Pointers and Memory

By Nick Parlante

Abstract
This document explains how pointers and memory work and how to use them—from the basic concepts through all the major programming techniques. For each topic there is a combination of discussion, sample C code, and drawings.

Audience
This document can be used as an introduction to pointers for someone with basic programming experience. Alternately, it can be used to review and to fill in gaps for someone with a partial understanding of pointers and memory. Many advanced programming and debugging problems only make sense with a complete understanding of pointers and memory — this document tries to provide that understanding. This document concentrates on explaining how pointers work. For more advanced pointer applications and practice problems, see the other resources below.

Pace
Like most CS Education Library documents, the coverage here tries to be complete but fast. The document starts with the basics and advances through all the major topics. The pace is fairly quick — each basic concept is covered once and usually there is some example code and a memory drawing. Then the text moves on to the next topic. For more practice, you can take the time to work through the examples and sample problems. Also, see the references below for more practice problems.

Topics
Topics include: pointers, local memory, allocation, deallocation, dereference operations, pointer assignment, deep vs. shallow copies, the ampersand operator (&), bad pointers, the NULL pointer, value parameters, reference parameters, heap allocation and deallocation, memory ownership models, and memory leaks. The text focuses on pointers and memory in compiled languages like C and C++. At the end of each section, there is some related but optional material, and in particular there are occasional notes on other languages, such as Java.

Other CS Education Library Documents
- Point Fun With Binky Video (http://cslibrary.stanford.edu/104/)
  A silly video about pointer basics.

- Linked list Basics (http://cslibrary.stanford.edu/103/)
  Introduces the basic techniques for building linked lists in C.

- Linked List Problems (http://cslibrary.stanford.edu/105/)
  18 classic linked list problems with solutions — a great way to practice with realistic, pointer intensive C code, and there's just no substitute for practice!

- Essential C (http://cslibrary.stanford.edu/101/)
  Complete coverage of the C language, including all of the syntax used in this document.

Table of Contents

Edition

The first edition of this document was on Jan 19, 1999. This Feb 21, 2000 edition represents only very minor changes. The author may be reached at nick.parlante@cs.stanford.edu. The CS Education Library may be reached at cslibrary@cs.stanford.edu.

Dedication

This document is distributed for the benefit and education of all. That someone seeking education should have the opportunity to find it. May you learn from it in the spirit in which it is given — to make efficiency and beauty in your designs, peace and fairness in your actions.

Preface To The First Edition

This article has appeared to hover at around 80% done for 6 months! Every time I add one section, I think of two more which also need to be written. I was motivated to keep working on it since there are so many other articles which use memory, &, ... in passing where I wanted something to refer to. I hope you find it valuable in its current form. I'm going to ship it quickly before I accidentally start adding another section!

Section 1 — Basic Pointers

Pointers — Before and After

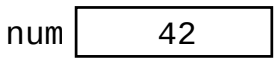
There's a lot of nice, tidy code you can write without knowing about pointers. But once you learn to use the power of pointers, you can never go back. There are too many things that can only be done with pointers. But with increased power comes increased responsibility. Pointers allow new and more ugly types of bugs, and pointer bugs can crash in random ways which makes them more difficult to debug. Nonetheless, even with their problems, pointers are an irresistibly powerful programming construct. (The following explanation uses the C language syntax where a syntax is required; there is a discussion of Java at the section.)

Why Have Pointers?

Pointers solve two common software problems. First, pointers allow different sections of code to share information easily. You can get the same effect by copying information back and forth, but pointers solve the problem better. Second, pointers enable complex "linked" data structures like linked lists and binary trees.

What Is A Pointer?

Simple `int` and `float` variables operate pretty intuitively. An `int` variable is like a box which can store a single `int` value such as 42. In a drawing, a simple variable is a box with its current value drawn inside.



A pointer works a little differently— it does not store a simple value directly. Instead, a pointer stores a **reference** to another value. The variable the pointer refers to is sometimes known as its "pointee". In a drawing, a pointer is a box which contains the beginning of an arrow which leads to its pointee. (There is no single, official, word for the concept of a pointee — pointee is just the word used in these explanations.)

The following drawing shows two variables: `num` and `numPtr`. The simple variable `num` contains the value 42 in the usual way. The variable `numPtr` is a pointer which contains a reference to the variable `num`. The `numPtr` variable is the pointer and `num` is its pointee. What is stored inside of `numPtr`? Its value is not an `int`. Its value is a reference to an `int`.