

Dereferee

Instrumenting C++ Pointers with Meaningful Runtime Diagnostics

Standard C++

int* get_array(int size) {

return array;

void foo() {

int* array = new int[size];

int* $p = get_array(5);$

Motivation

Many computer science programs incorporate C++ at some stage in their curricula. We have seen in our program that one of the **most challenging** aspects of the language is memory management and proper use of pointers. The learning curve appears to be equally steep for students regardless of their background; these concepts are foreign to students **new** to programming, and even those who have experience in a language such as Java or Python have not been required to strictly manage the lifetime of dynamic memory as they must in C++.

Unfortunately, the tools typically used to teach C++, such as gcc and Microsoft Visual Studio, are professional tools that do not provide adequate feedback for inexperienced users in an educational setting. Furthermore, the advanced debugging and diagnostic features that they offer are **overwhelming to new users**, so they regress to less useful ad hoc techniques, such as inserting output statements and repeatedly making small, questionably targeted modifications in order to track down and fix bugs in their code. This behooves us to develop a model where we can **push** to the students as much information as possible about the errors that they are making, rather than requiring them to **pull** these data actively.

Dereferee is a C++ toolkit that was developed to address these issues by providing highly detailed diagnostics for pointer- and memory-related issues, while at the same time interfering as little as possible with the student's learning and development processes.

Minimally Invasive Design

Dereferee's checked pointer class supports a full arsenal of overloaded **operators** to ensure that checked pointers are functional in every way that standard C++ pointers are. We provide a richer set than many other instrumented pointer implementations, including even the **mathematical** and relational operators in order to diagnose problems arising from improper array usage and pointer arithmetic.

Other instrumented pointer libraries use preprocessor tricks such as redefining the **new** token in order to assist in collecting diagnostic information. While this is a simple and workable approach, it also has the effect of **breaking some valid C++ syntactic constructs.** While such cases may be unlikely to arise in an educational setting, this would not be acceptable for a more general toolkit design.

Support is also provided for **implicit conversions** between **checked(Type*)** and standard C++ **Type*** pointers. This **mixed-mode** design makes it possible to instrument only those parts of a project that are known problem areas, to convert large projects component-bycomponent while still garnering diagnostic benefits at each stage of the process, or to seamlessly **interface with third-party code** that is unaware of Dereferee.

Lastly, since we have designed Dereferee to assist all developers in their debugging and not just those in the classroom, we have made it possible to disable the instrumentation with just one simple preprocessor definition. Set this once you have tested your code to your satisfaction and the **checked(Type*)** macro will evaluate merely to **Type***. Thus, you can gain **all** of the diagnostic benefits provided during development with no performance penalties in your final product.

Simplicity and Transparency

Dereferee was designed for **transparency** and **ease of use.** Adding it to a project requires only three simple steps:

- Include <dereferee.h>

Build and run your program, and Dereferee will notify you of any memory-related errors that may exist in your code, even **uncovering some problem areas** that may have gone silently undetected under an uninstrumented runtime environment. Additionally, the toolkit will produce a **report** at the end of execution that shows memory usage statistics and details of any **memory leaks** that were detected.

Understanding Pointers

During the execution of a program, a pointer can be considered to have an associated state. This state describes which operations can be legally performed on the pointer and which would result in **incorrect or undefined behavior.** These states are as follows:

- declared.
- due to pointer arithmetic.

Each instrumented pointer keeps track of its state, and this state is **tested before each** operation (dereference, assignment, arithmetic, and so on) so that an error can be reported if the pointer was misused.

The diagram to the right shows how the most common pointer operations affect the state of the pointer throughout the execution of a program.



When a program fails due to pointer or memory misuse, the crash that results is not a terribly useful diagnostic to determine what went wrong. Worse, the fault may not cause an immediate failure, but rather **silently propagate** to a later point or **go** undetected entirely. By using Dereferee, you can catch a number of memory-related errors **precisely at the point of failure** and determine the exact reason for them:

- Deleting a dead pointer

 Link to the Dereferee static library and the listener module appropriate for your platform and compiler configuration

Change pointer declarations in your code from Type* to checked(Type*)

• Alive: The pointer points to a currently allocated block of memory. • Null: The pointer is null.

