

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answers to Exercises

Maintained by [Richard Heathfield](#)

Last updated: 12 March 2002

This site was inaugurated on 1 January 2000 as a repository for definitive answers to the exercises in "The C Programming Language" by Kernighan and Ritchie (2nd edition) - ISBN 0-13-110362-8. Since it's a relatively new site, not all of the exercises have been done. That will change over time, of course.

Naturally, it's not impossible that you might notice a mistake, or be able to think of a better solution than the one presented on this site. If so, please let me know, and I'll consider your amendment for inclusion. I will try my best to ensure that full credit is given to all [contributors](#) to this site. Any uncredited listings (modulo errors of omission) are supplied by me personally, so beware! Complaints to email, please...

[Maintenance History](#)

[Contributors](#)

[Bugs and Fixes](#)

[Copyright Issues](#)

[Naming Conventions](#)

[Download](#) all existing non-essay solutions (C files, zipped, 116,202 bytes).

[Unsolved exercises](#)

Solutions

[Chapter 1 - A Tutorial Introduction](#)

[Chapter 2 - Types, Operators and Expressions](#)

[Chapter 3 - Control Flow](#)

[Chapter 4 - Functions and Program Structure](#)

[Chapter 5 - Pointers and Arrays](#)

[Chapter 6 - Structures](#)

[Chapter 7 - Input and Output](#)

[Chapter 8 - The UNIX System Interface](#)

[Home](#)

You are visitor number **456855** - call again soon!

"The C Programming Language" - Answers to Exercises

Contributors

My biggest thanks of all go, of course, to Dennis Ritchie and Brian Kernighan, authors of possibly the finest language tutorial and reference book ever written.

This site wouldn't even exist if it weren't for the efforts of a great many people, who contributed either by providing solutions or by suggesting corrections. Some have chosen to remain anonymous, but the rest (as far as I know) are listed here. Please don't ask me what all these guys did; if you can't find a solution from them, you may assume they provided constructive criticism of some kind:

Name or Nick	Email
Ola Angelsmark	olaan@ida.liu.se
Regis Barbanchon	regis@info.unicaen.fr
Russ Bobbitt	bobbitts@hotmail.com
Colin Barker	colin.barker@wanadoo.fr
Stig Brautaset	stigbrau@online.no
RC	RC no spam 4 meth x at high-temperature mail dot com
Dann Corbit	DCorbit@SolutionsIQ.com
Rick Dearman	rick@ricken.demon.co.uk
Stefan Farfeleder	e0026813@stud3.tuwien.ac.at
Flippant Squirrel	flippant_squirrel@yahoo.com
Paul Griffiths	paul@paulgriffiths.demon.co.uk
Steven Huang	steven.huang@iname.com
jimstad	jimstad@cyberway.com.sg
Lawrence Kirby	fred@genesis.demon.co.uk
Rick Litherland	ralither@bellsouth.net
Wayne Lubin	wclubin@telocity.com
Marshall S McLeish	jugg@home.com
Chris Mears	cmears@bigpond.com

Ben Pfaff	pfaffben@msu.edu
Dennis Piccioni	dennis-p@dataaccess.com
Gregory Pietsch	gkp1@flash.net
Lew Pitcher	lpitcher@yesic.com
MJSR	rawcswi@my-deja.com
Craig Schroeder	schroeder@santel.net
Partha Seetala	seetala@cs.umn.edu
Chris Sidi	sidi@angband.org
Andrew Tesker	atesker@cs.umb.edu
Chris Torek	torek@elf.bsdi.com
Bob Wightman	bob@pattinson.demon.co.uk
Bryan Williams	gfd34@dial.pipex.com
Lars Wirzenius	liw@iki.fi
Roman Yablonovsky	Roman.Yablonovsky@ssa.gov
386sx	386sx@my-deja.com

If you would like your name, or email address, or both, to be removed from this page (or munged), please let me know by email.

[Back](#)

[Home](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answers to Exercises Maintenance History

Maintained by [Richard Heathfield](#)

Last updated: 12 March 2002

Many people have sent me solutions which have not yet appeared on the site. I regret this and apologise for it. I'll get to them as soon as I can. In an effort to clear the backlog, I have at last relinquished my policy of dealing with items in strict chronological order. Instead, I'll do the easy stuff first, which I hope will have the effect of reducing the psychological burden of tackling the less easy stuff. Here is a list of the most recent changes:

12 March 2002 - Added Steven Huang's solution to 5-7 (first submitted on the 9th December 2000).

11 March 2002 - Rick Dearman's solution to 2-1 had a couple of glitches remaining; Stefan Farfeleder sent in a fix for these a few aeons ago, which I have at last included. Also, added Steven Huang's solution to 4-7, submitted on (cough) 9 December 2000. And I added a fix to my own 1-20 solution, first sent to me by Roman Yablonovsky in October 2000. (Steven Huang sent me the same suggestion, in December 2000.) Added Stig Brautaset's solutions to 1-9 and 1-24, submitted in July 2001. Oh, and Gregory Pietsch wrote to me in January 2001 to point out that his exercise 5-11 is actually in the form of *two* programs, and could I split them up please? Done that. Oh yes, and I massively re-organised the layout of the site - the size of the index page was getting beyond a joke. At the expense of an extra click to get to a solution, you no longer have to page down through, say, six chapters of stuff to find the Chapter 7 solutions...

18 Feb 2002 - Added Bob Wightman's Cat 1 solution to 4-6 (submitted in November 2000). Also fixed a credit error (Gregory Pietsch's 7-3 wasn't ascribed properly, further down this page).

15 Feb 2002 - Wayne Lubin submitted a query re Paul Griffiths's 3-4 solution, which I forwarded to Paul. He responded with a correction almost immediately, but it's taken me about six months to do anything about it!

14 Feb 2002 - added "386sx"'s solution to 1-16, which tackles a slightly stricter interpretation of the exercise than my own solution.

13 Feb 2002 - added Steven Huang's solution to 4-8, submitted on 9 December 2000, and Gregory

Pietsch's solution to 7-3, submitted on 29 Jan 2001.

11 Feb 2002 - added Steven Huang's solutions to 5-13, submitted on 9 December 2000, and to 7-8, submitted on 7 Dec 2000. Greg P submitted a revision of his 5-6 answer on 29 Jan 2001 - incorporated here at last.

8 Feb 2002 - fixed yesterday's mods (sigh), and added a solution to 6-5 by Gregory Pietsch, first submitted back in November 2000. (I'm giving up on these ouches now...)

7 Feb 2002 - fixed yesterday's mods!, and added a solution to a new problem, 5-14. Solution is by Steven Huang, and was originally sent to me on 8 December 2000. (Ouch again.)

6 Feb 2002 - added a couple of Cat 1 solutions from Stefan Farfeleder. He originally sent these in on 1 December 2000. (Ouch.)

1 Jan 2000 to 5 Feb 2002 - whatever. :-) Of course, it was within this period that the site received at least one solution for over seventy of the exercises.

This site's source code is syntax-coloured using code provided and maintained by Bryan Williams.

[Back](#)

[Home](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answers to Exercises - Bug Reporting and Fixing Policy (in theory!)

BUGS

Submitters, please note:

In an effort to ensure the quality of the solutions posted here, I have introduced the following mechanism for processing bug reports:

- 1) On receiving a bug report which appears credible, I will annotate the relevant solution with the bug report, and email the originator of the solution.
- 2) If the originator of the solution has either provided a fix (in the form of a complete listing to replace the existing listing) or shown why the bug report is incorrect, within six weeks of being informed of the bug report, then I will update the relevant page accordingly. If not, I reserve the right to remove the solution from the page altogether (but I'll make every reasonable effort to contact the contributor again before taking this step).

[Back](#)

[Home](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answers to Exercises - Copyright Issues

The copyright in all quotations from "The C Programming Language" belongs to Bell Telephone Laboratories Inc. I have received an assurance from Dennis Ritchie that neither he nor Brian Kernighan has any objection to the site. All the questions are direct quotes from the book, as is the source code provided as a starter for contributors wishing to offer solutions for some of the Chapter 4 exercises. Anyone tempted to offer solutions in breach of Tondo and Gimpel's copyright is warned that a couple of `comp.lang.c` regulars check each solution against their copy of Tondo and Gimpel, and will send me emails if they detect any plagiarism. Solutions breaching Tondo and Gimpel's copyright will be removed as soon as they are detected.

[Back](#)

[Home](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answers to Exercises - File naming conventions

The HTML page names are constructed as follows:

krx - prefix (Kernighan and Ritchie eXercises)

c - chapter number (1 to 8)

ee - exercise number

So the first exercise is at krx101.html.

The C files also have version numbers with the following meanings:

0v - ANSI/ISO C89 compliant. The example only uses the subset of C already covered at the point in the book at which the exercise appears.

1v - ANSI/ISO C89 compliant. The example uses aspects of C which may not have been covered at the point in the book at which the exercise appears.

2v - ANSI/ISO C99 compliant. The example only uses the subset of C already covered at the point in the book at which the exercise appears. It is compliant with the C99 standard (e.g. doesn't use implicit `int`).

3v - ANSI/ISO C99 compliant. The example uses some aspects of C which exist only in the C99 release of the language.

The minor version number allows for different treatments of the same exercise by different people, in cases where this may be helpful. The minor version number is in base 36 (0 through Z). That ought to be enough for anybody. :-)

[Back](#)

[Home](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answers to Exercises - Unsolved Solutions

The following solutions have not yet been addressed at any level. If you'd like to provide a solution, please do let me know. You know where to find me.

Chapter 1 (0/24 exercises): At least one solution has been provided for each exercise.

Chapter 2 (0/10 exercises): At least one solution has been provided for each exercise.

Chapter 3 (0/6 exercises): At least one solution has been provided for each exercise.

Chapter 4 (2/14 exercises): 9,10

Chapter 5 (7/20 exercises): 12,15,16,17,18,19,20

Chapter 6 (2/6 exercises): 2,6

Chapter 7 (3/9 exercises): 4,5,7

Chapter 8 (4/8 exercises): 2,5,7,8

Total exercises: 97

Solved exercises: 79

Yet to do: 18

[Back](#)

[Home](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answers to Exercises

Chapter 1 - A Tutorial Introduction

Ex No	Pg No	Exercise	Solution	Solved by
1	8	Run the "hello world" program on your system. Experiment with leaving out parts of the program, to see what error messages you get.	<u>Listing krx101</u>	Me
2	8	Experiment to find out what happens when <code>printf</code> 's argument string contains <code>\c</code> , where <code>c</code> is some character not listed above.	<u>Listing krx102</u>	Me
3	13	Modify the temperature conversion program to print a heading above the table.	<u>Listing krx103</u>	Me
4	13	Write a program to print the corresponding Celsius to Fahrenheit table.	<u>Listing krx104</u>	Me
5	14	Modify the temperature conversion program to print the table in reverse order, that is, from 300 degrees to 0.	<u>Listing krx105</u>	Category 0 Solutions by my good self and by Chris Sidi.
6	17	Verify that the expression <code>getchar () != EOF</code> is 0 or 1.	<u>Listing krx106</u>	Me
7	17	Write a program to print the value of <code>EOF</code> .	<u>Listing krx107</u>	Me
8	20	Write a program to count blanks, tabs, and newlines.	<u>Listing krx108</u>	Me
9	20	Write a program to copy its input to its output, replacing each string of one or more blanks by a single blank.	<u>Listing krx109</u>	Category 0 Solutions by my good self, by Chris Sidi, and by Stig Brautaset.

Write a program to copy its input to its output, replacing each tab by `\t`, each backspace by `\b`, and each backslash by `\\`. This makes tabs and backspaces visible in an unambiguous way.

10 20

[Listing
krx110](#)

A couple of solutions here. One from me, and one from Gregory Pietsch.

How would you test the word count program? What kinds of input are most likely to uncover bugs if there are any?

11 21

[Listing
krx111](#)

Solution by Dann Corbit.
Solution to Dann's follow-up challenge by Gregory Pietsch.

Write a program that prints its input one word per line.

12 21

[Listing
krx112](#)

Me

Write a program to print a histogram of the lengths of words in its input. It is easy to draw the histogram with the bars horizontal; a vertical orientation is more challenging.

13 24

[Listing
krx113](#)

Me

Write a program to print a histogram of the frequencies of different characters in its input.

14 24

[Listing
krx114](#)

Me

Rewrite the temperature conversion program of Section 1.2 to use a function for conversion.

15 27

[Listing
krx115](#)

Me

Revise the main routine of the longest-line program so it will correctly print the length of arbitrarily long input lines, and as much as possible of the text.

16 30

[Listing
krx116](#)

"386sx" and myself.

Write a program to print all input lines that are longer than 80 characters.

17 31

[Listing
krx117](#)

"MJSR"

Write a program to remove all trailing blanks and tabs from each line of input, and to delete entirely blank lines.

18 31

[Listing
krx118](#)

Solution by Ben Pfaff,
modification by Chris Sidi

Write a function `reverse(s)` that reverses the character string `s`. Use it to write a program that reverses its input a line at a time.

19 31

[Listing
krx119](#)

Me

20	34	Write a program <code>datab</code> that replaces tabs in the input with the proper number of blanks to space to the next tab stop. Assume a fixed set of tab stops, say every n columns. Should n be a variable or a symbolic parameter?	Listing krx120	Me
21	34	Write a program <code>entab</code> that replaces strings of blanks with the minimum number of tabs and blanks to achieve the same spacing. Use the same stops as for <code>datab</code> . When either a tab or a single blank would suffice to reach a tab stop, which should be given preference?	Listing krx121	Rick Dearman
22	34	Write a program to "fold" long input lines into two or more shorter lines after the last non-blank character that occurs before the n -th column of input. Make sure your program does something intelligent with very long lines, and if there are no blanks or tabs before the specified column.	Listing krx122	Category 1 Solution by Rick Dearman
23	34	Write a program to remove all comments from a C program. Don't forget to handle quoted strings and character constants properly. C comments do not nest.	Listing krx123	This was the first exercise for which solutions were submitted <i>en masse</i> from comp.lang.c , and as a result, quite a few solutions have been provided here. You may find it interesting to compare and contrast different approaches to the same problem.
24	34	Write a program to check a C program for rudimentary syntax errors like unbalanced parentheses, brackets and braces. Don't forget about quotes, both single and double, escape sequences, and comments. (This program is hard if you do it in full generality.)	Listing krx124	Solutions by Rick Dearman and others

[Home](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answers to Exercises

Chapter 2 - Types, Operators and Expressions

Ex No	Pg No	Exercise	Solution	Solved by
1	36	Write a program to determine the ranges of <code>char</code> , <code>short</code> , <code>int</code> , and <code>long</code> variables, both signed and unsigned, by printing appropriate values from standard headers and by direct computation. Harder if you compute them: determine the ranges of the various floating-point types.	<u>Listing krx201</u>	Rick Dearman
2	42	Write a loop equivalent to the <code>for</code> loop above without using <code>&&</code> or <code> </code> .	<u>Listing krx202</u>	"flippant_squirrel" (!) and Craig Schroeder
3	46	Write the function <code>htoi(s)</code> , which converts a string of hexadecimal digits (including an optional <code>0x</code> or <code>0X</code>) into its equivalent integer value. The allowable digits are <code>0</code> through <code>9</code> , <code>a</code> through <code>f</code> , and <code>A</code> through <code>F</code> .	<u>Listing krx203</u>	Me

Write an alternate version of `squeeze(s1, s2)` that deletes each character in the string `s1` that matches any character in the *string* `s2`.

Write the function `any(s1, s2)`, which returns the first location in the string `s1` where any character from the string `s2` occurs, or `-1` if `s1` contains no characters from `s2`. (The standard library function `strpbrk` does the same job but returns a pointer to the location.)

Write a function `setbits(x, p, n, y)` that returns `x` with the `n` bits that begin at position `p` set to the rightmost `n` bits of `y`, leaving the other bits unchanged.

Write a function `invert(x, p, n)` that returns `x` with the `n` bits that begin at position `p` inverted (i.e., 1 changed into 0 and vice versa), leaving the others unchanged.

Write a function `rightrot(x, n)` that returns the value of the integer `x` rotated to the right by `n` bit positions.

[Listing
krx204](#)

Me

[Listing
krx205](#)

The pedestrian solution is mine, and the clever one is by Partha Seetala.

[Listing
krx206](#)

Me

[Listing
krx207](#)

Gregory Pietsch

[Listing
krx208](#)

Gregory Pietsch

In a two's complement number system, $x \&= (x-1)$ deletes the rightmost 1-bit in x . Explain why. Use this observation to write a faster version of `bitcount`.

Rewrite the function `lower`, which converts upper case letters to lower case, with a conditional expression instead of `if-else`.

[Listing
krx209](#)

Gregory Pietsch

[Listing
krx210](#)

Bryan Williams

[Back](#)
[Home](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answers to Exercises

Chapter 3 - Control Flow

Ex No	Pg No	Exercise	Solution	Solved by
1	58	Our binary search makes two tests inside the loop, when one would suffice (at the price of more tests outside). Write a version with only one test inside the loop and measure the difference in run-time.	<u>Listing krx301</u>	Paul Griffiths, Colin Barker, Andrew Tesker
2	60	Write a function <code>escape(s, t)</code> that converts characters like newline and tab into visible escape sequences like <code>\n</code> and <code>\t</code> as it copies the string <code>t</code> to <code>s</code> . Use a <code>switch</code> . Write a function for the other direction as well, converting escape sequences into the real characters.	<u>Listing krx302</u>	Paul Griffiths
3	63	Write a function <code>expand(s1, s2)</code> that expands shorthand notations like <code>a-z</code> in the string <code>s1</code> into the equivalent complete list <code>abc...xyz</code> in <code>s2</code> . Allow for letters of either case and digits, and be prepared to handle cases like <code>a-b-c</code> and <code>a-z0-9</code> and <code>a-z</code> . Arrange that a leading or trailing <code>-</code> is taken literally.	<u>Listing krx303</u>	Paul Griffiths
4	64	In a two's complement number representation, our version of <code>itoa</code> does not handle the largest negative number, that is, the value of <code>n</code> equal to $-(2 \text{ to the power } (\text{wordsize} - 1))$. Explain why not. Modify it to print that value correctly regardless of the machine on which it runs.	<u>Listing krx304</u>	Paul Griffiths

Write the function `itob(n,s,b)` that converts the integer `n` into a base `b` character representation in the string `s`. In particular, `itob(n,s,16)` formats `n` as a hexadecimal integer in `s`.

Write a version of `itoa` that accepts three arguments instead of two. The third argument is a minimum field width; the converted number must be padded with blanks on the left if necessary to make it wide enough.

[Listing
krx305](#)

Paul Griffiths

[Listing
krx306](#)

Paul Griffiths

[Back](#)
[Home](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answers to Exercises

Chapter 4 - Functions and Program Structure

If you are intending to attempt solutions 4-3 through 4-10, you will find testing your solution to be a somewhat less laborious task if you download the following code:

[Kernighan and Ritchie's Polish calculator source](#)

It was when I was typing up this lot that the problem of copyright first occurred to me - after all, it's quite a substantial quote from the book. Fortunately for me, Dennis Ritchie had no objections (but he did check to make sure there wouldn't be any other verbatim quotes of comparable size!).

Ex No	Pg No	Exercise	Solution	Solved by
1	71	Write the function <code>strrindex(s, t)</code> , which returns the position of the <i>rightmost</i> occurrence of <code>t</code> in <code>s</code> , or -1 if there is none.	<u>Listing krx401</u>	Rick Dearman
2	73	Extend <code>atof</code> to handle scientific notation of the form <code>123.45e-6</code> where a floating-point number may be followed by <code>e</code> or <code>E</code> and an optionally signed exponent.	<u>Listing krx402</u>	Dann Corbit
3	79	Given the basic framework, it's straightforward to extend the calculator. Add the modulus (<code>%</code>) operator and provisions for negative numbers.	<u>Listing krx403</u>	Bob Wightman
4	79	Add commands to print the top element of the stack without popping, to duplicate it, and to swap the top two elements. Add a command to clear the stack.	<u>Listing krx404</u>	Bob Wightman

5	79	Add access to library functions like <code>sin</code> , <code>exp</code> , and <code>pow</code> . See <code><math.h></code> in Appendix B, Section 4.	Listing krx405	Bob Wightman
6	79	Add commands for handling variables. (It's easy to provide twenty-six variables with single-letter names.) Add a variable for the most recently printed value.	Listing krx406	Cat 1 Solution by Bob Wightman
7	79	Write a routine <code>ungets(s)</code> that will push back an entire string onto the input. Should <code>ungets</code> know about <code>buf</code> and <code>bufp</code> , or should it just use <code>ungetch</code> ?	Listing krx407	Steven Huang
8	79	Suppose that there will never be more than one character of pushback. Modify <code>getch</code> and <code>ungetch</code> accordingly.	Listing krx408	Steven Huang
9	79	Our <code>getch</code> and <code>ungetch</code> do not handle a pushed-back EOF correctly. Decide what their properties ought to be if an EOF is pushed back, then implement your design.	No solution yet posted.	----
10	79	An alternate organization uses <code>getline</code> to read an entire input line; this makes <code>getch</code> and <code>ungetch</code> unnecessary. Revise the calculator to use this approach.	No solution yet posted.	----
11	83	Modify <code>getop</code> so that it doesn't need to use <code>ungetch</code> . Hint: use an internal static variable.	Listing krx411	Gregory Pietsch
12	88	Adapt the ideas of <code>printd</code> to write a recursive version of <code>itoa</code> ; that is, convert an integer into a string by calling a recursive routine.	Listing krx412	Gregory Pietsch
13	88	Write a recursive version of the function <code>reverse(s)</code> , which reverses the string <code>s</code> in place.	Listing krx413	Gregory Pietsch
14	91	Define a macro <code>swap(t, x, y)</code> that interchanges two arguments of type <code>t</code> . (Block structure will help.)	Listing krx414	Gregory Pietsch, Lars Wirzenius

[Back](#)
[Home](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answers to Exercises

Chapter 5 - Pointers and Arrays

Ex No	Pg No	Exercise	Solution	Solved by
1	97	As written, <code>getint</code> treats a + or - not followed by a digit as a valid representation of zero. Fix it to push such a character back on the input.	<u>Listing krx501</u>	Gregory Pietsch
2	97	Write <code>getfloat</code> , the floating-point analog of <code>getint</code> . What type does <code>getfloat</code> return as its function value?	<u>Listing krx502</u>	Chris Mears, Gregory Pietsch
3	107	Write a pointer version of the function <code>strcat</code> that we showed in Chapter 2: <code>strcat(s,t)</code> copies the string <code>t</code> to the end of <code>s</code> .	<u>Listing krx503</u>	Me
4	107	Write the function <code>strend(s,t)</code> , which returns 1 if the string <code>t</code> occurs at the end of the string <code>s</code> , and zero otherwise.	<u>Listing krx504</u>	Bryan Williams
5	107	Write versions of the library functions <code>strncpy</code> , <code>strncat</code> , and <code>strncmp</code> , which operate on at most the first <code>n</code> characters of their argument strings. For example, <code>strncpy(s,t,n)</code> copies at most <code>n</code> characters of <code>t</code> to <code>s</code> . Full descriptions are in Appendix B.	<u>Listing krx505</u>	Lars Wirzenius

6	107	Rewrite appropriate programs from earlier chapters and exercises with pointers instead of array indexing. Good possibilities include <code>getline</code> (Chapters 1 and 4), <code>atoi</code> , <code>itoa</code> , and their variants (Chapters 2, 3, and 4), <code>reverse</code> (Chapter 3), and <code>strindex</code> and <code>getop</code> (Chapter 4).	Listing krx506	Gregory Pietsch
7	110	Rewrite <code>readlines</code> to store lines in an array supplied by <code>main</code> , rather than calling <code>alloc</code> to maintain storage. How much faster is the program?	Listing krx507	Steven Huang
8	112	There is no error-checking in <code>day_of_year</code> or <code>month_day</code> . Remedy this defect.	Listing krx508	Lars Wirzenius
9	114	Rewrite the routines <code>day_of_year</code> and <code>month_day</code> with pointers instead of indexing.	Listing krx509	Lars Wirzenius, Gregory Pietsch
10	118	Write the program <code>expr</code> , which evaluates a reverse Polish expression from the command line, where each operator or operand is a separate argument. For example, <code>expr 2 3 4 + *</code> evaluates $2 \times (3+4)$.	Listing krx510	Lars Wirzenius
11	118	Modify the programs <code>entab</code> and <code>datab</code> (written as exercises in Chapter 1) to accept a list of tab stops as arguments. Use the default tab settings if there are no arguments.	Listing krx511	Gregory Pietsch
12	118	Extend <code>entab</code> and <code>datab</code> to accept the shorthand <code>entab -m +n</code> to mean tab stops every n columns, starting at column m . Choose convenient (for the user) default behavior.	No solution yet posted.	----

Write the program `tail`, which prints the last `n` lines of its input. By default, `n` is 10, say, but it can be changed by an optional argument, so that

```
tail -n
```

prints the last `n` lines. The program should behave rationally no matter how unreasonable the input or the value of `n`. Write the program so it makes the best use of available storage; lines should be stored as in the sorting program of Section 5.6, not in a two-dimensional array of fixed size.

[Listing krx513](#)

Gregory Pietsch, Steven Huang

Modify the sort program to handle a `-r` flag, which indicates sorting in reverse (decreasing) order. Be sure that `-r` works with `-n`.

[Listing krx514](#)

Steven Huang

Add the option `-f` to fold upper and lower case together, so that case distinctions are not made during sorting; for example, `a` and `A` compare equal.

No solution yet posted.

Add the `-d` ("directory order") option, which makes comparisons only on letters, numbers and blanks. Make sure it works in conjunction with `-f`.

No solution yet posted.

Add a field-handling capability, so sorting may be done on fields within lines, each field sorted according to an independent set of options. (The index for this book was sorted with `-df` for the index category and `-n` for the page numbers.)

No solution yet posted.

Make `dc1` recover from input errors.

No solution yet posted.

19	126	Modify <code>undcl</code> so that it does not add redundant parentheses to declarations.	No solution yet posted.	----
20	126	Expand <code>dcl</code> to handle declarations with function argument types, qualifiers like <code>const</code> , and so on.	No solution yet posted.	----

[Back](#)[Home](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answers to Exercises

Chapter 6 - Structures

Ex No	Pg No	Exercise	Solution	Solved by
1	136	Our version of <code>getword</code> does not properly handle underscores, string constants, comments, or preprocessor control lines. Write a better version.	Listing krx601	Ben Pfaff
2	143	Write a program that reads a C program and prints in alphabetical order each group of variable names that are identical in the first 6 characters but different somewhere thereafter. Don't count words within strings and comments. Make 6 a parameter that can be set from the command line.	No solution yet posted.	----
3	143	Write a cross-referencer that prints a list of all words in a document, and, for each word, a list of the line numbers on which it occurs. Remove noise words like "the," "and," and so on.	Listing krx603	Me
4	143	Write a program that prints the distinct words in its input sorted into decreasing order of frequency of occurrence. Precede each word by its count.	Listing krx604	Bryan Williams
5	145	Write a function <code>undef</code> that will remove a name and definition from the table maintained by <code>lookup</code> and <code>install</code> .	Listing krx605	Paul Griffiths, Gregory Pietsch

6

145

Implement a simple version of the `#define` processor (i.e., no arguments) suitable for use with C programs, based on the routines of this section. You may also find `getch` and `ungetch` helpful.

No solution yet posted.

[Back](#)

[Home](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answers to Exercises

Chapter 7 - Input and Output

Ex No	Pg No	Exercise	Solution	Solved by
1	153	Write a program that converts upper case to lower or lower case to upper, depending on the name it is invoked with, as found in <code>argv[0]</code> .	Listing krx701	Richard Heathfield, Bryan Williams
2	155	Write a program that will print arbitrary input in a sensible way. As a minimum, it should print non-graphic characters in octal or hexadecimal according to local custom, and break long text lines.	Listing krx702	Me
3	156	Revise <code>minprintf</code> to handle more of the facilities of <code>printf</code> .	Listing krx703	Gregory Pietsch
4	159	Write a private version of <code>scanf</code> analogous to <code>minprintf</code> from the previous section.	No solution yet posted.	----
5	159	Rewrite the postfix calculator of Chapter 4 to use <code>scanf</code> and/or <code>sscanf</code> to do the input and number conversion.	No solution yet posted.	----
6	165	Write a program to compare two files, printing the first line where they differ.	Listing krx706	"flippant_squirrel"(!)

Modify the pattern finding program of Chapter 5 to take its input from a set of named files or, if no files are named as arguments, from the standard input. Should the file name be printed when a matching line is found?

Write a program to print a set of files, starting each new one on a new page, with a title and a running page count for each file.

