

MISRA-C Compliance Matrix Using PC Lint

by

Eur Ing **Chris Hills** BSc(Hons), C. Eng., MIEE, FRGS

Revision 0.2
23 December 2001

Part of the **QuEST** series:- **QA4**



Hitex (UK) Ltd.
Warwick Uni Science Park
Coventry, CV4 7EZ
www.hitex.co.uk
chills@hitex.co.uk



chris@phaedrus.org
quest.phaedrus.org

MISRA-C Compliance Matrix Using PC Lint

by

Eur Ing **Chris Hills** BSc(Hons), C. Eng., MIEE, FRGS

Revision 0.2
23 December 2001

The copies of this paper (and subsequent versions) and any power point slides will be available or <http://quest.phaedsys.org> the authors personal web site. Quest@phaedsys.org

This paper will be developed further.

The ART in Embedded Engineering
comes through good
engineering discipline.

Quality Embedded Software Techniques

QuEST is a series of papers based around the theme of *Quality Embedded Software*. Not for a specific industry or type of work but for all embedded C. It is usually faster, more efficient and surprisingly a lot more fun when things work well.

QuEST Series

- QuEST 0** Introduction & SCIL-Level
- QuEST 1** Embedded C Traps and Pitfalls
- QuEST 2** Microcontroller Debuggers
- QuEST 3** Advanced Embedded Testing For Fun

Additional Information

- QA1** SCIL-Level
- QA2** Tile Hill Style Guide
- QA3** QuEST-C
- QA4** PC-Lint MISRA-C Compliance Matrix

Contents

0	MISRA-C Compliance Matrix.....	7
1	PC-Lint & MISRA-C Checking.....	9
2	A Compliance Matrix Using PC-Lint.....	11
3	PC-Lint MISRA-C Rule checking statistics.....	17
4	PC-LINT MISRA-C Rule Enforcement.....	19
5	References.....	31

0 MISRA-C Compliance Matrix

Version 0.2 23 December 2001

Since its publication in 1998 MISRA-C has gained an unprecedented level of acceptance and use. Not only in the automotive business but in all manner of embedded systems across the world. This is partly due to the fact that working Engineers who wanted something they could read easily wrote it with that in mind.

However, the ease of use of the rules in chapter 7 of the guide has somewhat over shadowed the information in the first six chapters. There is a great deal of useful information in these chapters, not least on the use and implementation of the rules

Chapter 5 is entitled "Using MISRA-C" It lists sections entitled:

The Software Engineering Context	(See QuEST Vol. 1)
Style Guide	(See QuEST QA2)
Tool Selection & Validation	(See QuEST Vol. 3 & 4)
Source complexity Metrics	(See QuEST Vol. 1)
Test Coverage	(See QuEST Vol. 4)

It also has a section (5.3) entitled "Adopting the subset". The first sub-section (5.3.1) concerns the Matrix. This is quite simply a chart containing a line for each rule and several columns one for each tool used in the process. Where a rule is enforced or checked this is entered in the box. The Example gives the "message number"

The idea is not to have a tick in every box in the column but at least one tick in each row. That is to say each rule is checked at least once somewhere in the process.

During the work by the MISRA-C Working Group (of which the author is a member) It has been determined that not all the rules are mechanically enforceable. Some are only possible to check by code review. Some rules are mutually exclusive. Therefore at the current time it is not possible to claim 100% MISRA-C compliance. However the Working Group is working on it.

If you have any comments, spot any errors etc please email the author at <mailto:quest@phaedsys.org>

The charts on the following pages list, for each MISRA-C Rule whether it is Advisory or Required followed by where the rule is checked. In this Matrix here are three options,:

- The Static Analyser
- The Compiler
- Manual Checking (code review)

The exact level of compliance depends on the tools used. In this case the static analyser. PC-Lint from Gimpel (www.gimpel.com/in the USA and www.hitex.co.uk/in Europe)

This chart will be updated as Pc-Lint and MISRA-C are updated.

1 PC-Lint & MISRA-C Checking

It was never intended that the MISRA-C rules would be 100% static detectable. Rules, for example like Rule 2. In fact, some such as Rule 4 “*Provision should be made for appropriate run-time checking*” are only testable by manual code review and human judgment (probably a majority vote at that!). Some rules like Rule 14 would require the re-writing of standard library headers and functions. Rule 1 is not possible in most 8 and 16 bit embedded systems anyway. Therefore, no tool can claim 100% MISRA-C compliance.

