



Edge Coding Guidelines

Contents

1	Version Numbering	9
2	Tools	11
3	Rules	13
3.1	ISO/Ansi Standard Conformance	13
3.2	Filenames and Modules	13
3.3	Language Features	14
3.4	Other Syntactic/Semantic Requirements	15
3.5	Style Rules	15

Abstract

This document is part of Edge, a parallel CFD code developed by FOI, see [Eliasson \(2001\)](#) and [Edge homepage](#).

This document describes the conventions that will be used when implementing new features in Edge. Old code will be fixed to also comply with the rules defined here. This includes allowed syntactic and semantic language features, filename and module conventions, indentation conventions and more. The document also describes how the code can be checked to confirm that it complies with the coding guidelines.

1 Version Numbering

The release numbers follow the rule

Major.Patch.Internal

where the **Major** number is increased once a new major update is performed. The **Patch** number is increased when new features have been added and the **Internal** number is increased when a new patch for bug fixes etc is released.

Note that between **Internal** releases no changes to the Edge parameter file is allowed.

2 Tools

To verify that the source code complies with the rules, a combination of different tools are used. Some rules may be impossible to test completely, in those cases we must trust each programmer to do his best to inspect his own code before committing it to the revision control repository.

The meaning of the tool abbreviations used in the requirement tables in 3 are listed below.

ifort	<p>The intel 8.0 fortran compiler supports the following features for debugging:</p> <ol style="list-style-type: none">1. Array index bounds and character constant bounds may be checked at runtime.2. Conformance with fortran 95 and/or fortran 90 may be checked.3. Use of obsolescent features may be reported as compilation warnings.4. Unused variables may in some cases be reported as compilation warnings. <p>The compilation flags for these checks are <code>-check all -std95 -warn</code></p>
if95	<p>The Lahey Fujitsu fortran compiler supports the following features for debugging:</p> <ol style="list-style-type: none">1. Array index bounds and character constant bounds may be checked at runtime.2. Conformance with fortran 95 and/or fortran 90 may be checked.3. Unused variables and “dead code” may in some cases be reported as compilation warnings. <p>The compilation flags for these checks are <code>-chk a,e,s,u -warn -info --f95</code>.</p>
valgrind	<p>Valgrind is a freeware memory debugger (32-bit only) that can find some runtime errors:</p> <ol style="list-style-type: none">1. Use of uninitialized values/pointers in some cases.2. Write or read outside allocated memory blocks.3. Example: <code>valgrind --tool=memcheck executable arguments</code>
f95chk	<p>This tool is a script that covers many of the remaining rule checks. It is written in python.</p>
make	<p>Same as f95chk.</p>
chk-fix	
release	<p>Used before releasing a new version to check that version numbers etc have been updated.</p>

3 Rules

The coding standard is formulated as tables of rules. Each rule has a name in the margin representing the tool to be used to test the requirement and a column describing the requirement. There is also one column containing the number to be used as reference for the rule.

3.1 ISO/Ansi Standard Conformance

ifort	§1.1 Source code must comply with fortran 95 ISO/ANSI standard.
ifort	§1.2 Features obsolescent in fortran 95 are not allowed.
ifort	§1.3 Free form source code must be used.
N/A	§1.4 A C preprocessor will not be required to compile any file.

3.2 Filenames and Modules

N/A	§2.1 Fortran filename must have extension .f90 .
f95chk	§2.2 All subprograms (subroutines or functions) that will be used as external subprograms must be contained in a module. Each module must be contained in a file with the same name as the module (plus extension '.f90'). This implies that there can be only one module per file. The program unit where the external procedure will be used must include the corresponding module in a USE statement, preferably with an ONLY-list containing the names imported from that module. This rule increases type-safety and readability. It also makes it easier to write makefiles since the module files can be generated by a simple makefile rule. Example:

```
foo_m.f90
=====
MODULE FOO_M
CONTAINS
SUBROUTINE FOO(N)
  INTEGER N
  WRITE(*,*) N
END DO
END SUBROUTINE FOO
END MODULE FOO_M
```

```

something_else.f90
=====
PROGRAM BAR()
  USE FOO_M, ONLY : FOO
  CALL FOO(10)

```

3.3 Language Features

- f95chk** §3.1 Character string constants must not span several lines. Instead, use concatenation operator. Example:
instead of

```

WRITE(*,*) 'This is a continued &
           &line'

```

write

```

WRITE(*,*) 'This is a continued '//&
           'line'

```

- ifort** §3.2 No variables may be implicitly declared. Every program unit (function, subroutine or program) must contain an IMPLICIT NONE statement.