• Out of scope: The pointer variable has gone out of scope or has not yet been

• **Dead:** A dead pointer is one that **does not point to a valid block of memory.** In order to give as detailed information as is possible, we break this into three cases, depending on the reason that the pointer is dead: the pointer was **never** initialized, the memory it points to was deleted, or it was moved out of bounds

for (int i = 0; i < 5; i++) { p[i] = i * i; delete [] p; D = NULL

 $\mathbf{p} = \mathbf{l}$

Types of Errors Detected by Dereferee

• Using a dead pointer as the source of an assignment to a checked pointer Calling delete on memory allocated with new[], or delete[] on memory allocated with **new** (array/non-array mismatch)

• Dereferencing a dead pointer

• Dereferencing a null pointer

• Indexing (with **operator**]) a pointer to memory that was not allocated as an array Indexing an array out of bounds

Dereferee also provides iterator-like semantics for pointer arithmetic, raising an error if a pointer is moved **outside the bounds** of its original memory block, except in the case of permitting a pointer to "one past the end" of an array to exist, but not be accessed (analogous to the **end()** iterator of a collection).

The toolkit also strictly enforces several pre-conditions on pointers that are described in the C++ standard but are not tested by most runtimes for performance reasons. These include consistency checks when comparing the values of pointers; for example, it does not make sense to compare two pointers with an inequality operator if they do not point into the same array, and Dereferee will warn of this.

Lastly, if the last live pointer to a block of memory goes out of scope or is re-assigned, then this is indicative of a memory leak and Dereferee reports the problem **immediately** at the point where the leak occurred.

Instrumented with Dereferee

```
#include <dereferee.h>
checked(int*) get_array(int size) {
    checked(int*) array = new int[size];
     return array;
void foo() {
    checked(int*) p = get_array(5);
    for(int i = 0; i < 5; i++) {
         p[i] = i * i
    delete [] p;
```



Extensibility

Dereferee has been designed to be **highly extensible.** A user can write a "listener" module that hooks into the memory manager in order to customize the toolkit's functionality, as described below.

Controlling How and Where Errors Are Reported

The pre-written listener modules included with Dereferee send error messages to standard output or standard error. This may not be appropriate in all cases; a user could instead write a custom module that displays errors in a graphical user interface, or collects statistical **information** about the number of nature of errors that occur for later analysis.

In our introductory programming courses, we use a listener module that integrates with the **CxxTest unit** testing framework (see sidebar) so that memory-related errors are handled in the same manner as a user-supplied test assertion failure.

Obtaining Backtraces to Assist in Debugging

As discussed earlier, many libraries that provide debugging support for memory management redefine the **new** token and use the **____FILE___** and **____LINE___** macros to associate with each block of memory the source code location at which it was allocated. Since Dereferee **avoids** using the preprocessor in this manner in order to maintain as much syntactic compatibility as possible, a different approach was required.

In order to provide users with as much information as possible, the memory manager requests an **execution backtrace** for each call to **new**. Since this is naturally a platform-specific task, we push the responsibility to the listener so that new modules can be **easily written to** support any platform that is desired.

As of this writing, Dereferee **ships with canned support** for the following compiler and platform configurations: Microsoft Visual C++ 2005 or higher under Windows; and gcc 3.4 or higher under Mac OS X, Cygwin, and any variant of Unix that supports the BFD library and **/proc** filesystem. Furthermore, the Visual C++ listener module integrates with the environment's debugger; if Dereferee raises an error during execution, the program is suspended on a breakpoint so that the cause of the error can be immediately investigated.

Acknowledgments

Dereferee owes its existence to prior work by Scott M. Pike and Bruce W. Weide of The Ohio State University and Joseph E. Hollingsworth of Indiana University Southeast. Their checked pointer toolkit, *Checkmate,* inspired and formed the basis for the work that eventually became Dereferee.