The “top end” tools costing several orders of magnitude more than PC-Lint suggest that they can hit 85% of the [required] rules where PC-Lint hits 82%. Any tools or vendors claiming much more than 85% of all 127 rules should be regarded with suspicion. Like PC-Lint, any tool you use should be able to show you **how** it is testing each rule. Do not settle for anything less.

At the time of writing (December 2001) Chris Hills, a Technical Specialist at Hitex UK, who is on the MISRA-C working Group has said that “***Whilst the MISRA-C Guide is a lot better than anything else out there for the average Engineer, it still has a lot of ambiguities that are a problem to tool vendors. It is for this reason that in 2001/2002 the MISRA-C working Group will be clarifying The Rules and producing example test cases for them.***”

This is why there is currently (December 2001) no definitive test suite available or compliance certification for MISRA-C. It is highly unlikely that there will be either before early 2003.

2 A Compliance Matrix Using PC-Lint

MISRA Rule	Required Advisory	Tools			Tool checked	REQUIRED NOT CHECKED
		PC-Lint	Compiler	Manual		
1	R	Yes			Yes	
2	A			YES	NO	
3	A	Yes +			Yes	*+
4	A			YES	NO	
5	R	Yes			Yes	
6	R	*****	YES		YES	
7	R	Yes			Yes	
8	R	*****	YES		YES	
9	R	Yes			Yes	
10	A			YES	NO	
11	R	Yes			Yes	
12	A	Yes			Yes	
13	A	Yes			Yes	
14	R	Yes			Yes	
15	A			YES	NO	
16	R			YES	NO	*****
17	R	Yes			Yes	
18	A	Yes			Yes	
19	R			YES	NO	*****
20	R	Yes			Yes	
21	R	Yes +			Yes	*+
22	A	Yes			Yes	
23	A	Yes			Yes	
24	R	Yes			Yes	
25	R	Yes			Yes	

Yes+ == no Specific MISRA-Rule Message

***** == REQUIRED RULE Not checked by PC-Lint Required manual checking

MISRA Rule	Required Advisory	Tools			Tool checked	REQUIRED NOT CHECKED
		PC-Lint	Compiler	Manual		
26	R	Yes			Yes	
27	A	Yes +			Yes	
28	A	Yes +			Yes	
29	R	Yes			Yes	
30	R	Yes			Yes	
31	R	Yes			Yes	
32	R	Yes +			Yes	*+
33	R	Yes +			Yes	*+
34	R			YES	NO	*****
35	R	Yes			Yes	
36	A			YES	NO	
37	R	Yes			Yes	
38	R	Yes			Yes	
39	R	Yes			Yes	
40	A	Yes +			Yes	
41	A			YES	NO	
42	R	Yes			Yes	
43	R	Yes			Yes	
44	A	Yes +			Yes	
45	R	Yes			Yes	
46	Y	Yes			Yes	
47	A	Yes +			Yes	
48	A	Yes +			Yes	
49	A	Yes +			Yes	
50	R	Yes			Yes	

Yes+ == no Specific MISRA-Rule Message

***** == REQUIRED RULE Not checked by PC-Lint Required manual checking

MISRA Rule	Required Advisory	Tools			Tool checked	REQUIRED NOT CHECKED
		PC-Lint	Compiler	Manual		
51	A	Yes			Yes	
52	R	Yes			Yes	
53	R	Yes			Yes	
54	R	Yes			Yes	
55	A	Yes +			Yes	
56	R	Yes			Yes	
57	R	Yes +			Yes	*+
58	R	Yes +			Yes	*+
59	R	Yes +			Yes	*+
60	A	Yes +			Yes	
61	R	Yes			Yes	
62	R	Yes			Yes	
63	A	Yes +			Yes	
64	R	Yes			Yes	
65	R	Yes +			Yes	*+
66	A			YES	NO	
67	A			YES	NO	
68	R	Yes +			Yes	*+
69	R	Yes			Yes	
70	R		Note 1			*****
71	R	Yes			Yes	
72	R	Yes			Yes	
73	R	Yes +			Yes	*+
74	R					*****
75	R	Yes			Yes	

Yes+ == no Specific MISRA-Rule Message

***** == REQUIRED RULE Not checked by PC-Lint Required manual checking

Note 1

Some embedded compilers do not permit recursion.

