonzero, unsigned, integer constant is required.

488 E invalid usage of repeat

489 E invalid usage of scale factor

- A scalefactor is only allowed for floating point edit descriptors.

490 W nonstandard edit descriptor

491 E missing or invalid width

- A nonzero, unsigned, integer constant is required.

492 E invalid edit descriptor

- No valid edit descriptor was detected.

493 E external i/o not allowed in a pure procedure

494 I namelist unreferenced

- A namelist has been specified but is never referenced (used).

495 I more than once in namelist group

496 E namelist group undefined

- A namelist group is referenced but it has not been specified.

497 E stream and async i/o only allowed on ext. files and not on * units

498 E namelist i/o only allowed for sequential i/o

499 E accompanying subprogram statement missing or incorrect

500 E no main program

- The complete option was specified but no main program is present.

501 I recursive reference

502 I possible recursive reference

503 E more than one main program

- A main program is a program unit of which the first statement is not a BLOCKDATA, SUBROUTINE, or FUNCTION statement. Therefor, besides of a program unit beginning with a PROGRAM statement, a main program will also be detected when e.g. two consecutive END statements have been specified.

504 E more than one unnamed BLOCKDATA

- Only one unnamed blockdata program unit is allowed.

505 E multiple definition of BLOCKDATA

- The name of the blockdata program unit has already been defined as the name of a blockdata program unit.

506 E multiple declaration of program unit or entry

- The name has been defined already before as a PROGRAM, SUBROUTINE, FUNCTION or ENTRY name.
- The name of a program, subroutine, function, or entry name has already been used.

507 E multiple declaration of statement function

508 I entries are not disjoint

- There could be transfer of control to the current or other entry blocks.

509 E no name specified

- A procedure, (sub)module or type name is expected.

510 E   multiple declaration of interface, this one ignored

511 E   explicit interface required

512 E   invalid subroutine or function reference

- A procedure reference is not allowed in this context.

- The function needs an explicit interface and must not be referenced in this context.

513 E   invalid usage of procedure name

- The name of the current subprogram or entry cannot be used as an actual argument.

- An internal procedure name cannot be used as an actual argument.

- A procedure name must not be specified in a type-declaration-statement with a language-binding.

514 E   subroutine/function conflict

- The procedure is referenced as a subroutine but has been referenced or defined as a function before.

- The procedure is referenced as a function but has been referenced or defined as a subroutine before.

515 E   invalid subprogram type

516 E   invalid usage of EXTERNAL

- A procedure name, as specified in an EXTERNAL statement, cannot be used at the left side of an assignment statement or as a statement function.

517 E   procedure actual argument must be declared EXTERNAL or INTRINSIC

- A procedure name, used as an actual argument, must be declared EXTERNAL or INTRINSIC.

518 I   referenced procedure not declared EXTERNAL

519 I   name of external procedure is same as module procedure name

520 E   external or dummy procedure expected

- The interface of the containing subprogram must not be specified in an interface block.

- The interface of an internal or module procedure must not be specified in an interface block.

521 E   invalid usage of generic name

- The generic name of a procedure cannot be used as an actual argument. Use the appropriate specific name.

522 E an interface with (module) procedure statements must be generic

523 E procedure already in list of specific procedures of this interface

524 W mixing of subroutines and functions in generic interface not allowed

- The Fortran standard does not allow to combine specific functions and subroutines in a generic procedure. Some compilers allow this as a syntax extension.

525 E defined operator procedure must be a function

526 E defined assignment procedure must be a subroutine

527 E no matching intrinsic or specific procedure found

528 I no procedure interfaces specified in interface block

529 E recursive reference

- A function is referenced recursively while recursive functions are not supported in the Fortran language level specified.

- A module is referenced circularly.

530 W possible recursive reference

- A path has been detected through which the procedure may reference itself.

531 I function is impure

- An argument and/or common-block object is being changed in this procedure.

- A local variable is saved.

- A non-local variable is changed in this procedure.

- A variable is initialized in a type or data statement.

532 E type conflict with type of function

- All entries within a character function must be of type character.

- The type specified when referencing the function differs from the specification of the function.

533 E type length conflict with type length of function

- All entries within function must have the same type length.

- The type length while referencing the function differs from the specification of the function.

534 E type of function inconsistent with first occurrence

- The type of the function differs from that at the first reference encountered.

535 E function type length inconsistent with first occurrence

  • The type length of the function differs from that at the first reference encountered.

536 I function type length inconsistent with first occurrence

  • The type length of the dummy function differs from that at the first reference encountered.

537 E shape of function reference differs from shape at first reference

538 E shape of function reference differs from shape of function result

539 E procedure must have private accessibility

  • If one or more of the dummy arguments or the function result is of private type the procedure must be private.

540 E multiple specification of prefix specification

541 E invalid combination of prefix specifications

  • A procedure cannot be specified elemental and recursive.

  • PURE and IMPURE cannot be specified both.

542 E procedure must be pure

  • Any procedure referenced in a pure subprogram, a forall statement, FORALL construct, or DO CONCURRENT construct shall be pure.

543 E invalid usage of prefix specification

544 E dummy argument of elemental procedure must be scalar

545 E dummy arg. of elemental proc. must not be a pointer or allocatable

546 E elemental procedure must be scalar

547 E elemental procedure must not be a pointer or allocatable

548 E dummy procedure argument not allowed in elemental procedure

549 I referenced intrinsic procedure not declared INTRINSIC

550 E invalid usage of alternate return

  • An alternate return is only allowed in a subroutine which is not elemental.

551 E invalid dummy argument list

552 E invalid usage of arguments

  • In an EXTERNAL or INTRINSIC specification a single procedure name without arguments is required.

- In a dummy argument list a dummy procedure must not have arguments.

- In the reference of an external procedure in USEROPEN no arguments are allowed.

553 E invalid usage of dummy argument

- The name of a dummy procedure argument has been used as the name of a statement function.

- A pointee cannot be a dummy argument.

554 E invalid dummy argument

- A dummy procedure argument cannot be a constant or expression.

555 E more than once in argument list

- A dummy argument is specified more than once in the dummy argument list.

- An argument keyword is specified more than once in the actual argument list.

556 I argument unreferenced in statement function

- A dummy argument of a statement function is not referenced in the statement function.

557 I dummy argument not used

558 E missing argument list

- In an expression or in an output statement a function must have an actual argument list. This argument list can be empty.

- In a FUNCTION statement an argument list is required. This list can be empty.

559 E argument missing, or no corresponding actual argument found

- A null argument is nonstandard Fortran.

- A non-optional actual argument is missing.

- No actual argument with the dummy argument keyword is found.

560 E incorrect number of arguments

561 E incorrect argument type

562 E incorrect argument attributes

563 E number of arguments inconsistent with first occurrence

- The number of actual arguments differs from that at the first reference encountered.

564 I number of arguments inconsistent with first occurrence

- The number of arguments of the dummy procedure differs from that at the first reference encountered.

565 E number of arguments inconsistent with specification

- The number of actual arguments differs from that in the specification of the procedure.

566 E argument keyword missing in actual argument list

- When in an argument list a keyword has been used, all subsequent arguments must be specified using keywords.

567 E argument keyword does not match a dummy argument

568 E argument class inconsistent with first occurrence

- The actual argument is a function, subroutine, external name, record, or label, but at the first reference encountered, the argument is of a different class.

569 I type inconsistent with first occurrence

- The actual argument of the dummy procedure is a function, subroutine, external name, record, or label, but at the first reference encountered, the argument is of a different class.

- The type of a common-block object differs from that in the first list encountered.

