EasyChair Preprint

№ 9224

Static Code Analysis for C and C++ Using VectorCAST Lint Tool

Veeresh Havalad, K J Priyanka, B Shweta, R Muttanna, S Gokulan, A Mohammed and Yogendra C Dasar

EasyChair preprints are intended for rapid dissemination of research results and are integrated with the rest of EasyChair.

November 3, 2022

Static Code Analysis for C and C++ using VectorCAST lint tool

Veeresh Havalad, Priyanka K J, Shweta B, Muttanna R, Gokulan, Mohammed A, Yogendra C Dasar
Mando Softtech India [SW Verification]

Abstract:

VectorCAST/Lint uses the Industry Standard Lint Analysis Engine for C/C++ source code. Lint analysis identifies coding and logic errors in code that is otherwise syntactically correct. VectorCAST/Lint can examine individual source files or an entire application and provides comprehensive reporting capabilities. The VectorCAST/Lint tool is preconfigured to provide out-of-the box checking for the MISRA C, MISRA C 2004, and MISRA C++ 2008 standards. The VectorCAST/Lint message browser makes it easy to review errors and warnings in the context of the source file where they occur. Additionally, VectorCAST/Lint's message browser displays detailed explanations of the concepts behind the analysis. VectorCAST/Lint merges the technology of two industry standard tools into a single IDE providing unprecedented efficiency for embedded developers.

Keywords: VectorCAST, MISRA Guidelines, Static Testing

1. INTRODUCTION

Static Testing is a software testing technique which is used to check defects in software application without executing the code. Static testing is done to avoid errors at an early stage of development as it is easier to identify the errors and solve the errors. It also helps finding errors that may not be found by Dynamic Testing.
The two main types of static testing techniques are
Manual examinations: Manual examinations include analysis of code done manually, also known as **REVIEWS.**
Automated analysis using tools: Automated analysis are basically static analysis which is done using tools.

VectorCAST provides ways to integrate the tools you already use to enforce coding standards, define requirements, and build your applications. The VectorCAST embedded software testing platform is a family of products that automates testing activities across the software development lifecycle.

VectorCAST is integrated with Gimpel Software's Lint source code analysis engine. Lint is a static source code analyzer that performs module-based or whole-program source code analysis on C/C++ codebases and automatically identifies problems at their source, prior to compiling. The VectorCAST/Lint integration is configured for checking MISRA C (C1), MISRA C (C2), MISRA C (2012), and MISRA C++ standards, and includes an extensive list of embedded compiler option files. The VectorCAST/Lint integration is available in C/C++ environments and Cover environments.

The costs of fixing a bug depends on the phase of the software development the issue is discovered. Even though the actual reported numbers vary in the literature, it is a common understanding that the costs are heavily increasing with every development stage. Logically, in order to reduce cost and to increase quality, testing activities are brought forward in the process as much as possible.

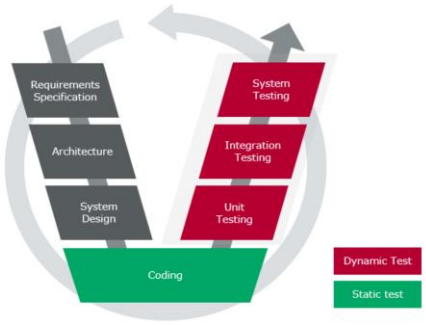

Fig 1: Development life cycle

2. HOW DOES STATIC ANALYSIS WORK?

Static code analyzers use a compiler-like front-end to build a syntactic and semantic model of the software. The syntactic model is then analyzed against a set of rules or "checkers" to see if the code is in violation. These checkers use pattern-matching algorithms to detect errors such as poor use of language constructs, use of insecure functions, and violations of coding guidelines. The specific set of checkers used is configurable by the user. Pre-set configurations are provided for convenience, for instance for coding standards such as MISRA C. More sophisticated checkers employ semantic analysis that uses data and control flow to detect complex bugs and security vulnerabilities. To do this, the static analyzer builds an execution model of the software, considers possible paths through the code, and evaluates use of data as it flows from source (like user input) to its destination (such as an API call or system call). Analyzing every single possible condition and path would be too time consuming, so the analyzer uses heuristics to detect the most likely paths for evaluation.

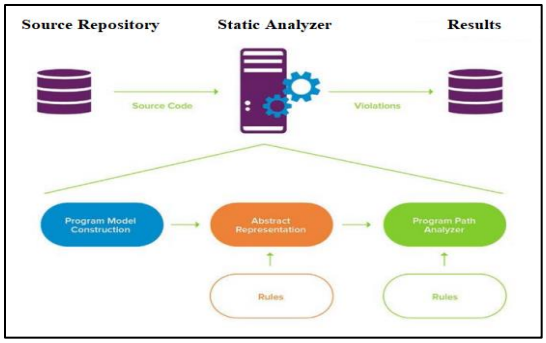

Fig 2: Static analysis work flow

This is the systematic process of analyzing the source code of the software without actually executing it. In this process, the source code for the software is analyzed for different parameters. This testing aims to detect errors in the execution phase helps in the prevention of errors at later stages.

