A Brain-Friendly Guide

Head First C

Discover the secrets of the C coding gurus

Learn how make can change your life

Avoid embarrassing pointer mistakes

See how variadic functions helped Sue be more flexible

Fool around in the C Standard Library

Build a retro classic arcade game

David Griffiths &
Dawn Griffiths

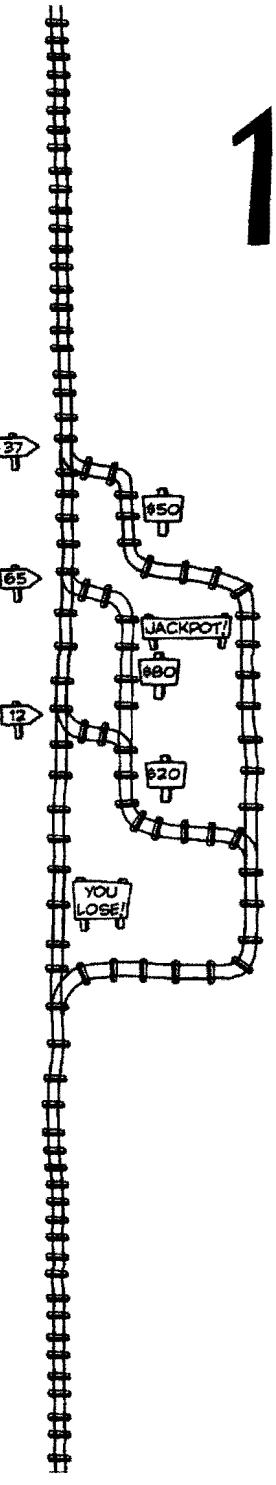
O'REILLY®

Table of Contents (Summary)

Table of Contents (the real thing)

Intro

Your brain on C. Here *you* are trying to *learn* something, while here your *brain* is, doing you a favor by making sure the learning doesn't *stick*. Your brain's thinking, "Better leave room for more important things, like which wild animals to avoid and whether naked snowboarding is a bad idea." So how *do* you trick your brain into thinking that your life depends on knowing C?



getting started with C

1 Diving in

Want to get inside the computer's head?

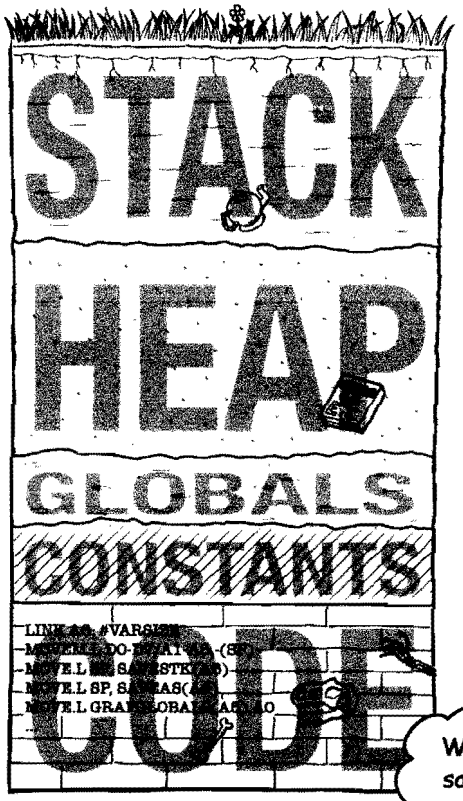
Need to write **high-performance code** for a new game? Program an **Arduino**? Or use that advanced **third-party library** in your iPhone app? If so, then C's here to help. C works at a **much lower level** than most other languages, so understanding C gives you a much better idea of **what's really going on**. C can even help you better understand other languages as well. So dive in and grab your compiler, and you'll soon get started in no time.

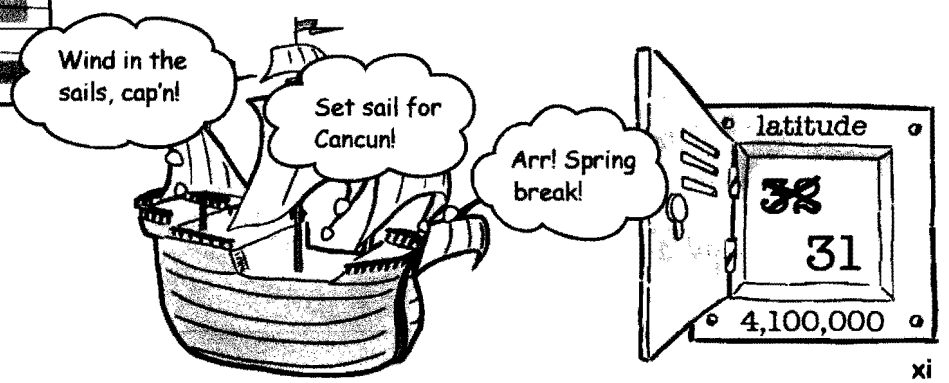
memory and pointers

2 What are you pointing at?

If you really want to kick butt with C, you need to understand how C handles memory.