Functions like `isupper` can be implemented to save space or to save time. Explore both possibilities.

No solution yet posted.

[Listing krx708](#)

[Listing krx709](#)

Steven Huang

Gregory Pietsch

[Back](#)
[Home](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answers to Exercises

Chapter 8 - The UNIX System Interface

Please note: these solutions use the Unix system interface, so they're not portable. Nevertheless, they're in K&R, so they appear here despite their inherent non-portability. In any case, they will *become* portable as soon as everyone's using Unix, as God intended.

Ex No	Pg No	Exercise	Solution	Solved by
1	174	Rewrite the program <code>cat</code> from Chapter 7 using <code>read</code> , <code>write</code> , <code>open</code> and <code>close</code> instead of their standard library equivalents. Perform experiments to determine the relative speeds of the two versions.	Listing krx801	Andrew Tesker
2	178	Rewrite <code>fopen</code> and <code>_fillbuf</code> with fields instead of explicit bit operations. Compare code size and execution speed.	No solution yet posted.	----
3	179	Design and write <code>_flushbuf</code> , <code>fflush</code> and <code>fclose</code> .	Listing krx803	Gregory Pietsch
4	179	The standard library function <pre>int fseek(FILE *fp, long offset, int origin)</pre> is identical to <code>lseek</code> except that <code>fp</code> is a file pointer instead of a file descriptor and the return value is an <code>int</code> status, not a position. Write <code>fseek</code> . Make sure that your <code>fseek</code> coordinates properly with the buffering done for the other functions of the library.	Listing krx804	Gregory Pietsch
5	184	Modify the <code>fsize</code> program to print the other information contained in the inode entry.	No solution yet posted.	----

6	189	The standard library function <code>calloc(n, size)</code> returns a pointer to <code>n</code> objects of size <code>size</code> , with the storage initialized to zero. Write <code>calloc</code> , by calling <code>malloc</code> or by modifying it.	Listing krx806	Bryan Williams
7	189	<code>malloc</code> accepts a size request without checking its plausibility; <code>free</code> believes that the block it is asked to free contains a valid size field. Improve these routines so they take more pains with error checking.	No solution yet posted.	----
8	189	Write a routine <code>bfree(p, n)</code> that will free an arbitrary block <code>p</code> of <code>n</code> characters into the free list maintained by <code>malloc</code> and <code>free</code> . By using <code>bfree</code> , a user can add a static or external array to the free list at any time.	No solution yet posted.	----

[Back](#)
[Home](#)

You are visitor number **456855** - call again soon!



Please support the anti-patent petition. Thank you.

Hi. I'm a programmer. This is my web site. If you find something useful here, great. If not, c'est la vie.

[CLINT - the library with no name](#)

This C library contains a number of useful routines for C programmers. For example, it has (rudimentary support for) a stretchy string type, and some reasonable (and portable) allocation wrappers. Opaque types (more or less) throughout, for safe programming. Enjoy!

[Other C Stuff](#)

Some more C stuff, a sort of hotch-potch of code that I've written for the heck of it, with no particular theme to it. It includes an error message management library, a directory parser demo (Windows only), and a couple of routines for reading arbitrarily-sized words and lines from a text input stream. Oh yes - and a noughts-and-crosses game. :-)

[The C Programming Language - Answers to Exercises](#)

Last updated 12 March 2002.

In 1988, Kernighan (pronounced Kern'-i-han' - the 'g' is silent) and Ritchie wrote the second edition of "The C Programming Language" - the revised tutorial and reference for the C language, which Ritchie himself designed. In that book, 97 exercises were posed. [Follow this link](#) to find a list of answers to those questions, written partly by me and partly by people from all over the Net. This list of answers will slowly become definitive (I give it ten years), as more and more people from Usenet slag off the existing answers. Not all the exercises have answers yet (in fact, 79 of the 97 exercises in the book have so far been addressed).

[Recommended C Books](#)

If you'd like to know which are the good C books, check out this list. No dummies or men in a hurry need bother - you won't find "C for Dummies" or "Teach Yourself C in 2.06 nanoseconds" on this list...

["C Unleashed" Author List and Chapter Summary](#)

I wrote a book. It's a good one. Lots of people helped me. It's called "C Unleashed" and it's available from all good bookshops. If it isn't available from your bookshop, I guess it can't be a very good shop. :-) Anyway, if you want to know more, click the link.

[Cryptology stuff](#)

This is the newest bit of the site, and is still quite raggedy. Still, if you're after some source code for key generation (Borland only) or monoalphabetic cipher cracking, here's some you can check out.

[CDX-2 Encryption](#)

The long-awaited update to CDX-1. This version incorporates S-boxes for added security, and is considerably quicker than CDX-1. In fact, it's so fast it's usable! Even if you're not interested in cryptography, you may find the bit rotation technique mildly diverting.

[CDX-1 Encryption](#)

After a couple of people sent me uncrackable encryption algorithms, neither of which took a particularly long time to crack, I decided to have a go at writing my own algorithm. CDX-1 uses a variable-length key and bit rotation with N rounds of encryption, where N is the key length. CDX-1 is quite slow, but nobody's managed to crack it yet and that's the main thing. If I find time, I'll have a go at speeding it up whilst retaining its security. At the above link, you'll find complete source code and a sample cipher. (Good luck cracking it...)

Update 19/4/2000: CDX-1 has a flaw. It still hasn't been cracked, but there's a possible attack on multiple messages of the same length encrypted using the same key (not terribly unlikely, if you consider network datagrams as the plaintext), because plaintexts which differ by only one bit produce ciphertexts which only differ by one bit.

As a result, I can no longer recommend CDX-1 to you unless you are going to use a one-time-pad for the key generation technique, in which case you probably don't need CDX anyway.

[Chat application \(client/server\) in VB](#)

I didn't write this. Some time ago, I was asked by four or five people in rapid succession how one would go about writing a chat application in Visual Basic. At the time, I was quite busy, so I asked Mark Pickering to write one for me; he very kindly did so, and gave me permission to publish it. I sent it to about a dozen people via email before I worked out that it would be a lot less hassle to simply post it on the Web. So here it is.

[Accessing Access in VB - adding data to Access tables](#)

Again, this one's by popular(ish) demand. I may add a second article on VB database programming if enough people express interest. So far, nobody has. ;-)

[Yahoo! Programming Room FAQs](#)

I spend far too much time in Yahoo! Chat, in the Programming room. I, and the other regulars in there, were getting sick of the same questions being asked over and over again, hence this FAQ. It's been described as "an FAQ with attitude". Enjoy!

[Check this out :-\)](#)

This MIDI file is a rather bizarre version of a tune which anyone from the UK will recognise immediately. It's less well known in the States. (Update: apparently the Yanks *do* know this tune after all.) If you have a MIDI keyboard, you're much more likely to 'get' it - on a soundcard it sounds a bit tinny.

[My bookshelf](#)

I've put off this project for a long time. But enough people have asked me to provide a brief synopsis of the books on my shelf that I've decided to at least make a start. So far, only a few books have made it onto my virtual bookshelf. I will continue to add others from my collection as and when I find time to type in the information. I've included short reviews of each book. **Update 11 September 2000: I've neglected this for far longer than I intended. I'm making no promises, but the intent is still to update, extend and improve this part of the site.**

[A small number of obfuscated C programs.](#)

The links that follow are mainly here for my own convenience, as short-cuts to places I don't need quite often enough to remember where they are, but often enough that I'd otherwise have to look them up... Feel free to use them if you are curious, interested, on a mission, or just want to get anywhere but here!

[Sarah Thompson's Home Page](#)

Every now and again, you meet a really bright person. If you're bright enough to know you're not all that bright, you might even recognise it when it happens to you. Sarah is a really amazingly bright person. I had the good fortune to work with her on some parsing software for a UK bank, and I learned a huge amount from her. If you're lucky enough to hire her, pay her lots, tell her what industry you're in, and then leave her alone to get on with making you lots of brilliant market-leading software. If not, well, go say hi to her Web site instead. :-)

[Loopy's Home Page](#)

Loopy is a networking guy. A good one. The best I know, in fact. He has given me a massive amount of networking support, he co-wrote Yahello! (my C++ Builder chat client, which I may get back to work on soon), and he taught me CGI programming. At his site you'll find all kinds of useful information about networking.

[Programmers who like beer, try here...](#)

The lyrics to "99 bottles of beer on the wall" in over 200 different programming languages! Some of these programs are fiendishly clever. Worth a look.

[comp.lang.c FAQs \(official\)](#)

[comp.lang.c FAQs \(mirror\)](#)

This is the official FAQ list for [comp.lang.c](#) . If you have a question about C programming, check here *first*, before you try any other resource. This document is highly authoritative, mainly because some of the best C programmers in the world have been using it regularly as a resource for helping others - in the process, they've spent several years getting the bugs out of it. If there are any mistakes remaining in this FAQ list, I'd be astonished. No printed book can claim as much - not even K&R, and indeed not even the printed version of this FAQ!

[C Programming - INfrequently Asked Questions!](#)

If you visit Seeb's IAQ page, please think twice before emailing him with corrections. All is not as it seems.

[The Jargon File](#)

The Jargon File contains thousands of references to hacker jargon (for a start, look up "hacker" - you might be surprised); be careful if you visit this site - you can easily spend a couple of hours there without realising it; it's a fascinating lexicography and cultural statement.

[Snippets](#)

This is an amazing resource. Dozens of useful programs written in C, with full source code provided. Credit card number validation, julian date routines, binary coded decimal conversion, checksumming, ISBN validation, and much much more, courtesy of Bob Stout and his gang.

[File Formats](#)

www.wotsit.org has docs on just about every file format known to mankind. Well, not quite. But when you're looking for a file format, this is *definitely* the first place to check.

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 1-1, page 8

Solution by Richard Heathfield

Run the "hello, world" program on your system. Experiment with leaving out parts of the program, to see what error messages you get.

Murphy's Law dictates that there is no single correct answer to the very first exercise in the book. Oh well. Here's a "hello world" program:

```
#include <stdio.h>

int main(void)
{
    printf("hello, world\n");
    return 0;
}
```

As you can see, I've added a `return` statement, because `main` always returns `int`, and it's good style to show this explicitly.

Here's a list of simple compile lines for a variety of popular compilers:

GNU C

```
gcc -W -Wall -ansi -pedantic -o hello hello.c
```

Microsoft C, up to version 5.1

```
cl -W3 -Zi -Od -AL hello.c
```

Microsoft C, version 6, Microsoft Visual C++ 1.0, 1.5

```
cl -W4 -Zi -Od -AL hello.c
```

Microsoft Visual C++ 2.0 and later

```
cl -W4 -Zi -Od hello.c
```

Turbo C++

```
tcc -A -ml hello.c (I think)
```

Borland C++, 16 bit versions

```
bcc -A -ml hello.c
```

Borland C++, 32 bit versions

```
bcc32 -A hello.c
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 1-2, page 8

Solution by [Richard Heathfield](#)

Experiment to find out what happens when `printf` 's argument string contains `\c`, where `c` is some character not listed above.

By 'above', the question is referring to:

`\n` (newline)

`\t` (tab)

`\b` (backspace)

`\"` (double quote)

`\\` (backslash) We have to tread carefully here, because using a non-specified escape sequence invokes [undefined behaviour](#).

The following program attempts to demonstrate all the legal escape sequences, not including the ones already shown (except `\n`, which I actually need in the program), and not including hexadecimal and octal escape sequences.

```
#include <stdio.h>
```

```
int main(void)
{
    printf("Audible or visual alert. \a\n");
    printf("Form feed. \f\n");
    printf("This escape, \r, moves the active position to the initial position of the\n");
    printf("current line.\n");
    printf("Vertical tab \v is tricky, as its behaviour is unspecified under certain\n");
    printf("conditions.\n");

    return 0;
}
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 1-3, page 13

Solution by Richard Heathfield

Modify the temperature conversion program to print a heading above the table.

```
#include <stdio.h>

int main(void)
{
    float fahr, celsius;
    int lower, upper, step;

    lower = 0;
    upper = 300;
    step = 20;

    printf("F      C\n\n");
    fahr = lower;
    while(fahr <= upper)
    {
        celsius = (5.0 / 9.0) * (fahr - 32.0);
        printf("%3.0f %6.1f\n", fahr, celsius);
        fahr = fahr + step;
    }
    return 0;
}
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 1-4, page 13

Solution by [Richard Heathfield](#)

Write a program to print the corresponding Celsius to Fahrenheit table.

```
#include <stdio.h>

int main(void)
{
    float fahr, celsius;
    int lower, upper, step;

    lower = 0;
    upper = 300;
    step = 20;

    printf("C      F\n\n");
    celsius = lower;
    while(celsius <= upper)
    {
        fahr = (9.0/5.0) * celsius + 32.0;
        printf("%3.0f %6.1f\n", celsius, fahr);
        celsius = celsius + step;
    }
    return 0;
}
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 1-5, page 14

Solutions by [Richard Heathfield](#) and [Chris Sidi](#)

Modify the temperature conversion program to print the table in reverse order, that is, from 300 degrees to 0.

This version uses a while loop:

```
#include <stdio.h>

int main(void)
{
    float fahr, celsius;
    int lower, upper, step;

    lower = 0;
    upper = 300;
    step = 20;

    printf("C      F\n\n");
    celsius = upper;
    while(celsius >= lower)
    {
        fahr = (9.0/5.0) * celsius + 32.0;
        printf("%3.0f %6.1f\n", celsius, fahr);
        celsius = celsius - step;
    }
    return 0;
}
```

This version uses a for loop:

```
#include <stdio.h>
```

```

int main(void)
{
    float fahr, celsius;
    int lower, upper, step;

    lower = 0;
    upper = 300;
    step = 20;

    printf("C      F\n\n");
    for(celsius = upper; celsius >= lower; celsius = celsius - step)
    {
        fahr = (9.0/5.0) * celsius + 32.0;
        printf("%3.0f %6.1f\n", celsius, fahr);
    }
    return 0;
}

```

Chris Sidi notes that Section 1.3 Has a short For statement example, and "Based on that example, I think the solution to 1.5:

- a) should do fahr to celsius conversion (whereas the solutions on your page do celsius to fahr)
- b) should be similar to the example and as small." He offers this solution:

```

#include <stdio.h>

/* print Fahrenheit-Celsius table */
int
main()
{
    int fahr;

    for (fahr = 300; fahr >= 0; fahr = fahr - 20)
        printf("%3d %6.1f\n", fahr, (5.0/9.0)*(fahr-32));

    return 0;
}

```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 1-6, page 17

Solution by [Richard Heathfield](#)

Verify that the expression `getchar() != EOF` is 0 or 1.

```

/* This program prompts for input, and then captures a character
 * from the keyboard. If EOF is signalled (typically through a
 * control-D or control-Z character, though not necessarily),
 * the program prints 0. Otherwise, it prints 1.
 *
 * If your input stream is buffered (and it probably is), then
 * you will need to press the ENTER key before the program will
 * respond.
 */

#include <stdio.h>

int main(void)
{
    printf("Press a key. ENTER would be nice :-)\n\n");
    printf("The expression getchar() != EOF evaluates to %d\n", getchar() != EOF);
    return 0;
}

```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 1-7, page 17

Solution by [Richard Heathfield](#)

Write a program to print the value of EOF .

```
#include <stdio.h>

int main(void)
{
    printf("The value of EOF is %d\n\n", EOF);

    return 0;
}
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 1-8, page 20

Solution by Richard Heathfield

Write a program to count blanks, tabs, and newlines.

```
#include <stdio.h>

int main(void)
{
    int blanks, tabs, newlines;
    int c;
    int done = 0;
    int lastchar = 0;

    blanks = 0;
    tabs = 0;
    newlines = 0;

    while(done == 0)
    {
        c = getchar();

        if(c == ' ')
            ++blanks;

        if(c == '\t')
            ++tabs;

        if(c == '\n')
            ++newlines;

        if(c == EOF)
        {
            if(lastchar != '\n')
            {
                ++newlines; /* this is a bit of a semantic stretch, but it copes
                             * with implementations where a text file might not
                             * end with a newline. Thanks to Jim Stad for pointing
```

```
        * this out.
        */

    }
    done = 1;
}
lastchar = c;
}

printf("Blanks: %d\nTabs: %d\nLines: %d\n", blanks, tabs, newlines);
return 0;
}
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 1-9, page 20

Solutions by [Richard Heathfield](#), [Chris Sidi](#), and [Stig Brautaset](#)

Write a program to copy its input to its output, replacing each string of one or more blanks by a single blank.

```

#include <stdio.h>

int main(void)
{
    int c;
    int inspace;

    inspace = 0;
    while((c = getchar()) != EOF)
    {
        if(c == ' ')
        {
            if(inspace == 0)
            {
                inspace = 1;
                putchar(c);
            }
        }

        /* We haven't met 'else' yet, so we have to be a little clumsy */
        if(c != ' ')
        {
            inspace = 0;
            putchar(c);
        }
    }

    return 0;
}

```

Chris Sidi writes: "instead of having an "inspace" boolean, you can keep track of the previous character and see if both the current character and previous character are spaces:"

```

#include <stdio.h>

```

```

/* count lines in input */
int
main()
{
    int c, pc; /* c = character, pc = previous character */

    /* set pc to a value that wouldn't match any character, in case
    this program is ever modified to get rid of multiples of other
    characters */

    pc = EOF;

    while ((c = getchar()) != EOF) {
        if (c == ' ')
            if (pc != ' ') /* or if (pc != c) */
                putchar(c);

        /* We haven't met 'else' yet, so we have to be a little clumsy */
        if (c != ' ')
            putchar(c);
        pc = c;
    }

    return 0;
}

```

Stig writes: "I am hiding behind the fact that break is mentioned in the introduction!"

```

#include <stdio.h>

int main(void)
{
    int c;
    while ((c = getchar()) != EOF) {
        if (c == ' ') {
            putchar(c);
            while((c = getchar()) == ' ' && c != EOF)
                ;
        }
        if (c == EOF)
            break; /* the break keyword is mentioned
            * in the introduction...
            * */
    }
}

```

```
        putchar(c);  
    }  
    return 0;  
}
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 1-10, page 20

Category 0 Solution by [Gregory Pietsch](#) Category 1 Solution by [Richard Heathfield](#)

Write a program to copy its input to its output, replacing each tab by `\t`, each backspace by `\b`, and each backslash by `\\`. This makes tabs and backspaces visible in an unambiguous way.

Category 0

Gregory Pietsch pointed out that my solution was actually Category 1. He was quite right. Better still, he was kind enough to submit a Category 0 solution himself. Here it is:

```
/* Gregory Pietsch <gkpl@flash.net> */

/*
 * Here's my attempt at a Category 0 version of 1-10.
 *
 * Gregory Pietsch
 */

#include <stdio.h>

int main()
{
    int c, d;

    while ( (c=getchar()) != EOF) {
        d = 0;
        if (c == '\\') {
            putchar('\\');
            putchar('\\');
            d = 1;
        }
        if (c == '\t') {
            putchar('\\');
            putchar('t');
            d = 1;
        }
    }
}
```

```

        if (c == '\\b') {
            putchar('\\');
            putchar('b');
            d = 1;
        }
        if (d == 0)
            putchar(c);
    }
    return 0;
}

```

Category 1

This solution, which I wrote myself, is the sadly discredited Cat 0 answer which has found a new lease of life in Category 1.

```

#include <stdio.h>

#define ESC_CHAR '\\\\'

int main(void)
{
    int c;

    while((c = getchar()) != EOF)
    {
        switch(c)
        {
            case '\\b':
                /* The OS on which I tested this (NT) intercepts \b characters. */
                putchar(ESC_CHAR);
                putchar('b');
                break;
            case '\\t':
                putchar(ESC_CHAR);
                putchar('t');
                break;
            case ESC_CHAR:
                putchar(ESC_CHAR);

```

```
        putchar(ESC_CHAR);
        break;
default:
    putchar(c);
    break;
    }
}
return 0;
}
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 1-11, page 21

Solution by Dann Corbit

How would you test the word count program? What kinds of input are most likely to uncover bugs if there are any?

It sounds like they are really trying to get the programmers to learn how to do a unit test. I would submit the following:

0. input file contains zero words
1. input file contains 1 enormous word without any newlines
2. input file contains all white space without newlines
3. input file contains 66000 newlines
4. input file contains word/{huge sequence of whitespace of different kinds}/word
5. input file contains 66000 single letter words, 66 to the line
6. input file contains 66000 words without any newlines
7. input file is `/usr/dict` contents (or equivalent)
8. input file is full collection of moby words
9. input file is binary (e.g. its own executable)
10. input file is `/dev/nul` (or equivalent)

66000 is chosen to check for integral overflow on small integer machines.

Dann suggests a followup exercise 1-11a: write a program to generate inputs (0,1,2,3,4,5,6)

I guess it was inevitable that I'd receive a solution for this followup exercise! Here is Gregory Pietsch's program to generate Dann's suggested inputs:

```
#include <assert.h>
#include <stdio.h>
```

```

int main(void)
{
    FILE *f;
    unsigned long i;
    static char *ws = " \f\t\v";
    static char *al = "abcdefghijklmnopqrstuvwxyz";
    static char *i5 = "a b c d e f g h i j k l m "
                      "n o p q r s t u v w x y z "
                      "a b c d e f g h i j k l m "
                      "n o p q r s t u v w x y z "
                      "a b c d e f g h i j k l m "
                      "n\n";

    /* Generate the following: */
    /* 0. input file contains zero words */
    f = fopen("test0", "w");
    assert(f != NULL);
    fclose(f);

    /* 1. input file contains 1 enormous word without any newlines */
    f = fopen("test1", "w");
    assert(f != NULL);
    for (i = 0; i < ((66000ul / 26) + 1); i++)
        fputs(al, f);
    fclose(f);

    /* 2. input file contains all white space without newlines */
    f = fopen("test2", "w");
    assert(f != NULL);
    for (i = 0; i < ((66000ul / 4) + 1); i++)
        fputs(ws, f);
    fclose(f);

    /* 3. input file contains 66000 newlines */
    f = fopen("test3", "w");
    assert(f != NULL);
    for (i = 0; i < 66000; i++)
        fputc('\n', f);
    fclose(f);

    /* 4. input file contains word/
     *     {huge sequence of whitespace of different kinds}

```

```

    *      /word
    */
f = fopen("test4", "w");
assert(f != NULL);
fputs("word", f);
for (i = 0; i < ((66000ul / 26) + 1); i++)
    fputs(ws, f);
fputs("word", f);
fclose(f);

/* 5. input file contains 66000 single letter words,
   *      66 to the line
   */
f = fopen("test5", "w");
assert(f != NULL);
for (i = 0; i < 1000; i++)
    fputs(i5, f);
fclose(f);

/* 6. input file contains 66000 words without any newlines */
f = fopen("test6", "w");
assert(f != NULL);
for (i = 0; i < 66000; i++)
    fputs("word ", f);
fclose(f);

return 0;
}

```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 1-12, page 21

Solution by [Richard Heathfield](#)

Write a program that prints its input one word per line.

```
#include <stdio.h>
int main(void)
{
    int c;
    int inspace;

    inspace = 0;
    while((c = getchar()) != EOF)
    {
        if(c == ' ' || c == '\t' || c == '\n')
        {
            if(inspace == 0)
            {
                inspace = 1;
                putchar('\n');
            }
            /* else, don't print anything */
        }
        else
        {
            inspace = 0;
            putchar(c);
        }
    }
    return 0;
}
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 1-13, page 24

Solution by Richard Heathfield

Write a program to print a histogram of the lengths of words in its input. It is easy to draw the histogram with the bars horizontal; a vertical orientation is more challenging.

```
/* This program was the subject of a thread in comp.lang.c, because of the way it
handled EOF.
 * The complaint was that, in the event of a text file's last line not ending with a
newline,
 * this program would not count the last word. I objected somewhat to this
complaint, on the
 * grounds that "if it hasn't got a newline at the end of each line, it isn't a text
file".
 *
 * These grounds turned out to be incorrect. Whether such a file is a text file
turns out to
 * be implementation-defined. I'd had a go at checking my facts, and had - as it
turns out -
 * checked the wrong facts! (sigh)
 *
 * It cost me an extra variable. It turned out that the least disturbing way to
modify the
 * program (I always look for the least disturbing way) was to replace the
traditional
 * while((c = getchar()) != EOF) with an EOF test actually inside the loop body.
This meant
 * adding an extra variable, but is undoubtedly worth the cost, because it means the
program
 * can now handle other people's text files as well as my own. As Ben Pfaff said at
the
 * time, "Be liberal in what you accept, strict in what you produce". Sound advice.
 *
 * The new version has, of course, been tested, and does now accept text files not
ending in
 * newlines.
 *
 * I have, of course, regenerated the sample output from this program. Actually,
there's no
 * "of course" about it - I nearly forgot.
 */
```

```
#include <stdio.h>
```

```
#define MAXWORDLEN 10
```

```

int main(void)
{
    int c;
    int inspace = 0;
    long lengtharr[MAXWORDLEN + 1];
    int wordlen = 0;

    int firstletter = 1;
    long thisval = 0;
    long maxval = 0;
    int thisidx = 0;
    int done = 0;

    for(thisidx = 0; thisidx <= MAXWORDLEN; thisidx++)
    {
        lengtharr[thisidx] = 0;
    }

    while(done == 0)
    {
        c = getchar();

        if(c == ' ' || c == '\t' || c == '\n' || c == EOF)
        {
            if(inspace == 0)
            {
                firstletter = 0;
                inspace = 1;

                if(wordlen <= MAXWORDLEN)
                {
                    if(wordlen > 0)
                    {
                        thisval = ++lengtharr[wordlen - 1];
                        if(thisval > maxval)
                        {
                            maxval = thisval;
                        }
                    }
                }
            }
            else
            {
                thisval = ++lengtharr[MAXWORDLEN];
                if(thisval > maxval)
                {
                    maxval = thisval;
                }
            }
        }
        if(c == EOF)
        {
            done = 1;
        }
    }
}

```

```

    }
}
else
{
    if(inspace == 1 || firstletter == 1)
    {
        wordlen = 0;
        firstletter = 0;
        inspace = 0;
    }
    ++wordlen;
}
}

for(thisval = maxval; thisval > 0; thisval--)
{
    printf("%4d | ", thisval);
    for(thisidx = 0; thisidx <= MAXWORDLEN; thisidx++)
    {
        if(lengthharr[thisidx] >= thisval)
        {
            printf("* ");
        }
        else
        {
            printf(" ");
        }
    }
    printf("\n");
}
printf("      +");
for(thisidx = 0; thisidx <= MAXWORDLEN; thisidx++)
{
    printf("----");
}
printf("\n      ");
for(thisidx = 0; thisidx < MAXWORDLEN; thisidx++)
{
    printf("%2d ", thisidx + 1);
}
printf(">%d\n", MAXWORDLEN);

return 0;
}

```

Here's the output of the program when given its own source as input:

```
113 | *
```


112		*		
111		*		
110		*		
109		*		
108		*		
107		*		
106		*		
105		*		
104		*		
103		*		
102		*		
101		*		
100		*		
99		*		
98		*		
97		*		
96		*		
95		*		
94		*	*	
93		*	*	
92		*	*	
91		*	*	
90		*	*	
89		*	*	
88		*	*	
87		*	*	
86		*	*	
85		*	*	
84		*	*	
83		*	*	
82		*	*	
81		*	*	
80		*	*	
79		*	*	
78		*	*	
77		*	*	
76		*	*	
75		*	*	
74		*	*	
73		*	*	
72		*	*	
71		*	*	
70		*	*	
69		*	*	
68		*	*	
67		*	*	
66		*	*	
65		*	*	
64		*	*	
63		*	*	*
62		*	*	*
61		*	*	*

60	*	*	*								
59	*	*	*								
58	*	*	*								
57	*	*	*								
56	*	*	*								
55	*	*	*								
54	*	*	*								
53	*	*	*								
52	*	*	*	*							
51	*	*	*	*	*						
50	*	*	*	*	*						
49	*	*	*	*	*						
48	*	*	*	*	*						
47	*	*	*	*	*						
46	*	*	*	*	*						
45	*	*	*	*	*						
44	*	*	*	*	*						
43	*	*	*	*	*		*				
42	*	*	*	*	*		*				
41	*	*	*	*	*		*				
40	*	*	*	*	*		*				
39	*	*	*	*	*		*				
38	*	*	*	*	*		*				
37	*	*	*	*	*		*				
36	*	*	*	*	*		*				
35	*	*	*	*	*		*				
34	*	*	*	*	*		*				
33	*	*	*	*	*		*				
32	*	*	*	*	*		*				
31	*	*	*	*	*		*				
30	*	*	*	*	*		*			*	
29	*	*	*	*	*		*			*	
28	*	*	*	*	*	*	*			*	
27	*	*	*	*	*	*	*			*	
26	*	*	*	*	*	*	*			*	
25	*	*	*	*	*	*	*	*		*	
24	*	*	*	*	*	*	*	*		*	
23	*	*	*	*	*	*	*	*		*	
22	*	*	*	*	*	*	*	*	*	*	
21	*	*	*	*	*	*	*	*	*	*	
20	*	*	*	*	*	*	*	*	*	*	
19	*	*	*	*	*	*	*	*	*	*	
18	*	*	*	*	*	*	*	*	*	*	
17	*	*	*	*	*	*	*	*	*	*	
16	*	*	*	*	*	*	*	*	*	*	
15	*	*	*	*	*	*	*	*	*	*	
14	*	*	*	*	*	*	*	*	*	*	*
13	*	*	*	*	*	*	*	*	*	*	*
12	*	*	*	*	*	*	*	*	*	*	*
11	*	*	*	*	*	*	*	*	*	*	*
10	*	*	*	*	*	*	*	*	*	*	*
9	*	*	*	*	*	*	*	*	*	*	*

8		*	*	*	*	*	*	*	*	*	*
7		*	*	*	*	*	*	*	*	*	*
6		*	*	*	*	*	*	*	*	*	*
5		*	*	*	*	*	*	*	*	*	*
4		*	*	*	*	*	*	*	*	*	*
3		*	*	*	*	*	*	*	*	*	*
2		*	*	*	*	*	*	*	*	*	*
1		*	*	*	*	*	*	*	*	*	*
	+	-----									
		1	2	3	4	5	6	7	8	9	10 >10

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 1-14, page 24

Solution by Richard Heathfield

Write a program to print a histogram of the frequencies of different characters in its input.

Naturally, I've gone for a vertical orientation to match exercise 13. I had some difficulty ensuring that the printing of the X-axis didn't involve cheating. I wanted to display each character if possible, but that would have meant using `isprint()`, which we haven't yet met. So I decided to display the value of the character instead. (The results below show the output on an ASCII system - naturally, a run on an EBCDIC machine would give different numbers.) I had to jump through a few hoops to avoid using the `%` operator which, again, we haven't yet met at this point in the text.