- Whether or not the code meets the industry coding standards?
- Are there any loopholes in the dry run of the code?
- Is the syntax correct?
- Is the source code optimized or needs more optimizations?
- Is there any dead code or unreachable code?
- Are there any unassigned variables?
- Infinite loops present in the code.

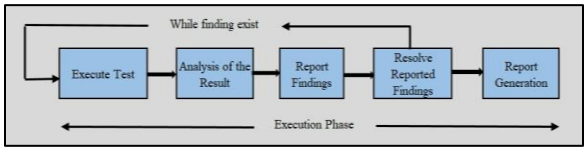

Fig 3: Life cycle of static analysis

3. TYPES OF ERRORS

Some of the major static errors found in the static analysis using lint tool in the early stage are listed below.

- Indexing beyond arrays.
- Dereferencing null pointers.
- (Potentially) dangerous data type combinations.
- Unreachable code.
- Non-portable constructs.

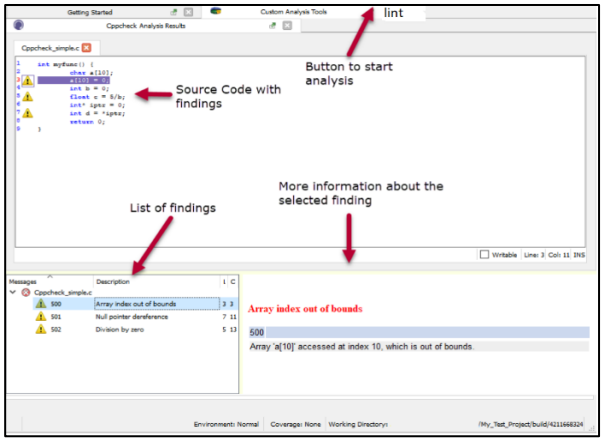

Fig 4: Error analysis using lint

The Lint Analysis Results window has three parts:

- Source Code tab, which displays lint_example.c
- Lint Messages, which displays the issues found in the unit INTEGRATING LINT 294.
- Message Detail, which shows the path to the unit, and displays details about a particular message, when one is selected. We are going to correct each Warning issue, and then re-analyze.

Most error messages have an associated error number. By looking up the number in the list below you can obtain additional information about the cause of the error.

	C	C++	Warning Level
Syntax Errors	1 - 199	1001 - 1199	1
Internal Errors	200 - 299	1200 - 1299	
Fatal Errors	300 - 399		
Warnings	400 - 699	1400 - 1699	2
Informational	700 - 899	1700 - 1899	3
Elective Notes	900 - 999	1900 - 1999	4

Fig 5: Error messages number

4. USING VECTORCAST/LINT'S MISRA COMPLIANCE CHECKING

The Motor Industry Reliability Association (MIRA) released a programming guideline for C in 1998 (sometimes referred to as MISRA C1), and a revised version was released in 2004 (MISRA C2). In 2008, MIRA released guidelines for C++ (MISRA C++). PC-Lint/FlexeLint have supported checks for the available MISRA guidelines since early 2001, and we intend for Lint to provide ongoing and increasing support for these guidelines.

MISRA violation rules can be viewed in the below figure in VectorCAST\Lint

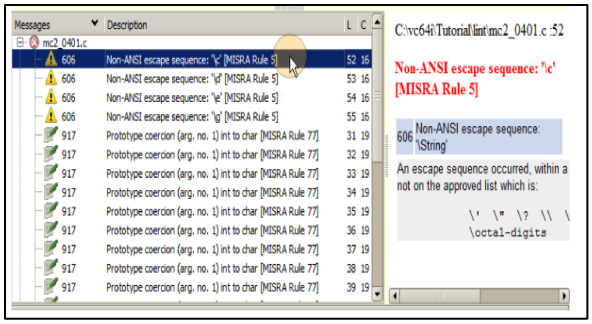

Fig 6: MISRA Violation message

5. ADVANTAGES OF STATIC TESTING USING LINT

- Identify a wide range of defects and vulnerabilities
- Deep analysis to find potential bugs and suspicious code
- Quickly identify root cause and provide actionable fixes
- Support for coding Standards such as MISRA, AUTOSAR, and CERT C
- Certified for ISO 26262 and IEC 61508
- Early detection and correction of any coding errors.
- Reduces cost in early stages of development in terms of the amount of rework needed to fix any errors.
- Reduced timescales for development.

6. CONCLUSION

Throughout this paper static analysis tools have been discussed in depth. These tools are today irreplaceable in the development cycle and very important in verification activities. Each phase of the development cycle can benefit from the use static analysis tools, ultimately resulting in a better and safer product. Static analysis tools are beneficial to the software production and verification processes, although the way they are deployed and used may be different. They help keep development costs down by finding issues as early as possible in the development cycle.

References

- PC-lint Plus | Static Code Analysis for C and C++ | Vector
- VCAST6.4.6/Lint/Docs/flexelint.pdf
- VCAST6.4.6/docs/PDF/vcast_interactive_tutorials.pdf