- f95chk** §3.3 All dummy arguments to functions and subroutines must include the INTENT clause to show if data is meant to go IN, OUT or INOUT. This improves readability and allows the compiler to optimize the code better, and to check for unintended usage. Example:

```

SUBROUTINE FOO(A,B,C)
  IMPLICIT NONE
  INTEGER, INTENT(IN) :: A
  INTEGER, INTENT(OUT) :: B
  INTEGER, INTENT(INOUT) :: C

```

- f95chk** §3.4 Shared DO-loop terminations and numbered DO-loops are not allowed. There is no need for this obscure syntax. Use END DO instead. If necessary, use named DO-loops instead. Example:
instead of

```

DO 10, I=1,100
  DO 10, J=1,100
    WRITE(*,*) XX(I,J)
10 CONTINUE

```

write

```

DO I=1,100
  DO J=1,100
    WRITE(*,*) XX(I,J)
  END DO
END DO

```

f95chk	§3.5	BLOCK DATA program units are not allowed. MODULE provides better functionality.
f95chk	§3.6	COMMON statements are not allowed. MODULE provides better functionality.
f95chk	§3.7	EQUIVALENT statements are not allowed. This is error-prone and may prevent optimization.
f95chk	§3.8	SEQUENCE statements are not allowed.

3.4 Other Syntactic/Semantic Requirements

valgrind	§4.3	All allocated memory must be deallocated at program termination. This makes it possible to find memory leaks with a memory debugger.
valgrind	§4.4	Read or write operations outside an allocated memory blocks are not allowed.
ifort,if95	§4.5	Read or write outside index bounds in arrays or character variables are not allowed.
valgrind(?)	§4.6	No uninitialized/undefined values may be used in a way that could affect the execution.
ifort	§4.7	Local variables that are not used must be removed. (Unused dummy arguments are allowed).
f95chk	§4.8	Pointer variables must be initialized with NULL() unless they are obviously initialized in some other way before they are used. The fortran 95 pointer initialization syntax is recommended. Example: instead of

```
REAL, POINTER :: A(:, :)
```

write

```
REAL, POINTER :: A(:, :)=>NULL()
```

3.5 Style Rules

f95chk	§5.1	Indentation will be two steps for each nested block-level. To save space, the outermost block content is never indented. Also, within a module, the outermost subroutine/function blocks are not indented. Example:
---------------	------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

```
MODULE FOO_M
  SUBROUTINE FOO()
    INTEGER N
    DO N=1,10
      WRITE(*,*) N
    END DO
  END SUBROUTINE FOO
END MODULE FOO_M
```

f95chk §5.2 Short comments will use a single '!' at the end of the line. Longer comments will be written at the beginning of a line (column 1) with one empty '!' above and one below the actual comment text. The comment text will start in column 3. Example:

```

      INTEGER I      ! This is a short comment.
      IF (N .EQ. 0) THEN
        I=1
      !
      ! This is a more in-depth comment.
      !

```

f95chk §5.3 Every block must be terminated by an END BLOCK statement (where BLOCK is replaced by the appropriate block type). If the block has a name it must also be appended (END BLOCK NAME).

none §5.4 Indentation convention is up to each programmer. However, if a file uses a certain convention any changes to that file must follow these conventions so that each file has the same indentation convention everywhere.

none §5.5 At the beginning of each program unit (module, program, subroutine or function) there must be a comment header describing the purpose of this code. All comments that come immediately on or after the line defining the program unit will be used to generate source code documentation. Dummy arguments to subroutines/functions are also documented the same way. Example:

```

      SUBROUTINE SOLVE(I,R)
      !
      ! This comment will appear in the documentation for SOLVE
      !
      IMPLICIT NONE
      INTEGER, INTENT(IN) :: I ! This short comment documents I
      REAL, INTENT(INOUT) :: R(:, :)
      !
      ! This comment will document argument R
      !

```

References

EDGE HOMEPAGE <http://www.edge.foi.se>.

ELIASSON, P. 2001 EDGE, a Navier–Stokes Solver for Unstructured Grids. Scientific Report FOI-R--0298--SE. Computational Aerodynamics Department, Aeronautics Division, FOI.

FOI is an assignment-based authority under the Ministry of Defence. The core activities are research, method and technology development, as well as studies for the use of defence and security. The organization employs around 1350 people of whom around 950 are researchers. This makes FOI the largest research institute in Sweden. FOI provides its customers with leading expertise in a large number of fields such as security-policy studies and analyses in defence and security, assessment of different types of threats, systems for control and management of crises, protection against and management of hazardous substances, IT-security and the potential of new sensors.



FOI
Swedish Defence Research Agency
SE-164 90 STOCKHOLM

Tel: + 46 8 555 03000
Fax: + 46 8 555 03100

www.foi.se