570 E argument class inconsistent with specification

- The actual argument is a function, subroutine, external name, or label, but in the specification of the procedure the argument is of a different class.

571 E argument type inconsistent with first occurrence

- The type of an actual argument differs from that at the first reference encountered.

572 W type inconsistent with first occurrence

- The type of an actual argument of the dummy procedure differs from that at the first reference encountered.

- The type of a common-block object differs from that in the first list encountered.

573 E argument type inconsistent with specification

- The type of an actual argument differs from that in the specification of the procedure.

574 E argument type inconsistent with first occurrence (int/log)

- The type of an actual argument differs from that at the first reference encountered. (Mixing of integer and logical types of equal lengths.)

575 I   argument type inconsistent with first occurrence (int/log)

- The type of an actual argument of the dummy procedure differs from that at the first reference encountered. (Mixing of integer and logical types of equal lengths.)

576 E   argument type inconsistent with specification (int/log)

- The type of an actual argument differs from that in the specification of the procedure. (Mixing of integer and logical types of equal lengths.)

577 E   argument type inconsistent with first occurrence (int/real)

- The type of an actual argument differs from that at the first reference encountered. (Mixing of integer and real types of equal lengths.)

578 I   argument type inconsistent with first occurrence (int/real)

- The type of an actual argument of the dummy procedure differs from that at the first reference encountered. (Mixing of integer and real types of equal lengths.)

579 E   argument type inconsistent with specification (int/real)

- The type of an actual argument differs from that in the specification of the procedure. (Mixing of integer and real types of equal lengths.)

580 E   argument type length inconsistent with first occurrence

- The type length of an actual argument differs from that at the first reference encountered.

581 I   type length inconsistent with first occurrence

- The type length of an argument of a dummy procedure differs from that at the first reference encountered.

- The type length of a common-block object differs from that in the first list encountered.

- The type length is explicit in one instance and implicit in another.

582 E   argument type length inconsistent with specification

- The type length of an actual argument differs from that in the specification of the procedure.

583 E   type of function argument inconsistent with first occurrence

- The type of a function actual argument differs from that at the first reference encountered.

584 I   type of function argument inconsistent with first occurrence

- The type of a function actual argument of the dummy procedure differs from that at the first reference encountered.

585 E argument type kind inconsistent with first occurrence

- The type kind of an actual argument differs from that at the first reference encountered.

586 I type kind inconsistent with first occurrence

- The type kind of an argument of a dummy procedure differs from that at the first reference encountered.

- The type kind of a common-block object differs from that in the first list encountered.

- The type kind is explicit in one instance and implicit in another.

- The type kind has been specified in one instance, the type length in the other.

587 E type of function argument inconsistent with specification

- The type of a function actual argument differs from that in the specification of the procedure.

588 E argument type kind inconsistent with specification

- The type kind of an actual argument differs from that in the specification of the procedure.

589 E shape of this argument must be supplied as argument

- Adjustable shapes must be specified in each entry in which the array occurs.

590 E array versus scalar conflict

- An actual argument is an array name while at a previous reference the argument is a scalar, or vice versa.

- An actual argument is an array name while the dummy argument is a scalar, or vice versa.

- An actual argument is an array element of an assumed-shape or pointer array while the dummy argument is an array.

591 I array versus scalar conflict

- The argument of a dummy procedure is an array name, while at a previous reference the argument was a scalar, or vice versa.

592 I arg. is an array element while it was an array in the previous ref.

593 I arg. is an array while it was an array element in the previous ref.

594 I the actual argument is an array element while the dummy is an array

595 I shape of argument differs from first occurrence

596 E shape of argument differs from specification

597 I shape of argument differs from specification

598 E actual array or character variable shorter than dummy

- The array or character datum as specified in the procedure is longer than the size specified the referencing program unit.

599 W array or character length differs form first occurrence

600 E attributes of argument inconsistent with first occurrence

601 E attributes of argument inconsistent with declaration

602 E invalid assignment: actual argument is constant or expression

- The dummy procedure argument is an output or input/output argument and will modify the actual argument.

603 E invalid assignment:variable more than once in actual argument list

- The variable occurs more than once in an actual argument list so it must not be modified. The dummy procedure argument is an output or input/output argument and will modify the actual argument.

604 E invalid assignment: the actual argument is an active DO variable

- The dummy procedure argument is an output or input/output argument and will modify the actual argument which is an active DO variable.

605 I possible invalid assignment: act.arg. is constant or expression

- The procedure might modify this argument.

606 I possible invalid assignment:var. more than once in act.arg.list

- The variable occurs more than once in the actual argument list and might be modified during the reference.

607 I possible invalid assignment: actual arg. is active DO variable

- The actual argument is an active DO variable and might be modified during the procedure reference.

608 I no INTENT specified, specify INTENT(IN) in the referenced subprogram

609 E dummy argument must not be OPTIONAL

610 E optional dummy argument unconditionally used

- An optional dummy argument may only be referenced, defined, allocated, or deallocated if it is present in the actual argument list of the referencing program unit, unless as an actual argument of a procedure reference if the corresponding dummy argument is also optional and not a pointer.

611 E   actual argument is an optional dummy argument,the dummy argument not

- The procedure is unconditionally referenced while the actual argument is an optional dummy argument of the referencing procedure which may not be present.

612 E   optional dummy argument expected

613 E   INTENT not allowed for pointer arguments

614 E   INTENT(IN) required for this dummy argument

- The arguments of a defined operator function must be defined INTENT(IN).

- The second argument of a defined assignment subroutine must be defined INTENT(IN).

- The arguments of a pure or elemental function must be defined INTENT(IN).

615 E   INTENT(OUT) or INTENT(INOUT) required for this dummy argument

- The first argument of a defined assignment subroutine must be defined INTENT(OUT) or INTENT(INOUT).

616 E   referenced input or input/output argument is not defined

- The argument was not defined when the procedure was referenced and not defined in the procedure before it was unconditionally referenced.

617 I   conditionally referenced argument is not defined

- The argument was not unconditionally defined when the procedure was referenced and it was not defined in the procedure before it was conditionally referenced.

618 I   possibly ref. input or input/output argument is possibly not defined

- The argument was not unconditionally defined when the procedure was referenced and not defined in the procedure before it was referenced.

- The argument was not defined when the procedure was referenced and was possibly not defined in the procedure before it was referenced.

622 E   dummy function must be specified as entry argument

- A dummy function must be specified in the argument list of each ENTRY statement from where the function is referenced.

623 I   intrinsic procedure is specific

- By referencing the generic intrinsic procedure instead, the code will be better readable, transportable and easier to adapt to different type parameters.

624 E conflict with intrinsic-procedure name

- A generic procedure has been referenced while the name of the generated specific procedure is already in use as a user defined, dummy, or statement function.

- The name of a common block must not be the name of an intrinsic procedure.

625 W nonstandard Fortran intrinsic procedure

626 E no intrinsic procedure

- A non-intrinsic procedure has been specified in an INTRINSIC statement.

627 E this intrinsic function is not allowed as actual argument

- The intrinsic functions to determine the minimum and maximum and the type conversion functions must not be passed as an argument.

628 E type conflicts with intrinsic function of the same name

- An intrinsic function has been generated or referenced while an intrinsic function with the same name and different data type has already been declared or used.

629 E invalid number of arguments for intrinsic procedure

630 E invalid argument type for intrinsic procedure

- The type of the argument of a specific procedure is incorrect.

- No specific procedure could be generated of which the argument type matches the actual argument type.

- A specific procedure has been generated with an argument type which matches the argument type of the first argument, but the type of (one of) the other arguments does not match.

631 E invalid argument type length for intrinsic procedure

- The type length of the argument of a specific procedure is incorrect.