```
#include <stdio.h>

/* NUM_CHARS should really be CHAR_MAX but K&R haven't covered that at this stage in
the book */
#define NUM_CHARS 256

int main(void)
{
    int c;
    long freqarr[NUM_CHARS + 1];

    long thisval = 0;
    long maxval = 0;
    int thisidx = 0;

    for(thisidx = 0; thisidx <= NUM_CHARS; thisidx++)
    {
        freqarr[thisidx] = 0;
    }

    while((c = getchar()) != EOF)
    {
        if(c < NUM_CHARS)
        {
            thisval = ++freqarr[c];
            if(thisval > maxval)
            {
                maxval = thisval;
            }
        }
        else
        {
            thisval = ++freqarr[NUM_CHARS];
            if(thisval > maxval)
            {
```

```

        maxval = thisval;
    }
}

for(thisval = maxval; thisval > 0; thisval--)
{
    printf("%4d  |", thisval);
    for(thisidx = 0; thisidx <= NUM_CHARS; thisidx++)
    {
        if(freqarr[thisidx] >= thisval)
        {
            printf("*");
        }
        else if(freqarr[thisidx] > 0)
        {
            printf(" ");
        }
    }
    printf("\n");
}
printf("      +");
for(thisidx = 0; thisidx <= NUM_CHARS; thisidx++)
{
    if(freqarr[thisidx] > 0)
    {
        printf("-");
    }
}
printf("\n      ");
for(thisidx = 0; thisidx < NUM_CHARS; thisidx++)
{
    if(freqarr[thisidx] > 0)
    {
        printf("%d", thisidx / 100);
    }
}
printf("\n      ");
for(thisidx = 0; thisidx < NUM_CHARS; thisidx++)
{
    if(freqarr[thisidx] > 0)
    {
        printf("%d", (thisidx - (100 * (thisidx / 100))) / 10 );
    }
}
printf("\n      ");
for(thisidx = 0; thisidx < NUM_CHARS; thisidx++)
{
    if(freqarr[thisidx] > 0)
    {
        printf("%d", thisidx - (10 * (thisidx / 10)));
    }
}

```

```
}  
if(freqarr[NUM_CHARS] > 0)  
{  
    printf(">%d\n", NUM_CHARS);  
}  
  
printf("\n");  
  
return 0;  
}
```

Here's the output of the program when given its own source as input:

```
474 | *  
473 | *  
472 | *  
471 | *  
470 | *  
469 | *  
468 | *  
467 | *  
466 | *  
465 | *  
464 | *  
463 | *  
462 | *  
461 | *  
460 | *  
459 | *  
458 | *  
457 | *  
456 | *  
455 | *  
454 | *  
453 | *  
452 | *  
451 | *  
450 | *  
449 | *  
448 | *  
447 | *  
446 | *  
445 | *  
444 | *  
443 | *  
442 | *  
441 | *  
440 | *
```

439		*
438		*
437		*
436		*
435		*
434		*
433		*
432		*
431		*
430		*
429		*
428		*
427		*
426		*
425		*
424		*
423		*
422		*
421		*
420		*
419		*
418		*
417		*
416		*
415		*
414		*
413		*
412		*
411		*
410		*
409		*
408		*
407		*
406		*
405		*
404		*
403		*
402		*
401		*
400		*
399		*
398		*
397		*
396		*
395		*
394		*
393		*
392		*
391		*
390		*
389		*
388		*

387		*
386		*
385		*
384		*
383		*
382		*
381		*
380		*
379		*
378		*
377		*
376		*
375		*
374		*
373		*
372		*
371		*
370		*
369		*
368		*
367		*
366		*
365		*
364		*
363		*
362		*
361		*
360		*
359		*
358		*
357		*
356		*
355		*
354		*
353		*
352		*
351		*
350		*
349		*
348		*
347		*
346		*
345		*
344		*
343		*
342		*
341		*
340		*
339		*
338		*
337		*
336		*

335		*
334		*
333		*
332		*
331		*
330		*
329		*
328		*
327		*
326		*
325		*
324		*
323		*
322		*
321		*
320		*
319		*
318		*
317		*
316		*
315		*
314		*
313		*
312		*
311		*
310		*
309		*
308		*
307		*
306		*
305		*
304		*
303		*
302		*
301		*
300		*
299		*
298		*
297		*
296		*
295		*
294		*
293		*
292		*
291		*
290		*
289		*
288		*
287		*
286		*
285		*
284		*

283		*
282		*
281		*
280		*
279		*
278		*
277		*
276		*
275		*
274		*
273		*
272		*
271		*
270		*
269		*
268		*
267		*
266		*
265		*
264		*
263		*
262		*
261		*
260		*
259		*
258		*
257		*
256		*
255		*
254		*
253		*
252		*
251		*
250		*
249		*
248		*
247		*
246		*
245		*
244		*
243		*
242		*
241		*
240		*
239		*
238		*
237		*
236		*
235		*
234		*
233		*
232		*

231		*
230		*
229		*
228		*
227		*
226		*
225		*
224		*
223		*
222		*
221		*
220		*
219		*
218		*
217		*
216		*
215		*
214		*
213		*
212		*
211		*
210		*
209		*
208		*
207		*
206		*
205		*
204		*
203		*
202		*
201		*
200		*
199		*
198		*
197		*
196		*
195		*
194		*
193		*
192		*
191		*
190		*
189		*
188		*
187		*
186		*
185		*
184		*
183		*
182		*
181		*
180		*

179		*
178		*
177		*
176		*
175		*
174		*
173		*
172		*
171		*
170		*
169		*
168		*
167		*
166		*
165		*
164		*
163		*
162		*
161		*
160		*
159		*
158		*
157		*
156		*
155		*
154		*
153		*
152		*
151		*
150		*
149		*
148		*
147		*
146		*
145		*
144		*
143		*
142		*
141		*
140		*
139		*
138		*
137		*
136		*
135		*
134		*
133		*
132		*
131		*
130		*
129		*
128		*

127		*	
126		*	
125		*	
124		*	
123		*	
122		*	
121		*	
120		*	
119		*	
118		*	
117		*	
116		*	
115		*	
114		*	
113		*	
112		*	
111		*	
110		*	
109		*	*
108		*	*
107		*	*
106		*	*
105		*	*
104		*	*
103		*	*
102		*	*
101		*	*
100		*	*
99		*	*
98		*	*
97		**	*
96		**	*
95		**	*
94		**	*
93		**	*
92		**	*
91		**	*
90		**	*
89		**	*
88		**	*
87		**	*
86		**	*
85		**	*
84		**	*
83		**	*
82		**	*
81		**	*
80		**	*
79		**	*
78		**	*
77		**	*
76		**	*

```

75 | **                                     *
74 | **                                     *
73 | **                                     *
72 | **                                     *
71 | **                                     *          *
70 | **                                     *          *
69 | **                                     *          *
68 | **                                     *          *
67 | **                                     *          *
66 | **                                     *          *
65 | **                                     *          *
64 | **                                     *          *
63 | **                                     *          *
62 | **                                     *          *
61 | **                                     *          *
60 | **                                     *          *
59 | **                                     *        * *
58 | **                                     *        * *
57 | **                                     *        * *
56 | **                                     *        * *
55 | **                                     *        * *
54 | **                                     *        * *
53 | **                                     *        * *
52 | **                                     *        * *
51 | **                                     **       * *
50 | **                                     **       * *
49 | **                                     **       * * *
48 | **                                     **       * * *
47 | **                                     **       * * *
46 | **                                     **       * * *
45 | **                                     **       * * *
44 | **                                     **       * * *
43 | **                                     * **      * * *
42 | **                                     *  * * **  * * *
41 | **                                     *  * * **  * * *
40 | **          **                       *  * * **  * * *
39 | **          **                       *  * * **  * * *
38 | **          **                       *  * * **  * * *
37 | **          **                       *  * * **  * * *   *
36 | **          **                       *  * * **  * * *   *
35 | **          **                       *  * * **  * * *   *
34 | **          **                       *  * * **  * * *   *
33 | **          **                       *  * * **  * * *   *
32 | **          **                       *  * * **  *  * * *   *
31 | **          **                       *  * * **  *  * * *   *
30 | **          **                       *  * * **  *  * * *   *
29 | **          **                       *  * * **  *  * * *   *
28 | ** *        **                       *  * * * ** *  * * *   *
27 | ** *        **                       *  * * * ** *  * * *   *
26 | ** *        **                       *  * * * ** *  * * *   *
25 | ** *        **                       *  * * * ** *  * * *   *
24 | ** *        **                       *  * * * ** *  * * *   *

```

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 1-15, page 27

Solution by Richard Heathfield

Rewrite the temperature conversion program of Section 1.2 to use a function for conversion.

```
#include <stdio.h>

float FtoC(float f)
{
    float c;
    c = (5.0 / 9.0) * (f - 32.0);
    return c;
}

int main(void)
{
    float fahr, celsius;
    int lower, upper, step;

    lower = 0;
    upper = 300;
    step = 20;

    printf("F      C\n\n");
    fahr = lower;
    while(fahr <= upper)
    {
        celsius = FtoC(fahr);
        printf("%3.0f %6.1f\n", fahr, celsius);
        fahr = fahr + step;
    }
    return 0;
}
```


[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 1-16, page 30

Solutions by Richard Heathfield and "386sx"

Revise the main routine of the longest-line program so it will correctly print the length of arbitrarily long input lines, and as much as possible of the text.

```
/* This is the first program exercise where the spec isn't entirely
 * clear. The spec says, 'Revise the main routine', but the true
 * length of an input line can only be determined by modifying
 * getline. So that's what we'll do. getline will now return the
 * actual length of the line rather than the number of characters
 * read into the array passed to it.
 */

#include <stdio.h>

#define MAXLINE 1000 /* maximum input line size */

int getline(char line[], int maxline);
void copy(char to[], char from[]);

/* print longest input line */
int main(void)
{
    int len;                /* current line length */
    int max;                /* maximum length seen so far */
    char line[MAXLINE];     /* current input line */
    char longest[MAXLINE]; /* longest line saved here */

    max = 0;

    while((len = getline(line, MAXLINE)) > 0)
    {
        printf("%d: %s", len, line);

        if(len > max)
```

```

    {
        max = len;
        copy(longest, line);
    }
}
if(max > 0)
{
    printf("Longest is %d characters:\n%s", max, longest);
}
printf("\n");
return 0;
}

/* getline: read a line into s, return length */
int getline(char s[], int lim)
{
    int c, i, j;

    for(i = 0, j = 0; (c = getchar()) != EOF && c != '\n'; ++i)
    {
        if(i < lim - 1)
        {
            s[j++] = c;
        }
    }
    if(c == '\n')
    {
        if(i <= lim - 1)
        {
            s[j++] = c;
        }
        ++i;
    }
    s[j] = '\0';
    return i;
}

/* copy: copy 'from' into 'to'; assume 'to' is big enough */
void copy(char to[], char from[])
{
    int i;

    i = 0;

```

```

while((to[i] = from[i]) != '\0')
{
    ++i;
}
}

```

Chris Sidi, however, was not convinced - he thought this answer was "too easy", so he checked with bwk, who agreed. Chris writes: "Looks like Mr. Kernighan meant for "main routine" in Exercise 1-16 to refer to function main(), saying your solution of modifying getline() is "too easy." :) (Though I think your solution shouldn't be removed from the Answers web site, just complimented with another one that only modifies main())"

Cue Mr "386sx", riding to the rescue on a white horse...

```

/* Exercise 1-16 */

#include <stdio.h>

#define MAXLINE 20

int getline(char s[], int lim);
void copy(char to[], char from[]);

int main(void)
{
    char line[MAXLINE];
    char longest[MAXLINE];
    char temp[MAXLINE];
    int len, max, prevmax, getmore;

    max = prevmax = getmore = 0;
    while((len = getline(line, MAXLINE)) > 0)
    {
        if(line[len - 1] != '\n')
        {
            if(getmore == 0)
                copy(temp, line);
            prevmax += len;
        }
    }
}

```

```

        if(max < prevmax)
            max = prevmax;
        getmore = 1;
    }
    else
    {
        if(getmore == 1)
        {
            if(max < prevmax + len)
            {
                max = prevmax + len;
                copy(longest, temp);
                longest[MAXLINE - 2] = '\n';
            }
            getmore = 0;
        }

        else if(max < len)
        {
            max = len;
            copy(longest, line);
        }
        prevmax = 0;
    }
}
if(max > 0)
{
    printf("%s", longest);
    printf("len = %d\n", max);
}

return 0;
}

int getline(char s[], int lim)
{
    int c, i;

    for(i = 0;
        i < lim - 1 && ((c = getchar()) != EOF && c != '\n');
        ++i)
        s[i] = c;

```

```
    if(c == '\n')
    {
        s[i] = c;
        ++i;
    }
    else if(c == EOF && i > 0)
    {
        /* gotta do something about no newline preceding EOF */
        s[i] = '\n';
        ++i;
    }
    s[i] = '\0';
    return i;
}

void copy(char to[], char from[])
{
    int i;

    i = 0;
    while((to[i] = from[i]) != '\0')
        ++i;
}
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 1-17, page 31

Solution by "MJSR"

Write a program to print all input lines that are longer than 80 characters.

```
#include <stdio.h>

#define MINLENGTH 81

int readbuff(char *buffer) {
    size_t i=0;
    int c;
    while (i < MINLENGTH) {
        c = getchar();
        if (c == EOF) return -1;
        if (c == '\n') return 0;
        buffer[i++] = c;
    }
    return 1;
}

int copyline(char *buffer) {
    size_t i;
    int c;
    int status = 1;
    for(i=0; i<MINLENGTH; i++)
        putchar(buffer[i]);
    while(status == 1) {
        c = getchar();
        if (c == EOF)
            status = -1;
        else if (c == '\n')
            status = 0;
        else
            putchar(c);
    }
}
```

```
    putchar( '\\n' );  
    return status;  
}  
  
int main(void) {  
    char buffer[MINLENGTH];  
    int status = 0;  
    while (status != -1) {  
        status = readbuff(buffer);  
        if (status == 1)  
            status = copyline(buffer);  
    }  
    return 0;  
}
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 1-18, page 31

Solution by Ben Pfaff and a modification thereof by Chris Sidi

Write a program to remove all trailing blanks and tabs from each line of input, and to delete entirely blank lines.

```
/* K&R2 1-18 p31: Write a program to remove trailing blanks and tabs
   from each line of input, and to delete entirely blank lines.
```

The program specification is ambiguous: does "entirely blank lines" mean lines that contain no characters other than newline, or does it include lines composed of blanks and tabs followed by newline? The latter interpretation is taken here.

This implementation does not use any features not introduced in the first chapter of K&R2. As a result, it can't use pointers to dynamically allocate a buffer to store blanks that it has seen, so it must limit the number of blanks that are allowed to occur consecutively. (This is the value of MAXQUEUE, minus one.)

It is intended that this implementation "degrades gracefully." Even though a particular input might have 1000 or more blanks or tabs in a row, causing a problem for a single pass, multiple passes through the file will correct the problem. The program signals the need for such an additional pass by returning a failure code to the operating system. (EXIT_FAILURE isn't mentioned in the first chapter of K&R, but I'm making an exception here.) */

```
#include <stdio.h>
#include <stdlib.h>

#define MAXQUEUE 1001

int advance(int pointer)
{
    if (pointer < MAXQUEUE - 1)
        return pointer + 1;
    else
        return 0;
}

int main(void)
{
    char blank[MAXQUEUE];
```

```

int head, tail;
int nonspace;
int retval;
int c;

retval = nonspace = head = tail = 0;
while ((c = getchar()) != EOF) {
    if (c == '\n') {
        head = tail = 0;
        if (nonspace)
            putchar('\n');
        nonspace = 0;
    }
    else if (c == ' ' || c == '\t') {
        if (advance(head) == tail) {
            putchar(blank[tail]);
            tail = advance(tail);
            nonspace = 1;
            retval = EXIT_FAILURE;
        }

        blank[head] = c;
        head = advance(head);
    }
    else {
        while (head != tail) {
            putchar(blank[tail]);
            tail = advance(tail);
        }
        putchar(c);
        nonspace = 1;
    }
}

return retval;
}

```

Chris Sidi writes:

Ben,

I thought your solution to 1-18 was really neat (it didn't occur to me when I was doing the exercise), the way it degrades gracefully and multiple passes can get rid of huge blocks of whitespace.

However, if there is a huge block of non-trailing whitespace (eg "A",2000 spaces, "B\n") your program returns an error when there's not a need for it. And if someone were to use your program till it passes it will loop infinitely:

```
$ perl -e 'print "A"," "x2000,"B\n";' > in
$ until ./a.out < in > out; do echo failed, running another pass; cp out
in; done
failed, running another pass
failed, running another pass
failed, running another pass
[snip]
```

Below I have added a variable `spaceJustPrinted` to your program and check to see if the spaces printed early are trailing. I hope you like the minor improvement. (Though I can understand if you don't give a [1] :))

[1] expletive deleted - RJH.

```
/* K&R2 1-18 p31: Write a program to remove trailing blanks and tabs
from each line of input, and to delete entirely blank lines.
```

The program specification is ambiguous: does "entirely blank lines" mean lines that contain no characters other than newline, or does it include lines composed of blanks and tabs followed by newline? The latter interpretation is taken here.

This implementation does not use any features not introduced in the first chapter of K&R2. As a result, it can't use pointers to dynamically allocate a buffer to store blanks that it has seen, so it must limit the number of blanks that are allowed to occur consecutively. (This is the value of `MAXQUEUE`, minus one.)

It is intended that this implementation "degrades gracefully." Even though a particular input might have 1000 or more trailing blanks or tabs in a row, causing a problem for a single pass, multiple passes through the file will correct the problem. The program signals the need for such an additional pass by returning a failure code to the operating system. (`EXIT_FAILURE` isn't mentioned in the first chapter of K&R, but I'm making an exception here.) */

```
#include <stdio.h>
#include <stdlib.h>
```

```
#define MAXQUEUE 1001
```

```
int advance(int pointer)
```

```

{
    if (pointer < MAXQUEUE - 1)
        return pointer + 1;
    else
        return 0;
}

int main(void)
{
    char blank[MAXQUEUE];
    int head, tail;
    int nonspace;
    int retval;
    int c;
    int spaceJustPrinted; /*boolean: was the last character printed whitespace?*/

    retval = spaceJustPrinted = nonspace = head = tail = 0;

    while ((c = getchar()) != EOF) {
        if (c == '\n') {
            head = tail = 0;
            if (spaceJustPrinted == 1) /*if some trailing whitespace was printed...*/
                retval = EXIT_FAILURE;

            if (nonspace) {
                putchar('\n');
                spaceJustPrinted = 0; /* this instruction isn't really necessary since
                                     spaceJustPrinted is only used to determine the
                                     return value, but we'll keep this boolean
                                     truthful */
                nonspace = 0; /* moved inside conditional just to save a needless
                               assignment */
            }
        }
        else if (c == ' ' || c == '\t') {
            if (advance(head) == tail) {
                putchar(blank[tail]); /* these whitespace chars being printed early
                                     are only a problem if they are trailing,
                                     which we'll check when we hit a \n or EOF */

                spaceJustPrinted = 1;
                tail = advance(tail);
                nonspace = 1;
            }

            blank[head] = c;
            head = advance(head);
        }
        else {

```

```
    while (head != tail) {
        putchar(blank[tail]);
        tail = advance(tail);
    }
    putchar(c);
    spaceJustPrinted = 0;
    nonspace = 1;
}

/* if the last line wasn't ended with a newline before the EOF,
we'll need to figure out if trailing space was printed here */
if (spaceJustPrinted == 1) /*if some trailing whitespace was printed...*/
    retval = EXIT_FAILURE;

return retval;
}
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 1-19, page 31

Solution by Richard Heathfield

Write a function `reverse(s)` that reverses the character string `s`. Use it to write a program that reverses its input a line at a time.

```
#include <stdio.h>

#define MAX_LINE 1024

void discardnewline(char s[])
{
    int i;
    for(i = 0; s[i] != '\0'; i++)
    {
        if(s[i] == '\n')
            s[i] = '\0';
    }
}

int reverse(char s[])
{
    char ch;
    int i, j;

    for(j = 0; s[j] != '\0'; j++)
    {
    }

    --j;

    for(i = 0; i < j; i++)
    {
        ch = s[i];
        s[i] = s[j];
        s[j] = ch;
    }
}
```

```
    --j;
}

return 0;
}

int getline(char s[], int lim)
{
    int c, i;

    for(i = 0; i < lim - 1 && (c = getchar()) != EOF && c != '\n'; ++i)
    {
        s[i] = c;
    }

    if(c == '\n')
    {
        s[i++] = c;
    }

    s[i] = '\0';

    return i;
}

int main(void)
{
    char line[MAX_LINE];

    while(getline(line, sizeof line) > 0)
    {
        discardnewline(line);
        reverse(line);
        printf("%s\n", line);
    }
    return 0;
}
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 1-20, page 34

Solution by Richard Heathfield

Thanks to Rick Dearman for pointing out that my solution used `fgets()` which has not been introduced by page 34. I've fixed the solution to use K&R's `getline()` function instead. Further thanks to Roman Yablonovsky who, in Oct 2000, pointed out that the solution was buggy, and hinted at a fix. Basically, the problem he spotted was that my solution failed to keep track of the cumulative effect of multiple tabs in a single line. I've adopted his fix (which was in fact also slightly buggy, but I've fixed that too).

Write a program `detab` that replaces tabs in the input with the proper number of blanks to space to the next tab stop. Assume a fixed set of tab stops, say every `n` columns. Should `n` be a variable or a symbolic parameter?

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_BUFFER    1024
#define SPACE        ' '
#define TAB           '\t'

int CalculateNumberOfSpaces(int Offset, int TabSize)
{
    return TabSize - (Offset % TabSize);
}

/* K&R's getline() function from p29 */
int getline(char s[], int lim)
{
    int c, i;

    for(i = 0; i < lim - 1 && (c = getchar()) != EOF && c != '\n'; ++i)
        s[i] = c;
```

```
    if(c == '\\n')
    {
        s[i] = c;
        ++i;
    }
    s[i] = '\\0';

    return i;
}

int main(void)
{
    char Buffer[MAX_BUFFER];
    int TabSize = 5; /* A good test value */

    int i, j, k, l;

    while(getline(Buffer, MAX_BUFFER) > 0)
    {
        for(i = 0, l = 0; Buffer[i] != '\\0'; i++)
        {
            if(Buffer[i] == TAB)
            {
                j = CalculateNumberOfSpaces(l, TabSize);
                for(k = 0; k < j; k++)
                {
                    putchar(SPACE);
                    l++;
                }
            }
            else
            {
                putchar(Buffer[i]);
                l++;
            }
        }
    }

    return 0;
}
```

In answer to the question about whether n should be variable or symbolic, I'm tempted to offer the answer 'yes'. :-) Of course, it should be variable, to allow for modification of the value at runtime, for example via a command line argument, without requiring recompilation.

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 1-21, page 34

Category 0 Solution by [Rick Dearman](#)
 Category 1 Solution by [Stefan Farfeleder](#)

Write a program `entab` that replaces strings of blanks with the minimum number of tabs and blanks to achieve the same spacing. Use the same stops as for `detab`. When either a tab or a single blank would suffice to reach a tab stop, which should be given preference?

Rick Dearman's Cat 0 solution:

```

/*****
  KnR 1-21
  -----
  Write a program "entab" which replaces strings of
  blanks with the minimum number of tabs and blanks
  to achieve the same spacing.

  Author: Rick Dearman
  email: rick@ricken.demon.co.uk

*****/
#include <stdio.h>

#define MAXLINE 1000 /* max input line size */
#define TAB2SPACE 4 /* 4 spaces to a tab */

char line[MAXLINE]; /*current input line*/

int getline(void); /* taken from the KnR book. */

int
main( )
{
    int i,t;
    int spacecount,len;

```

```

while ( ( len = getline()) > 0 )
{
    spacecount = 0;
    for( i=0; i < len; i++)
    {
        if(line[i] == ' ')
            spacecount++; /* increment counter for each space */
        if(line[i] != ' ')
            spacecount = 0; /* reset counter */
        if(spacecount == TAB2SPACE) /* Now we have enough spaces
                                     ** to replace them with a tab
                                     */
        {
            /* Because we are removing 4 spaces and
            ** replacing them with 1 tab we move back
            ** three chars and replace the ' ' with a \t
            */
            i -= 3; /* same as "i = i - 3" */
            len -= 3;
            line[i] = '\t';
            /* Now move all the char's to the right into the
            ** places we have removed.
            */
            for(t=i+1;t<len;t++)
                line[t]=line[t+3];
            /* Now set the counter back to zero and move the
            ** end of line back 3 spaces
            */
            spacecount = 0;
            line[len] = '\0';
        }
    }
    printf("%s", line);
}
return 0;
}

/* getline: specialized version */
int getline(void)
{
    int c, i;
    extern char line[];

```

```

for ( i=0;i<MAXLINE-1 && ( c=getchar()) != EOF && c != '\n'; ++i)
    line[i] = c;
if(c == '\n')
{
    line[i] = c;
    ++i;
}
line[i] = '\0';
return i;
}

```

Stefan Farfeleder's Cat 1 solution:

```

/* 1-21.c */

#include <stdio.h>

#define TABSTOP 4

int main(void)
{
    size_t spaces = 0;
    int ch;
    size_t x = 0;           /* position in the line */
    size_t tabstop = TABSTOP; /* get this from the command-line
                               * if you want to */

    while ((ch = getchar()) != EOF)
    {
        if (ch == ' ')
        {
            spaces++;
        }
        else if (spaces == 0) /* no space, just printing */
        {
            putchar(ch);
            x++;
        }
        else if (spaces == 1) /* just one space, never print a tab */

```

```

{
    putchar(' ');
    putchar(ch);
    x += 2;
    spaces = 0;
}
else
{
    while (x / tabstop != (x + spaces) / tabstop)
        /* are the spaces reaching behind the next tabstop ? */
    {
        putchar('\t');
        x++;
        spaces--;
        while (x % tabstop != 0)
        {
            x++;
            spaces--;
        }
    }

    while (spaces > 0) /* the remaining ones are real space */
    {
        putchar(' ');
        x++;
        spaces--;
    }
    putchar(ch); /* now print the non-space char */
    x++;
}
if (ch == '\n')
{
    x = 0; /* reset line position */
}
}

return 0;
}

```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 1-22, page 34

Solution by [Rick Dearman](#)

Write a program to "fold" long input lines into two or more shorter lines after the last non-blank character that occurs before the n -th column of input. Make sure your program does something intelligent with very long lines, and if there are no blanks or tabs before the specified column.