MISRA Rule	Required Advisory	Tools			Tool checked	REQUIRED NOT CHECKED	
		PC-Lint	Compiler	Manual			
76	R	Yes +			Yes	*+	
77	R	Yes			Yes		
78	R	Yes			Yes		
79	R	Yes			Yes		
80	R	Yes			Yes		
81	A			YES	NO	*+	
82	A			YES	NO		
83	R	Yes			Yes		
84	A	Yes			Yes		
85	A			YES	NO		
86	A			YES	NO		
87	R	Yes +			Yes		
88	R	Yes			Yes		
89	R	Yes			Yes		
90	R			YES	NO		*****
91	R	Yes +			Yes	*+	
92	A	Yes			Yes		
93	A			YES	NO		
94	R	Yes			Yes		
95	R	Yes			Yes		
96	R	Yes			Yes		
97	A	Yes			Yes		
98	A	Yes +			Yes		
99	R			YES	NO		*****
100	R	Yes +			Yes		

Yes+ == no Specific MISRA-Rule Message

***** == REQUIRED RULE Not checked by PC-Lint Required manual checking

MISRA Rule	Required Advisory	Tools			Tool checked	REQUIRED NOT CHECKED
		PC-Lint	Compiler	Manual		
101	A			YES	NO	
102	A			YES	NO	
103	R	Yes			Yes	
104	R			YES	NO	*****
105	R			YES	NO	*****
106	R	Yes			Yes	
107	R	Yes			Yes	
108	R	Yes			Yes	
109	R		YES	YES	YES	*****
110	R	Yes +			Yes	*+
111	R	Yes			Yes	
112	R	Yes			Yes	
113	R			YES	NO	*****
114	R	Yes			Yes	
115	R			YES	NO	*****
116	R			YES	NO	*****
117	R	Yes ***			Yes ***	***
118	R	Yes			Yes	
119	R	Yes			Yes	*+
120	R	Yes			Yes	*+
121	R	Yes			Yes	
122	R	Yes			Yes	
123	R	Yes			Yes	
124	R	Yes			Yes	
125	R	Yes			Yes	
126	R	Yes			Yes	
127	R	Yes			Yes	

Yes++ can be checked but requires specific setting up with Pc-Lint

Yes+ == no Specific MISRA-Rule Message

***** == REQUIRED RULE Not checked by PC-Lint Required manual checking

3 PC-Lint MISRA-C Rule checking statistics

As of December 2001 PC-Lint tests 74% overall and 82% of the "Required" rules, which is about average among the current crop of MISRA-C checking, tools.

	#	of total	percentage
Required	76	of 93	82%
Advisory	18	of 34	53%
Total	94	of 127	74%

There are several "Required" rules, 12 by my count, that are not detected by PC-Lint or likely to be seen by the compiler that will have to be manually checked in code review.

Therefore, Code reviews should be specifically tasked with looking for these problems. A Style Guide (such as the Tile Hill Embedded C Style Guide) can be used to make this job easier.

4 PC-LINT MISRA-C Rule Enforcement

The text below is the (slightly) more readable form of the au-misra.lnt file used for checking the MISRA rules. It shows how each rule is tested.

Note: Using +e834 activates additional messages regarding rule 46 and 47. Using +e829 and the -headerwarn() options activate additional messages regarding rules 121 and 124. Using +e960 activates additional messages regarding rules 19, 32, 33, 42, 54, 57, 58, 59, 65, 68, 69, 73, 87, 88, 91, 98, 100 and 110. Using +e961 activates additional messages regarding rules 18, 28, 40, 44, 47, 55, 60, 63 and 92. When using PC-lint, although the options +e421, +e578 +e834, +e829, +e960 and +e961 need be used once, they are repeated here to demonstrate the reach of each option. Furthermore, semantic specification may assist rule 117 compliance.

Rule 1 (req) -A strict ISO

+e950

-append(950,[MISRA Rule 1])

Rule 2 (adv)

Rule 3 (adv)

-rw(asm, __asm, __asm) remove asm built-in's

-dasm=_ignore_init define asm as a ...

+rw(_ignore_init) function introduction

Rule 4 (adv)

Rule 5 (req)

+e606 non-ANSI escape sequence

-append(606,[Misra Rule 5])

Rule 6 (req)

Rule 7 (req)

-ftg inhibit use of trigraphs

+e739 activate trigraph in string message

-append(739,[Misra Rule 7])

Rule 8 (req) -fwc deactivate wchar_t no option to detect L"..."