- No specific procedure could be generated of which the argument type length matches the actual argument type length.

- A specific procedure has been generated with an argument type length which matches the argument type length of the first argument, but the type length of (one of) the other arguments does not match.

632 I intrinsic function is explicitly typed

- Intrinsic functions are implicitly typed and need not to appear in a type statement.

633 E   invalid usage of built-in function

  • This built-in function can only be used in an actual argument list.

634 E   invalid assignment, variable more than once in statement

  • If a variable occurs more than once in a statement it must not be modified during evaluation of the statement (Fortran 77). The dummy procedure argument is an output argument and will modify the actual argument.

635 I   possible invalid assignment:variable more than once in statement

  • The variable occurs more than once in the statement in which the procedure is referenced and might be modified during the reference (Fortran 77).

636 E   INTENT must be specified for this dummy argument

  • The intent of the arguments of a pure subprogram must be specified.

  • The intent of the arguments of an elemental subprogram that do not have the VALUE attribute must be specified.

637 E   specific procedure has no unique argument list

638 E   invalid redefinition of intrinsic operation or assignment

639 I   type is not the type of the generic intrinsic function

  • Specifying a type for a generic intrinsic function does not, in itself, remove the generic property from that function.

640 E   generic procedure reference could not uniquely be solved

641 E   argument must be an allocatable variable

642 E   argument must have the POINTER attribute

643 E   argument must have the POINTER or TARGET attribute

644 I   none of the entities, imported from the module, is used

645 E   module must not reference itself directly or indirectly

646 E   (MODULE NOT FOUND)

  • The module information is not found.

  • The library entry found is not a module.

647 E   multiple specification of (sub)module

  • A (sub)module with the same name has already been analyzed.

648 E   conflict between (sub)module and program unit or entry name

649 I module already referenced without only or rename list

650 E invalid rename clause

- No generic name, operator, or assignment expected.

- local_name=>module_name expected.

651 I already imported from host or same module

- The entity is in an ONLY list and has already been imported from the same module in the same or host scoping unit.

- The entity is already imported from the host scoping unit

652 I entity imported from more than one module: do not reference

653 E entity is not a public entity of the imported module

654 I (sub)module unused

- The complete option has been specified and the module is not imported in any of the analysed program units, or a submodule is not used.

665 I eq. or ineq. comparison of floating point data with constant

- Because of limited precision and different implementations of real and complex numbers the result of this comparison may be unpredictable.

666 E undefined operation

667 E undefined: dummy argument not in entry argument list

- The variable has been referenced but when entered through the previous ENTRY statement no value has been assigned to the variable.

668 I possibly undefined: dummy argument not in entry argument list

- The variable has been conditionally referenced but when entered through the previous ENTRY statement no value has been assigned to the variable.

669 I not locally associated, specify SAVE in the module to retain data

- A target must be associated with a pointer before being defined or referenced. The variable is not associated in this program unit and is use associated but not saved. From Fortran 2008 on module data are saved by default.

670 E actual argument must be a variable

- The dummy procedure argument is an output or input/output argument and could modify the actual argument.

671 E variable more than once in actual argument list

- The dummy procedure argument is an output or input/output argument and could modify the actual argument.

672 E active DO variable invalid for this actual argument

- The dummy procedure argument is an output or input/output argument and could modify the actual argument.

673 I not locally referenced

- The variable is not referenced in this subprogram. It could have been referenced by another subprogram using the module.

674 I procedure, program unit, or entry not referenced

- A procedure or program unit (entry) has been explicitly specified but is not referenced.

675 I named constant not used

- A named constant has been defined but is never referenced.

676 I none of the objects of the common block is used

677 I none of the objects of the common block is referenced

678 I none of the entities stored in the library file is used

679 I common-block object not used

680 I common-block object unreferenced

681 I not used

- An entity has been declared and possibly allocated, initialized or assigned, but is never used.

682 E procedure not defined

- The specified module procedure is not defined in the module.

683 E common-block object not defined before referenced

684 I common-block object possibly not defined before referenced

- The common-block object was conditionally defined.

685 I generic name was not needed to generate a specific procedure

686 E conflict with constant name

- The name of a common block must not be the same as the name of a constant.

687 E type length must be specified by a constant expression

- The type length of this object must be known at compile time.

688 E implicit characteristics are inconsistent with those in host context

- The type of the entity has been declared in the host scoping unit however, in the current scoping unit it appears to be a statement function. You must declare this entity locally.

- The type of the object has been declared in the host scoping unit however, in the current scoping unit it appears to be an EXTERNAL or INTRINSIC procedure. You must declare the entity in the host scoping unit as EXTERNAL or INTRINSIC.

689 I type length inconsistent with type length of function

- All entries within a function must have the same type length. One has the default length, the other has an explicitly specified type length.

- The type length while referencing the function is inconsistently specified compared to the specification of the function. One has the default length, the other has an explicitly specified type length.

690 I type length inconsistent with type length at first reference

- The type length while referencing the function is inconsistently specified compared to the first reference. One has the default length, the other has an explicitly specified type length.

- The type length of an actual argument is inconsistently specified compared to the first reference encountered. One has the default length, the other has an explicitly specified type length.

- The type length of a common-block object is inconsistently specified compared to the first reference encountered. One has the default length, the other has an explicitly specified type length.

691 I type length inconsistent with specification

- type length of an actual argument is inconsistently specified compared to the specification of the procedure. One has the default type length, the other has an explicitly specified type length.

692 E result of procedure must be scalar

693 E storage association conflict with object with the TARGET attribute

- An object with the TARGET attribute may become storage associated only with another object that has the TARGET attribute and the same type and type parameters.

694 E explicitness of dummy proc. argument inconsistent with first occurr.

- If the interface of a dummy procedure argument is explicit in one instance it must be explicit in each instance.

695 E no defined assignment supplied for this type

- If a defined assignment for one or more of the derived type components is present, you must supply a defined assignment for the type.

696 E entity is not an accessible entity in the host scoping unit

697 E name not explicitly typed, implicit type assumed

- The object has not been explicitly typed and:

- IMPLICIT NONE has been specified.

698 I implicit conversion to more accurate type

699 I implicit conversion of real or complex to integer

- Precision is lost due to conversion to integer.

700 E object undefined

- An attribute is specified for an object which has not been specified.

701 I type length of element inconsistent with first element

- The type length of this array element is inconsistently specified compared to that of the first element. One has the default length, the other has an explicitly specified type length.

702 E scalar default character expression expected

703 E a procedure cannot have the POINTER or TARGET attribute

704 E more than once in derived-type parameter list

705 E the VALUE attribute can not be specified for this object

- The VALUE attribute can only be specified for a scalar dummy argument

- The VALUE attribute can not be specified for a character datum with a length other then one.

706 E a protected object must not be changed outside its module

707 I module procedure not referenced from outside its module

- The module procedure can be declared private.

708 E END INTERFACE statement missing

709 E source expression not allowed for a typed allocation

- type-spec and a source expression cannot be specified both.

710 E only one source expression allowed in a sourced allocation

- SOURCE= and MOLD= cannot be specified both.

711 I declared RECURSIVE but not recursively referenced

712 E ancestor or parent (sub)module name missing

713 E interface name missing

714 I abstract interface not referenced

- An abstract procedure interface has been specified but it is not used.

715 E type-bound procedures not allowed in sequence or interoperable type

716 E a component cannot have the name of a type parameter

- KIND or LEN must be specified for a derived-type parameter declaration.

- Only KIND and LEN are valid derived-type parameter attributes.