Anthony Allevato and Stephen Edwards allevato@vt.edu, edwards@cs.vt.edu Virginia Tech Department of Computer Science http://web-cat.org

Related Work

The Web-CAT project began as a web-based automated grading system for assessing how rigorously students test their code, but it now hosts a number of educationally-oriented tools for students in CS1 and CS2 courses as well. Some of our other work is featured below.

Electronic Submission Plug-Ins for Eclipse and Microsoft Visual Studio

With Web-CAT currently being used at 23 institutions and growing, we provide plug-ins for the most popular development environments to allow students to automatically zip up and submit their projects to the grader. Once submitted, the student's results will be displayed in their web browser.

• Workbench	Electronic Submission		
 Ant Build Order Checkclipse DrJava 	Please enter the URL provided by your instructor that contains the assignment definitions to be used by the electronic submission plug-in in the field below.		
Electronic Submission Gild G	<u>A</u> ssignment definition URL: Default <u>u</u> sername: <u>O</u> utgoing (SMTP) mail server: <u>E</u> -mail address:	http://www.cs.foo.edu/cs100/submissions.xml student mail.foo.edu student@foo.edu Restore Defaults Apply	
Import Export		OK Cancel	

When using the plug-in for the first time, the user "sets and forgets" their configuration i their environment's preferences. These settings include a URL from which to obtain the

The Web-CAT grader provides a URL that automatically generates this content based or

he courses and assignments that are currently published on the server, so an instructo

needs to take no additional action to make their assignments available to the submitter

"targets" (projects and assignments) to which the user can submit their files.

	E- 😂 CS 100
	Lab 07 Lab 08 E - Dverdue Assignments
Assignment:	
Username:	student
Password:	******

Once the user has selected a project to submit, they choose the targe assignment on the server and enter their log-in credentials, if required The files in their project are then automatically zipped up and ransmitted, without having to leave their development environment.

These plug-ins are not restricted to the Web-CAT grader. They can be used to make submissions via HTTP, FTP, or e-mail, and can be extended to support other proprietary systems as well.

Integrated Unit Testing for C++ in Eclipse and **Microsoft Visual Studio**

In our introductory programming courses, we place a great deal of emphasis on unit testing and assess students on how thoroughly they test the components that they write. When they make the transition from Java to C++, we wish to provide them with the same level of simplicity and integration that they were afforded previously.

We have adopted the **CxxTest[†]** framework for unit testing in C++, due to its relative syntactic simplicity compared to other similar C++ libraries. A set of plug-ins for Eclipse automatically collects the test cases that a student has written as a part of the project and then executes them at the end of each successful build, concisely displaying the results in a convenient panel. Other convenience features, such as a wizard to create a test suite for a class that automatically defines stubs for the methods being tested, are also provided.

A CxxTest integration package for Microsoft Visual Studio is also currently under development.

Runs: 1		ole 🔲 Properties		
📑 Hierarchy 🕜			Details	
🗄 🗄 BankAccou	ntTests			

n Eclipse, the CxxTest results view has been designed to mimic the rder to ease the transition from Java to C++ for students.

	account.ac	Eposit(/),								
	// To my }	knowledge, 5 + 7 != 13.								
38	TS_ASSERT_	_EQUALS(account.getBalance(), 13);								
	}		~							
	<		>							
8	Problems 🧟 Tasks 📃 Co	onsole 🔲 Properties 🔩 🛛 🔤								
R	Runs: 6 🛛 Errors: 0 🖾 Failures: 1									
E	Hierarchy 😚 Memory	Details								
	BankAccountTests testInitialBalances testDeposits testWithdrawals testVariousActivity testOverdrawn testBadCase	∎? Failed assertion (line 103): expected account.getBalance() == 13, but found	12 != 13							
		st case failure, the reason for the failure is displayed in the rs are placed on the source code lines where the errors occur								

that they can be easily located by the use

[†] The CxxTest unit testing framework can be found on the web at *http://cxxtest.sourceforge.net*.

Web-CAT is supported in part by NSF under grants DUE-0618663 and DUE-0633594, and by Microsoft Corporation.