Rule 9 (req)

-fnc inhibit nested comments

+e602 comment within comment

-append(602,[Misra Rule 9])

Rule 10 (adv)

Rule 11 (req)

-idlen(31) report on names identical in the first 31 characters
+e621 Identifier clash within n characters - length set by -idlen
-append(621,[Misra Rule 11])

Rule 12 (adv)

+e578 enable reports of name hiding
+e580 enable reports of name hiding
-append(578,[Misra Rules 12, 21, and 27])
-append(580,[Misra Rule 12])

Rule 13 (adv)

+e970
-append(970,[MISRA Rule 13])

Rule 14 (req)

+e971
-append(971,[MISRA Rule 14])

Rule 15 (adv)

Rule 16 (req)

Rule 17 (req)

+e623 redefining the storage class of symbol
-append(623,[Misra Rule 17])

Rule 18 (adv)

+e912
-append(912,[MISRA Rules 18 and 48])
+e919
+e915
-append(919,[MISRA Rule 18])
-append(915,[MISRA Rule 18])
+e961

Rule 19 (req)

Rule 20 (req)

+e40 undeclared identifier
-append(40,[MISRA Rule 20])
+e718 Symbol undeclared
-append(718,[Misra Rules 20 and 71])
+e746
-append(746,[Misra Rules 20 and 71])

Rule 21 (req)

+e578 Declaration of Symbol hides Symbol

Rule 22 (adv)

+e956

-append(956,[MISRA Rule 22])

Rule 23 (adv)

+e765 symbol previously used as static

-append(765,[Misra Rule 23])

Rule 24 (req)

+e512 symbol previously used as static

-append(512,[Misra Rule 24])

Rule 25 (req)

+e14 symbol previously defined

-append(14,[Misra Rule 25])

Rule 26 (req)

+e15 symbol redeclared

-append(15,[Misra Rule 26])

Rule 27 (adv)

+e578 Declaration of Symbol hides Symbol

Rule 28 (adv)

+e961

Rule 29 (req)

+e64

-append(64,[MISRA Rule 29])

Rule 30 (req)

+e644 Symbol may not have been initialized

+e771 Symbol conceivably not initialized

+e530 Symbol not initialized

-append(644,[MISRA Rule 30])

-append(771,[MISRA Rule 30])

-append(530,[MISRA Rule 30])

Rule 31 (req)

+e940 omitted braces within an initializer

-append(940,[MISRA Rule 31])

Rule 32 (req)

+e960

Rule 33 (req)

+e960

Rule 34 (req)

Rule 35 (req)

+e720 Boolean test of assignment

-append(720,[MISRA Rules 35 and 49])

+e820

-append(820,[MISRA Rule 35])

Rule 36 (adv)

Rule 37 (req)

Bitwise operations shall not be performed on signed integer types.

+e701 shift left of signed quantity

+e702 shift right of signed quantity

-append(701,[MISRA Rule 37])

-append(702,[MISRA Rule 37])

Rule 38 (req)

+e572 excessive shift value

-append(572,[MISRA Rule 38])

Rule 39 (req)

+e501 expected signed type

-append(501,[MISRA Rule 39])

Rule 40 (adv)

+e961

Rule 41 (adv)

Rule 42 (req)

+e147

-append(147,[MISRA Rule 42])

+e960

Rule 43 (req)

+e524 loss of precision

-append(524,[MISRA Rule 43])

+e653 possible loss of fraction

-append(653,[MISRA Rules 43 and 48])

Rule 44 (adv)

+e961

Rule 45 (req)

+e923 cast pointer/non-pointer

-append(923,[MISRA Rule 45])

Rule 46 (req)

+e834

+e564 order of evaluation

-append(564,[MISRA Rule 46])

Rule 47 (adv)

+e961

+e834

Rule 48 (adv)

+e912 implicit binary conversion

+e653 implicit possible loss of fraction

Rule 49 (adv)

+e720

Rule 50 (req)

+e777 testing floats for equality

-append(777,[MISRA Rule 50])

Rule 51 (adv)

+e648 overflow in computing constant

-append(648,[MISRA Rule 51])

Rule 52 (req)

+e527 unreachable

-append(527,[MISRA Rule 52])

+e506

-append(506,[MISRA Rule 52])

+e681

-append(681,[MISRA Rule 52])

+e827

-append(827,[MISRA Rule 52])

Rule 53 (req)

+e505

+e522

-append(505,[MISRA Rule 53])

-append(522,[MISRA Rule 53])

Rule 54 (req)

+e722 suspicious use of ;

-append(722,[MISRA Rule 54])