717 E derived-type parameter not defined

- Each derived-type parameter must be specified with the KIND or LEN attribute.

718 E a CLASS component must be allocatable or a pointer

719 E a procedure component must be a pointer

720 E no components specified in derived-type definition

721 E no type-bound procedures specified

722 E external or module procedure expected

723 E type-bound procedure undefined

724 E DEFERRED attribute required

725 E DEFERRED attribute not allowed

726 E component keyword missing in structure-constructor

- When in a structure-constructor a keyword has been used, all subsequent components must be specified using keywords.

727 E keyword missing in type-param-spec-list

- When in a parameter list a keyword has been used, all subsequent parametes must be specified using keywords.

728 E incorrect, or missing language-binding-spec: BIND(C) expected

- the language-binding-spec must be BIND(C)

729 E no enumerators in enumeration

730 E END ENUM missing

731 E   interface name not allowed in this context

732 E   procedure attributes not allowed in this context

733 E   delimiter not allowed in this context

734 E   statement only allowed in a (non separate) interface body

735 E   explicit or abstract interface required

736 E   this intrinsic function not allowed as interface name

737 E   TYPE IS, CLASS IS, or CLASS DEFAULT expected after SELECT TYPE

738 E   associate name expected

739 E   association list missing

740 E   selector missing

741 E   invalid assignment

742 E   the selector must be polymorphic

743 E   passed-object dummy argument not found

744 E   incorrect number of derived-type parameters

745 E   invalid argument kind type parameter for intrinsic procedure

- The kind type parameter of the argument of a specific procedure is incorrect.

- No specific procedure could be generated of which the argument kind type parameter matches the actual argument type kind.

- A specific procedure has been generated with an argument kind type parameter which matches the argument type kind of the first argument, but the type kind of (one of) the other arguments do not match.

746 I   type kind or length inconsistently specified

- The type kind or length of the argument is explicit, the type kind or length of others is default, or specified as DOUBLE PRECISION.

- The type kind or length of this object in one instance of the common block is explicit, the type kind or length in the others is default, or specified as DOUBLE PRECISION.

747 E   each element in an array constructor must be of the same kind

748 I   element kind inconsistent with kind of first element

- The kind of this array element is inconsistently specified compared to that of the first element. One has the default kind, the other has an explicitly specified kind.

749 E   mixing of protected and non-protected objects in equivalence

750 W unsupported kind type parameter, default assumed

- The kind type parameter of this type is not supported by the emulated compiler.

751 W unsupported kind, default assumed

- No supported kind can be found that matches.

752 W unsupported character set

- No supported kind can be found for this character set.

753 E each element must have the same kind type parameters

754 E no objects to allocate or to deallocate

755 E unrecognized keyword

756 E type-spec or SOURCE= required

- One or more of the allocate-objects have deferred-type parameters

757 I no entities imported from module

758 E invalid target for a procedure pointer

759 E procedure already in list of final subroutines of this derived type

760 E final procedure has no unique argument list

761 E type parameter specified more than once or unknown

762 E empty parameter list

763 E deferred type parameter not allowed

764 E assumed type parameter not allowed

765 E each length type parameter must be assumed

766 E SEQUENCE type, or BIND attribute not allowed

767 E type must be an extension of the selector

768 E NOPASS must be specified

769 E passed-object argument required.

770 E argument must be a data-object

771 E derived type io procedure must be a subroutine

772 E type must be abstract

773 E argument must be scalar

774 E argument must be polymorphic

775 E   argument must not be polymorphic

776 E   the accessibility of the generic spec must be the same as originaly

777 I   the accessibility is inconsistently specified

778 E   types are not compatible

779 E   a CLASS entity must be dummy, allocatable or a pointer

780 E   entity is not accessible

781 E   entity must be interoperable

782 E   type kind conflict with type kind of function

- All entries within a function must have the same type kind.

- The type kind while referencing the function differs from the specification of the function.

783 E   function type kind inconsistent with first occurrence

- The type kind of the function differs from that at the first reference encountered.

784 I   type kind inconsistent with type kind of function

- All entries within a function must have the same type kind. One has the default kind, the other has an explicitly specified kind.

- The type kind while referencing the function is inconsistently specified compared to the specification of the function. One has the default kind, the other has an explicitly specified kind.

785 I   type kind inconsistent with type kind at first reference

- The type kind while referencing the function is inconsistently specified compared to the first reference. One has the default kind, the other has an explicitly specified kind.

- The type kind of an actual argument is inconsistently specified compared to the first reference encountered. One has the default kind, the other has an explicitly specified kind.

- The type kind of a common-block object is inconsistently specified compared to the first reference encountered. One has the default kind, the other has an explicitly specified kind.

786 I   type kind inconsistent with specification

- The type kind of an actual argument is inconsistently specified compared to the specification of the procedure. One has the default kind, the other has an explicitly specified type kind.

- The type kind has been specified in one instance, the type length in the other.

787 E invalid usage of abstract type

788 E invalid overriding of binding

789 E component name not unique

790 E component not defined

791 E the derived type must be extensible

792 E entity cannot be an explicit-shape array

793 E INTENT not allowed for nonpointer dummy procedure arguments

794 E entity cannot have the POINTER attribute

795 E entity cannot have the PROTECTED attribute

796 E dummy argument with assumed type parameter expected

797 E dummy argument must not be an elemental procedure

798 E invalid specification of shape

799 E named language binding not allowed

800 E multiple declaration of procedure

801 E derived-type name expected

802 E list of type-bound procedures not allowed

- In Fortan 2003 a list is not supported.

803 E invalid usage of unlimited format item

804 E scalar default integer or character constant expression expected

805 O could not determine type parameter

806 E invalid coarray specification

807 E ENTRY within a FORALL construct

808 E NULL() expected

809 E NULL() or procedure name expected

810 E TYPE IS, CLASS IS, or CLASS DEFAULT at invalid SELECT TYPE level

811 E invalid argument value

812 I derived-type component not used

- None of the objects of the type uses this component.

813 I derived-type component not referenced

- None of the objects of the type references this component.

814 I   derived-type component not defined

- None of the objects of the type defines this component.

815 I   derived-type component not allocated

- None of the objects of the type allocates this component.

816 I   derived-type component not associated

- None of the objects of the type associates this component.

817 E   incorrect type for a coarray

818 E   cannot extend parent type

819 E   nonpointer nonallocatable scalar expected

820 E   array with the POINTER attribute expected

821 E   target must be contiguous

822 E   missing coarray specification

823 E   function result cannot be a coarray

824 E   type of function result must not have a coarray ultimate component

825 E   a coarray must be a dummy argument, allocatable, in main, or saved

826 E   must be a dummy argument or saved

827 E   deferred-coshape specification not allowed

828 E   deferred-coshape specification required

829 E   array pointer or assumed-shape array expected

830 E   actual argument must be a contiguous array

831 E   entity cannot be a coarray

832 E   type not allowed for an INTENT(OUT) argument

833 E   a coarray cannot have the POINTER attribute

834 E   invalid usage of coindex or image selector

835 E   invalid number of cosubscripts

836 E   missing coshape specification

837 E   SAVE without entity list invalid in a BLOCK construct

838 I   input or input/output argument is not defined

- The argument was defined as an input or input/output argument and was not defined when the procedure was referenced.

- The argument was not or conditionally referenced before defined in the procedure and was not defined when the procedure was referenced.