Category 1 Solution

```

/*****
KnR 1-22
-----
Write a program that wraps very long lines of input
into two or more shorter lines.

Author: Rick Dearman
email: rick@ricken.demon.co.uk

*****/
#include <stdio.h>

#define MAXLINE 1000 /* max input line size */

char line[MAXLINE]; /*current input line*/

int getline(void); /* taken from the KnR book. */

int
main( )
{
    int t,len;
    int location,spaceholder;

```

```

const int FOLDLENGTH=70; /* The max length of a line */

while (( len = getline()) > 0 )
{
    if( len < FOLDLENGTH )
    {
    }
    else
    {
        /* if this is an extra long line then we
        ** loop through it replacing a space nearest
        ** to the foldarea with a newline.
        */
        t = 0;
        location = 0;
        while(t<len)
        {
            if(line[t] == ' ')
                spaceholder = t;

            if(location==FOLDLENGTH)
            {
                line[spaceholder] = '\n';
                location = 0;
            }
            location++;
            t++;
        }
        printf ( "%s", line);
    }
    return 0;
}

/* getline: specialized version */
int getline(void)
{
    int c, i;
    extern char line[];

    for ( i=0;i<MAXLINE-1 && ( c=getchar()) != EOF && c != '\n'; ++i)
        line[i] = c;
}

```

```
if(c == '\n')
{
    line[i] = c;
    ++i;
}
line[i] = '\0';
return i;
}
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 1-23, page 34

Solutions by

[Rick Dearman](#)

[Ben Pfaff](#) (two solutions)

[Lew Pitcher](#)

[Gregory Pietsch](#)

[Chris Torek](#)

[Chris Mears](#)

Critique by

[Rick Litherland](#)

(see end of page)

Write a program to remove all comments from a C program. Don't forget to handle quoted strings and character constants properly. C comments do not nest.

This was the first exercise to be posted as a fun "competition" on comp.lang.c, on 1 June 2000. As a result, there was a small flurry of submissions. Not all of them are completely working solutions. See the very end of this page for a test program which breaks most of them. :-)

Category 0 Solutions

From Rick Dearman

Now handles "/* comment in string */" correctly, but does not remove the comment from

```
return /* comment inside return statement */ 0;
```

```

/*****
"Write a program to remove all comments from a C program.
Don't forget to handle quoted strings and character
constants properly. C comments do not nest."

```

Author: Rick Dearman (rick@ricken.demon.co.uk)

```

*****/

```

```

#include <stdio.h>

```

```

#define MAXLINE 1000 /* max input line size */

```

```

char line[MAXLINE]; /*current input line*/

```

```

int getline(void); /* taken from the KnR book. */

```

```

int
main()
{
    int in_comment, len;
    int in_quote;
    int t;

    in_comment = in_quote = t = 0;
    while ((len = getline()) > 0 )
    {
        t=0;
        while(t < len)
        {
            if( line[t] == '"' )
                in_quote = 1;

            if( ! in_quote )
            {
                if( line[t] == '/' && line[t+1] == '*' )
                {
                    t=t+2;
                    in_comment = 1;
                }
                if( line[t] == '*' && line[t+1] == '/' )
                {
                    t=t+2;
                    in_comment = 0;
                }
                if(in_comment == 1)
                {
                    t++;
                }
            }
        }
    }
}

```

```

        else
        {
            printf ("%c", line[t]);
            t++;
        }
    }
    else
    {
        printf ("%c", line[t]);
        t++;
    }
}

return 0;
}

/* getline: specialized version */
int getline(void)
{
    int c, i;
    extern char line[];

    for ( i=0; i<MAXLINE-1 && ( c=getchar()) != EOF && c != '\n'; ++i)
        line[i] = c;
    if(c == '\n')
    {
        line[i] = c;
        ++i;
    }
    line[i] = '\0';
    return i;
}

```

From Ben Pfaff

This version is a bugfix for the code var/'\2'

```

/* K&R2 1-23: Write a program to remove all comments from a C program.
   Don't forget to handle quoted strings and character constants
   properly.  C comments do not nest.

```

This solution does not deal with other special cases, such as
 trigraphs, line continuation with \, or <> quoting on #include,

since these aren't mentioned up 'til then in K&R2. Perhaps this is cheating.

Note that this program contains both comments and quoted strings of text that looks like comments, so running it on itself is a reasonable test. It also contains examples of a comment that ends in a star and a comment preceded by a slash. Note that the latter will break C99 compilers and C89 compilers with `//` comment extensions.

Interface: The C source file is read from `stdin` and the comment-less output is written to `stdout`. `*/`

```
#include <stdio.h>

int
main(void)
{
#define PROGRAM 0
#define SLASH 1
#define COMMENT 2
#define STAR 3
#define QUOTE 4
#define LITERAL 5

    /* State machine's current state, one of the above values. */
    int state;

    /* If state == QUOTE, then ' or ". Otherwise, undefined. */
    int quote;

    /* Input character. */
    int c;

    state = PROGRAM;
    while ((c = getchar()) != EOF) {
        /* The following cases are in guesstimated order from most common
           to least common. */
        if (state == PROGRAM || state == SLASH) {
            if (state == SLASH) {
                /* Program text following a slash. */
                if (c == '/*')
                    state = COMMENT;
                else {
                    putchar('/');
                    state = PROGRAM;
                }
            }
        }
    }
}
```

```

    if (state == PROGRAM) {
        /* Program text. */
        if (c == '\\' || c == '"') {
            quote = c;
            state = QUOTE;
            putchar(c);
        }
        else if (c == "/*"[0])
            state = SLASH;
        else
            putchar(c);
    }
}
else if (state == COMMENT) {
    /* Comment. */
    if (c == "/*"[1])
        state = STAR;
}
else if (state == QUOTE) {
    /* Within quoted string or character constant. */
    putchar(c);
    if (c == '\\')
        state = LITERAL;
    else if (c == quote)
        state = PROGRAM;
}
else if (state == SLASH) {
}
else if (state == STAR) {
    /* Comment following a star. */
    if (c == '/')
        state = PROGRAM;
    else if (c != '*')
        state = COMMENT;
}
else /* state == LITERAL */ {
    /* Within quoted string or character constant, following \. */
    putchar(c);
    state = QUOTE;
}
}

if (state == SLASH)
    putchar('/', /* */
        1);

return 0;
}

```



```

/*
    Local variables:
    compile-command: "checkergcc -W -Wall -ansi -pedantic knr123-0.c -o knr123-0"
    End:
*/

```

From Lew Pitcher

```

/* Lew Pitcher <lpitcher@yesic.com> */

/**
**  derem - remove C comments
**
**  (attempt to solve K&R Exercise 1-22)
**
**  As I only have v1 copy of K&R, I cannot
**  be sure what is covered in K&R ANSI chapter 1.
**  So, I restrict myself to the components covered
**  in K&R v1 chapter 1, but modified for requisite ANSI
**  features (int main() and return value).
**
**  Components covered in v1 K&R chapter 1 include:
**  while (), for (), if () else
**  getchar(), putchar(), EOF
**  character constants, character escapes
**  strings
**  array subscripting
**
**  Not directly covered are
**  string subscripting ( "/"*"[0] )
**  initializers ( int state = PROGRAM; )
**/

/** */

#include <stdio.h>

#define PROGRAM          0
#define BEGIN_COMMENT    1
#define COMMENT          2
#define END_COMMENT      3
#define QUOTE            4

```

```

int main(void)
{
    int this_char, quote_char;
    int state;

    state = PROGRAM;

    while ((this_char = getchar()) != EOF)
    {
        if (state == PROGRAM)
        {
            if (this_char == '/')
                state = BEGIN_COMMENT;
            else if ((this_char == '"' || (this_char == '\\'))
            {
                state = QUOTE;
                putchar(quote_char = this_char);
            }
            else
                putchar(this_char);
        }
        else if (state == BEGIN_COMMENT)
        {
            if (this_char == '*')
                state = COMMENT;
            else
            {
                putchar('/'); /* for the '/' of the comment */
                if (this_char != '/')
                {
                    state = PROGRAM;
                    putchar(this_char);
                }
                else
                    state = COMMENT; /* stuttered */
            }
        }
        else if (state == QUOTE)
        {
            putchar(this_char);
            if (this_char == '\\')
                putchar(getchar()); /* escaped character */
            else if (this_char == quote_char)
                state = PROGRAM;
        }
        else if (state == COMMENT)
        {
            if (this_char == '*')
                state = END_COMMENT;
        }
        else if (state == END_COMMENT)

```

```

        {
            if (this_char == '/')
                state = PROGRAM;
            else if (this_char != '*')           /* stuttered */
                state = COMMENT;
        }
    }

    return 0;
}

```

From Gregory Pietsch

```

/* Gregory Pietsch <gkpl@flash.net> */

#include <stdio.h>

char p[] =
"0/!10\"040\"'050.001/011*!21\"/41\"'/51./02*!32.!23/ \"
"03*!33.!24\"004\"\\064.045\"'005\"\\075.056.047.05";

int main(){int c,i,d;char s,n;s='0';while((c=getchar())
!=EOF){d=0;for(i=0;p[i]!='\0'&&d==0;i=i+4){if(p[i]==s&&
(p[i+1]==c|p[i+1]=='.')){if(p[i+2]=='0')putchar(c);else
if(p[i+2]=='/'){putchar('/');putchar(c);}else if(p[i+2]
==' ')putchar(' ');n=p[i+3];d=1;}}s=n;}return 0;}

```

Category 1 Solutions

From Ben Pfaff (again)

This version has the var/'\2' bug fix.

```

/* K&R2 1-23: Write a program to remove all comments from a C program.
   Don't forget to handle quoted strings and character constants
   properly. C comments do not nest.

```

This solution does not deal with other special cases, such as trigraphs, line continuation with \, or <> quoting on #include, since these aren't mentioned up 'til then in K&R2. Perhaps this is

cheating.

Note that this program contains both comments and quoted strings of text that looks like comments, so running it on itself is a reasonable test. It also contains examples of a comment that ends in a star and a comment preceded by a slash. Note that the latter will break C99 compilers and C89 compilers with `//` comment extensions.

Interface: The C source file is read from `stdin` and the comment-less output is written to `stdout`. `*/`

```
#include <stdio.h>

int
main(void)
{
    /* State machine's current state. */
    enum {
        PROGRAM,
        SLASH,
        COMMENT,
        STAR,
        QUOTE,
        LITERAL
    } state;

    /* If state == QUOTE, then ' or ". Otherwise, undefined. */
    int quote;

    state = PROGRAM;
    for (;;) {
        int c = getchar();
        if (c == EOF) {
            if (state == SLASH)
                putchar('/' /***/
                        1 / 1 /\1');
            break;
        }

        switch (state) {
        case SLASH:
            /* Program text following a slash. */
            if (c == "/*"[1]) {
                state = COMMENT;
                break;
            }
            putchar('/');
            state = PROGRAM;
        }
```

```

    /* Fall through. */

case PROGRAM:
    /* Program text. */
    if (c == '\\' || c == '"') {
        quote = c;
        state = QUOTE;
        putchar(c);
    }
    else if (c == "/" *"[0])
        state = SLASH;
    else
        putchar(c);
    break;

case COMMENT:
    /* Comment. */
    if (c == '*')
        state = STAR;
    break;

case STAR:
    /* Comment following a star. */
    if (c == '/')
        state = PROGRAM;
    else if (c != '*') {
        state = COMMENT;
        putchar (' ');
    }
    break;

case QUOTE:
    /* Within quoted string or character constant. */
    putchar(c);
    if (c == '\\')
        state = LITERAL;
    else if (c == quote)
        state = PROGRAM;
    break;

case LITERAL:
    /* Within quoted string or character constant, following \. */
    putchar(c);
    state = QUOTE;
    break;

default:
    abort();
}

```

```

    }

    return 0;
}

/*
Local variables:
compile-command: "checkergcc -W -Wall -ansi -pedantic knr123.c -o knr123"
End:
*/

```

From Chris Torek

```

/* torek@elf.bsdi.com (Chris Torek) */

/*
"Write a program to remove all comments from a C program. Don't forget
to handle quoted strings and character constants properly. C comments do
not nest."

```

Well, what the heck. I mailed this a day or two ago, but here is the posted version. I modified the problem a bit: it removes comments from full ANSI C89 or C99 programs, handling trigraphs and \-newline sequences. It attempts to preserve any trigraphs in the output, even while examining them in the "C code" as their translated characters. (I am not sure why I bothered doing all of them, when only ??/ matters here.) It keeps output line numbers in sync with input line numbers, so that if the output is compiled, any error messages will refer back to the proper input source line.

Lightly tested.

```

*/

#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/*
 * This flag controls whether we do trigraph processing.
 */
int      trigraphs = 1;

/*

```

```

* This flag controls whether a comment becomes "whitespace" (ANSI C)
* or "nothing at all" (some pre-ANSI K&R C compilers).
*/
int      whitespace = 1;

/*
* This flag controls whether we do C89 or C99.  (C99 also handles C++.)
*/
int      c99;

/*
* These are global so that options() can get at them, and for later
* error messages if needed.
*/
const char *inname, *outname;

int options(const char *, char **);
void usage(void);

void      process(FILE *, FILE *);

#ifdef __GNUC__
void      panic(const char *) __attribute__((noreturn));
#else
void      panic(const char *);
#endif

int main(int argc, char **argv) {
    int i;
    FILE *in, *out;

    for (i = 1; i < argc; i++) {
        if (argv[i][0] == '-')
            i += options(argv[i] + 1, argv + i + 1);
        else if (inname == NULL)
            inname = argv[i];
        else
            usage();
    }
    if (inname != NULL) {
        if ((in = fopen(inname, "r")) == NULL) {
            fprintf(stderr, "cannot open %s for reading\n", inname);
            exit(EXIT_FAILURE);
        }
    } else {
        inname = "stdin";
        in = stdin;
    }
    if (outname != NULL) {

```

```

        if ((out = fopen(outname, "w")) == NULL) {
            fprintf(stderr, "cannot open %s for writing\n",
                outname);
            exit(EXIT_FAILURE);
        }
    } else {
        outname = "stdout";
        out = stdout;
    }
    process(in, out);
    fclose(in);
    fclose(out);
    exit(EXIT_SUCCESS);
}

```

```

/*
 * This scans for -o type options.  Options that have an argument
 * can either take it immediately or as a subsequent argument (e.g.,
 * -ofoo means the same thing as -o foo).  We return 0 for "handled
 * them normally", 1 for "handled them normally but needed more
 * arguments".
 *
 * Currently this function is more powerful than really needed, but
 * if we ever decide to have more arguments...
 */
int options(const char *afterdash, char **moreargs) {
    int nmore = 0, c;

    while ((c = *afterdash++) != '\0') {
        if (c == 'o') {
            if (*afterdash) {
                outname = afterdash;
                afterdash = "";
            } else if (moreargs[nmore] != NULL)
                outname = moreargs[nmore++];
            else
                usage();
        } else if (c == 't')
            trigraphs = 0;
        else if (c == 'w')
            whitespace = 0;
        else if (c == '9')
            c99 = 1;
        else
            usage();
    }
    return nmore;
}

```



```

void usage(void) {
    fprintf(stderr, "usage: uncomment [-9tw] [-o outfile] [infile]\n");
    exit(EXIT_FAILURE);    /* ??? */
}

/*
 * States, level 0:
 *     normal
 *     trigraph processing: Q1 Q2 (for ??x)
 *
 * States, level 1:
 *     backslash-newline processing: BACK (seen \, may consume NL)
 *
 * States, level 2:
 *     normal
 *     character constant: CC (seen '), CCBACK (seen \ inside CC)
 *     string constant: SC, SCBACK
 *     comment: SLASH, COMM, COMMSTAR (for /, in-comment, & seen-star)
 *     C99: SLASHSLASH
 */

enum l0state {
    L0_NORMAL,
    L0_Q1, L0_Q2
};
enum l1state {
    L1_NORMAL,
    L1_BACK
};
enum l2state {
    L2_NORMAL,
    L2_CC, L2_CCBACK,
    L2_SC, L2_SCBACK,
    L2_SLASH, L2_COMM, L2_COMMSTAR,
    L2_SLASHSLASH
};

struct state {
    FILE *in;
    enum l0state l0state;
    int npushback;
    char pushback[4];
    char pushorig[4];    /* nonzero => trigraph pushback */
    int lastgetc;
    int lineno;
};

/*
 * Set up "initial" state.

```

```

*/
static void state0(struct state *sp, FILE *in) {
    sp->in = in;
    sp->l0state = L0_NORMAL;
    sp->npushback = 0;
    sp->lastgetc = 0;
    sp->lineno = 1;
}

static void pushback(struct state *sp, int c, char origc) {
    assert(sp->npushback < sizeof sp->pushback);
    sp->pushback[sp->npushback] = c;
    sp->pushorig[sp->npushback++] = origc;
}

/*
 * Get a character, doing trigraph processing. Set *origc to 0 for normal
 * characters, or the actual input character pre-trigraph-mapping
 * for trigraph input.
 *
 * As a side effect, this can wind up getting up to 3 characters, maybe
 * stuffing two of them into the pushback buffer sp->buf[]. It also bumps
 * sp->lineno when a previously-read newline has been passed over.
 */
static int getl0char(struct state *sp, char *origc) {
    int c, newc;
    enum l0state state;

    state = sp->l0state;
    *origc = 0;
    while ((c = getc(sp->in)) != EOF) {
        if (sp->lastgetc == '\n')
            sp->lineno++;
        sp->lastgetc = c;
        switch (state) {

            case L0_NORMAL:
                /* ? => get another character; otherwise we are ok */
                if (c == '?') {
                    state = L0_Q1;
                    continue;
                }
                assert(sp->l0state == L0_NORMAL);
                return c;

            case L0_Q1:
                /* ?? => get another character */
                if (c == '?') {
                    state = L0_Q2;

```

```

        continue;
    }
    /* ?X => return ?, look at X later */
    pushback(sp, c, 0);
    sp->l0state = L0_NORMAL;
    return '?';

case L0_Q2:
    /*
     * ??X, where X is trigraph => map
     * ??X, where X is non-trigraph => tricky
     * ??? => also tricky
     */
    switch (c) {
case '=':
        newc = '#';
        break;
case '(':
        newc = '[';
        break;
case '/':
        newc = '\\';
        break;
case ')':
        newc = ']';
        break;
case '\\':
        newc = '^';
        break;
case '<':
        newc = '{';
        break;
case '!':
        newc = '|';
        break;
case '>':
        newc = '}';
        break;
case '?':
        /*
         * This one is slightly tricky. Three '?'s
         * mean that the '?' we read two characters
         * ago gets returned, and the two remaining
         * '?'s leave us in Q2 state.
         */
        sp->l0state = L0_Q2;
        return '?';
default:
        /*

```

```

        * This one returns the first ?, leaves
        * the second ? to be re-examined, and
        * leaves the last character to be re-examined.
        * In any case we are back in "normal" state.
        */
        pushback(sp, c, 0);
        pushback(sp, '?', 0);
        sp->l0state = L0_NORMAL;
        return '?';
    }
    /* mapped a trigraph char -- return new char */
    *origc = c;
    sp->l0state = L0_NORMAL;
    return newc;

    default:
        panic("getl0char state");
    }
}
sp->lastgetc = EOF;
return EOF;
}

void warn(struct state *, const char *);

void process(FILE *in, FILE *out) {
    enum l1state l1state = L1_NORMAL;
    enum l2state l2state = L2_NORMAL;
    int c, pendnls;
    char origc, backc;
    struct state state;

    state0(&state, in);
    pendnls = 0;
    backc = 0;                /* defeat gcc warning */

    /*
     * Slight sort-of-bug: files ending in \ cause two "final" getc()s.
     */
    do {
        if (state.npushback) {
            c = state.pushback[--state.npushback];
            origc = state.pushorig[state.npushback];
        } else if (trigraphs) {
            c = getl0char(&state, &origc);
        } else {
            c = getc(in);
            origc = 0;
            if (state.lastgetc == '\n')

```

```

        state.lineneno++;
        state.lastgetc = c;
    }

    /*
     * Do backslash-newline processing.
     */
    switch (llstate) {

    case L1_NORMAL:
        if (c == '\\') {
            llstate = L1_BACK;
            backc = origc;
            continue;
        }
        break;

    case L1_BACK:
        /*
         * If backc is nonzero here, the backslash that
         * got us into this state was spelled ??/ --
         * if we eat a newline (and hence the backslash),
         * we forget that the eaten newline was spelled
         * this way.  This is sort of a bug, but so it goes.
         */
        llstate = L1_NORMAL;
        if (c == '\\n') {
            pendnls++;
            continue;
        }
        if (c != EOF)
            pushback(&state, c, origc);
        c = '\\';
        origc = backc;
        break;

    default:
        panic("bad llstate");
    }

    /*
     * Now ready to do "C proper" processing.
     */

```

```

#define SYNCLINES()    while (pendnls) putc('\\n', out), pendnls--
#define OUTPUT(ch, tri) ((tri) ? fprintf(out, "??%c", tri) : putc(ch, out))
#define COPY()        OUTPUT(c, origc)

```

```

    switch (l2state) {
    case L2_NORMAL:

```

```

        switch (c) {
        case '\\':
            l2state = L2_CC;
            break;
        case '"':
            l2state = L2_SC;
            break;
        case '/':
            l2state = L2_SLASH;
            continue;
        default:
            break;
        }
        SYNCLINES();
        if (c != EOF)
            COPY();
        break;

    case L2_CC:
        switch (c) {
        case EOF:
            warn(&state, "EOF in character constant");
            break;
        case '\n':
            warn(&state, "newline in character constant");
            break;
        case '\\':
            l2state = L2_CCBACK;
            break;
        case '\\':
            l2state = L2_NORMAL;
            break;
        default:
            break;
        }
        if (c != EOF)
            COPY();
        break;

    case L2_CCBACK:
        switch (c) {
        case EOF:
            warn(&state, "EOF in character constant");
            break;
        case '\n':
            warn(&state, "newline in character constant");
            break;
        default:
            break;
        }

```

```

    }
    l2state = L2_CC;
    if (c != EOF)
        COPY();
    break;

case L2_SC:    /* much like CC */
    switch (c) {
    case EOF:
        warn(&state, "EOF in string constant");
        break;
    case '\n':
        warn(&state, "newline in string constant");
        break;
    case '\\':
        l2state = L2_SCBACK;
        break;
    case '"':
        l2state = L2_NORMAL;
        break;
    default:
        break;
    }
    if (c != EOF)
        COPY();
    break;

case L2_SCBACK:
    switch (c) {
    case EOF:
        warn(&state, "EOF in string constant");
        break;
    case '\n':
        warn(&state, "newline in string constant");
        break;
    default:
        break;
    }
    l2state = L2_SC;
    if (c != EOF)
        COPY();
    break;

case L2_SLASH:
    if (c == '*')
        l2state = L2_COMM;
    else if (c99 && c == '/')
        l2state = L2_SLASHSLASH;
    else {

```

```

        SYNCLINES();
        OUTPUT('/', 0);
        if (c != '/') {
            if (c != EOF)
                COPY();
            l2state = L2_NORMAL;
        }
    }
    break;

case L2_COMM:
    switch (c) {
        case '*':
            l2state = L2_COMMSTAR;
            break;
        case '\n':
            pendnls++;
            break;
        case EOF:
            warn(&state, "EOF inside comment");
            break;
    }
    break;

case L2_COMMSTAR:
    switch (c) {
        case '/':
            l2state = L2_NORMAL;
            /*
             * If comments become whitespace,
             * and we have no pending newlines,
             * must emit a blank here.
             *
             * The comment text is now all eaten.
             */
            if (whitespace && pendnls == 0)
                putc(' ', out);
            SYNCLINES();
            break;
        case '*':
            /* stay in L2_COMMSTAR state */
            break;
        case EOF:
            warn(&state, "EOF inside comment");
            break;
        case '\n':
            pendnls++;
            /* FALLTHROUGH */
        default:

```



```

        l2state = L2_COMM;
    }
    break;

case L2_SLASHSLASH:
    switch (c) {
    case EOF:
        /* ??? do we really care? */
        warn(&state, "EOF inside //-comment");
        break;
    case '\n':
        l2state = L2_NORMAL;
        pendnls++;          /* cheesy, but... */
        SYNCLINES();
    default:
        break;
    }
    break;

default:
    panic("bad l2state");
}
} while (c != EOF);
SYNCLINES();
}

```

```

void warn(struct state *sp, const char *msg) {
    fprintf(stderr, "uncomment: %s(%d): %s\n", inname, sp->lineno, msg);
}

```

```

void panic(const char *msg) {
    fprintf(stderr, "panic: %s\n", msg);
    abort();
    exit(EXIT_FAILURE);
}

```

From Chris Mears

Here's Chris's updated version, without the bugs (says he). :-)

```

/*
 * C comment stripper.
 *
 * Strips comments from C or C++ code.
 */

```

```

#include <stdio.h>

enum state_t { normal, string, character, block_comment, line_comment};

enum token_t { none, backslash, slash, star, tri1, tri2, tri_backslash};

static int print_mode(enum state_t s)
{
    return (s == normal || s == string || s == character);
}

void cstrip(FILE *infile, FILE *outfile)
{
    int ch;
    int comment_newline = 0;
    enum state_t state = normal;
    enum token_t token = none;
    enum token_t last_token = none;

    if (!infile || !outfile || (infile == outfile)) {
        return;
    }

    while ((ch = fgetc(infile)) != EOF) {
        switch (ch) {
            case '/':
                if (token == tri2) {
                    token = tri_backslash;
                    if (print_mode(state))
                        fputc(ch, outfile);
                } else if (state == string || state == character) {
                    fputc(ch, outfile);
                    token = slash;
                } else if (state == block_comment && token == star) {
                    state = normal;
                    token = none;
                }

                /* Replace block comments with whitespace. */
                if (comment_newline) {
                    fputc('\n', outfile);
                } else {
                    fputc(' ', outfile);
                }
            } else if (state == normal && token == slash) {
                state = line_comment;
                token = slash;
            } else {
                token = slash;
            }
        }
    }
}

```

```

    }

    break;

case '\\':
    if (state == normal && token == slash)
        fputc('/', outfile);
    if (print_mode(state))
        fputc(ch, outfile);

    if (token == backslash || token == tri_backslash) {
        token = none;
    } else {
        last_token = token;
        token = backslash;
    }

    break;

case '":
    if (state == normal && token == slash)
        fputc('/', outfile);
    if (state == string && token != backslash)
        state = normal;
    else if (state == normal && token != backslash)
        state = string;

    if (print_mode(state))
        fputc(ch, outfile);

    token = none;

    break;

case '\':
    if (state == normal && token == slash)
        fputc('/', outfile);
    if (state == character && token != backslash)
        state = normal;
    else if (state == normal && token != backslash)
        state = character;

    if (print_mode(state))
        fputc(ch, outfile);

    token = none;

    break;

```

```

case '\n':
    /* This test is independent of the others. */
    if (state == block_comment)
        comment_newline = 1;

    if (state == normal && token == slash)
        fputc('/', outfile);

    if (token == backslash || token == tri_backslash)
        token = last_token;
    else if (state == line_comment &&
             token != backslash) {
        state = normal;
        token = none;
    } else {
        token = none;
    }

    if (print_mode(state))
        fputc(ch, outfile);

    break;

case '*':
    if (state == normal && token == slash) {
        state = block_comment;
        token = none;
        comment_newline = 0;
    } else {
        token = star;
    }

    if (print_mode(state))
        fputc(ch, outfile);

    break;

case '?':
    if (state == normal && token == slash)
        fputc('/', outfile);

    if (token == tri1) {
        token = tri2;
    } else if (token == tri2) {
        token = tri2;    /* retain state */
    } else {
        /* We might need the last token if this
         * trigraph turns out to be a backslash.
         */

```

```

        last_token = token;
        token = tr1;
    }

    if (print_mode(state))
        fputc(ch, outfile);

    break;

default:
    if (state == normal && token == slash)
        fputc('/', outfile);

    if (print_mode(state))
        fputc(ch, outfile);

    token = none;

    break;
} /* switch */

} /* while */

return;
}

/* Small driver program. */

int main(void)
{
    cstrip(stdin, stdout);

    return 0;
}

```

Here's a critique of the above, sent in by Rick Litherland. (Please note: when Rick posted this, I hadn't yet posted Chris Mears's updated version of the code.)

(Since I find it hard to pick the solution number out of KRX12300.C at a glance, I'll refer to the solutions as uncomment00, uncomment01, and so on.)

[Rick - KR means K&R. X means eXercise. 1 means Chapter 1. 23 means exercise 23. The next digit is the category number - 0 == Cat 0 (ANSI C89, with code restricted to what K&R have discussed at this point in the book). The final

digit is the solution number. 0 is the first I received in that category, 1 is the second, and so on. (RJH)]

uncomment03 (Gregory Pietsch)

=====

I can find only one possible flaw in this, namely that it does not allow for a slash in program text being immediately followed by a quotation mark. One could reasonably argue that this is not a flaw at all, because that would never happen in sensible code. On the other hand, it can happen in legal code, as demonstrated by the following complete (if useless) program.

