Reducing Novice C Programmers’ Frustration through Improved Runtime Error Checking

Matthew Heinsen Egan, Chris McDonald
Computer Science and Software Engineering
University of Western Australia
Crawley, WA 6009, Australia
m.heinsen.egan@graduate.uwa.edu.au, chris.mcdonald@uwa.edu.au

ABSTRACT

We present SeeC, a novice-focused tool for the C programming language that records the execution of student programs, detects runtime errors, and enables students to review their programs’ execution in a graphical environment.

Categories and Subject Descriptors
D.2.5 [Software Engineering]: Testing and Debugging; K.3.2 [Computers and Education]: Computer and Information Science Education

General Terms
Human Factors, Languages

Keywords
Novice programmers; debuggers

1. INTRODUCTION

Despite C’s evolution and standardization, newcomers to C still experience demoralizing difficulty with runtime errors, the language’s limited runtime error checking, and its concept of undefined behaviour. Frequently seen errors include array-bounds violations, use of uninitialized memory, dereferencing invalid pointers, passing invalid parameters to standard library functions, and passing non-terminated character arrays to string handling functions.

2. SEECC

SeeC is a novice-focused environment which detects and describes all of these errors during the execution of students’ programs. SeeC records a program’s execution and a graphical interface facilitates students’ navigation, backwards and forwards, through their program’s execution history. Within our IDE, students can reason backwards from a runtime error to determine its true cause, can deterministically replay a program, and can deeply investigate a program’s behaviour.

SeeC uses LLVM [1] to perform compile-time instrumentation of student programs: inserting code for execution tracing and error detection, and redirecting function calls from the C standard library to error-checking interfaces. Function parameters are checked against information from the tracing system, allowing detection of invalid usage such as passing a non-terminated character array to a function requiring a C string. SeeC produces superior error messages describing the program’s misuse of the standard library, rather than just errors that result from that misuse. Our approach leads to students’ reduced frustration and greater understanding of program behaviour.

3. COMPARISON WITH MEMCHECK

Numerous debuggers and error detectors for C are used in educational settings, such as the Valgrind tool Memcheck, which detects some runtime memory errors [2]. While Valgrind’s dynamic instrumentation modifies pre-compiled binaries, our source-aware compile-time instrumentation provides access to detailed semantic information, enabling improved error detection with precise reporting. SeeC informs students of their exact C expression and context responsible for an error, rather than just displaying the containing line.

We compared the error detection of SeeC and Memcheck by testing the correctness of 170 student project solutions collected during the 2012 presentation of our University’s 1st year course covering the C programming language and Operating Systems. SeeC detected errors in 43 student programs for which Memcheck detected no errors. 7 of these were “benign” uses of uninitialized memory, permitted by Memcheck. For all programs in which Memcheck detected errors, SeeC also detected errors.

4. ACKNOWLEDGEMENTS

This research is partially supported by an Australian Postgraduate Award and the Apple University Consortium (AU).

5. REFERENCES