839 E incorrect usage of coindexed object

840 E target has invalid rank

841 I module object not used outside the module

- The object can be decalred PRIVATE

842 E component must have the POINTER and/or ALLOCATABLE attribute

843 E statement not allowed within a CRITICAL or DO CONCURRENT construct

- A RETURN or an image control statement is not allowed within a CRITICAL or DO CONCURRENT construct

844 E no corresponding CRITICAL statement found

845 E missing END CRITICAL

846 E a coarray cannot not be (de)allocated within this construct

- A coarray cannot be (de)allocated within a CRITICAL or DO CONCURRENT construct

847 E invalid transfer of control out of construct

848 E invalid list of edit descriptors

849 E scalar character constant expression expected

850 E ancestor module must not be intrinsic

851 E module nature conflict

853 E (sub)module not found

854 E inconsistent attribute

855 E inconsistent dummy argument name

856 E inconsistent characteristics

857 I intrinsic module has the same name as a nonintrinsic module

858 I nonintrinsic module has the same name as an intrinsic module

859 I variable, used as actual argument, unreferenced

- The variable is defined through an actual argument in a referenced procedure but not referenced in the referencing program unit.

860 E scalar default character constant expression expected

861 E inconsistent BIND(C) attribute or binding label

- When a common blockor external procedure has been specified with the BIND(C) attribute in a certain subprogram, it must be specified with the BIND(C) attribute and the same binding label in every subprogram in which the common block or external procedure has been specified.

862 E binding label is not unique

863 E initialization expression expected

864 E an assumed-type entity must be a dummy variable

865 E an assumed-type variable can only be used as an actual argument

866 E an assumed-rank variable can only be used as an actual argument

867 E assumed-shape or assumed-rank argument expected

868 E assumed-rank entity must be a dummy variable

869 E invalid usage of procedure pointer

- A procedure pointer must not be dereferenced in an expression.

870 I dummy argument has no INTENT attribute

871 E INTENT(IN) dummy argument must not be modified

- The INTENT(IN) attribute for a non pointer dummy argument specifies that it shall not be changed during the execution of the procedure.

872 E INTENT(IN) dummy argument pointer must not be modified

- The INTENT(IN) attribute for a pointer dummy argument specifies that during the execution of the procedure its association shall not be changed.

873 I INTENT(OUT) dummy argument is not defined

874 I INTENT(OUT) dummy argument pointer is not associated or nullified

875 I INTENT(INOUT) dummy argument is not modified in this procedure

- The INTENT can be changed to INTENT(IN).

876 I INTENT(INOUT) pointer association is not modified in this procedure

- The INTENT can be changed to INTENT(IN).

877 I INTENT(INOUT) dummy argument is defined before referenced

- The INTENT can be changed to INTENT(OUT).

878 I INTENT(INOUT) dummy argument pointer is modified before referenced

- The INTENT can be changed to INTENT(OUT).

879 E an explicit RESULT variable must be declared for direct recursion.

880 E specification expression expected

881 E missing END ASSOCIATE('s)

882 E pointer association is not defined

883 E pointer association of one or more component(s) is not defined

884 I (SOURCE POSSIBLY IN FIXED FORM. DO NOT SPECIFY THE FREE-FORM OPTION)

885 E array element or scalar structure component expected

886 E expression in CASE statement not in range of selector

887 I array unreferenced

- An array has been defined but is not referenced.

888 I array not used

- An array has been declared and possibly allocated, initialized or assigned, but is never used.

Appendix E

References

1. American National Standard Programming Language FORTRAN, American National Standards Institute, Inc, X3.9-1978, New York, New York, 1978.

2. International Standard ISO/IEC 1539, Second edition 1991-07-01. Reference number ISO/IEC 1539 : 1991 (E), International Standards Organization, Geneva, 1991.

3. American National Standard Language Fortran 90, American National Standards Institute, Inc., X 3.198-1992, New York, 1992.

4. International Standard ISO/IEC 1539-1, Reference number ISO/IEC 1539-1 : 1997 (E), International Standards Organization, Geneva, 1997.

5. International Standard ISO/IEC 1539-1, Reference number ISO/IEC 1539-1 : 2004 (E), International Standards Organization, Geneva, 2004.

6. International Standard ISO/IEC 1539-1, Reference number ISO/IEC 1539-1 : 2010 (E), International Standards Organization, Geneva, 2010.

7. E.W.Kruyt, FORCHECK, A Fortran 77 Programming Aid, Proceedings of the Digital Equipment Users Society, pp 199-204, Hamburg, 1986.

8. PDP-11 FORTRAN, Language Reference Manual, Digital Equipment Corporation, AA-1855D-TC, Maynard, Massachusetts, December 1979.

9. PDP-11 FORTRAN-77, Language Reference Manual, Digital Equipment Corporation, AA-L979-TC, Maynard, Massachusetts, September 1981.

10. VAX FORTRAN, Language Reference Manual, Digital Equipment Corporation, AA-D034E-TE, Maynard, Massachusetts, June 1988.

11. VAX FORTRAN, User manual, Digital Equipment Corporation, AA-D035D-TE, Maynard, Massachusetts, June 1988.

12. FORTRAN for RISC, FORTRAN Language Reference Manual for RISC Processors, AA-NA31A-TE, Digital Equipment Corporation, 1989.

13. FORTRAN for RISC, Guide to FORTRAN Language Programming for RISC Processors, AA-NA30A-TE, Digital Equipment Corporation, 1989.

14. DEC Fortran, Language Reference Manual, AA-PU45A-TK, Digital Equipment Corporation, Maynard, Massachusetts, 1992.

15. DEC Fortran 90, Language Reference Manual, AA-Q66SB-TK, Digital Equipment Corporation, Maynard, Massachusetts, 1995.

16. Digital Fortran, Language Reference Manual, AA-Q66SC-TK, Digital Equipment Corporation, Maynard, Massachusetts, 1997.

17. Compaq Fortran, Language Reference Manual, AA-Q66SD-TK, Compaq Computer Corporation, Houston, Texas, 1999.

18. VS FORTRAN Version 2 Release 5: Language and Library Reference, IBM, fifth edition (august 1989), SC26-4221-5.

19. VS FORTRAN Version 2 Release 4: Programming Guide, IBM, fifth edition (august 1989), SC26-4222-4.

20. UNISYS OS1100 ASCII Fortran Programming Reference Manual Relative to Release Level 11R2 UP-8244.4 and UP-8244.4A Unisys Corporation, St. Paul, MN, December 1987.

21. Camilla B. Haase and Jerry W. Ornstein, Fortran 77 Reference Guide, Translator Family Release T1.0-21.0, DOC4029-5LA, Prime Computer Inc., January 1988.

22. CONVEX FORTRAN Language Reference Manual, Document No. 720-000050-203, Seventh Edition, CONVEX Computer Corporation, October 1988.

23. CONVEX FORTRAN User's Guide, Document No. 720-000030-203, Eighth Edition, CONVEX Computer Corporation, October 1988.

24. FORTRAN Version 1 for NOS/VE, Language Definition, Usage, Publication Number 60485913, Revision J, Control Data Corporation, 1988.

25. Domain Fortran, Language Reference, Document No. 000530-A01, Hewlett Packard Co., December 1990.

26. Sun FORTRAN Programmer's Guide, Part No: 800-2163-10, Revision A, Sun Microsystems Inc., 1988.

27. FORTRAN/9000 Reference, HP 9000 Series 300/400 Computers, HP Part Number B1688-90600, Hewlett-Packard Company, 1990.

28. HP Fortran 77/HP-UX Programmer's Guide, HP Part Number 92430-9004, Hewlett-Packard Company, 1988.

29. FORTRAN/9000 Reference, HP 9000 Series 700 Computers, HP Part Number B2408-90001, Hewlett-Packard Company, 1991.

30. Fortran 90 Programmer's Reference, HP Document Number B3908-90002, Hewlett-Packard Company, 1998.

31. RM/FORTRAN, Language Reference Manual (Version 2.4), Ryan-McFarland Corporation, 1987.

32. RM/FORTRAN, User's Guide, Version 2.4 (DOS), Ryan-McFarland Corporation, 1987.

33. IBM Personal Computer Professional FORTRAN Reference Manual, International Business Machines Corporation, first edition, 1984.