```
#include <stdio.h>
int main(void)
{
    /* print the number three */
    printf("%d\n", 6/'\2');
    /* remember to return a value from main */
    return 0;
}
```

When this is fed to uncomment03, the output is

```
#include <stdio.h>
int main(void)
{

    printf("%d\n", 6/'\2');
    /* remember to return a value from main */
    return 0;
}
```

Clearly, uncomment03 realises that the second comment is too important to remove. Um, sorry, that was a feeble excuse for a joke. What's happening is that uncomment03 doesn't recognise the beginning of the character constant '\2', so it takes the closing quote as the start of a "character constant" that is never terminated. The peculiar idiom 6/'\2' for 3 can be replaced by the even more brain-damaged 6/"\2"[0] with the same effect. Since uncomment03 is table-driven, it's easy to make it recognise these situations by adding two new rules to the table.

```
/* modified krx12303.c */
#include <stdio.h>

char p[] =
"0/!10\"@40\"'@50.@01/@11*!2"
"1\"/41\"'/5"                /* added by RAL */
"1./02*!32.!23/ 03*!33.!24\"@04\\\"@64.@45\"'@05\\\"@75.@56.@47.@5";
```

```
int main(){int c,i,d;char s,n;s='0';while((c=getchar())
!=EOF){d=0;for(i=0;p[i]!='\0'&&d==0;i=i+4){if(p[i]==s&&
(p[i+1]==c | p[i+1]=='. ')){if(p[i+2]=='@')putchar(c);else
```

```

if(p[i+2]=='/') {putchar('/');putchar(c);}else if(p[i+2]
==' ')putchar(' ');n=p[i+3];d=1;}}s=n;}return 0;}
/* end of modified krx12303.c */

```

uncomment02 (Lew Pitcher)

```
=====
```

uncomment11 (Chris Torek)

```
=====
```

These have the same problem (or non-problem, according to your point of view) as uncomment03. If it were regarded as a problem, it could probably be fixed quite easily, though not (I think) as neatly as with uncomment03; I haven't looked at these carefully enough to be sure.

uncomment01, uncomment10 (Ben Pfaff)

```
=====
```

An oversight has the effect that if a slash in program text is followed by anything other than a star or another slash, the following character is dropped. For example, with input

```
int a = 4/2;
```

the output is

```
int a = 4/;
```

The correction is the same in both cases; replace

```

/* Program text following a slash. */
if (c == '*')
    state = COMMENT;
else {
    putchar('/');
    if (c != '/')
        state = PROGRAM;
}

```

by

```

/* Program text following a slash. */
if (c == '*')
    state = COMMENT;
else {
    putchar('/');
}

```

```

    if (c != '/') {
        putchar(c);
        state = PROGRAM;
    }
}

```

After this, these programs will have the same problem (or not) as the previous three.

uncomment12 (Chris Mears)

=====

This is a completely different kettle of fish. If you run this with Ben Pfaff's solution as input, the output is quite bizarre; some comments have just their initial and final slashes removed, for instance. I've managed to find two things contributing to this. The first is illustrated by the input

```
int c = '/';
```

with output

```
int c = '';
```

This can be fixed by changing the lines

```

case '/':
    if (state == string) {

```

to

```

case '/':
    if (state == string || state == character) {

```

However, with or without this change, the input

```
char *p = "\\"; /* This is not a comment. */
```

is left unchanged. What happens is that the closing quote of the string literal isn't recognised as such because of the preceding backlash, despite the backlash before that. The handling of backslashes is split between three cases (at least), and is complicated enough that I don't feel competent to propose a remedy.

This program breaks most of the above submissions:

```
/* krx123tp.c - a test program to serve as input to krx123*.c
```



```

*
* This is a shameless copy of Ben Pfaff's solution, to which I have
* added a few extra statements to further test the candidate programs
* for this exercise. As Ben says, this program already contains lots
* of examples of comments and not-quite-comments. I've just made it
* a little tougher.
*
*/

```

```

/* K&R2 1-23: Write a program to remove all comments from a C program.
Don't forget to handle quoted strings and character constants
properly. C comments do not nest.

```

This solution does not deal with other special cases, such as trigraphs, line continuation with `\`, or `<>` quoting on `#include`, since these aren't mentioned up 'til then in K&R2. Perhaps this is cheating.

Note that this program contains both comments and quoted strings of text that looks like comments, so running it on itself is a reasonable test. It also contains examples of a comment that ends in a star and a comment preceded by a slash. Note that the latter will break C99 compilers and C89 compilers with `//` comment extensions.

Interface: The C source file is read from `stdin` and the comment-less output is written to `stdout`. `**/`

```

#include <stdio.h>

```

```

int
main(void)
{
    /* State machine's current state. */
    enum {
        PROGRAM,
        SLASH,
        COMMENT,
        STAR,
        QUOTE,
        LITERAL
    } state;

    /* If state == QUOTE, then ' or ". Otherwise, undefined. */
    int quote;

    state = PROGRAM;
    for (;;) {
        int c = getchar();

```

```

if (c == EOF) {
    if (state == SLASH)
        putchar('/', /* */
            1 / 1 /\1');
    break;
}

if(0)
    printf("%d\n", 6/\2');
/* line of code, and comment, added by RJH 10 July 2000 */

switch (state) {
case SLASH:
    /* Program text following a slash. */
    if (c == "/*"[1]) {
        state = COMMENT;
        break;
    }
    putchar('/');
    state = PROGRAM;
    /* Fall through. */

case PROGRAM:
    /* Program text. */
    if (c == '\'' || c == '"') {
        quote = c;
        state = QUOTE;
        putchar(c);
    }
    else if (c == "/*"[0])
        state = SLASH;
    else
        putchar(c);
    break;

case COMMENT:
    /* Comment. */
    if (c == '*')
        state = STAR;
    break;

case STAR:
    /* Comment following a star. */
    if (c == '/')
        state = PROGRAM;
    else if (c != '*') {
        state = COMMENT;
        putchar (' ');
    }
}

```

```

        break;

    case QUOTE:
        /* Within quoted string or character constant. */
        putchar(c);
        if (c == '\\')
            state = LITERAL;
        else if (c == quote)
            state = PROGRAM;
        break;

    case LITERAL:
        /* Within quoted string or character constant, following \. */
        putchar(c);
        state = QUOTE;
        break;

    default:
        abort();
}

```

```

return /* this comment added by RJH 10 July 2000 */ 0;

```

```

}

```

```

/*

```

```

Local variables:

```

```

compile-command: "checkergcc -W -Wall -ansi -pedantic knr123.c -o knr123"

```

```

End:

```

```

*/

```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 1-24, page 34

Category 0 Solution by [Rick Dearman](#)

Category 1 Solutions by [Stefan Farfeleder](#) and [Stig Brautaset](#)

Write a program to check a C program for rudimentary syntax errors like unbalanced parentheses, brackets and braces. Don't forget about quotes, both single and double, escape sequences, and comments. (This program is hard if you do it in full generality.)

Rick Dearman's Category 0 solution:

```

/*****
    KnR 1-24
    -----
    Write a program to check the syntax of a C program
    for matching {} () " ' []

    Author: Rick Dearman
    email: rick@ricken.demon.co.uk

*****/
#include <stdio.h>

#define MAXLINE 1000 /* max input line size */
char line[MAXLINE]; /*current input line*/

int getline(void); /* taken from the KnR book. */

int
main()
{
    int len=0;
    int t=0;
    int brace=0, bracket=0, parenthesis=0;
    int s_quote=1, d_quote=1;

    while ((len = getline()) > 0 )
    {
        t=0;
        while(t < len)
        {
            if( line[t] == '[' )
            {
                brace++;
            }
            if( line[t] == ']' )

```

```

        {
            brace--;
        }
        if( line[t] == '(' )
        {
            parenthesis++;
        }
        if( line[t] == ')' )
        {
            parenthesis--;
        }
        if( line[t] == '\'' )
        {
            s_quote *= -1;
        }
        if( line[t] == '"' )
        {
            d_quote *= -1;
        }
        t++;
    }
}

if(d_quote !=1)
    printf ("Mismatching double quote mark\n");
if(s_quote !=1)
    printf ("Mismatching single quote mark\n");
if(parenthesis != 0)
    printf ("Mismatching parenthesis\n");
if(brace != 0)
    printf ("Mismatching brace mark\n");
if(bracket != 0)
    printf ("Mismatching bracket mark\n");
if( bracket==0 && brace==0 && parenthesis==0 && s_quote == 1 && d_quote == 1)
    printf ("Syntax appears to be correct.\n");
return 0;
}

/* getline: specialized version */
int getline(void)
{
    int c, i;
    extern char line[];

    for ( i=0; i<MAXLINE-1 && ( c=getchar()) != EOF && c != '\n'; ++i)
        line[i] = c;
    if(c == '\n')
    {
        line[i] = c;
        ++i;
    }
    line[i] = '\0';
}

```

```
return i;
```

```
}
```

Stefan Farfeleder's Category 1 solution:

```
/* 1-24.c */
```

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#define MAX_STACK 1024
```

```
enum
```

```
{
    CODE,          /* nothing of the following */
    COMMENT,       /* inside a comment */
    QUOTE1,        /* inside '\'' */
    QUOTE2,        /* inside '"' */
};
```

```
int main(void)
```

```
{
    int ch;
    int state = CODE;
    char stack[MAX_STACK];
    size_t top = 0; /* points to the top of the stack :-) */
    size_t line = 1;
    int error = 0; /* for ok-message */

    while ((ch = getchar()) != EOF)
    {
        if (ch == '\n')
        {
            line++;
        }

        switch (state)
        {
        case CODE:
            if (ch == '\')
            {
                state = QUOTE1;
            }
            else if (ch == '"')
            {
                state = QUOTE2;
            }
            else if (ch == '/')
            {
                int second = getchar();
            }
        }
    }
}
```

```

        if (second == '*')
        {
            state = COMMENT;
        }
        else
        {
            ungetc(second, stdin);
        }
    }
    else if (ch == '(' || ch == '[' || ch == '{')
    {
        if (top < MAX_STACK)
        {
            stack[top++] = ch;
        }
        else
        {
            printf("Stack too small!\n");
            return EXIT_FAILURE; /* exit gracefully :-) */
        }
    }
    else if (ch == ')' || ch == ']' || ch == '}')
    {
        if (top == 0) /* found closing brace but stack is empty */
        {
            printf("Line %lu: Closing '%c' found without "
                   "counterpart.\n", (unsigned long)line, ch);
            error = 1;
        }
        else
        {
            char open = stack[--top];

            if ((ch == ')' && open != '(') ||
                (ch == ']' && open != '[') ||
                (ch == '}' && open != '{'))
            {
                printf("Line %lu: Closing '%c' does not match "
                       "opening '%c'.\n", (unsigned long)line, ch, open);
                error = 1;
            }
        }
    }
}
break;
case COMMENT:
    if (ch == '*')
    {
        int second = getchar();

        if (second == '/')
        {

```

```

        state = CODE;
    }
    else
    {
        ungetc(second, stdin);
    }
}
break;
case QUOTE1:
    if (ch == '\\')
    {
        (void)getchar(); /* an escaped char inside ' throw it away */
    }
    else if (ch == '\')
    {
        state = CODE;
    }
    break;
case QUOTE2:
    if (ch == '\\')
    {
        (void)getchar(); /* an escaped char inside " throw it away */
    }
    else if (ch == '"')
    {
        state = CODE;
    }
    break;
}
}

if (state == COMMENT)
{
    printf("Code ends inside comment!\n");
}
else if (state == QUOTE1)
{
    printf("Code ends inside single quotes!\n");
}
else if (state == QUOTE2)
{
    printf("Code ends inside double quotes!\n");
}
else if (top == 0 && error == 0)
{
    printf("Code seems to be ok.\n");
}
if (top > 0) /* still something in the stack */
{
    size_t i;
    for (i = 0; i < top; i++)
    {

```



```

        printf("Opening '%c' found without counterpart.\n", stack[i]);
    }
}

return 0;
}

```

Stig Brautaset's Cat 1 solution:

```

/* This is my first rudimentary C syntax checker. It checks for syntax errors,
 * like closing a set of brackets using the wrong type. It is not *very* good
 * at it, but it does not bother about comments, and it does know something
 * about escape sequences and character strings/constants.
 *
 * It uses a simple static stack to keep track of the braces, and it also uses
 * a stack to keep track of the errors on each line. Someday I might change
 * that to use a queue for the error-tracking, because as it is now, it outputs
 * the rightmost error on the line first, and then it steps leftwards (if there
 * is more than one error on each line).
 *
 * I might also implement my dynamically allocated stack and queue implementa-
 * tions, so that running out of space in the stack is not an issue. I might
 * also skip it, since it has little to do with the exercise in question.
 *
 * The program is especially bad at error-recovery. If it finds an error, (or
 * something it believes to be an error) subsequent errors reported might be a
 * bit dubious.
 */

#include <stdio.h>

#define MAXVAL 1000
#define MAXLINE 1000

typedef struct {
    int top;
    int val[MAXVAL];
    int pos[MAXVAL];
} stackstr;

/* very simple stack push function */
int push(stackstr *stk, int foo, int bar)
{
    if (stk->top == MAXVAL) {
        printf("stack overflow. NOT putting more values on the stack.\n");
        return 1;
    }
    stk->val[stk->top] = foo;
    stk->pos[stk->top] = bar;
    stk->top++;
}

```

```

    return 0;
}

/* very simple function to pop values off a stack */
int pop(stackstr *stk, int *foo, int *bar)
{
    if (stk->top == 0) {
        return 1;
    }
    stk->top--;
    *foo = stk->val[stk->top];
    *bar = stk->pos[stk->top];

    return 0;
}

/* we go through the input one line at a time, and this function
 * gets the line to test
 */
int getline(char *s, int lim)
{
    int i, c;

    for (i = 0; i < lim - 1 && (c = getchar()) != EOF && c != '\n'; i++)
        *(s + i) = c;

    if (c == '\n')
        *(s + i++) = c;
    *(s + i) = '\0';

    return i;
}

void scanline(stackstr *stk, stackstr *errstk, char *s, int len)
{
    int i, c, d, foo;
    static int string = 0, comment = 0, isconst = 0, escape = 0;

    for (i = 0; i < len; i++) {
        c = *(s + i);

        if (!comment) {
            if (c == '\\') { /* we have an escape */
                /* test for a valid escape sequence */
                if ((d = *(s + ++i)) == '\\') || d == 'n' || d == '0'
                    || d == 'r' || d == '?'
                    || d == 't' || d == '\'' || d == '\"' || d
                    == 'b' || d == 'x') {
                    continue; /* ok, valid escape sequence --
don't bother about it */
                } else {

```

```

                                push(errstk, 5, i); /* illegal escape
sequence */
                                }
                                } else if (c == '\\"') { /* is it a text string then? */
                                    if (!string)
                                        string = 1;
                                    else
                                        string = 0;
                                } else if (c == '\\\'') { /* is it a constant? */
                                    if (!isconst)
                                        isconst = 1;
                                    else
                                        isconst = 0;
                                }
                            }

if (!isconst && !string && !comment && c == '/') {
    if ((d = *(s + ++i)) == '*')
        comment = 1;
} else if (comment && c == '*') {
    if ((d = *(s + ++i)) == '/') {
        comment = 0;
        continue; /* done with the comment stuff -- start
over */
    }
}

/* only bother about ({[ ]})'s that's not in
 * a string, constant or comment
 */
if (!isconst && !string && !comment) {
    if (c == '(' || c == '{' || c == '[') {
        push(stk, c, 0);
    } else if (c == ']' || c == '}' || c == ')') {
        if (pop(stk, &d, &foo)) {
            push(errstk, 4, i);
        }
        if (c == ')' && d != '(') {
            push(stk, d, 0);
            push(errstk, 1, i);
        } else if (c == ']' && d != '[') {
            push(stk, d, 0);
            push(errstk, 2, i);
        } else if (c == '}' && d != '{') {
            push(stk, d, 0);
            push(errstk, 3, i);
        }
    }
}
}
}
}

```

```

/* print errors on the line (if there were any) */
void print_err(stackstr *errstk, int lineno)
{
    int errno, pos;

    /* yes I know... this way the errors come "backwards" :) */
    while (!pop(errstk, &errno, &pos)) {
        printf("on line number %d: ", lineno);
        switch(errno) {
            case 1:
                printf("closing unopened parantheses, column %d\n", pos+1);
                break;
            case 2:
                printf("closing unopened square bracket, column %d\n", pos
+1);
                break;
            case 3:
                printf("closing unopened curly braces, column %d\n", pos+1);
                break;
            case 4:
                printf("trying to close unopened block/control structure,
column %d\n", pos+1);
                break;
            case 5:
                printf("illigal escape sequence, column %d\n", pos+1);
                break;
            default:
                printf("undeterminable error\n");
                break;
        }
    }
}

int main(void)
{
    stackstr errstk = {0}, stk = {0};
    int c, linenbr = 0, errcount = 0, linelen;
    char line[MAXLINE];

    while ((linelen = getline(line, MAXLINE)) > 0) {
        linenbr++;
        scanline(&stk, &errstk, line, linelen);
        if (errstk.top) {
            print_err(&errstk, linenbr);
            errcount++;
        }
    }

    if (errcount)
        printf("%d lines contained error(s)\n", errcount);
    else

```

```
        printf("Well, *I* didn't find any syntax errors, but don't take my  
word for it...:)\n");  
  
    return 0;  
}
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 2-1, page 36

Category 0 Solution by [Rick Dearman](#) - with corrections by Russ Bobbitt, marked `/* RB */` and by Stefan Farfeleder, marked `/* SF */`

Write a program to determine the ranges of `char`, `short`, `int`, and `long` variables, both signed and unsigned, by printing appropriate values from standard headers and by direct computation. Harder if you compute them: determine the ranges of the various floating-point types.

```
#include <stdio.h>
#include <limits.h>

int
main ()
{
    printf("Size of Char %d\n", CHAR_BIT);
    printf("Size of Char Max %d\n", CHAR_MAX);
    printf("Size of Char Min %d\n", CHAR_MIN);
    printf("Size of int min %d\n", INT_MIN);
    printf("Size of int max %d\n", INT_MAX);
    printf("Size of long min %ld\n", LONG_MIN);           /* RB */
    printf("Size of long max %ld\n", LONG_MAX);          /* RB */
    printf("Size of short min %d\n", SHRT_MIN);
    printf("Size of short max %d\n", SHRT_MAX);
    printf("Size of unsigned char %u\n", UCHAR_MAX);    /* SF */
    printf("Size of unsigned long %lu\n", ULONG_MAX);   /* RB */
    printf("Size of unsigned int %u\n", UINT_MAX);      /* RB */
    printf("Size of unsigned short %u\n", USHRT_MAX);   /* SF */

    return 0;
}
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 2-2, page 42

Solutions by "Flippant Squirrel" :-) and Craig Schroeder

Exercise 2-2 discusses a for loop from the text. Here it is:

```
for(i=0; i<lim-1 && (c=getchar()) != '\n' && c != EOF; ++i)
    s[i] = c;
```

Write a loop equivalent to the for loop above without using && or || .

```
#include <stdio.h>

#define MAX_STRING_LENGTH 100

int main(void)
{
    /*
    for (i = 0; i < lim-1 && (c=getchar()) != '\n' && c != EOF; ++i)
        s[i] = c;
    */

    int i = 0,
        lim = MAX_STRING_LENGTH,
        c;
    char s[MAX_STRING_LENGTH];

    while (i < (lim - 1))
    {
        c = getchar();

        if (c == EOF)
            break;
        else if (c == '\n')
            break;
```



```

        s[i++] = c;
    }

    s[i] = '\0';    /* terminate the string */

    return 0;
}

```

Here's a Category 1 solution from Craig Schroeder, which is not so much exegetic as - um - cute. :-)

```

#include <stdio.h>

#define lim 80

int main()
{
    int i, c;
    char s[lim];

    /* There is a sequence point after the first operand of ?: */

    for(i=0; i<lim-1 ? (c=getchar()) != '\n' ? c != EOF : 0 : 0 ; ++i)
        s[i] = c;

    return s[i] ^= s[i]; /* null terminate and return. */
}

```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 2-3, page 46

Solutions by Richard Heathfield and Marshall S McLeish

Write the function `htoi(s)`, which converts a string of hexadecimal digits (including an optional `0x` or `0X`) into its equivalent integer value. The allowable digits are `0` through `9`, `a` through `f`, and `A` through `F`.

Here's my solution:

```
/* Write the function htoi(s), which converts a string of hexadecimal
 * digits (including an optional 0x or 0X) into its equivalent integer
 * value. The allowable digits are 0 through 9, a through f, and
 * A through F.
 *
 * I've tried hard to restrict the solution code to use only what
 * has been presented in the book at this point (page 46). As a
 * result, the implementation may seem a little naive. Error
 * handling is a problem. I chose to adopt atoi's approach, and
 * return 0 on error. Not ideal, but the interface doesn't leave
 * me much choice.
 *
 * I've used unsigned int to keep the behaviour well-defined even
 * if overflow occurs. After all, the exercise calls for conversion
 * to 'an integer', and unsigned ints are integers!
 */

/* These two header files are only needed for the test driver */
#include <stdio.h>
#include <stdlib.h>

/* Here's a helper function to get me around the problem of not
 * having strchr
 */

int hexalpha_to_int(int c)
{
    char hexalpha[] = "aAbBcCdDeEfF";
    int i;
    int answer = 0;

    for(i = 0; answer == 0 && hexalpha[i] != '\0'; i++)
    {
        if(hexalpha[i] == c)
        {
            answer = 10 + (i / 2);
        }
    }
}
```

```

    }
}

return answer;
}

unsigned int htoi(const char s[])
{
    unsigned int answer = 0;
    int i = 0;
    int valid = 1;
    int hexit;

    if(s[i] == '0')
    {
        ++i;
        if(s[i] == 'x' || s[i] == 'X')
        {
            ++i;
        }
    }

    while(valid && s[i] != '\0')
    {
        answer = answer * 16;
        if(s[i] >= '0' && s[i] <= '9')
        {
            answer = answer + (s[i] - '0');
        }
        else
        {
            hexit = hexalpha_to_int(s[i]);
            if(hexit == 0)
            {
                valid = 0;
            }
            else
            {
                answer = answer + hexit;
            }
        }

        ++i;
    }

    if(!valid)
    {
        answer = 0;
    }
}

```

```

    return answer;
}

/* Solution finished. This bit's just a test driver, so
 * I've relaxed the rules on what's allowed.
 */

int main(void)
{
    char *endp = NULL;
    char *test[] =
    {
        "F00",
        "bar",
        "0100",
        "0x1",
        "0XA",
        "0X0C0BE",
        "abcdef",
        "123456",
        "0x123456",
        "deadbeef",
        "zog_c"
    };

    unsigned int result;
    unsigned int check;

    size_t numtests = sizeof test / sizeof test[0];

    size_t thistest;

    for(thistest = 0; thistest < numtests; thistest++)
    {
        result = htoi(test[thistest]);
        check = (unsigned int)strtoul(test[thistest], &endp, 16);

        if((*endp != '\0' && result == 0) || result == check)
        {
            printf("Testing %s. Correct. %u\n", test[thistest], result);
        }
        else
        {
            printf("Testing %s. Incorrect. %u\n", test[thistest], result);
        }
    }

    return 0;
}

```

And here's Marshall's:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

long hchartoi (char hexdig, int pos);    /* converts a hex char to decimal knowing
its 0 based place value */
long htoi (char hexstring[]);           /* converts a string of hex bits to
integer ... */

int main(void)
{
    char *endp = NULL;
    char *test[] =
    {
        "F00",
        "bar",
        "0100",
        "0x1",
        "0XA",
        "0X0C0BE",
        "abcdef",
        "123456",
        "0x123456",
        "deadbeef",
        "zog_c"
    };

    long int result;
    long int check;

    size_t numtests = sizeof test / sizeof test[0];

    size_t thistest;

    for(thistest = 0; thistest < numtests; thistest++)
    {
        result = htoi(test[thistest]);
        check = strtol(test[thistest], &endp, 16);

        if((*endp != '\0' && result == -1) || result == check)
        {
            printf("Testing %s. Correct. %ld\n", test[thistest], result);
        }
        else
        {

```

```

        printf("Testing %s. Incorrect. %ld\n", test[thistest], result);
    }
}

return 0;
}

long htoi (char s[])
{
    char *p = &s[strlen(s)-1];
    long deci = 0, dig = 0;
    int pos = 0;

    while (p >= s) {
        if ((dig = hchartoi(*p, pos)) < 0 ) {
            printf("Error\n");
            return -1;
        }
        deci += dig;
        --p;
        ++pos;
    }
    return deci;
}

/* convert hex char to decimal value */
long hchartoi (char hexdig, int pos)
{
    char hexdigits[] = "0123456789ABCDEF";
    char *p = &hexdigits[0];
    long deci = 0;
    int i;

    while (*p != toupper(hexdig) && deci < 16) {
        ++p;
        ++deci;
    }
    if (*p == toupper(hexdig)) {
        for (i = 0; i < pos; i++)
            deci *= 16;
        return deci;
    }
}

```

```
    return -1;  
}
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 2-4, page 48

Solution by [Richard Heathfield](#)

Write an alternate version of `squeeze(s1,s2)` that deletes each character in the string `s1` that matches any character in the string `s2`.

```

/*
 * Exercise 2-4 Page 48
 *
 * Write an alternate version of squeeze(s1,s2) that deletes each
 * character in s1 that matches any character in the string s2.
 *
 */

/* squeeze2: delete all characters occurring in s2 from string s1. */

void squeeze2(char s1[], char s2[])
{
    int i, j, k;
    int instr2 = 0;

    for(i = j = 0; s1[i] != '\0'; i++)
    {
        instr2 = 0;
        for(k = 0; s2[k] != '\0' && !instr2; k++)
        {
            if(s2[k] == s1[i])
            {
                instr2 = 1;
            }
        }

        if(!instr2)
        {
            s1[j++] = s1[i];
        }
    }
    s1[j] = '\0';
}

/* test driver */

```



```

#include <stdio.h>
#include <string.h>

int main(void)
{
    char *leftstr[] =
    {
        " ",
        "a",
        "antidisestablishmentarianism",
        "beautifications",
        "characteristically",
        "deterministically",
        "electroencephalography",
        "familiarisation",
        "gastrointestinal",
        "heterogeneousness",
        "incomprehensibility",
        "justifications",
        "knowledgeable",
        "lexicographically",
        "microarchitectures",
        "nondeterministically",
        "organizationally",
        "phenomenologically",
        "quantifications",
        "representationally",
        "straightforwardness",
        "telecommunications",
        "uncontrollability",
        "vulnerabilities",
        "wholeheartedly",
        "xylophonically", /* if there is such a word :-) */
        "youthfulness",
        "zoologically"
    };
    char *rightstr[] =
    {
        " ",
        "a",
        "the",
        "quick",
        "brown",
        "dog",
        "jumps",
        "over",
        "lazy",

```

```
"fox",  
"get",  
"rid",  
"of",  
"windows",  
"and",  
"install",  
"linux"  
};  
  
char buffer[32];  
size_t numlefts = sizeof leftstr / sizeof leftstr[0];  
size_t numrights = sizeof rightstr / sizeof rightstr[0];  
size_t left = 0;  
size_t right = 0;  
  
for(left = 0; left < numlefts; left++)  
{  
    for(right = 0; right < numrights; right++)  
    {  
        strcpy(buffer, leftstr[left]);  
  
        squeeze2(buffer, rightstr[right]);  
  
        printf("[%s] - [%s] = [%s]\n", leftstr[left], rightstr[right], buffer);  
    }  
}  
return 0;  
}
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 2-5, page 48

Solution 1 by [Richard Heathfield](#)

Solution 2 by [Partha Seetala](#)

Write the function `any(s1, s2)`, which returns the first location in the string `s1` where any character from the string `s2` occurs, or `-1` if `s1` contains no characters from `s2`. (The standard library function `strpbrk` does the same job but returns a pointer to the location.)

Here is my solution, which is very simple but quite naive and inefficient. It has a worst-case time complexity of $O(nm)$ where n and m are the lengths of the two strings.

