

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



O'REILLY®

David Griffiths &  
Dawn Griffiths

# Table of Contents (Summary)

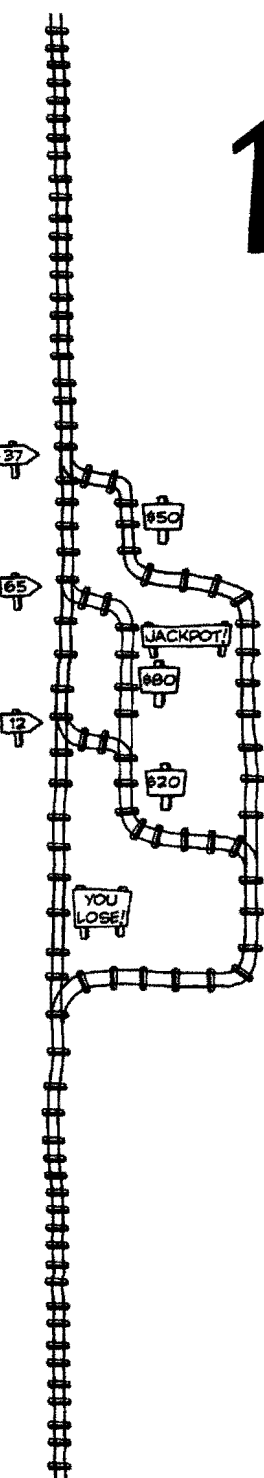
	Intro	xxvii
1	Getting Started with C: <i>Diving in</i>	1
2	Memory and Pointers: <i>What are you pointing at?</i>	41
2.5	Strings: <i>String theory</i>	83
3	Creating Small Tools: <i>Do one thing and do it well</i>	103
4	Using Multiple Source Files: <i>Break it down, build it up</i>	157
	C Lab 1: <i>Arduino</i>	207
5	Structs, Unions, and Bitfields: <i>Rolling your own structures</i>	217
6	Data Structures and Dynamic Memory: <i>Building bridges</i>	267
7	Advanced Functions: <i>Turn your functions up to 11</i>	311
8	Static and Dynamic Libraries: <i>Hot-swappable code</i>	351
	C Lab 2: <i>OpenCV</i>	389
9	Processes and System Calls: <i>Breaking boundaries</i>	397
10	Interprocess Communication: <i>It's good to talk</i>	429
11	Sockets and Networking: <i>There's no place like 127.0.0.1</i>	467
12	Threads: <i>It's a parallel world</i>	501
	C Lab 3: <i>Blasteroids</i>	523
i	Leftovers: <i>The top ten things (we didn't cover)</i>	539
ii	C Topics: <i>Revision roundup</i>	553

# 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?

Who is this book for?	xxviii
We know what you're thinking	xxix
Metacognition	xxx
Bend your brain into submission	xxxiii
Read me	xxxiv
The technical review team	xxxvi
Acknowledgments	xxxvii



*getting started with C*

## 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.

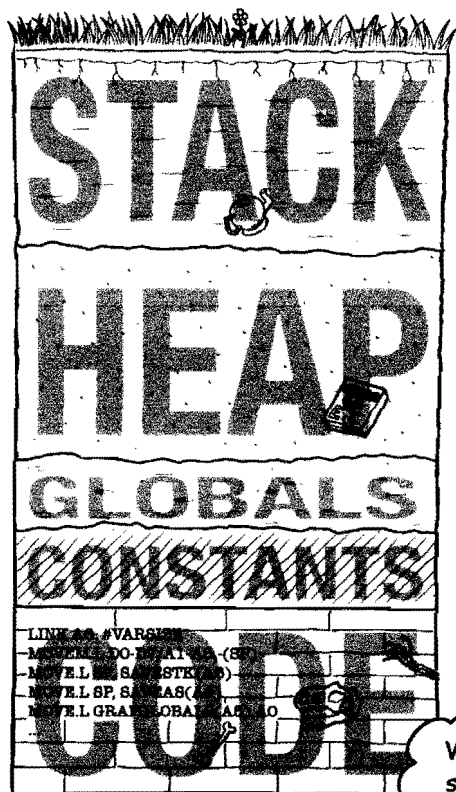
C is a language for small, fast programs	2
But what does a complete C program look like?	5
But how do you run the program?	9
Two types of command	14
Here's the code so far	15
Card counting? In C?	17
There's more to booleans than equals...	18
What's the code like now?	25
Pulling the ol' switcheroo	26
Sometimes once is not enough...	29
Loops often follow the same structure...	30
You use break to break out...	31
Your C Toolbox	40