34. Microsoft FORTRAN Version 5.1 for MS OS/2 and MS-DOS Operating Systems, Reference, Document No. LN21013-0591, Microsoft Corporation, 1991.

35. Microsoft Fortran Power Station, Professional Development System, Version 1.0, for MS-DOS and Windows Operating systems, Language Guide, document No. DB38033-0293, Microsoft Corporation, 1993.

36. Microsoft Fortran Power Station, Version 4.0, Development System for Windows 95 and Windows NT workstation, Programmer's Guide, document No. DD64081-0995, Microsoft Corporation, 1995.

37. F77L, Fortran Language System, Reference Manual, Revision E, Lahey Computer Systems, Inc, August 1989.

38. F77L-EM/32, Fortran Language System, Reference Manual, Revision B, Lahey Computer Systems, Inc, June 1989.

39. Fortran 90, Language Reference, Revision B, Lahey Computer Systems, Inc, 1995.

40. Lahey/Fujitsu Fortran 95 Language Reference, Revision D, Lahey Computer Systems, Inc, 1998.

41. Lahey/Fujitsu Fortran 95 User's Guide, Lahey Computer Systems, Inc, 1998.

42. NDP Fortran, Reference Manual, MicroWay, Inc., Kingston, Massachusetts, USA, April 1990.

43. CF77 Compiling System, Volume 1: Fortran Reference Manual, SR-3071 4.0, Cray Research, Inc., Mendota Heights, Main, USA, June 1990.

44. CF90 Fortran Language Reference Manual, 1995, SR-3902, SR-3903, and SR-3905 2.0. Cray Research, Inc., Mendota Heights, Main, USA, June 1990.

45. David Bailey, David M. Vallance, Olga Vapenikova, Sara L. Pulford, FTN77/386 Reference Manual, The University of Salford, 1989.

46. FTN95 User's Guide, Salford Software Ltd, 1998.

47. XL Fortran for AIX, Language Reference, SC09-2348-00, IBM Corporation, June 1996.

48. XL Fortran for AIX, User's Guide, SC09-2349-00, IBM Corporation, June 1996.

49. Watcom FORTRAN 77, Language Reference, 5rd Edition, WATCOM International Corp., Waterloo, Canada, 1995, ISBN 1-55094-104-6.

50. Control Data 4000 Series. FORTRAN Programmer's Guide and Language Reference Manual. Publication Number 62940786. Control Data Corporation, Minneapolis, 1990.

51. Fortran 77 Language Reference Manual, document No. 007-0710-040, Silicon Graphics, Inc. Mountain View, California, 1991.

52. NagWare f90 Compiler (VMS), Release 2.0, The Numerical Algorithms Group Limited, Oxford, UK, 1993, ISBN 1-85206-098-0.

53. FORTRAN 77 for Windows 95/Windows NT, Reference Manual, Absoft Corporation, Rochester Hills, MI, USA, 1995.

54. MIPSpro Fortran 77 Language Reference Manual, Document Number 007-2362-003, Silicon Graphics, Inc., 1994-1996.

55. Fujitsu Fortran 90 User's Guide, September 1995, Part No: J2Z0-0080-01-EN, Fujitsu Open Systems Solutions, Inc, San Jose, CA, USA.

56. Intel Fortran, Programmer's reference, Version Number: FWL-700-04, Intel Corporation, USA, 2002

Appendix F

Glossary

active DO variable. A DO variable within the range of a DO loop.

actual argument. An expression, a variable, a procedure, or an alternate return specifier that is specified in a procedure reference.

aggregate field. A composite, or structured, data item, that is, a (Fortran 77 extension) record structure or a record substructure.

alphanumeric. A letter or a digit. As an extension the dollar sign is in some implementations considered a letter.

analysis message. An information, warning, or error message concerning the syntax or static semantics of the analyzed source program.

ANSI. American National Standards Institute.

argument. A parameter passed between a calling program unit and a procedure. It can be an actual argument or a dummy argument.

argument association. The relationship between an actual argument and a dummy argument during the execution of a procedure reference.

argument keyword. A dummy argument name which may be used in a procedure reference.

array. A set of scalar data, all of the same type and type parameters, whose individual elements are arranged in a rectangular pattern.

array element. One of the scalar data that make up an array. It is identified by the array name followed by a subscript indicating the position in the array.

array section. A subobject of an array consisting of a set of array elements.

assignment statement. A statement of the form 'variable = expression'.

association. Name association, pointer association, storage association, or inheritance association.

assumed-shape array A nonpointer dummy array that takes it shape from the associated actual argument.

assumed-size array A dummy array whose size is assumed from the associated actual argument. Its last upper bound is specified by an asterisk.

attribute A property of a data object that may be specified in a type declaration statement.

batch job. A number of commands placed in a file and submitted to be processed.

blank common. An unnamed common block.

block. A sequence of executable constructs embedded in another executable construct, bounded by statements that are particular to the construct, and treated as an integral unit.

block-data program unit. A program unit that provides initial values for data objects in named common blocks.

bounds. For a named array, the limits within which the values of the subscripts of its array elements must lie.

byte. A storage unit, generally consisting of eight bits, which can contain a single character.

call tree. See "reference structure".

character. A letter, digit, or other symbol.

character length parameter. The type parameter that specifies the number of characters for an entity of type character.

character string. A sequence of characters.

character storage unit. The unit of storage for holding a scalar that is not a pointer and is of type default character and character length one.

class. A set of types extended from a specific type.

collating sequence. An ordering of all the different characters of a particular kind type parameter.

command input. The entry of commands to instruct a program to perform the required actions.

command file. A file containing command input.

command file entry. The entry of commands through specification of a command file.

command line entry. The entry of commands through typing command lines.

common block. A block of physical storage that may be accessed by any of the scoping units in a program.

common-block object. An entity in a common block denoted by a name: a variable or record

(Fortran 77 extension).

common-block size. The number of bytes the common block will occupy.

compiler. A program that translates a program, written in a higher programming language, into code understood by the computer.

compiler directive. An instruction to the compiler to assist processing of source statements.

compile time. The time during which the compiler processes the source file.

complex constant. An ordered pair of signed or unsigned real or integer constants separated by a comma and enclosed in parentheses. The first constant of the pair is the real part of the complex constant, the second is the imaginary part.

complex type. An approximation of the value of a complex number, consisting of an ordered pair of real data items separated by a comma and enclosed in parentheses. The first item represents the real part of the complex number, the second represents the imaginary part.

component. A constituent of a derived type.

conditional compilation. Source code lines can be either included in the compilation process or be left out by applying a compiler directive and a command line option. The simplest compiler directive to tag lines to compile conditionally is a D in the first column of the source line.

configuration file. A file containing instructions to adapt a program to the user's requirements.

conformable. Two arrays are said to be conformable if they have the same shape. A scalar is conformable with any array.

conformance. A program conforms to the standard if it uses only those forms and relationships described therein, and if the program has an interpretation according to the standard. A program unit conforms to the standard if it can be included in a program in a manner that allows the program to be standard conforming.

constant. A data object whose value must not change during execution of a program. It may be a named constant or a literal constant.

constant expression. An expression satisfying rules that ensure that its value does not vary during program execution.

construct. A sequence of statements starting with an ASSOCIATE, DO, FORALL, IF, SELECT CASE, SELECT TYPE, or WHERE statement and ending with the corresponding terminal statement.

construct entity. An entity defined by a lexical token whose scope is a construct.

cross-reference table. A table in which all references to certain entities are listed.

data entity. An entity that has or may have a data value. It may be a data object, the result of the evaluation of an expression, or the result of a function reference.

data object. A data entity that is a constant, a variable, a record (Fortran 77 extension), or a subobject of a constant.

data type. See type. **debug line.** A source code line containing a character denoting conditional compilation in its first column.

default initialization. If initialization is specified in a type definition, an object of the type will be automatically initialized.

defined. For a data object, the property of having or being given a valid value.

deleted feature. A feature in a previous Fortran standard that is considered to be redundant and largely unused.

derived type. A type whose data have components, each of which is either of intrinsic type or of another derived type.

designator. A name, followed by zero or more component selectors, array section selectors, array element selectors, and substring selectors.

digit. One of the characters 0 to 9.