```
/*
 * Exercise 2-5 Page 48
 *
 * Write the function any(s1,s2), which returns the first location
 * in the string s1 where any character from the string s2 occurs,
 * or -1 if s1 contains no characters from s2. (The standard library
 * function strpbrk does the same job but returns a pointer to the
 * location.)
 */

int any(char s1[], char s2[])
{
    int i;
    int j;
    int pos;

    pos = -1;

    for(i = 0; pos == -1 && s1[i] != '\0'; i++)
    {
        for(j = 0; pos == -1 && s2[j] != '\0'; j++)
        {
            if(s2[j] == s1[i])
            {
                pos = i;
            }
        }
    }

    return pos;
}

/* test driver */

/* We get a helpful boost for testing from the question text, because we are
```

```

* told that the function's behaviour is identical to strpbrk except that it
* returns a pointer instead of a position. We use this fact to validate the
* function's correctness.
*/

```

```

#include <stdio.h>
#include <string.h>

```

```

int main(void)
{
    char *leftstr[] =
    {
        "",
        "a",
        "antidisestablishmentarianism",
        "beautifications",
        "characteristically",
        "deterministically",
        "electroencephalography",
        "familiarisation",
        "gastrointestinal",
        "heterogeneousness",
        "incomprehensibility",
        "justifications",
        "knowledgeable",
        "lexicographically",
        "microarchitectures",
        "nondeterministically",
        "organizationally",
        "phenomenologically",
        "quantifications",
        "representationally",
        "straightforwardness",
        "telecommunications",
        "uncontrollability",
        "vulnerabilities",
        "wholeheartedly",
        "xylophonically",
        "youthfulness",
        "zoologically"
    };
    char *rightstr[] =
    {
        "",
        "a",
        "the",
        "quick",
        "brown",
        "dog",
        "jumps",
        "over",
        "lazy",
    };
}

```

```

    "fox",
    "get",
    "rid",
    "of",
    "windows",
    "and",
    "install",
    "linux"
};

size_t numlefts = sizeof leftstr / sizeof leftstr[0];
size_t numrights = sizeof rightstr / sizeof rightstr[0];
size_t left = 0;
size_t right = 0;

int passed = 0;
int failed = 0;

int pos = -1;
char *ptr = NULL;

for(left = 0; left < numlefts; left++)
{
    for(right = 0; right < numrights; right++)
    {
        pos = any(leftstr[left], rightstr[right]);
        ptr = strpbrk(leftstr[left], rightstr[right]);

        if(-1 == pos)
        {
            if(ptr != NULL)
            {
                printf("Test %d/%d failed.\n", left, right);
                ++failed;
            }
            else
            {
                printf("Test %d/%d passed.\n", left, right);
                ++passed;
            }
        }
        else
        {
            if(ptr == NULL)
            {
                printf("Test %d/%d failed.\n", left, right);
                ++failed;
            }
            else
            {
                if(ptr - leftstr[left] == pos)
                {

```

```

        printf("Test %d/%d passed.\n", left, right);
        ++passed;
    }
    else
    {
        printf("Test %d/%d failed.\n", left, right);
        ++failed;
    }
}
}
}
}
}
printf("\n\nTotal passes %d, fails %d, total tests %d\n",
        passed,
        failed,
        passed + failed);
return 0;
}

```

Here's a much better solution, by Partha Seetala. This solution has a worst- case time complexity of only $O(n + m)$ which is considerably better.

It works in a very interesting way. He first defines an array with one element for each possible character in the character set, and then takes the *second* string and 'ticks' the array at each position where the second string contains the character corresponding to that position. It's then a simple matter to loop through the first string, quitting as soon as he hits a 'ticked' position in the array.

```

#include <stdio.h> /* for NULL */

int any(char *s1, char *s2)
{
    char array[256]; /* rjh comments
                     * (a) by making this char array[256] = {0}; the first loop
becomes unnecessary.
                     * (b) for full ANSIness, #include <limits.h>, make the
array unsigned char,
                     *      cast as required, and specify an array size of
UCHAR_MAX + 1.
                     * (c) the return statements' (parentheses) are not
required.
                     */

    int i;
    if (s1 == NULL) {
        if (s2 == NULL) {
            return(0);
        } else {
            return(-1);
        }
    }
}

```

```

    }

    for(i = 0; i < 256; i++) {
        array[i] = 0;
    }

    while(*s2 != '\0') {
        array[*s2] = 1;
        s2++;
    }

    i = 0;
    while(s1[i] != '\0') {
        if (array[s1[i]] == 1) {
            return(i);
        }
        i++;
    }
    return(-1);
}

/* test driver by Richard Heathfield */

/* We get a helpful boost for testing from the question text, because we are
 * told that the function's behaviour is identical to strpbrk except that it
 * returns a pointer instead of a position. We use this fact to validate the
 * function's correctness.
 */

#include <string.h>

int main(void)
{
    char *leftstr[] =
    {
        "",
        "a",
        "antidisestablishmentarianism",
        "beautifications",
        "characteristically",
        "deterministically",
        "electroencephalography",
        "familiarisation",
        "gastrointestinal",
        "heterogeneousness",
        "incomprehensibility",
        "justifications",
        "knowledgeable",
        "lexicographically",
        "microarchitectures",
        "nondeterministically",
        "organizationally",
    }

```

```

    "phenomenologically",
    "quantifications",
    "representationally",
    "straightforwardness",
    "telecommunications",
    "uncontrollability",
    "vulnerabilities",
    "wholeheartedly",
    "xylophonically",
    "youthfulness",
    "zoologically"
};

char *rightstr[] =
{
    "",
    "a",
    "the",
    "quick",
    "brown",
    "dog",
    "jumps",
    "over",
    "lazy",
    "fox",
    "get",
    "rid",
    "of",
    "windows",
    "and",
    "install",
    "linux"
};

size_t numlefts = sizeof leftstr / sizeof leftstr[0];
size_t numrights = sizeof rightstr / sizeof rightstr[0];
size_t left = 0;
size_t right = 0;

int passed = 0;
int failed = 0;

int pos = -1;
char *ptr = NULL;

for(left = 0; left < numlefts; left++)
{
    for(right = 0; right < numrights; right++)
    {
        pos = any(leftstr[left], rightstr[right]);
        ptr = strpbrk(leftstr[left], rightstr[right]);

        if(-1 == pos)

```



```

{
    if(ptr != NULL)
    {
        printf("Test %d/%d failed.\n", left, right);
        ++failed;
    }
    else
    {
        printf("Test %d/%d passed.\n", left, right);
        ++passed;
    }
}
else
{
    if(ptr == NULL)
    {
        printf("Test %d/%d failed.\n", left, right);
        ++failed;
    }
    else
    {
        if(ptr - leftstr[left] == pos)
        {
            printf("Test %d/%d passed.\n", left, right);
            ++passed;
        }
        else
        {
            printf("Test %d/%d failed.\n", left, right);
            ++failed;
        }
    }
}
}
}
}
printf("\n\nTotal passes %d, fails %d, total tests %d\n",
    passed,
    failed,
    passed + failed);
return 0;
}

```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 2-6, page 49

Solution by [Richard Heathfield](#)

Write a function `setbits(x,p,n,y)` that returns `x` with the `n` bits that begin at position `p` set to the rightmost `n` bits of `y`, leaving the other bits unchanged.

This one's scary.

```
#include <stdio.h>

unsigned setbits(unsigned x, int p, int n, unsigned y)
{
    return (x & ((~0 << (p + 1)) | (~(0 << (p + 1 - n)))) | ((y & ~(0 << n)) << (p
+ 1 - n));
}

int main(void)
{
    unsigned i;
    unsigned j;
    unsigned k;
    int p;
    int n;

    for(i = 0; i < 30000; i += 511)
    {
        for(j = 0; j < 1000; j += 37)
        {
            for(p = 0; p < 16; p++)
            {
                for(n = 1; n <= p + 1; n++)
                {
                    k = setbits(i, p, n, j);
                    printf("setbits(%u, %d, %d, %u) = %u\n", i, p, n, j, k);
                }
            }
        }
    }
    return 0;
}
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 2-7, page 49

Solution by [Gregory Pietsch](#)

Write a function `invert(x, p, n)` that returns `x` with the `n` bits that begin at position `p` inverted (i.e., 1 changed into 0 and vice versa), leaving the others unchanged.

```

unsigned invert(unsigned x, int p, int n)
{
    return x ^ (~(0U << n) << p);
}

/* main driver added, in a hurry while tired, by RJH. Better test driver suggestions
are welcomed! */

#include <stdio.h>

int main(void)
{
    unsigned x;
    int p, n;

    for(x = 0; x < 700; x += 49)
        for(n = 1; n < 8; n++)
            for(p = 1; p < 8; p++)
                printf("%u, %d, %d: %u\n", x, n, p, invert(x, n, p));
    return 0;
}

```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 2-8, page 49

Solutions by Gregory Pietsch and Bob Wightman

Write a function `rightrot(x,n)` that returns the value of the integer `x` rotated to the right by `n` bit positions.

Greg's Cat 0 solution

```
unsigned rightrot(unsigned x, unsigned n)
{
    while (n > 0) {
        if ((x & 1) == 1)
            x = (x >> 1) | ~(~0U >> 1);
        else
            x = (x >> 1);
        n--;
    }
    return x;
}

/* main driver added, in a hurry while tired, by RJH. Better test driver suggestions
are welcomed! */

#include <stdio.h>

int main(void)
{
    unsigned x;
    int n;

    for(x = 0; x < 700; x += 49)
        for(n = 1; n < 8; n++)
            printf("%u, %d: %u\n", x, n, rightrot(x, n));
    return 0;
}
```

Here's Bob Wightman's Cat 1 solution:

```
/* K&R exercise 2-8
```

It is class 1 due to the `/sizeof/` operator (`CHAR_BIT` is introduced with `<limits.h>` in Chapter 1). I could have used the conditional operator but thought that this is clearer.

Notes:

1. Implicit int removed (not absolutely necessary but...)
2. Checks for the size of the shift and reduces it to the range 0 - (number of bits in an int) - 1 This is to avoid right shifting the number into oblivion.
3. If either the value or the shift is zero then nothing need to be done to the parameter so just return it.

```
*/
```

```
unsigned int rightrot(unsigned int x, unsigned int n)
```

```
{
    /* calculate number of bits in type */
    size_t s = sizeof(x) * CHAR_BIT;
    size_t p;

    /* limit shift to range 0 - (s - 1) */
    if(n < s)
        p = n;
    else
        p = n % s;

    /* if either is zero then the original value is unchanged */
    if((0 == x) || (0 == p))
        return x;

    return (x >> p) | (x << (s - p));
}
```

```
/* Driver based on yours but runs the shift values beyond the size of an
unsigned integer on any system */
```

```
int main(void)
{
    unsigned int val;
    unsigned int pos;
    unsigned int max = sizeof (pos) * CHAR_BIT + 1;

    for(val = 0; val < 700; val += 49)
    {
        for(pos = 0; pos < max; ++pos)
        {
            printf("%u, %d: %u\n", x, n, rightrot(val, pos));
        }
    }
}
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 2-9, page 51

Solution by Gregory Pietsch

In a two's complement number system, $x \&= (x-1)$ deletes the rightmost 1-bit in x . Explain why. Use this observation to write a faster version of `bitcount`.

`bitcount` is written on p.50 as this:

```
/* bitcount:  count 1 bits in x */
int bitcount(unsigned x)
{
    int b;

    for (b = 0; x != 0; x >>= 1)
        if (x & 01)
            b++;
    return b;
}
```

Answer: If x is odd, then $(x-1)$ has the same bit representation as x except that the rightmost 1-bit is now a 0. In this case, $(x \& (x-1)) == (x-1)$. If x is even, then the representation of $(x-1)$ has the rightmost zeros of x becoming ones and the rightmost one becoming a zero. Anding the two clears the rightmost 1-bit in x and all the rightmost 1-bits from $(x-1)$. Here's the new version of `bitcount`:

```
/* bitcount:  count 1 bits in x */
int bitcount(unsigned x)
{
    int b;

    for (b = 0; x != 0; x &= (x-1))
        b++;
    return b;
}
```


[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 2-10, page 52

Solution by Bryan Williams

Rewrite the function lower, which converts upper case letters to lower case, with a conditional expression instead of if-else.

```

/*
    Exercise 2-10. Rewrite the function lower, which converts upper case letters
    to lower case, with a conditional expression instead of if-else.

    Assumptions : by conditional expression they mean an expression involving a
    ternary operator.

    Author: Bryan Williams
*/

#include <stdio.h>
#include <string.h>

#define TEST
#define ORIGINAL          0
#define SOLUTION          1
#define PORTABLE_SOLUTION 0

/*
    ok, the original routine we are trying to convert looks like this..
*/
#ifdef ORIGINAL
/* lower: convert c to lower case; ASCII only */
int lower(int c)
{
    if(c >= 'A' && c <= 'Z')
        return c + 'a' - 'A';
    else
        return c;
}
#endif

/*
    the natural solution for simply making this a conditional (ternary) return
    instead of an
    if ... else ...

```

```

*/
#if SOLUTION
/* lower: convert c to lower case; ASCII only */
int lower(int c)
{
    return c >= 'A' && c <= 'Z' ? c + 'a' - 'A' : c;
}
#endif

/*
    the more portable solution, requiring string.h for strchr but keeping the idea of
a    conditional return.
*/
#if PORTABLE_SOLUTION
/* lower: convert c to lower case */
int lower(int c)
{
    char *Uppercase = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    char *Lowercase = "abcdefghijklmnopqrstuvwxyz";
    char *p = NULL;

    return NULL == (p = strchr(Uppercase, c)) ? c : *(Lowercase + (p - Uppercase));
}
#endif

/*
    ok, this bit is just a test driver... exclude as required
*/
#ifdef TEST
int main(void)
{
    char *Tests = "AaBbcCD3EdFGHgIJKLhM2NOjPQRkSTlUVWfXYf0Z1";
    char *p = Tests;
    int Result = 0;

    while('\\0' != *p)
    {
        Result = lower(*p);
        printf("[%c] gives [%c]\\n", *p, Result);
        ++p;
    }

    /* and the obligatory boundary test */
    Result = lower(0);
    printf("'\\0' gives %d\\n", Result);

    return 0;
}

```

```
}
```

```
#endif
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 3-1, page 58

Solutions by Paul Griffiths Colin Barker and Andrew Tesker

Our binary search makes two tests inside the loop, when one would suffice (at the price of more tests outside). Write a version with only one test inside the loop and measure the difference in run-time.

Paul Griffiths' solution (krx30100.c):

```

/* Solution by Paul Griffiths (paul@paulgriffiths.demon.co.uk) */

/*
   EX3_1.C
   =====

   Suggested solution to Exercise 3-1

*/

#include <stdio.h>
#include <time.h>

int binsearch(int x, int v[], int n);      /* Original K&R function */
int binsearch2(int x, int v[], int n);    /* Our new function      */

#define MAX_ELEMENT 20000

/* Outputs approximation of processor time required
   for our two binary search functions. We search for
   the element -1, to time the functions' worst case
   performance (i.e. element not found in test data) */

int main(void) {
    int testdata[MAX_ELEMENT];
    int index;                /* Index of found element in test data */
    int n = -1;               /* Element to search for */
    int i;
    clock_t time_taken;

    /* Initialize test data */

```

```

for ( i = 0; i < MAX_ELEMENT; ++i )
    testdata[i] = i;

/*  Output approximation of time taken for
    100,000 iterations of binsearch()          */

for ( i = 0, time_taken = clock(); i < 100000; ++i ) {
    index = binsearch(n, testdata, MAX_ELEMENT);
}
time_taken = clock() - time_taken;

if ( index < 0 )
    printf("Element %d not found.\n", n);
else
    printf("Element %d found at index %d.\n", n, index);

printf("binsearch() took %lu clocks (%lu seconds)\n",
       (unsigned long) time_taken,
       (unsigned long) time_taken / CLOCKS_PER_SEC);

/*  Output approximation of time taken for
    100,000 iterations of binsearch2()          */

for ( i = 0, time_taken = clock(); i < 100000; ++i ) {
    index = binsearch2(n, testdata, MAX_ELEMENT);
}
time_taken = clock() - time_taken;

if ( index < 0 )
    printf("Element %d not found.\n", n);
else
    printf("Element %d found at index %d.\n", n, index);

printf("binsearch2() took %lu clocks (%lu seconds)\n",
       (unsigned long) time_taken,
       (unsigned long) time_taken / CLOCKS_PER_SEC);

return 0;
}

/*  Performs a binary search for element x
    in array v[], which has n elements          */

int binsearch(int x, int v[], int n) {
    int low, mid, high;

```

```

    low = 0;
    high = n - 1;
    while ( low <= high ) {
        mid = (low+high) / 2;
        if ( x < v[mid] )
            high = mid - 1;
        else if ( x > v[mid] )
            low = mid + 1;
        else
            return mid;
    }
    return -1;
}

/*  Implementation of binsearch() using
    only one test inside the loop          */

int binsearch2(int x, int v[], int n) {
    int low, high, mid;

    low = 0;
    high = n - 1;
    mid = (low+high) / 2;
    while ( low <= high && x != v[mid] ) {
        if ( x < v[mid] )
            high = mid - 1;
        else
            low = mid + 1;
        mid = (low+high) / 2;
    }
    if ( x == v[mid] )
        return mid;
    else
        return -1;
}

```

Colin Barker's solution (krx30101.c):

```

/*  Solution by Colin Barker (colin.barker@wanadoo.fr)
    *  using the driver from the solution by Paul Griffiths.
    */

/*

```

EX3_1.C

=====

Suggested solution to Exercise 3-1

```

*/

#include <stdio.h>
#include <time.h>

int binsearch(int x, int v[], int n);      /* Original K&R function */
int binsearch2(int x, int v[], int n);    /* Our new function      */

#define MAX_ELEMENT 20000

/* Outputs approximation of processor time required
   for our two binary search functions. We search for
   the element -1, to time the functions' worst case
   performance (i.e. element not found in test data) */

int main(void) {
    int testdata[MAX_ELEMENT];
    int index;                /* Index of found element in test data */
    int n = -1;               /* Element to search for */
    int i;
    clock_t time_taken;

    /* Initialize test data */

    for ( i = 0; i < MAX_ELEMENT; ++i )
        testdata[i] = i;

    /* Output approximation of time taken for
       100,000 iterations of binsearch() */

    for ( i = 0, time_taken = clock(); i < 100000; ++i ) {
        index = binsearch(n, testdata, MAX_ELEMENT);
    }
    time_taken = clock() - time_taken;

    if ( index < 0 )
        printf("Element %d not found.\n", n);
    else
        printf("Element %d found at index %d.\n", n, index);
}

```



```

printf("binsearch() took %lu clocks (%lu seconds)\n",
      (unsigned long) time_taken,
      (unsigned long) time_taken / CLOCKS_PER_SEC);

/*  Output approximation of time taken for
    100,000 iterations of binsearch2()          */

for ( i = 0, time_taken = clock(); i < 100000; ++i ) {
    index = binsearch2(n, testdata, MAX_ELEMENT);
}
time_taken = clock() - time_taken;

if ( index < 0 )
    printf("Element %d not found.\n", n);
else
    printf("Element %d found at index %d.\n", n, index);

printf("binsearch2() took %lu clocks (%lu seconds)\n",
      (unsigned long) time_taken,
      (unsigned long) time_taken / CLOCKS_PER_SEC);

return 0;
}

/*  Performs a binary search for element x
    in array v[], which has n elements          */

int binsearch(int x, int v[], int n) {
    int low, mid, high;

    low = 0;
    high = n - 1;
    while ( low <= high ) {
        mid = (low+high) / 2;
        if ( x < v[mid] )
            high = mid - 1;
        else if ( x > v[mid] )
            low = mid + 1;
        else
            return mid;
    }
    return -1;
}

int binsearch2(int x, int v[], int n)
{

```

```

int low, high, mid;

low = -1;
high = n;
while (low + 1 < high) {
    mid = (low + high) / 2;
    if (v[mid] < x)
        low = mid;
    else
        high = mid;
}
if (high == n || v[high] != x)
    return -1;
else
    return high;
}

```

Andrew Tesker's solution (krx30102.c):

```

/* Andrew Tesker
 *
 * krx30102.c
 */

#include <stdio.h>

/* find x in v[] */
int binsearch(int x, int v[], int n);

/*
    The main is here for the purpose of a built in test
 */

int main(void)
{
    int test[]={1,3,5,7,9,11,13};
    int i;
    for(i=(sizeof(test)/sizeof(int))-1; i>=0; --i)
        printf("looking for %d. Index=%d\n",test[i],binsearch(test[i], test, sizeof
(test)/sizeof(*test)));

    return 0;
}

```

```
}
/* n = size of array v */

int binsearch(int x, int v[], int n)
{
    int low, high, mid;

    low = 0;
    high = n-1;

    while(low < high) {
        mid = (low+high)/2;
        if(x <= v[mid])
            high=mid;

        else
            low = mid+1;
    }

    return (x == v[low])?low : -1;
}
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 3-2, page 60

Solution by Paul Griffiths

Write a function `escape(s, t)` that converts characters like newline and tab into visible escape sequences like `\n` and `\t` as it copies the string `t` to `s`. Use a `switch`. Write a function for the other direction as well, converting escape sequences into the real characters.

```
/*

EX3_2.C
=====

Suggested solution to Exercise 3-2

*/

#include <stdio.h>

void escape(char * s, char * t);
void unescape(char * s, char * t);

int main(void) {
    char text1[50] = "\aHello,\n\tWorld! Mistakee\b was \"Extra 'e'\"!\n";
    char text2[51];

    printf("Original string:\n%s\n", text1);

    escape(text2, text1);
    printf("Escaped string:\n%s\n", text2);

    unescape(text1, text2);
    printf("Unescaped string:\n%s\n", text1);

    return 0;
}

/*  Copies string t to string s, converting special
```

characters into their appropriate escape sequences.
 The "complete set of escape sequences" found in
 K&R Chapter 2 is used, with the exception of:

```
\? \' \ooo \xhh
```

as these can be typed directly into the source code,
 (i.e. without using the escape sequences themselves)
 and translating them is therefore ambiguous. */

```
void escape(char * s, char * t) {
    int i, j;
    i = j = 0;

    while ( t[i] ) {

        /* Translate the special character, if we have one */

        switch( t[i] ) {
            case '\n':
                s[j++] = '\\';
                s[j] = 'n';
                break;

            case '\t':
                s[j++] = '\\';
                s[j] = 't';
                break;

            case '\a':
                s[j++] = '\\';
                s[j] = 'a';
                break;

            case '\b':
                s[j++] = '\\';
                s[j] = 'b';
                break;

            case '\f':
                s[j++] = '\\';
                s[j] = 'f';
                break;

            case '\r':
```

```

        s[j++] = '\\';
        s[j] = 'r';
        break;

```

```

case '\\v':
    s[j++] = '\\';
    s[j] = 'v';
    break;

```

```

case '\\\\':
    s[j++] = '\\';
    s[j] = '\\';
    break;

```

```

case '\\\"':
    s[j++] = '\\';
    s[j] = '\"';
    break;

```

```

default:

```

```

    /* This is not a special character, so just copy it */

```

```

    s[j] = t[i];
    break;

```

```

}
++i;
++j;

```

```

}
s[j] = t[i];    /* Don't forget the null character */

```

```

}

```

```

/* Copies string t to string s, converting escape sequences
   into their appropriate special characters. See the comment
   for escape() for remarks regarding which escape sequences
   are translated. */

```

```

void unescape(char * s, char * t) {
    int i, j;
    i = j = 0;

    while ( t[i] ) {
        switch ( t[i] ) {
            case '\\':

```

```
/* We've found an escape sequence, so translate it */

switch( t[++i] ) {
case 'n':
    s[j] = '\n';
    break;

case 't':
    s[j] = '\t';
    break;

case 'a':
    s[j] = '\a';
    break;

case 'b':
    s[j] = '\b';
    break;

case 'f':
    s[j] = '\f';
    break;

case 'r':
    s[j] = '\r';
    break;

case 'v':
    s[j] = '\v';
    break;

case '\\':
    s[j] = '\\';
    break;

case '\"':
    s[j] = '\"';
    break;

default:

    /* We don't translate this escape
       sequence, so just copy it verbatim */
```

```
        s[j++] = '\\';
        s[j] = t[i];
    }
    break;

default:

    /* Not an escape sequence, so just copy the character */

    s[j] = t[i];
}
++i;
++j;
}
s[j] = t[i];    /* Don't forget the null character */
}
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 3-3, page 63

Solution by Paul Griffiths

Write a function `expand(s1, s2)` that expands shorthand notations like `a-z` in the string `s1` into the equivalent complete list `abc...xyz` in `s2`. Allow for letters of either case and digits, and be prepared to handle cases like `a-b-c` and `a-z0-9` and `-a-z`. Arrange that a leading or trailing `-` is taken literally.

```

/*
    EX3_3.C
    =====

    Suggested solution to Exercise 3-3
*/

#include <stdio.h>
#include <string.h>

void expand(char * s1, char * s2);

int main(void) {
    char *s[] = { "a-z-", "z-a-", "-1-6-",
                  "a-ee-a", "a-R-L", "1-9-1",
                  "5-5", NULL };
    char result[100];
    int i = 0;

    while ( s[i] ) {

        /*  Expand and print the next string in our array s[]  */

        expand(result, s[i]);
        printf("Unexpanded: %s\n", s[i]);
        printf("Expanded   : %s\n", result);
        ++i;
    }

    return 0;
}

```

```

/* Copies string s2 to s1, expanding
   ranges such as 'a-z' and '8-3'      */

void expand(char * s1, char * s2) {
    static char upper_alph[27] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    static char lower_alph[27] = "abcdefghijklmnopqrstuvwxyz";
    static char digits[11]     = "0123456789";

    char * start, * end, * p;
    int i = 0;
    int j = 0;

    /* Loop through characters in s2 */

    while ( s2[i] ) {
        switch( s2[i] ) {
            case '-':
                if ( i == 0 || s2[i+1] == '\0' ) {

                    /* '-' is leading or trailing, so just copy it */

                    s1[j++] = '-';
                    ++i;
                    break;
                }
                else {

                    /* We have a "range" to extrapolate. Test whether
                       the two operands are part of the same range. If
                       so, store pointers to the first and last characters
                       in the range in start and end, respectively. If
                       not, output and error message and skip this range. */

                    if ( (start = strchr(upper_alph, s2[i-1])) &&
                        (end   = strchr(upper_alph, s2[i+1])) )
                        ;
                    else if ( (start = strchr(lower_alph, s2[i-1])) &&
                        (end   = strchr(lower_alph, s2[i+1])) )
                        ;
                    else if ( (start = strchr(digits, s2[i-1])) &&
                        (end   = strchr(digits, s2[i+1])) )
                        ;
                    else {

                        /* We have mismatched operands in the range,

```

such as 'a-R', or '3-X', so output an error message, and just copy the range expression. */

```

    fprintf(stderr, "EX3_3: Mismatched operands '%c-%c'\n",
             s2[i-1], s2[i+1]);
    s1[j++] = s2[i-1];
    s1[j++] = s2[i+1];
    break;
}

/* Expand the range */

p = start;
while ( p != end ) {
    s1[j++] = *p;
    if ( end > start )
        ++p;
    else
        --p;
}
s1[j++] = *p;
i += 2;
}
break;

default:
    if ( s2[i+1] == '-' && s2[i+2] != '\0' ) {

        /* This character is the first operand in
           a range, so just skip it - the range will
           be processed in the next iteration of
           the loop. */

        ++i;
    }
    else {

        /* Just a normal character, so copy it */

        s1[j++] = s2[i++];
    }
    break;
}
}
s1[j] = s2[i];    /* Don't forget the null character */
}

```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 3-4, page 64

Solution by Paul Griffiths - error spotted by Wayne Lubin and fixed.

Wayne Lubin's query involved Paul's discussion of two's complement. The text has now been corrected (by Paul).

In a two's complement number representation, our version of `itoa` does not handle the largest negative number, that is, the value of `n` equal to $-(2 \text{ to the power } (\text{wordsize} - 1))$. Explain why not. Modify it to print that value correctly regardless of the machine on which it runs.