The C language gives you a lot more *control* over how your program uses the **computer's memory**. In this chapter, you'll strip back the covers and see exactly what happens when you **read and write variables**. You'll learn **how arrays work**, how to avoid some **nasty memory SNAFUs**, and most of all, you'll see how **mastering pointers and memory addressing** is key to becoming a kick-ass C programmer.



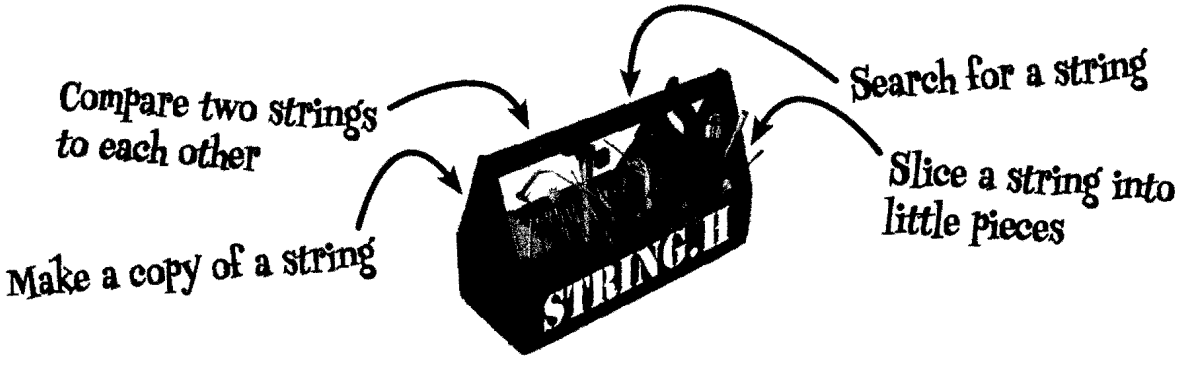


strings

2.5 String theory

There's more to strings than reading them.

You've seen how strings in C are actually `char` *arrays* but what does C allow you to *do* with them? That's where ***string.h*** comes in. *string.h* is part of the C Standard Library that's dedicated to **string manipulation**. If you want to ***concatenate*** strings together, ***copy*** one string to another, or ***compare*** two strings, the functions in *string.h* are there to help. In this chapter, you'll see how to create an **array of strings**, and then take a close look at how to ***search within strings*** using the **`strstr()`** function.

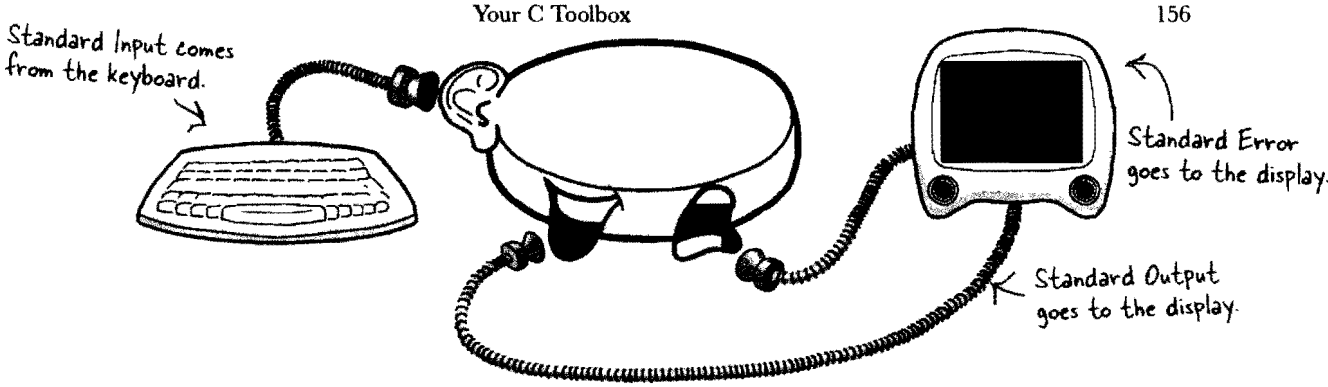


creating small tools

3 Do one thing and do it well

Every operating system includes small tools.

Small tools written in C perform **specialized small tasks**, such as reading and writing files, or filtering data. If you want to perform more complex tasks, you can even *link several tools together.* But how are these small tools built? In this chapter, you'll look at the building blocks of creating small tools. You'll learn how to control **command-line options**, how to manage **streams of information**, and **redirection**, getting tooled up in no time.



using multiple source files

4 Break it down, build it up

If you create a big program, you don't want a big source file. Can you imagine how difficult and time-consuming a single source file for an enterprise-level program would be to maintain? In this chapter, you'll learn how C allows you to break your source code into **small, manageable chunks** and then rebuild them into **one huge program**. Along the way, you'll learn a bit more about **data type subtleties** and get to meet your new best friend: **`make`**.