DO loop. A range of statements executed repeatedly by a DO statement.

double precision. The standard name for real data that is allocated two numeric storage units (8 bytes).

DO variable. A variable, specified in a DO statement that is initialized or increased prior to each execution of the statement or statements within the DO range.

dummy argument. An entity whose name appears in the parenthesized list following the procedure name in a FUNCTION, SUBROUTINE, ENTRY, or statement function statement (formal argument).

dummy array. A dummy argument that is an array.

dummy pointer. A dummy argument that is a pointer.

dummy data object. A dummy argument that is a data object.

dummy procedure. A dummy argument that is a procedure.

entity. The term entity is used for any of the following: a program unit, a procedure, an abstract interface, an operator, a generic interface, a common block, an external unit, a statement function, a type, a data entity, a statement label, a construct, or a namelist group.

entry. The location in the subprogram where execution of the statements starts when the entry name is referenced.

equivalence. The association of names referring to the same memory location.

equivalence list. A list of names to be associated.

executable statement. An instruction to perform or control one or more computational actions.

exit status. The resulting error level of the execution of a program.

explicit interface. For a procedure referenced in a scoping unit, the property of being an internal procedure, a module procedure, an intrinsic procedure, an external procedure that has an interface body, a recursive procedure reference in its own scoping unit, or a dummy procedure that has an interface body.

explicit type. The type of a name when specified by a type statement.

expression. A sequence of operands, operators, and parentheses. It may be a variable, a constant, a function reference, or may represent a computation.

extension. See Filename extension.

extent. The size of one dimension of an array.

external file. A sequence of records that exists in a medium external to the program.

external i/o. I/O operations performed on an external file.

external procedure. A procedure that is defined by an external subprogram or by means other than Fortran.

external subprogram. A subprogram that is not in a main program, module, or another subprogram.

field. An atomic unit of a record (Fortran 77 extension). It corresponds to a substructure, a variable or an array element.

file. An internal file or an external file.

file access type. The way an external file is accessed: sequential, direct, or stream.

file name extension. The denotation of a file type by extending the file name with a delimiter followed by a number of characters.

FORCHECK. A computer program to validate Fortran source programs through static analysis.

format type. The way the data is stored in an external file: formatted or unformatted. Formatted: stored as printable characters (e.g. ASCII or EBCDIC) Unformatted: stored in internal computer representation.

FORTRAN. An acronym of "Formula Translation" denoting a higher computer language.

FORTRAN 77. The American National Standard Programming Language FORTRAN, as specified by the American National Standards Institute in document X3.9-1978.

fortran 90. The Standard Programming Language Fortran, as specified by the ISO-1539:1991(E) document.

fortran 90. The Standard Programming Language Fortran, as specified by the ISO-1539-1:1997(E) document.

fortran-supplied procedure. See "intrinsic function".

function. A procedure that is invoked in an expression.

function result. The data object that returns the value of a function.

function subprogram. A sequence of statements beginning with a FUNCTION statement that is not an interface block and ending with the corresponding END statement.

generic identifier. A name that appears in an INTERFACE statement and is associated with all the procedures in the inerface block or that appears in a GENERIC statement and is associated with the specific type-bound procedures.

global entity. An entity identified with an identifier whose scope is a program.

global information. All information on global entities that is relevant to other program units of the program.

global Program Analysis. The analysis across program unit boundaries to verify the global entities.

hexadecimal constant. A literal constant that is represented by a sequence of digits and the letters A through F (base-16 notation).

hollerith constant. A string of any characters preceded by wH, where w is the number of characters in the string.

host. Host scoping unit.

host association. The process by which a contained scoping unit accesses entities of its host.

host scoping unit. A scoping unit that immediately surrounds another scoping unit.

identifier. See "Name".

implicit interface. A procedure referenced in a scoping unit other than its own is said to have an implicit interface if the procedure does not have an explicit interface there.

implicit Type. The default type of a name when no type has been specified by a type specification statement.

implied DO. An indexing specification (similar to a DO statement, but without specifying the word DO) with a list of data elements, rather than a set of statements, as its range.

include file. A file with statements that have to be included in the source code of the program at the place of the include statement which references the include file.

include path. A file directory at which the system tries to locate include files.

input record. A record of the input source file.

input file. A sequence of input records.

inquiry function. An function that is either intrinsic or is defined in an intrinsic module and whose result depends on properties of one or more of its arguments instead of their values.

intent. An attribute of a dummy data object that indicates whether it is used to transfer data into the procedure, out of the procedure, or both.

interface block. A sequence of statements from an `INTERFACE` statement to the corresponding `END INTERFACE` statement.

inter-subprogram information. All information on subprograms which is relevant to other program units of the program (global information).

interactive entry. Specification of program commands and options through a query.

interface of a procedure. See "procedure interface".

internal file. A character variable that is used to transfer and convert data from internal storage to internal storage.

internal i/o. I/O operations performed on an internal file.

internal procedure. A procedure that is defined by an internal subprogram.

internal subprogram. A subprogram in a main program or another subprogram.

intrinsic. An adjective applied to types, operations, assignment statements, procedures, and modules that are defined in the standard and may be used in any scoping unit without further definition or specification.

i/o. Pertaining to either input or output, or both.

i/o list. A list of items in an input or output statement specifying which data is to be read or to be written.

i/o operation code. A symbol denoting the category of input/output operation performed.

keyword. An argument keyword or a word with a special, predefined, meaning for the compiler.

kind type parameter. A parameter whose values label the available kinds of an intrinsic type, or a derived-type parameter that is declared to have the `KIND` attribute.

label. See "Statement label".

label type. The syntactic construct in which the statement label is used determines its type: end of a `DO` loop, identification of a `FORMAT` statement, or other.

labeled common. See "Named common".

length. Array length, character string length, type length, or record length.

length specification. The specification of the type length.

lexical token. A sequence of one or more characters with a specified interpretation.

library file. An external file consisting of an index and the global information on program units.

line. A sequence of characters containing (part of) Fortran statements, a comment, or an `INCLUDE` line.

list file. A sequential formatted file in which the numbered statements are presented with other information concerning the source code.

listing. See "List file".

literal constant. A constant without a name.

local entity. An entity identified by a lexical token whose scope is a scoping unit.

logical constant. A constant that can have one of two values: true or false.

logical expression. A combination of logical primaries and logical operators. The result is the value true or false.

logical operator. Any of the set of operators `.NOT.`, `.AND.`, `.OR.`, `.EQV.`, `.NEQV.`, `.XOR.`

logical primary. A primary that can have the value true or false. See also "primary".

main program. A program unit that is not a module, external subprogram, or block data program unit.