+e960

Rule 55 (adv)

+e961

Rule 56 (req)

+e801

-append(801,[MISRA Rule 56])

Rule 57 (req)

+e960

Rule 58 (req)

+e960

Rule 59 (req)

+e960

Rule 60 (adv)

+e961

Rule 61 (req)

+e616

-append(616,[MISRA Rule 61])

+e825

-append(825,[MISRA Rule 61])

Rule 62 (req)

+e744 switch statement has no default

-append(744,[MISRA Rule 62])

Rule 63 (adv)

+e961

Rule 64 (req)

+e764 switch does not have a case
-append(764,[MISRA Rule 64])

Rule 65 (req)

+e960

Rule 66 (adv)

Rule 67 (adv)

Rule 68 (req)

+e960

Rule 69 (req)

+e960

+e1916 ellipsis encountered
-append(1916,[MISRA Rule 69])

Rule 70 (req)

Rule 71 (req)

+e718 symbol undeclared
+e746 call not made in the presence of a prototype
+e937 old-style function declaration
-append(937,[MISRA Rules 71 and 76])
+e957
-append(957,[MISRA Rule 71])

Rule 72 (req)

+e18 symbol redeclared
-fvr varying return mode not allowed
+e516 argument type conflict
+e532 return mode of symbol inconsistent
-append(18,[MISRA Rule 72])
-append(516,[MISRA Rule 72])
-append(532,[MISRA Rule 72])

Rule 73 (req)

+e960

Rule 74 (req)

Rule 75 (req)

+e745 function has no explicit type
+e939 return type defaults to int
-append(745,[MISRA Rule 75])
-append(939,[MISRA Rule 75])

Rule 76 (req) +e937 old-style function declaration

Rule 77 (req)

+e747 significant prototype coercion

+e917 prototype coercion

+e918 prototype coercion of pointers

-append(747,[MISRA Rule 77])

-append(917,[MISRA Rule 77])

-append(918,[MISRA Rule 77])

Rule 78 (req)

+e118 too few arguments for prototype

+e119 too many arguments for prototype

-append(118,[MISRA Rule 78])

-append(119,[MISRA Rule 78])

Rule 79 (req)

+e82

-append(82,[MISRA Rules 79 and 84])

Rule 80 (req)

+e144 non-existent return value

-append(144,[MISRA Rule 80])

Rule 81 (adv)

Rule 82 (adv)

Rule 83 (req)

+e533 function should return a value

-append(533,[MISRA Rules 83 and 84])

Rule 84 (req)

+e533 function should not return a value

+e82 return <exp>; illegal with void function

Rule 85 (adv)

Rule 86 (adv)

Rule 87 (req)

+e960

Rule 88 (req)

+e960

Rule 89 (req)

+e12 Need < or \ " after #include
-append(12,[MISRA Rule 89])

Rule 90 (req)

Rule 91 (req)

+e960

Rule 92 (adv)

+e961

Rule 93 (adv)

Rule 94 (req)

+e131 syntax error in call of macro
-append(131,[MISRA Rule 94])

Rule 95 (req)

+e436
-append(436,[MISRA Rule 95])

Rule 96 (req)

+e773 expression-like macro not parenthesized
-append(773,[MISRA Rule 96])

Rule 97 (adv)

+e553 undefined preprocessor variable
-append(553,[MISRA Rule 97])

Rule 98 (req)

+e960

Rule 99 (req)

Rule 100 (req)

+e960

Rule 101 (adv)

Rule 102 (adv)

Rule 103 (req)

+e946 relational or subtract operator applied to pointers
-append(946,[MISRA Rule 103])

Rule 104 (req)

Rule 105 (req)

Rule 106 (req)

+e733 assigning address of auto to outer scope symbol

+e789 assigning address of auto to static

-append(733,[MISRA Rule 106])

-append(789,[MISRA Rule 106])

Rule 107 (req)

+e413

+e613

+e794

-append(413,[MISRA Rule 107])

-append(613,[MISRA Rule 107])

-append(794,[MISRA Rule 107])

Rule 108 (req)

+e43 vacuous type for variable

-append(43,[Misra Rule 108])

Rule 109 (req)

Rule 110 (req)

+e960

Rule 111 (req)

+e46 field type should be int

-append(46,[Misra Rule 111])

Rule 112 (req)

+e806 small bit field is signed rather than unsigned

-append(806,[Misra Rule 112])

Rule 113 (req)

Rule 114 (req)

+e683 complain about #define standard functions

-append(683,[Misra Rule 114])

Rule 115 (req)

Rule 116 (req)

Rule 117 (req)

PC-lint's and FlexeLint's built-in and customisable semantics allow conformance to this rule.