## memory and pointers

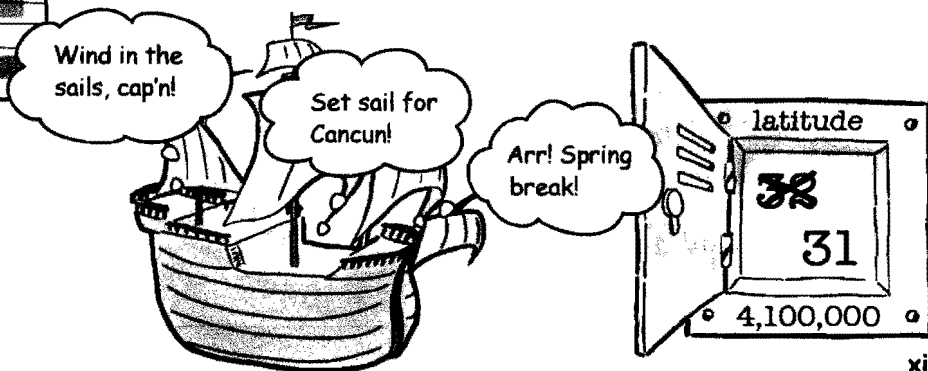
**What are you pointing at?****2**

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.



C code includes pointers	42
Digging into memory	43
Set sail with pointers	44
Try passing a pointer to the variable	47
Using memory pointers	48
How do you pass a string to a function?	53
Array variables are like pointers...	54
What the computer thinks when it runs your code	55
But array variables aren't quite pointers	59
Why arrays really start at 0	61
Why pointers have types	62
Using pointers for data entry	65
Be careful with scanf()	66
fgetc() is an alternative to scanf()	67
String literals can never be updated	72
If you're going to change a string, make a copy	74
Memory memorizer	80
Your C Toolbox	81



strings

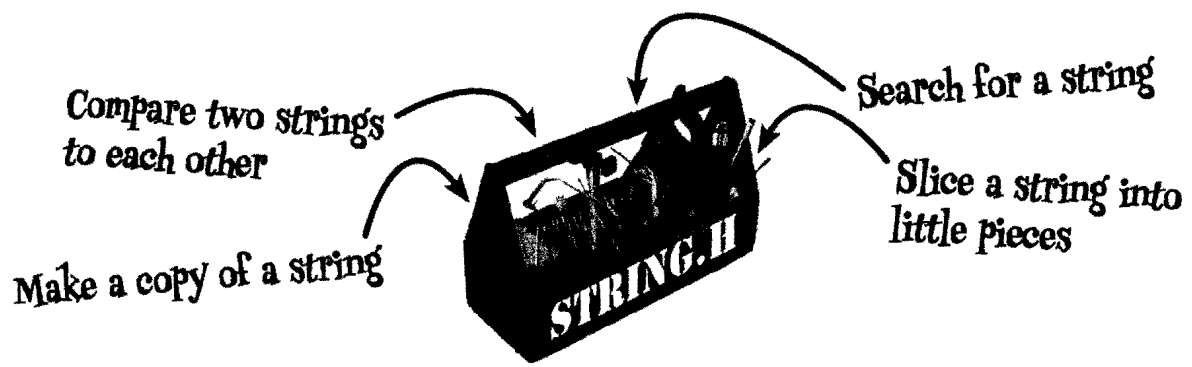
String theory

2.5

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.