Exercise 3-4 explanation: There are a number of ways of representing signed integers in binary, for example, signed-magnitude, excess-M, one's complement and two's complement. We shall restrict our discussion to the latter two. In a one's complement number representation, the binary representation of a negative number is simply the binary representation of its positive counterpart, with the sign of all the bits switched. For instance, with 8 bit variables:

SIGNED	BINARY	UNSIGNED
25	00011001	25
-25	11100110	230
127	01111111	127
-127	10000000	128

The implications of this are (amongst others) that there are two ways of representing zero (all zero bits, and all one bits), that the maximum range for a signed 8-bit number is -127 to 127, and that negative numbers are biased by $(2^n - 1)$ (i.e. -1 is represented by $(2^n - 1) - (+1)$). In our example, so:

Bias	=	$2^8 - 1$	=	255	=	11111111
Subtract 25					=	00011001
Equals					=	11100110

In a two's complement representation, negative numbers are biased by 2^n , e.g.:

```

Bias = 2^8   = 100000000
Subtract 25 = 00011001
Equals      = 11100111

```

In other words, to find the two's complement representation of a negative number, find the one's complement of it, and add one. The important thing to notice is that the range of an 8 bit variable using a two's complement representation is -128 to 127, as opposed to -127 to 127 using one's complement. Thus, the absolute value of the largest negative number cannot be represented (i.e. we cannot represent +128). Since the `itoa()` function in Chapter 3 handles negative numbers by reversing the sign of the number before processing, then adding a '-' to the string, passing the largest negative number will result in it being translated to itself:

```

-128                : 10000000
One's complement: 01111111
Subtract 1          : 10000000

```

Therefore, because $(n \neq 10)$ will be negative, the do-while loop will run once only, and will place in the string a '-', followed by a single character, $(\text{INT_MIN} \% 10 + '0')$. We can remedy these two bugs in the following way: 1 - change `'while ((n != 10) > 0)'` to `'while (n != 10)'`. Since any fractional part is truncated with integer division, `n` will eventually equal zero after successive divides by 10, and `'n != 10'` will evaluate to false sooner or later. 2 - change `'n % 10 + '0'` to `'abs(n % 10) + '0'`, to get the correct character. EX3_4.C shows the revised function, which will run correctly regardless of the number representation.

```

/*

EX3_4.C
=====

Suggested solution to Exercise 3-4

*/

#include <stdlib.h>
#include <stdio.h>
#include <limits.h>

void itoa(int n, char s[]);
void reverse(char s[]);

```

```
int main(void) {
    char buffer[20];

    printf("INT_MIN: %d\n", INT_MIN);
    itoa(INT_MIN, buffer);
    printf("Buffer : %s\n", buffer);

    return 0;
}

void itoa(int n, char s[]) {
    int i, sign;
    sign = n;

    i = 0;
    do {
        s[i++] = abs(n % 10) + '0';
    } while ( n /= 10 );
    if (sign < 0)
        s[i++] = '-';

    s[i] = '\0';
    reverse(s);
}

void reverse(char s[]) {
    int c, i, j;
    for ( i = 0, j = strlen(s)-1; i < j; i++, j--) {
        c = s[i];
        s[i] = s[j];
        s[j] = c;
    }
}
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 3-5, page 64

Solution by Paul Griffiths

Write the function `itob(n,s,b)` that converts the integer `n` into a base `b` character representation in the string `s`. In particular, `itob(n,s,16)` formats `n` as a hexadecimal integer in `s`.

```

/*
    EX3_5.C
    =====

    Suggested solution to Exercise 3-5

*/

#include <stdlib.h>
#include <stdio.h>

void itob(int n, char s[], int b);
void reverse(char s[]);

int main(void) {
    char buffer[10];
    int i;

    for ( i = 2; i <= 20; ++i ) {
        itob(255, buffer, i);
        printf("Decimal 255 in base %-2d : %s\n", i, buffer);
    }
    return 0;
}

/*  Stores a string representation of integer n
    in s[], using a numerical base of b. Will handle

```

```
up to base-36 before we run out of digits to use.  */
```

```
void itob(int n, char s[], int b) {
    static char digits[] = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    int i, sign;

    if ( b < 2 || b > 36 ) {
        fprintf(stderr, "EX3_5: Cannot support base %d\n", b);
        exit(EXIT_FAILURE);
    }

    if ((sign = n) < 0)
        n = -n;
    i = 0;
    do {
        s[i++] = digits[n % b];
    } while ((n /= b) > 0);
    if (sign < 0)
        s[i++] = '-';
    s[i] = '\0';
    reverse(s);
}

/*  Reverses string s[] in place  */

void reverse(char s[]) {
    int c, i, j;
    for ( i = 0, j = strlen(s)-1; i < j; i++, j--) {
        c = s[i];
        s[i] = s[j];
        s[j] = c;
    }
}
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 3-6, page 64

Solution by Paul Griffiths

Write a version of `itoa` that accepts three arguments instead of two. The third argument is a minimum field width; the converted number must be padded with blanks on the left if necessary to make it wide enough.

```
/*

EX3_6.C
=====

Suggested solution to Exercise 3-6

*/

#include <stdio.h>
#include <limits.h>

void itoa(int n, char s[], int width);
void reverse(char s[]);

int main(void) {
    char buffer[20];

    itoa(INT_MIN, buffer, 7);
    printf("Buffer:%s\n", buffer);

    return 0;
}

void itoa(int n, char s[], int width) {
    int i, sign;

    if ((sign = n) < 0)
```

```

        n = -n;
    i = 0;
    do {
        s[i++] = n % 10 + '0';
        printf("%d %% %d + '0' = %d\n", n, 10, s[i-1]);
    } while ((n /= 10) > 0);
    if (sign < 0)
        s[i++] = '-';

    while (i < width )        /* Only addition to original function */
        s[i++] = ' ';

    s[i] = '\0';
    reverse(s);
}

void reverse(char s[]) {
    int c, i, j;
    for ( i = 0, j = strlen(s)-1; i < j; i++, j--) {
        c = s[i];
        s[i] = s[j];
        s[j] = c;
    }
}

```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Base source code on which to build solutions to exercises 4-3 through 4-10.

This text is a pretty straight quote from K&R.

```

/* this program forms the basis on which exercises 4-3 through 4-10 build */

#include <stdio.h>
#include <stdlib.h> /* for atof() - in K&R, math.h is referenced - this is an
anachronism */

#define MAXOP 100 /* max size of operand or operator */
#define NUMBER '0' /* signal that a number was found */

int getop(char []);
void push(double);
double pop(void);

/* reverse Polish calculator */

int main(void)
{
    int type;
    double op2;
    char s[MAXOP];

    while((type = getop(s)) != EOF)
    {
        switch(type)
        {
            case NUMBER:
                push(atof(s));
                break;
            case '+':
                push(pop() + pop());
                break;
            case '*':
                push(pop() * pop());
                break;
            case '-':
                op2 = pop();
                push(pop() - op2);
                break;

```

```

    case '/':
        op2 = pop();
        if(op2 != 0.0)
            push(pop() / op2);
        else
            printf("error: zero divisor\n");
            break;
    case '\n':
        printf("\t%.8g\n", pop());
        break;
    default:
        printf("error: unknown command %s\n", s);
        break;
}
}

```

```

return 0;

```

```

#define MAXVAL 100 /* maximum depth of val stack */

```

```

int sp = 0; /* next free stack position */

```

```

double val[MAXVAL]; /* value stack */

```

```

/* push: push f onto value stack */

```

```

void push(double f)
{
    if(sp < MAXVAL)
        val[sp++] = f;
    else
        printf("error: stack full, can't push %g\n", f);
}

```

```

/* pop: pop and return top value from stack */

```

```

double pop(void)
{
    if(sp > 0)
        return val[--sp];
    else
    {
        printf("error: stack empty\n");
        return 0.0;
    }
}

```

```

#include <ctype.h>

```

```

int getch(void);

```

```

void ungetch(int);

/* getop: get next operator or numeric operand */
int getop(char s[])
{
    int i, c;

    while((s[0] = c = getch()) == ' ' || c == '\t')
        ;

    s[1] = '\0';
    if(!isdigit(c) && c != '.')
        return c; /* not a number */
    i = 0;
    if(isdigit(c)) /* collect integer part */
        while(isdigit(s[++i] = c = getch()))
            ;
    if(c == '.')
        while(isdigit(s[++i] = c = getch()))
            ;
    s[i] = '\0';
    if(c != EOF)
        ungetch(c);
    return NUMBER;
}

#define BUFSIZE 100

char buf[BUFSIZE]; /* buffer for ungetch */
int bufp = 0; /* next free position in buf */

int getch(void) /* get a (possibly pushed back) character */
{
    return (bufp > 0) ? buf[--bufp] : getchar();
}

void ungetch(int c) /* push character back on input */
{
    if(bufp >= BUFSIZE)
        printf("ungetch: too many characters\n");
    else
        buf[bufp++] = c;
}

```


[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 4-1, page 71

Solution by [Rick Dearman](#)

Write the function `strrindex(s, t)`, which returns the position of the rightmost occurrence of `t` in `s`, or `-1` if there is none.

```

/* Test driver by Richard Heathfield
 * Solution (strrindex function) by Rick Dearman
 */

#include <stdio.h>

/* Write the function strrindex(s,t), which returns the position
** of the rightmost occurrence of t in s, or -1 if there is none.
*/

int strrindex( char s[], char t )
{
    int i;
    int count = -1;

    for(i=0; s[i] != '\0'; i++)
    {
        if(s[i] == t)
        {
            count = i;
        }
    }

    return count;
}

typedef struct TEST
{
    char *data;
    char testchar;
    int expected;
}

```

```

} TEST;

int main(void)
{
    TEST test[] =
    {
        {"Hello world", 'o', 7},
        {"This string is littered with iiiis", 'i', 32},
        {"No 'see' letters in here", 'c', -1}
    };

    size_t numtests = sizeof test / sizeof test[0];
    size_t i;

    char ch = 'o';
    int pos;

    for(i = 0; i < numtests; i++)
    {
        pos = strrindex(test[i].data, test[i].testchar);

        printf("Searching %s for last occurrence of %c.\n",
               test[i].data,
               test[i].testchar);

        printf("Expected result: %d\n", test[i].expected);
        printf("%sorrect (%d).\n", pos == test[i].expected ? "C" : "Inc", pos);
        if(pos != -1)
        {
            printf("Character found was %c\n", test[i].data[pos]);
        }
    }

    return 0;
}

```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 4-2, page 73

Solution by [Dann Corbit](#)

Extend atof to handle scientific notation of the form 123.45e-6 where a floating-point number may be followed by e or E and an optionally signed exponent.

```

/*
**  Written by Dann Corbit as K&R 2, Exercise 4-2 (Page 73).
**  Keep in mind that this is *JUST* a student exercise, and is
**  light years away from being robust.
**
**  Actually, it's kind of embarrassing, but I'm too lazy to fix it.
**
**  Caveat Emptor, not my fault if demons fly out of your nose,
**  and all of that.
*/
#include <ctype.h>
#include <limits.h>
#include <float.h>
#include <signal.h>
#include <stdio.h>

int my_atof(char *string, double *pnumber)
{
    /* Convert char string to double data type. */
    double      retval;
    double      one_tenth = 0.1;
    double      ten = 10.0;
    double      zero = 0.0;
    int         found_digits = 0;
    int         is_negative = 0;
    char        *num;

    /* Check pointers. */
    if (pnumber == 0) {
        return 0;
    }

```

```

if (string == 0) {
    *pnumber = zero;
    return 0;
}
retval = zero;

num = string;

/* Advance past white space. */
while (isspace(*num))
    num++;

/* Check for sign. */
if (*num == '+')
    num++;
else if (*num == '-') {
    is_negative = 1;
    num++;
}
/* Calculate the integer part. */
while (isdigit(*num)) {
    found_digits = 1;
    retval *= ten;
    retval += *num - '0';
    num++;
}

/* Calculate the fractional part. */
if (*num == '.') {
    double          scale = one_tenth;
    num++;
    while (isdigit(*num)) {
        found_digits = 1;
        retval += scale * (*num - '0');
        num++;
        scale *= one_tenth;
    }
}
/* If this is not a number, return error condition. */
if (!found_digits) {
    *pnumber = zero;
    return 0;
}
/* If all digits of integer & fractional part are 0, return 0.0 */
if (retval == zero) {

```

```

    *pnumber = zero;
    return 1;                                /* Not an error condition, and no need to
                                              * continue. */
}
/* Process the exponent (if any) */
if ((*num == 'e') || (*num == 'E')) {
    int        neg_exponent = 0;
    int        get_out = 0;
    long       index;
    long       exponent = 0;
    double     getting_too_big = DBL_MAX * one_tenth;
    double     getting_too_small = DBL_MIN * ten;

    num++;
    if (*num == '+')
        num++;
    else if (*num == '-') {
        num++;
        neg_exponent = 1;
    }
    /* What if the exponent is empty?  Return the current result. */
    if (!isdigit(*num)) {
        if (is_negative)
            retval = -retval;

        *pnumber = retval;

        return (1);
    }
    /* Convert char exponent to number <= 2 billion. */
    while (isdigit(*num) && (exponent < LONG_MAX / 10)) {
        exponent *= 10;
        exponent += *num - '0';
        num++;
    }

    /* Compensate for the exponent. */
    if (neg_exponent) {
        for (index = 1; index <= exponent && !get_out; index++)
            if (retval < getting_too_small) {
                get_out = 1;
                retval = DBL_MIN;
            } else
                retval *= one_tenth;
    } else

```

```

        for (index = 1; index <= exponent && !get_out; index++) {
            if (retval > getting_too_big) {
                get_out = 1;
                retval = DBL_MAX;
            } else
                retval *= ten;
        }
    }
    if (is_negative)
        retval = -retval;

    *pnumber = retval;

    return (1);
}
/*
**  lame and evil wrapper function to give the exercise the requested
**  interface.  Dann Corbit will plead innocent to the end.
**  It's very existence means that the code is not conforming.
**  Pretend you are a C library implementer, OK?  But you would fix
**  all those bleeding gaps, I am sure.
*/
double atof(char *s)
{
    double          d = 0.0;
    if (!my_atof(s, &d))
    {
#ifdef DEBUG
        fputs("Error converting string in [sic] atof()", stderr);
#endif
        raise(SIGFPE);
    }
    return d;
}

#ifdef UNIT_TEST
char  *strings[] = {
    "1.0e43",
    "999.999",
    "123.456e-9",
    "-1.2e-3",
    "1.2e-3",
    "-1.2E3",
    "-1.2e03",
    "cat",

```

```
    " ",  
    0  
};  
int  main(void)  
{  
    int          i = 0;  
    for (; *strings[i]; i++)  
        printf("atof(%s) = %g\n", strings[i], atof(strings[i]));  
    return 0;  
}  
#endif
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 4-3, page 79

Solution by [Bob Wightman](#)

Given the basic framework, it's straightforward to extend the calculator. Add the modulus (%) operator and provisions for negative numbers.

In Bob's words: "Here's my attempt Adding the modulus is easily done by another case in main and using the fmod function. The standard library has been mentioned at this point so it should be valid to use this for type 0 compliance. math.h should be added to the list of #includes for fmod."

```
int main(void)
{
    int type;
    double op2;
    char s[MAXOP];
    int flag = TRUE;

    while((type = Getop(s)) != EOF)
    {
        switch(type)
        {
            /* other cases snipped for brevity */

            case '%':
                op2 = pop();
                if(op2)
                    push(fmod(pop(), op2));
                else
                    printf("\nError: Division by zero!");
                break;

        }
    }
    return EXIT_SUCCESS;
}
```

Bob goes on to say: "Deal with unary minus when retrieving tokens. This is based on the fact that a unary minus will have no intervening space between it and its operand."

```

/* Getop: get next operator or numeric operand. */
int Getop(char s[])
{
    #define PERIOD  '.'
    int i = 0;
    int c;
    int next;

    /* Skip whitespace */
    while((s[0] = c = getch()) == ' ' || c == '\t')
        ;
    s[1] = '\0';

    /* Not a number but may contain a unary minus. */
    if(!isdigit(c) && c != PERIOD && c != '-')
        return c;

    if(c == '-')
    {
        next = getch();
        if(!isdigit(next) && next != PERIOD)
        {
            return c;
        }
        c = next;
    }
    else
    {
        c = getch();
    }

    while(isdigit(s[++i] = c))
        c = getch();
    if(c == PERIOD) /* Collect fraction part. */
        while(isdigit(s[++i] = c = getch()))
            ;
    s[i] = '\0';
}

```

```
    if(c != EOF)
        unGetch(c);
    return NUMBER;
}
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 4-4, page 79

Solution by Bob Wightman

Add commands to print the top element of the stack without popping, to duplicate it, and to swap the top two elements. Add a command to clear the stack.

```
#include<stdlib.h>
#include<stdio.h>
#include<ctype.h>
#include<math.h>
```

```
#define MAXOP 100
#define NUMBER 0
#define TRUE 1
#define FALSE 0
```

```
/* This programme is a basic calculator.
```

Extra cases have been added to:

1. Show the top item of the stack without permanently popping it.
2. Swap the top two items on the stack.
3. Duplicate the top item on the stack.
4. Clear the stack.

I have used functions for each of the new cases rather than have the code inline in order to limit the physical size of the switch block.

In anticipation of the following exercise the following characters have been used for the operations (in the same order as above): ? ~ # ! rather than use alphabetic characters.

It is actually rather difficult to be original in this exercise.

This is exercise 4-4 from Kernighan & Ritchie, page 79.

```
*/
```

```
int Getop(char s[]);
void push(double val);
double pop(void);
void showTop(void);
void duplicate(void);
void swapItems(void);
void clearStack();

int main(void)
{
    int type;
    double op2;
    char s[MAXOP];
    int flag = TRUE;

    while((type = Getop(s)) != EOF)
    {
        switch(type)
        {
            case NUMBER:
                push(atof(s));
                break;
            case '+':
                push(pop() + pop());
                break;
            case '*':
                push(pop() * pop());
                break;
            case '-':
                op2 = pop();
                push(pop() - op2);
                break;
            case '/':
                op2 = pop();
                if(op2)
                    push(pop() / op2);
                else
                    printf("\nError: division by zero!");
                break;
            case '%':
                op2 = pop();
```

```

        if(op2)
            push(fmod(pop(), op2));
        else
            printf("\nError: division by zero!");
        break;
    case '?':
        showTop();
        break;
    case '#':
        duplicate();
        break;
    case '~':
        swapItems();
        break;
    case '!':
        clearStack();
    case '\n':
        printf("\n\t%.8g\n", pop());
        break;
    default:
        printf("\nError: unknown command %s.\n", s);
        break;
    }
}
return EXIT_SUCCESS;
}

#define MAXVAL 100

int sp = 0;           /* Next free stack position. */
double val[MAXVAL];   /* value stack. */

/* push: push f onto stack. */
void push(double f)
{
    if(sp < MAXVAL)
        val[sp++] = f;
    else
        printf("\nError: stack full can't push %g\n", f);
}

/*pop: pop and return top value from stack.*/
double pop(void)

```

```

{
    if(sp > 0)
        return val[--sp];
    else
    {
        printf("\nError: stack empty\n");
        return 0.0;
    }
}

void showTop(void)
{
    if(sp > 0)
        printf("Top of stack contains: %8g\n", val[sp-1]);
    else
        printf("The stack is empty!\n");
}

void duplicate(void)
{
    double temp = pop();

    push(temp);
    push(temp);
}

void swapItems(void)
{
    double item1 = pop();
    double item2 = pop();

    push(item1);
    push(item2);
}

/* pop only returns a value if sp is greater than zero. So setting the
stack pointer to zero will cause pop to return its error */

void clearStack(void)
{
    sp = 0;
}

```

```

int getch(void);
void unGetch(int);

/* Getop: get next operator or numeric operand. */
int Getop(char s[])
{
    int i = 0;
    int c;
    int next;

    /* Skip whitespace */
    while((s[0] = c = getch()) == ' ' || c == '\t')
        ;
    s[1] = '\0';

    /* Not a number but may contain a unary minus. */
    if(!isdigit(c) && c != '.' && c != '-')
        return c;

    if(c == '-')
    {
        next = getch();
        if(!isdigit(next) && next != '.')
        {
            return c;
        }
        c = next;
    }
    else
        c = getch();

    while(isdigit(s[++i] = c))
        c = getch();
    if(c == '.') /* Collect fraction part. */
        while(isdigit(s[++i] = c = getch()))
            ;

    s[i] = '\0';
    if(c != EOF)
        unGetch(c);
    return NUMBER;
}

```



```
#define BUFSIZE 100

char buf[BUFSIZE];
int bufp = 0;

/* Getch: get a ( possibly pushed back) character. */
int getch(void)
{
    return (bufp > 0) ? buf[--bufp]: getchar();
}

/* unGetch: push character back on input. */
void unGetch(int c)
{
    if(bufp >= BUFSIZE)
        printf("\nUnGetch: too many characters\n");
    else
        buf[bufp++] = c;
}
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 4-5, page 79

Solution by Bob Wightman

Add access to library functions like `sin`, `exp`, and `pow`. See `<math.h>` in Appendix B, Section 4.

```
#include<stdlib.h>
#include<stdio.h>
#include<ctype.h>
#include<math.h>
#include <string.h>
```

```
#define MAXOP 100
#define NUMBER      0
#define IDENTIFIER  1
#define TRUE 1
#define FALSE 0
```

```
/*
```

The new additions deal with adding functions from `math.h` to the calculator.

In anticipation of the following exercise the code deals with an identifier in the following manner:

If the identifier is recognised as one of the supported mathematical functions then that function from the library is called. If the identifier is not one of the supported functions, even if it is a valid function from `math.h` it is ignored.

The main changes are the introduction of another define value (`IDENTIFIER`) along with its associated case in the switch statement.

Getop has also been changed to deal with reading in alphabetical

characters.

This is exercise 4-5 from Kernighan & Ritchie, page 79.

```

*/

int Getop(char s[]);
void push(double val);
double pop(void);
void showTop(void);
void duplicate(void);
void swapItems(void);
void clearStack();
void dealWithName(char s[]);

int main(void)
{
    int type;
    double op2;
    char s[MAXOP];
    int flag = TRUE;

    while((type = Getop(s)) != EOF)
    {
        switch(type)
        {
            case NUMBER:
                push(atof(s));
                break;
            case IDENTIFIER:
                dealWithName(s);
                break;
            case '+':
                push(pop() + pop());
                break;
            case '*':
                push(pop() * pop());
                break;
            case '-':
                op2 = pop();
                push(pop() - op2);
                break;
            case '/':

```

```

        op2 = pop();
        if(op2)
            push(pop() / op2);
        else
            printf("\nError: division by zero!");
        break;
    case '%':
        op2 = pop();
        if(op2)
            push(fmod(pop(), op2));
        else
            printf("\nError: division by zero!");
        break;
    case '?':
        showTop();
        break;
    case '#':
        duplicate();
        break;
    case '~':
        swapItems();
        break;
    case '!':
        clearStack();
    case '\n':
        printf("\n\t%.8g\n", pop());
        break;
    default:
        printf("\nError: unknown command %s.\n", s);
        break;
    }
}
return EXIT_SUCCESS;
}

#define MAXVAL 100

int sp = 0;           /* Next free stack position. */
double val[MAXVAL];   /* value stack. */

/* push: push f onto stack. */
void push(double f)
{

```

```

    if(sp < MAXVAL)
        val[sp++] = f;
    else
        printf("\nError: stack full can't push %g\n", f);
}

/*pop: pop and return top value from stack.*/
double pop(void)
{
    if(sp > 0)
        return val[--sp];
    else
    {
        printf("\nError: stack empty\n");
        return 0.0;
    }
}

void showTop(void)
{
    if(sp > 0)
        printf("Top of stack contains: %8g\n", val[sp-1]);
    else
        printf("The stack is empty!\n");
}

/*
Alternatively:
void showTop(void)
{
    double item = pop();
    printf("Top of stack contains: %8g\n", item);
    push(item);
}
*/

void duplicate(void)
{
    double temp = pop();

    push(temp);
    push(temp);
}

```

```

}

void swapItems(void)
{
    double item1 = pop();
    double item2 = pop();

    push(item1);
    push(item2);
}

void clearStack(void)
{
    sp = 0;
}

/* deal with a string/name this may be either a maths function or for
future exercises: a variable */
void dealWithName(char s[])
{
    double op2;

    if( 0 == strcmp(s, "sin"))
        push(sin(pop()));
    else if( 0 == strcmp(s, "cos"))
        push(cos(pop()));
    else if (0 == strcmp(s, "exp"))
        push(exp(pop()));
    else if(!strcmp(s, "pow"))
    {
        op2 = pop();
        push(pow(pop(), op2));
    }
    else
        printf("%s is not a supported function.\n", s);
}

int getch(void);
void unGetch(int);

/* Getop: get next operator or numeric operand. */
int Getop(char s[])
{

```

```

int i = 0;
int c;
int next;
/*size_t len;*/

/* Skip whitespace */
while((s[0] = c = getch()) == ' ' || c == '\t')
    ;
s[1] = '\0';

if(isalpha(c))
{
    i = 0;
    while(isalpha(s[i++] = c ))
        c = getch();
    s[i - 1] = '\0';
    if(c != EOF)
        unGetch(c);
    return IDENTIFIER;
}

/* Not a number but may contain a unary minus. */
if(!isdigit(c) && c != '.' && c != '-')
    return c;

if(c == '-')
{
    next = getch();
    if(!isdigit(next) && next != '.')
    {
        return c;
    }
    c = next;
}
else
    c = getch();

while(isdigit(s[++i] = c))
    c = getch();
if(c == '.') /* Collect fraction part. */
    while(isdigit(s[++i] = c = getch()))
        ;
s[i] = '\0';

```

```
        if(c != EOF)
            unGetch(c);
        return NUMBER;
    }

#define BUFSIZE 100

char buf[BUFSIZE];
int bufp = 0;

/* Getch: get a ( possibly pushed back) character. */
int getch(void)
{
    return (bufp > 0) ? buf[--bufp]: getchar();
}

/* unGetch: push character back on input. */
void unGetch(int c)
{
    if(bufp >= BUFSIZE)
        printf("\nUnGetch: too many characters\n");
    else
        buf[bufp++] = c;
}
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 4-6, page 79

Solution by Bob Wightman

*Add commands for handling variables. (It's easy to provide twenty-six variables with single-letter names.)
Add a variable for the most recently printed value.*

```
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include <math.h>
#include <string.h>

#define MAXOP      100
#define NUMBER     0
/* 4-6 these are new for this exercise*/
#define IDENTIFIER 1
#define ENDSTRING  2
/* 4-6 end of new stuff */
#define TRUE       1
#define FALSE      0
#define MAX_ID_LEN 32
#define MAXVARS    30
```

```
/*
```

The new additions deal with adding variables to the calculator.

If the identifier is recognised as one of the supported mathematical functions then that function from the library is called. If the identifier is not one of the supported functions, even if it is a valid function from math.h it is ignored.

This is a class 1 solution as it uses structures which are not introduced until Chapter 6. This allows the use of "normal" names for variables rather than the suggested single letter though any identifier is limited to 31 characters.

The main changes are:

1. The introduction of two more define values (IDENTIFIER, ENDSTRING) along with associated cases in the switch statement.
2. Getop has also been changed to deal with reading in alphabetical characters and coping with the '=' sign.
3. A structure to hold the variable name and value.
4. Another case in the switch statement to deal with the '=' sign.
5. Altering the clearStack function to clear the array of structs as well as the stack.
6. The '<' operator now prints the last accessed variable.

Improvements:

The code could be made class 0 by the use of "parallel" arrays for the names and values rather than a struct but this would be messy and is the situation that structs were made for.

The use of a binary tree together with dynamically allocated memory would allow the arbitrary limit of 30 variables to be avoided. This would still be a class 1 solution.