module. A program unit that contains or accesses definitions to be accessed by other program units.

module procedure. A procedure that is defined by a module subprogram.

module subprogram. A subprogram that is in a module but is not an internal subprogram.

name. A lexical token consisting of a letter followed by up to 62 alphanumeric characters (letters, digits, and underscores). Note that in Fortran 77 this was called a symbolic name.

named. Having a name.

named constant. A constant that has a name. Note that in Fortran 77 this was called a symbolic constant.

nonexecutable statement. A statement that describes the characteristics of the program unit, of data, of editing information, or of statement functions, but does not cause an action to be taken by the program.

nonstandard syntax. Syntax which does not conform to the Fortran standard.

numeric constant. A constant that expresses an integer, real, double precision, or complex

number.

numeric type. Integer, real, or complex type.

obsolescent feature. A feature that is considered to have become redundant but that is still in frequent use.

operation code. A symbol denoting the kind of operation performed on a data object.

operational message. A message presented to signal a problem in the operation of the program.

operand. An expression that precedes or succeeds an operator.

operation. A computation involving one or two operands.

operator. A lexical token that specifies an operation.

option. A sub-command to select program features.

output file. A sequential formatted file in which all information requested is stored.

parameter. See "argument".

path. A full file specification.

pointer. An entity that has the `POINTER` attribute.

pointer assignment. The pointer association of a pointer with a target by the execution of a pointer assignment statement or the execution of an assignment statement for a data object of derived type having the pointer as a subobject.

pointer associated. The relationship between a pointer and a target following a pointer assignment or a valid execution of an `ALLOCATE` statement.

pointer association. The process by which a pointer becomes pointer associated with a target.

primary. An irreducible unit of data; a constant, variable, function reference, or expression enclosed in parentheses.

procedure. A computation that may be invoked during program execution. It may be a function or a subroutine. It may be an intrinsic procedure, an internal procedure, an external procedure, a module procedure, a dummy procedure, or a statement function.

procedure interface. The characteristics of a procedure, the name of the procedure, the name of each dummy argument, and the generic identifiers (if any) by which it may be referenced.

program. A set of program units that includes exactly one main program.

program interface. The way to instruct the program to perform the required actions.

program unit. The fundamental component of a program. A sequence of statements, comments and INCLUDE lines. It may be a main program, a module, an external subprogram, or a

block data program unit.

qualifier. See "option".

rank. The number of dimensions of an array. Zero for a scalar.

real type. An arithmetic type, capable of approximating the value of a real number.

record. 1) A sequence of values that is treated as a whole within a file. 2) A named data entity, consisting of one or more fields, contained in the program (Fortran 77 extension).

record length. 1) The number of bytes or storage units that make up an entity in a file. 2) The number of bytes a record (Fortran 77 extension) occupies.

recursive reference. A subprogram is recursively referenced when the subprogram is invoked from within that same subprogram, either directly or via other subprograms.

reference structure. The hierarchical call tree in which all references of subprograms are presented graphically.

reference. The appearance of an object designator in a context requiring the value at that point during execution, the appearance of a procedure designator, its operator symbol, or a defined assignment statement in a context requiring execution of the procedure at that point, or the appearance of a module name in a USE statement.

relational expression. An expression that consists of an arithmetic expression, followed by a relational operator, followed by another arithmetic expression or a character expression, followed by a relational operator, followed by another character expression. The result is a value that is true or false.

relational operator. Any of the set of operators: `.GT.`, `.GE.`, `.LT.`, `.LE.`, `.EQ.`, `.NE.`

saved. Variables, records (Fortran 77 extension) and named common blocks can be saved by specifying them in a `SAVE` statement to prevent them from becoming undefined after exit of a subprogram.

scalar. A single datum that is not an array and is not a record (Fortran 77 extension) or aggregate field (Fortran 77 extension).

scale factor. A specification in a `FORMAT` statement, which changes the location of the decimal point in a real number.

scope. That part of a program within which a lexical token has a single interpretation. It may be a program, a scoping unit, a construct, a single statement, or a part of a statement.

scoping unit. One of the following:
A program unit or subprogram, excluding any scoping units in it,
a derived-type definition, or an interface body, excluding any scoping units in it.
.

scratch file. An external file in which temporary information is stored.

size. The size of an array, record (Fortran extension), derived type, or common block is the total number of bytes that make up the entity.

source code. The original text which forms FORTRAN statements.

source code listing. See "list file".

source file. A file containing the original text of a program.

source program. The original text which forms a FORTRAN program.

specific function. An Fortran supplied (intrinsic) function which can be referenced directly or by referencing a generic function which invokes the specific function depending on the type of the actual arguments.

specification statement. One of the set of statements that provides the compiler with information about the data used in the source program. It supplies the information required to allocate data storage.

standard conforming. See "conformance".

statement. A sequence of lexical tokens. It may consist of a single line, but can be continued using a continuation character, or can be limited to occupy part of a line by a separation character.

statement entity. An entity identified by a lexical token whose scope is a single statement or part of a statement.

statement function. A procedure specified by a single statement.

statement label. A lexical token consisting of up to five digits that precedes a statement and may be used to refer to the statement.

static analysis. The analysis of the source code without execution of the program.

static analyzer. A tool to perform static analysis.

static semantics. The meaning of the code as far as it can be directly inferred from the code without knowing the algorithm.

storage association. The relationship between two storage sequences if a storage unit of one is the same as a storage unit of the other.

string. A character literal constant.

stride. The increment specified in a subscript triplet.

structure. A scalar data object of derived type (Fortran 90, or 95), or a group of statements that define the form of a record (Fortran 77 extension).

structure component. The part of an object of derived-type.

subobject. A portion of a data object that may be referenced or defined independently of other

portions.

subprogram. A function subprogram or a subroutine subprogram. Note that in Fortran 77 a block data program unit was called a subprogram.

subroutine. A procedure that is invoked by a `CALL` statement or by a defined assignment statement.

subroutine subprogram. A sequence of statements beginning with a `SUBROUTINE` statement that is not in an interface block and ending with the corresponding `END` statement.

subscript. One of the list of scalar integer expressions in an array element selector. Note that in Fortran 77 the whole list was called the subscript.

subscript triplet. An item in the list of an array section selector that contains a colon and specifies a regular sequence of integer values.

substring. A contiguous portion of a scalar character string.

suffix. See File name extension.

symbolic constant. See "Named constant".

symbolic name. See "Name".

syntax. The lexical structure of the language.

system Message. A message presented to inform the user of a problem during execution of the program.

target. A data entity that has the `TARGET` attribute, or an entity that is associated with a pointer.

truncation. The implicit conversion of a type to another type which occupies less storage, or conversion of a representation of a real number to an integer.

type. A named category of data that is characterized by a set of values, together with a way to denote these values and a collection of operators that interpret and manipulate the values. The set of data values depends on the values of the type parameters.

type declaration. The specification of the type for the name of a constant, variable, or function by use of an explicit type specification statement.

type length. The number of bytes an object of a specific type occupies.

type parameter. A parameter of a data type.

type statement. A statement to specify the type of a name.

unassigned. See "Undefined".

undefined. The property of a data object of not having a determinate value.

unit identifier. A means of referring to a file in order to use input/output statements.

unreferenced. The condition of a data object that no reference is made to that object.

use association. The association of names in different scoping units specified by a USE statement.

variable. A data object whose value can be defined and redefined during the execution of a program. It may be a named data object, an array element, an array section, a structure component, or a substring. Note that in Fortran 77 a variable was always scalar and named.

vector subscript. A section subscript that is an integer expression of rank one.

whole array. A named array, or an array component of a structure with no subscript list.

Index