Rule 118 (req)

+e421
-function(gets, calloc, malloc, realloc, free)
-append(421(calloc), [MISRA Rule 118])
-append(421(malloc), [MISRA Rule 118])
-append(421(realloc), [MISRA Rule 118])
-append(421(free), [MISRA Rule 118])

Rule 119 (req)

+derrno=errno_violates_MISRA_Rule_119

Rule 120 (req) +doffsetof=offsetof_violates_MISRA_Rule_120

Rule 121 (req)

+e829
+e421
-headerwarn(locale.h)
-function(gets, setlocale, localeconv)
-append(829(locale.h), [MISRA Rule 121])
-append(421(setlocale), [MISRA Rule 121])
-append(421(localeconv), [MISRA Rule 121])

Rule 122 (req)

+e421
-function(gets, longjmp, setjmp)
-append(421(longjmp), [MISRA Rule 122])
-append(421(setjmp), [MISRA Rule 122])

Rule 123 (req)

+e421
-function(gets, signal, raise)
-append(421(signal), [MISRA Rule 123])
-append(421(raise), [MISRA Rule 123])

Rule 124 (req)

+e829
-headerwarn(stdio.h)
-append(829(stdio.h), [MISRA Rule 124])

Rule 125 (req) -function(gets, atof, atoi, atol)

+e421
-append(421(atof), [MISRA Rule 125])
-append(421(atoi), [MISRA Rule 125])
-append(421(atol), [MISRA Rule 125])

Rule 126 (req) -function(gets,abort,exit,getenv,system)

+e421

-append(421(abort), [MISRA Rule 126])

-append(421(exit), [MISRA Rule 126])

-append(421(getenv), [MISRA Rule 126])

-append(421(system), [MISRA Rule 126])

Rule 127 (req)

+e429

-function(gets,time,strftime,clock,difftime,mktime)

-append(421(time), [MISRA Rule 127])

-append(421(strftime), [MISRA Rule 127])

-append(421(clock), [MISRA Rule 127])

-append(421(difftime), [MISRA Rule 127])

-append(421(mktime), [MISRA Rule 127])

5 References

Beach, M. *Hitex C51 Primer* 3rd Ed, Hitex UK, 1995, <http://www.hitex.co.uk>

COX B, *Software ICs and Objective C, Interactive Programming Environments*, McGraw Hill, 1984

Hatton L, *Safer C*, Mcgraw-Hill(1994)

Hills C A, *Microcontroller Debuggers* Chris Hills & Mike Beach, Hitex (UK) Ltd. April 1999 <http://www.hitex.co.uk> & quest.phaedsys.org

Hills CA & Beach M, Hitex, SCIL-Level A paper project managers, team leaders and Engineers on the classification of embedded projects and tools. Useful for getting accountants to spend money Download from www.scil-level.org

Home Office Reforming the Law on Involuntary Manslaughter : The governments Proposals <http://www.homeoffice.gov.uk/consult/lcbill.pdf>

[Johnson] S. C. Johnson, '*Lint, a Program Checker,*' in *Unix Programmer's Manual*, Seventh Edition, Vol. 2B, M. D. McIlroy and B. W. Kernighan, eds. AT&T Bell Laboratories: Murray Hill, NJ, 1979.

Kernighan Brian W, *The Practice of Programming*. Addison Wesley 1999

Koenig A C *Traps and Pitfalls*, Addison Wesley, 1989

K&R *The C programming Language* 2nd Ed., Prentice-Hall, 1988

MISRA Guidelines For The Use of The C Language in Vehicle Based Software. 1998 From <http://www.misra.org.uk/> and <http://www.hitex.co.uk/>

[Pressman] *Software Engineering A Practitioners Approach*. 3rd Ed McGrawHill 1992 ISBN 0-07-050814-3

Ritchie D. M. *The Development of the C Language* Bell Labs/Lucent Technologies Murray Hill, NJ 07974 USA 1993 available from his web site <http://cm.bell-labs.com/cm/cs/who/dmr/index.htm>. This is well worth reading.



Hitex (UK) Ltd.
Coventry, CV4 7EZ
www.hitex.co.uk
chills@hitex.co.uk



quest.phaedsys.org/
chris@phaedsys.org