Desperately seeking Frank	84
Create an array of arrays	85
Find strings containing the search text	86
Using the <code>strstr()</code> function	89
It's time for a code review	94
Array of arrays vs. array of pointers	98
Your C Toolbox	101



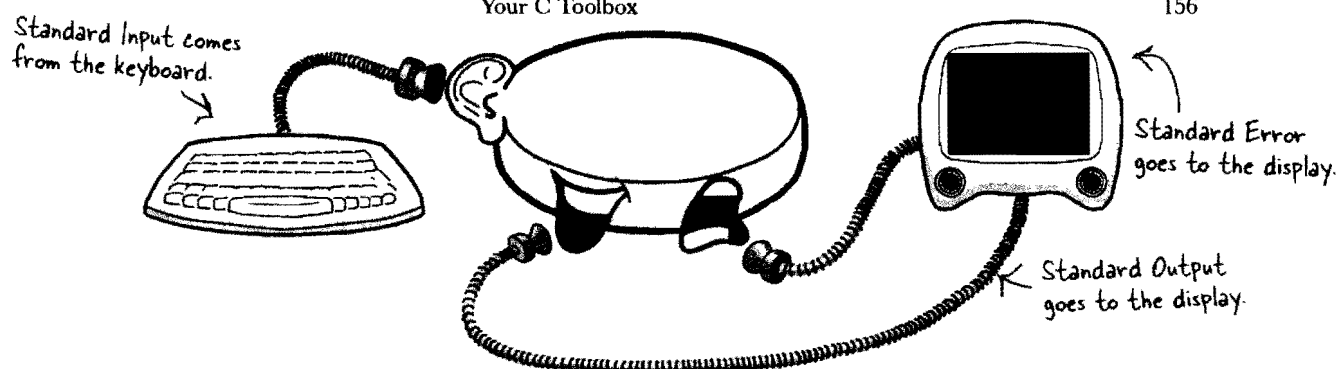
## creating small tools

## 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 toolled up in no time.

Small tools can solve big problems	104
Here's how the program should work	108
But you're not using files...	109
You can use redirection	110
Introducing the Standard Error	120
By default, the Standard Error is sent to the display	121
<code>fprintf()</code> prints to a data stream	122
Let's update the code to use <code>fprintf()</code>	123
Small tools are flexible	128
Don't change the <code>geo2json</code> tool	129
A different task needs a different tool	130
Connect your input and output with a pipe	131
The bermuda tool	132
But what if you want to output to more than one file?	137
Roll your own data streams	138
There's more to <code>main()</code>	141
Let the library do the work for you	149
Your C Toolbox	156



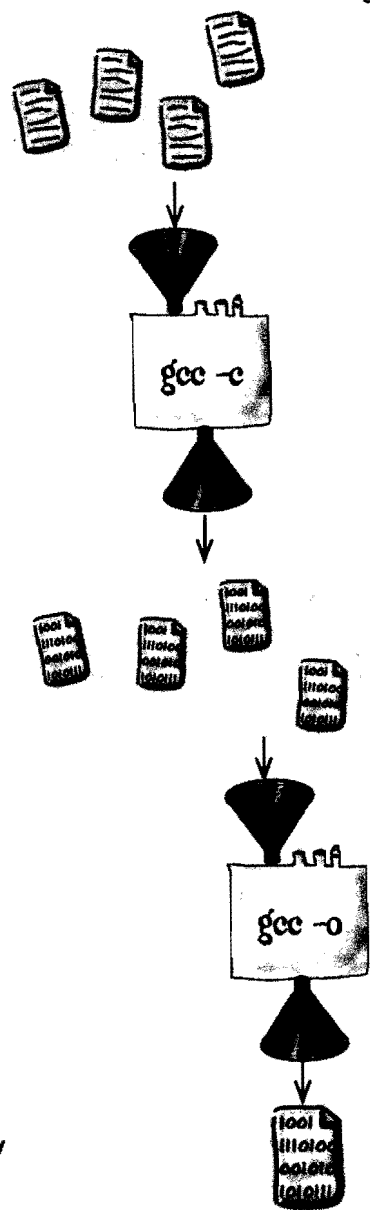
using multiple source files

Break it down, build it up

4

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**.



Your quick guide to data types	162
Don't put something big into something small	163
Use casting to put floats into whole numbers	164
Oh no...it's the out-of-work actors...	168
Let's see what's happened to the code	169
Compilers don't like surprises	171
Split the declaration from the definition	173
Creating your first header file	174
If you have common features...	182
You can split the code into separate files	183
Compilation behind the scenes	184
The shared code needs its own header file	186
It's not rocket science...or is it?	189
Don't recompile every file	190
First, compile the source into object files	191
It's hard to keep track of the files	196
Automate your builds with the make tool	198
How make works	199
Tell make about your code with a makefile	200
Liftoff!	205
Your C Toolbox	206