This is exercise 4-6 from Kernighan & Ritchie, page 79.

```
*/

/* 4-6 this is new for this program */
struct varType{
    char name[MAX_ID_LEN];
    double val;
};
/* 4-6 End of new stuff */

int Getop(char s[]);
void push(double val);
double pop(void);
void showTop(void);
void duplicate(void);
void swapItems(void);

/* 4-6 this is new for this program */
/* Changed clearStack(void) to clearStacks(struct varType var[])*
void clearStacks(struct varType var[]);
void dealWithName(char s[], struct varType var[]);
void dealWithVar(char s[], struct varType var[]);

int pos = 0;
```

```

struct varType last;

/* 4-6 End of new stuff */

int main(void)
{
    int type;
    double op2;
    char s[MAXOP];
    struct varType var[MAXVARS];

    /* Use the new function here */
    clearStacks(var);

    while((type = Getop(s)) != EOF)
    {
        switch(type)
        {
            case NUMBER:
                push(atof(s));
                break;
            case IDENTIFIER:
                dealWithName(s, var);
                break;
            case '+':
                push(pop() + pop());
                break;
            case '*':
                push(pop() * pop());
                break;
            case '-':
                op2 = pop();
                push(pop() - op2);
                break;
            case '/':
                op2 = pop();
                if(op2)
                    push(pop() / op2);
                else
                    printf("\nError: division by zero!");
                break;
            case '%':
                op2 = pop();
                if(op2)

```

```

        push(fmod(pop(), op2));
    else
        printf("\nError: division by zero!");
        break;
case '?':
    showTop();
    break;
case '#':
    duplicate();
    break;
case '~':
    swapItems();
    break;
case '!':
    clearStacks(var);
    break;
case '\\n':
    printf("\n\t%.8g\n", pop());
    break;
    /* 4-6 this is new for this program */
case ENDSTRING:
    break;
case '=':
    pop();
    var[pos].val = pop();
    last.val = var[pos].val;
    push(last.val);
    break;
case '<':
    printf("The last variable used was: %s (value == %g)\n",
           last.name, last.val);

    break;
    /* 4-6 End of new stuff */
default:
    printf("\nError: unknown command %s.\n", s);
    break;
}
}
return EXIT_SUCCESS;
}

```

```
#define MAXVAL 100
```

```
int sp = 0;           /* Next free stack position. */
```

```

double val[MAXVAL]; /* value stack. */

/* push: push f onto stack. */
void push(double f)
{
    if(sp < MAXVAL)
        val[sp++] = f;
    else
        printf("\nError: stack full can't push %g\n", f);
}

/*pop: pop and return top value from stack.*/
double pop(void)
{
    if(sp > 0)
    {
        return val[--sp];
    }
    else
    {
        printf("\nError: stack empty\n");
        return 0.0;
    }
}

void showTop(void)
{
    if(sp > 0)
        printf("Top of stack contains: %8g\n", val[sp-1]);
    else
        printf("The stack is empty!\n");
}

/*
Alternatively:
void showTop(void)
{
    double item = pop();
    printf("Top of stack contains: %8g\n", item);
    push(item);
}
*/

void duplicate(void)

```

```

{
    double temp = pop();

    push(temp);
    push(temp);
}

void swapItems(void)
{
    double item1 = pop();
    double item2 = pop();

    push(item1);
    push(item2);
}

/* 4-6 this is new for this program */
/* Altered to clear both the main stack and that of the variable
structure */
void clearStacks(struct varType var[])
{
    int i;

    /* Clear the main stack by setting the pointer to the bottom. */
    sp = 0;

    /* Clear the variables by setting the initial element of each name
to the terminating character. */
    for( i = 0; i < MAXVARS; ++i)
    {
        var[i].name[0] = '\0';
        var[i].val = 0.0;
    }
}

/* a string/name may be either a maths function or a variable */
void dealWithName(char s[], struct varType var[])
{
    double op2;

    if(!strcmp(s, "sin"))
        push(sin(pop()));
    else if(!strcmp(s, "cos"))
        push(cos(pop()));
}

```

```

else if (!strcmp(s, "exp"))
    push(exp(pop()));
else if(!strcmp(s, "pow"))
{
    op2 = pop();
    push(pow(pop(), op2));
}
/* Finally if it isn't one of the supported maths functions we have
a
    variable to deal with. */
else
{
    dealWithVar(s, var);
}
}

/* Our identifier is not one of the supported maths function so we have
to regard it as an identifier. */
void dealWithVar(char s[], struct varType var[])
{
    int i = 0;

    while(var[i].name[0] != '\0' && i < MAXVARS-1)
    {
        if(!strcmp(s, var[i].name))
        {
            strcpy(last.name, s);
            last.val = var[i].val;
            push(var[i].val);
            pos = i;
            return;
        }
        i++;
    }

    /* variable name not found so add it */
    strcpy(var[i].name, s);
    /* And save it to the last variable */
    strcpy(last.name, s);
    push(var[i].val);
    pos = i;
}

/* 4-6 End of new stuff */

```

```

int getch(void);
void unGetch(int);

/* Getop: get next operator or numeric operand. */
int Getop(char s[])
{
    int i = 0;
    int c;
    int next;

    /* Skip whitespace */
    while((s[0] = c = getch()) == ' ' || c == '\t')
    {
        ;
    }
    s[1] = '\0';

    if(isalpha(c))
    {
        i = 0;
        while(isalpha(s[i++] = c ))
        {
            c = getch();
        }
        s[i - 1] = '\0';
        if(c != EOF)
            unGetch(c);
        return IDENTIFIER;
    }

    /* Not a number but may contain a unary minus. */
    if(!isdigit(c) && c != '.' && c != '-')
    {
        /* 4-6 Deal with assigning a variable. */
        if('=' == c && '\n' == (next = getch()))
        {
            unGetch('\0');
            return c;
        }
        if('\0' == c)
            return ENDSTRING;

        return c;
    }
}

```



```

    if(c == '-')
    {
        next = getch();
        if(!isdigit(next) && next != '.')
        {
            return c;
        }
        c = next;
    }
    else
    {
        c = getch();
    }

    while(isdigit(s[++i] = c))
    {
        c = getch();
    }
    if(c == '.')                /* Collect fraction part. */
    {
        while(isdigit(s[++i] = c = getch()))
            ;
    }
    s[i] = '\0';
    if(c != EOF)
        unGetch(c);
    return NUMBER;
}

#define BUFSIZE 100

int buf[BUFSIZE];
int bufp = 0;

/* Getch: get a ( possibly pushed back) character. */
int getch(void)
{
    return (bufp > 0) ? buf[--bufp]: getchar();
}

/* unGetch: push character back on input. */
void unGetch(int c)
{

```

```
if(bufp >= BUFSIZE)
    printf("\nUnGetch: too many characters\n");
else
    buf[bufp++] = c;
}
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 4-7, page 79

Solution by Steven Huang

Write a routine `ungets(s)` that will push back an entire string onto the input. Should `ungets` know about `buf` and `bufp`, or should it just use `ungetch`?

```
/* K&R Exercise 4-7 */
/* Steven Huang */

#include <string.h>
#include <stdio.h>

#define BUFSIZE 100

char buf[BUFSIZE]; /* buffer for ungetch */
int bufp = 0; /* next free position in buf */

int getch(void) /* get a (possibly pushed back) character */
{
    return (bufp > 0) ? buf[--bufp] : getchar();
}

void ungetch(int c) /* push character back on input */
{
    if(bufp >= BUFSIZE)
        printf("ungetch: too many characters\n");
    else
        buf[bufp++] = c;
}

/*
ungets() actually takes a little bit of thought. Should the
first character in "s" be sent to ungetch() first, or should
it be sent last? I assumed that most code calling getch()
would be of this form:
```

```

char array[...];
int i;

while (...) {
    array[i++] = getch();
}

```

In such cases, the same code might call `ungets()` as:

```
ungets(array);
```

and expect to repeat the while loop to get the same string back. This requires that the last character be sent first to `ungetch()` first, because `getch()` and `ungetch()` work with a stack.

To answer K&R2's additional question for this problem, it's usually preferable for something like `ungets()` to just build itself on top of `ungetch()`. This allows us to change `ungetch()` and `getch()` in the future, perhaps to use a linked list instead, without affecting `ungets()`.

```

*/
void ungets(const char *s)
{
    size_t i = strlen(s);

    while (i > 0)
        ungetch(s[--i]);
}

int main(void)
{
    char *s = "hello, world.  this is a test.";
    int c;

    ungets(s);
    while ((c = getch()) != EOF)
        putchar(c);
    return 0;
}

```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 4-8, page 79

Solution by Steven Huang

Suppose there will never be more than one character of pushback. Modify `getch` and `ungetch` accordingly.

```
/* K&R Exercise 4-8 */
/* Steven Huang */

#include <stdio.h>

int buf = EOF; /* buffer for ungetch */

int getch(void) /* get a (possibly pushed back) character */
{
    int temp;

    if (buf != EOF) {
        temp = buf;
        buf = EOF;
    } else {
        temp = getchar();
    }
    return temp;
}

void ungetch(int c) /* push character back on input */
{
    if(buf != EOF)
        printf("ungetch: too many characters\n");
    else
        buf = c;
}

int main(void)
```

```
{
    int c;

    while ((c = getch()) != EOF) {
        if (c == '/') {
            putchar(c);
            if ((c = getch()) == '*') {
                ungetch('!');
            }
        }
        putchar(c);
    }
    return 0;
}
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 4-12, page 88

Solution by Gregory Pietsch

Adapt the ideas of `printf` to write a recursive version of `atoi` ; that is, convert an integer into a string by calling a recursive routine.

```
/*
```

`itoa()` is non-standard, but defined on p.64 as having this prototype:

```
void itoa(int n, char s[])
```

Instead of this, I thought I'd use a different prototype (one I got from the library manual of one of my compilers) since it includes all of the above:

```
char *itoa(int value, char *digits, int base);
```

Description: The `itoa()` function converts an integer value into an ASCII string of digits. The base argument specifies the number base for the conversion. The base must be a value in the range [2..36], where 2 is binary, 8 is octal, 10 is decimal, and 16 is hexadecimal. The buffer pointed to by `digits` must be large enough to hold the ASCII string of digits plus a terminating null character. The maximum amount of buffer space used is the precision of an `int` in bits + 2 (one for the sign and one for the terminating null).

Returns: `digits`, or `NULL` if error.

```
*/
```

```
#include <stdlib.h>
```

```
char *utoa(unsigned value, char *digits, int base)
{
    char *s, *p;
```



```

s = "0123456789abcdefghijklmnopqrstuvwxyz"; /* don't care if s is in
                                             * read-only memory
                                             */

if (base == 0)
    base = 10;
if (digits == NULL || base < 2 || base > 36)
    return NULL;
if (value < (unsigned) base) {
    digits[0] = s[value];
    digits[1] = '\0';
} else {
    for (p = utoa(value / ((unsigned)base), digits, base);
         *p;
         p++);
    utoa( value % ((unsigned)base), p, base);
}
return digits;
}

char *itoa(int value, char *digits, int base)
{
    char *d;
    unsigned u; /* assume unsigned is big enough to hold all the
                 * unsigned values -x could possibly be -- don't
                 * know how well this assumption holds on the
                 * DeathStation 9000, so beware of nasal demons
                 */

    d = digits;
    if (base == 0)
        base = 10;
    if (digits == NULL || base < 2 || base > 36)
        return NULL;
    if (value < 0) {
        *d++ = '-';
        u = -value;
    } else
        u = value;
    utoa(u, d, base);
    return digits;
}

```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 4-13, page 88

Solution by Gregory Pietsch

Write a recursive version of the function `reverse(s)`, which reverses the string `s` in place.

```
/*
EXERCISE 4-13 Gregory Pietsch
*/

static void swap(char *a, char *b, size_t n)
{
    while (n--) {
        *a ^= *b;
        *b ^= *a;
        *a ^= *b;
        a++;
        b++;
    }
}

void my_memrev(char *s, size_t n)
{
    switch (n) {
        case 0:
        case 1:
            break;
        case 2:
        case 3:
            swap(s, s + n - 1, 1);
            break;
        default:
            my_memrev(s, n / 2);
            my_memrev(s + ((n + 1) / 2), n / 2);
            swap(s, s + ((n + 1) / 2), n / 2);
    }
}
```

```
        break;
    }
}

void reverse(char *s)
{
    char *p;

    for (p = s; *p; p++)
        ;
    my_memrev(s, (size_t)(p - s));
}
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 4-14, page 91

Solutions by Gregory Pietsch and Lars Wirzenius

Define a macro `swap(t,x,y)` that interchanges two arguments of type `t`. (Block structure will help.)

Here are Greg's solutions for Cat 0 and Cat 1:

```

/* EXERCISE 4-14 Gregory Pietsch */

/* conditional compilation added by RJH */

#ifdef CATEGORY_0

#define swap(t,x,y) do{t z=x;x=y;y=z}while(0)

#else
#ifdef CATEGORY_1

/*
This works if I can use the assignment operator on type t.
I didn't know if I was allowed to use sizeof or not and still remain
Level 0, otherwise this one is better:
*/

#define swap(t,x,y) \
do { \
    (unsigned char *)a=(unsigned char *)&(x); \
    (unsigned char *)b=(unsigned char *)&(y); \
    size_t i = sizeof(t); \
    while (i--) { \
        *a ^= *b; \
        *b ^= *a; \
        *a ^= *b; \
        a++; \
        b++; \
    } \
} while (0)

#endif
#endif

/* editor's note: sizeof is first mentioned on p91, after this exercise,
 * and is not explained properly until p135, so it can be used in

```

```
* Category 0 solutions only for exercises 6-1 onward.
*/
```

...and here is a lively entry for Category 0, from Lars, which uses token pasting to derive a name for the temporary variable:

```
/*
 * Solution to exercise 4-14 in K&R2, page 91:
 *
 *      Define a macro swap(t,x,y) that interchanges two arguments of type t.
 *      (Block structure will help.)
 *
 * Feel free to modify and copy, if you really must, but preferably not.
 * This is just an exercise in preprocessor mechanics, not an example of
 * how it should really be used. The trickery is not worth it to save three
 * lines of code.
 *
 * To exchange the values of two variables we need a temporary variable and
 * this one needs a name. Any name we pick, the user of the macro might also
 * use. Thus, we use the preprocessor argument concatenation operator ## to
 * create the name from the actual variable names in the call. This guarantees
 * that the result won't be either of the actual arguments. In order to
 * make sure the result also does not fall into the implementation's name
 * space, we prefix the name with something safe.
 *
 * Lars Wirzenius <liw@iki.fi>
 */

#include <stdio.h>

#define swap(t, x, y) \
    do { \
        t safe ## x ## y; \
        safe ## x ## y = x; \
        x = y; \
        y = safe ## x ## y; \
    } while (0)

int main(void) {
    int ix, iy;
    double dx, dy;
    char *px, *py;
```

```

    ix = 42;
    iy = 69;
    printf("integers before swap: %d and %d\n", ix, iy);
    swap(int, ix, iy);
    printf("integers after swap: %d and %d\n", ix, iy);

    dx = 123.0;
    dy = 321.0;
    printf("doubles before swap: %g and %g\n", dx, dy);
    swap(double, dx, dy);
    printf("integers after swap: %g and %g\n", dx, dy);

    px = "hello";
    py = "world";
    printf("pointers before swap: %s and %s\n", px, py);
    swap(char *, px, py);
    printf("integers after swap: %s and %s\n", px, py);

    return 0;
}

```

...and here is yet another solution from Gregory:

```

#define swap(t,x,y) \
do { \
    (unsigned char *)_0=(unsigned char *)(&(x)); \
    (unsigned char *)_1=(unsigned char *)(&(y)); \
    unsigned long _2 = (unsigned long) \
        ((unsigned char *)(&(x)+1) \
        - (unsigned char *)(&(x))); \
    while (_2--) { \
        *_0 ^= *_1; \
        *_1 ^= *_0; \
        *_0 ^= *_1; \
        _0++; \
        _1++; \
    } \
} while (0)

```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 5-1, page 97

Solution by Gregory Pietsch

As written, `getint` treats a + or - not followed by a digit as a valid representation of zero. Fix it to push such a character back on the input.

Here is Greg's solution:

```
#include <ctype.h>

int getch(void);
void ungetch(int);

/* getint:  get next integer from input into *pn */
int getint(int *pn)
{
    int c, sign, sawsign;

    while (isspace(c = getch()))    /* skip white space */
        ;

    if (!isdigit(c) && c != EOF && c != '+' && c != '-') {
        ungetch(c);    /* it's not a number */
        return 0;
    }
    sign = (c == '-') ? -1 : 1;
    if (sawsign = (c == '+' || c == '-'))
        c = getch();
    if (!isdigit(c)) {
        ungetch(c);
        if (sawsign)
            ungetch((sign == -1) ? '-' : '+');
        return 0;
    }
    for (*pn = 0; isdigit(c); c = getch())
        *pn = 10 * *pn + (c - '0');
    *pn *= sign;
    if (c != EOF)
```

```
    ungetch(c);  
    return c;  
}
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 5-2, page 97

Solutions by [Chris Mears](#) and [Gregory Pietsch](#)

Write `getfloat`, the floating-point analog of `getint`. What type does `getfloat` return as its function value?

Here is Chris's solution:

```

/*
 * Exercise 5-2 from The C Programming Language, 2nd edition, by Kernighan
 * and Ritchie.
 *
 * "Write getfloat, the floating-point analog of getint. What type does
 * getfloat return as its function value?"
 */

/*
 * Here's the getint function, from section 5.2:
 */

#include <ctype.h>
#include <stdio.h>

int getch(void);
void ungetch(int);

/* getint: get next integer from input into *pn */
int getint(int *pn)
{
    int c, sign;

    while (isspace(c = getch())) /* skip white space */
        ;

    if (!isdigit(c) && c != EOF && c != '+' && c != '-') {
        ungetch(c); /* it is not a number */
        return 0;
    }
    sign = (c == '-') ? -1 : 1;
    if (c == '+' || c == '-')
        c = getch();
    for (*pn = 0; isdigit(c); c = getch())

```

```

        *pn = 10 * *pn + (c - '0');
    *pn *= sign;
    if (c != EOF)
        ungetch(c);
    return c;
}

```

```

/*
 * The getch and ungetch functions, from section 4.3, are also required.
 */

```

```

#include <stdio.h>

```

```

#define BUFSIZE 100

```

```

char buf[BUFSIZE];          /* buffer for ungetch */
int bufp = 0;               /* next free position in buf */

```

```

int getch(void)             /* get a (possibly pushed-back) character */
{
    return (bufp > 0) ? buf[--bufp] : getchar();
}

```

```

void ungetch(int c)         /* push character back on input */
{
    if (bufp >= BUFSIZE)
        printf("ungetch: too many characters\n");
    else
        buf[bufp++] = c;
}

```

```

/*
 * The getfloat function.
 *
 * Reads the next number from input, and puts it into *fp. Returns EOF for
 * end of file, zero if the next input is not a number, and a positive
 * value of the input contains a valid number.
 *
 * Based heavily on the getint function from K&R2.
 */

```

```

#include <ctype.h>
#include <math.h>

```

```

int getfloat(float *fp)
{
    int ch;

```

```

int sign;
int fraction;
int digits;

while (isspace(ch = getch()))    /* skip white space */
    ;

if (!isdigit(ch) && ch != EOF && ch != '+'
    && ch != '-' && ch != '.') {
    ungetch(ch);
    return 0;
}

sign = (ch == '-') ? -1 : 1;
if (ch == '+' || ch == '-') {
    ch = getch();
    if (!isdigit(ch) && ch != '.') {
        if (ch == EOF) {
            return EOF;
        } else {
            ungetch(ch);
            return 0;
        }
    }
}

*fp = 0;
fraction = 0;
digits = 0;
for ( ; isdigit(ch) || ch == '.' ; ch = getch()) {
    if (ch == '.') {
        fraction = 1;
    } else {
        if (!fraction) {
            *fp = 10 * *fp + (ch - '0');
        } else {
            *fp = *fp + ((ch - '0') / pow(10, fraction));
            fraction++;
        }
        digits++;
    }
}

*fp *= sign;

if (ch == EOF) {
    return EOF;
}

```

```

    } else {
        ungetch(ch);
        return (digits) ? ch : 0;
    }
}

/*
 * Test module.
 */

#include <stdio.h>

int main(void)
{
    int ret;

    do {
        float f;

        fputs("Enter a number: ", stdout);
        fflush(stdout);
        ret = getfloat(&f);
        if (ret > 0) {
            printf("You entered: %f\n", f);
        }
    } while (ret > 0);

    if (ret == EOF) {
        puts("Stopped by EOF.");
    } else {
        puts("Stopped by bad input.");
    }

    return 0;
}

```

...and here is Greg's solution:

```

/* Gregory Pietsch <gkp1@flash.net> Exercise 5-2 dated 2001-01-08 */

#include <ctype.h>
#include <limits.h>

```

```

/* also uses getch and ungetch from Section 4.3 */

/* number of significant digits in a double */
#define SIG_MAX 32

/* store double in d; return next character */
int getfloat(double *d)
{
    const char point = '.';      /* localeconv->decimal_point[0]; */
    int c;
    char buf[SIG_MAX], sign, sawsign, sawe, sawesign, esign;
    double x;
    static double fac[] = {0.0, 1.0e8, 1.0e16, 1.0e24, 1.0e32};
    double dpow;
    int ndigit, nsig, nzero, olead, opoint, n;
    char *pc;
    long lo[SIG_MAX / 8 + 1], lexp;
    long *pl;

    /* skip white space */
    while (isspace(c = getch()))
        ;
    if (sawsign = (c == '-' || c == '+')) {
        sign = c;
        c = getch();
    } else
        sign = '+';
    olead = -1;
    opoint = -1;
    ndigit = 0;
    nsig = 0;
    nzero = 0;
    while (c != EOF) {
        if (c == point) {
            if (0 <= opoint)
                break; /* already seen point */
            else
                opoint = ndigit;
        } else if (c == '0') {
            /* saw a zero */
            nzero++;
            ndigit++;
        } else if (!isdigit(c))
            break; /* found nondigit */
        else {
            /* got a nonzero digit */
            if (olead < 0)

```

```

        olead = nzero;
    else {
        /* deliver zeros */
        for ( ; 0 < nzero && nsig < SIG_MAX; --nzero)
            buf[nsig++] = 0;
    }
    ++ndigit;
    /* deliver digit */
    if (nsig < SIG_MAX)
        buf[nsig++] = (c - '0');
}
c = getch();
}
if (ndigit == 0) {
    /* no digits? */
    *d = 0.0;
    if (c != EOF)
        ungetch(c);
    if (0 <= opoint) {
        /* saw point */
        ungetch(c = point);
    }
    if (sawsign) {
        /* saw sign */
        ungetch(c = sign);
    }
    return c;
}
/* skip trailing digits */
for ( ; 0 < nsig && buf[nsig - 1] == 0; --nsig)
    ;
/* compute significand */
pc = buf;
pl = &(lo[nsig >> 3]);
for (*pl = 0, n = nsig; 0 < n; --n) {
    if ((n & 7) == 0)
        /* start new sum */
        *--pl = *pc++;
    else
        *pl = *pl * 10 + *pc++;
}
for (*d = (double)(lo[0]), n = 0; ++n <= (nsig >> 3); )
    if (lo[n] != 0)
        *d += fac[n] * (double)(lo[n]);
/* fold in any explicit exponent */
lexp = 0;
if (c == 'e' || c == 'E') {

```



```

    /* we have an explicit exponent */
    sawe = c;
    c = getch();
    if (sawesign = (c == '+' || c == '-')) {
        esign = c;
        c = getch();
    } else
        esign = '+';
    if (!isdigit(c)) {
        /* ill-formed exponent */
        if (c != EOF)
            ungetch(c);
        if (sawesign)
            ungetch(c = esign);
        c = sawe;
    } else {
        /* get exponent */
        while (isdigit(c)) {
            /* get explicit exponent digits */
            if (lexp < 100000)
                lexp = lexp * 10 + (c - '0');
            /* else overflow */
            c = getch();
        }
        if (esign == '-')
            lexp = -lexp;
    }
}
if (c != EOF)
    ungetch(c);
if (opoint < 0)
    lexp += ndigit - nsig;
else
    lexp += opoint - olead - nsig;
/* this is where I pray I don't lose precision */
esign = (lexp < 0) ? '-' : '+';
/* if anyone has a better way of handling overflow, tell me */
if (lexp < SHRT_MIN)
    lexp = SHRT_MIN;
if (lexp > SHRT_MAX)
    lexp = SHRT_MAX;
if (lexp < 0)
    lexp = -lexp;
if (lexp != 0) {
    dpow = (esign == '-') ? 0.1 : 10.0;
    while (lexp != 0) {
        /* form 10.0 to the lexp power */

```

```
        if ((lexp & 1) != 0) /* lexp is positive */
            *d *= dpow;
        lexp >>= 1;
        dpow *= dpow;
    }
}
/* if there was a minus sign in front, negate *d */
if (sign == '-')
    *d = -(*d);
return c;
}
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 5-3, page 107

Solution by [Richard Heathfield](#)

Write a pointer version of the function `strcat` that we showed in Chapter 2: `strcat(s, t)` copies the string `t` to the end of `s`.

```

/* ex 5-3, p107 */

#include <stdio.h>

void strcpy(char *s, char *t)
{
    while(*s++ = *t++);
}

void strcat(char *s, char *t)
{
    while(*s)
    {
        ++s;
    }
    strcpy(s, t);
}

int main(void)
{
    char testbuff[128];

    char *test[] =
    {
        "",
        "1",
        "12",
        "123",
        "1234"
    };

    size_t numtests = sizeof test / sizeof test[0];
    size_t thistest;
    size_t inner;

    for(thistest = 0; thistest < numtests; thistest++)
    {
        for(inner = 0; inner < numtests; inner++)

```

```

    {
        strcpy(testbuff, test[thistest]);
        strcat(testbuff, test[inner]);

        printf("[%s] + [%s] = [%s]\n", test[thistest], test[inner], testbuff);
    }
}

return 0;
}

```

Give nineteen programmers a spec, and you'll get at least twenty completely different programs. As a tiny example of this, here's a totally different solution, by [Bryan Williams](#).

```

/*

Exercise 5-3. Write a pointer version of the function strcat that we showed in
Chapter 2: strcat(s,t) copies the string t to the end of s.

implementation from chapter 2:

/ * strcat:  concatenate t to end of s; s must be big enough * /
void strcat(char s[], char t[])
{
    int i, j;

    i = j = 0;
    while (s[i] != '\0') / * find end of s * /
        i++;
    while ((s[i++] = t[j++]) != '\0') / * copy t * /
        ;
}

Author : Bryan Williams

*/

/* strcat:  concatenate t to end of s; s must be big enough; pointer version */
void strcat(char *s, char *t)
{
    /* run through the destination string until we point at the terminating '\0'
    */
    while(*s != '\0')
    {

```

```
    ++s;
}

/* now copy until we run out of string to copy */
while('\0' != (*s = *t))
{
    ++s;
    ++t;
}
}

#define DRIVER          6

#ifdef DRIVER
#include <stdio.h>

int main(void)
{
    char S1[8192] = "String One";
    char S2[8192] = "String Two";

    printf("String one is (%s)\n", S1);
    printf("String two is (%s)\n", S2);

    strcat(S1, S2);
    printf("The combined string is (%s)\n", S1);

    return 0;
}

#endif
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 5-4, page 107

Solution by Bryan Williams

Write the function `strend(s,t)`, which returns 1 if the string `t` occurs at the end of the string `s`, and zero otherwise.

```

/*
    Exercise 5-4. Write the function strend(s,t), which returns 1 if the string t
    occurs at the end of the string s, and zero otherwise.

    Author : Bryan Williams
*/

int strlen(char *s) /* added by RJH; source: K&R p99 */
{
    int n;

    for(n = 0; *s != '\0'; s++)
    {
        n++;
    }
    return n;
}

int strcmp(char *s, char *t) /* added by RJH; source: K&R p106 */
{
    for(; *s == *t; s++, t++)
        if(*s == '\0')
            return 0;
    return *s - *t;
}

int strend(char *s, char *t)
{
    int Result = 0;
    int s_length = 0;
    int t_length = 0;

    /* get the lengths of the strings */
    s_length = strlen(s);
    t_length = strlen(t);

    /* check if the lengths mean that the string t could fit at the string s */
    if(t_length <= s_length)

```

```

{
    /* advance the s pointer to where the string t would have to start in string s */
    s += s_length - t_length;

    /* and make the compare using strcmp */
    if(0 == strcmp(s, t))
    {
        Result = 1;
    }
}

return Result;
}

#include <stdio.h>

int main(void)
{
    char *s1 = "some really long string.";
    char *s2 = "ng.";
    char *s3 = "ng";

    if(strend(s1, s2))
    {
        printf("The string (%s) has (%s) at the end.\n", s1, s2);
    }
    else
    {
        printf("The string (%s) doesn't have (%s) at the end.\n", s1, s2);
    }
    if(strend(s1, s3))
    {
        printf("The string (%s) has (%s) at the end.\n", s1, s3);
    }
    else
    {
        printf("The string (%s) doesn't have (%s) at the end.\n", s1, s3);
    }

    return 0;
}

```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 5-5, page 107

Solution by Lars Wirzenius

Write versions of the library functions `strncpy`, `strncat`, and `strncmp`, which operate on at most the first n characters of their argument strings. For example, `strncpy(s, t, n)` copies at most n characters of t to s . Full descriptions are in Appendix B.

Note: Lars uses `EXIT_FAILURE` in his test code, but not in the actual solution code. As far as I can tell, then, this is a Category 0 solution.

```

/*
 * Solution to exercise 5-5 in K&R2, page 107:
 *
 * Write versions of the library functions strncpy, strncat,
 * and strncmp, which operate on at most the first n characters
 * of their argument strings. For example, strncpy(s,t,n) copies
 * at most n characters of t to s. Full descriptions are in
 * Appendix B.
 *
 * Note that the description in the exercise is not precise. Here are
 * descriptions from Appendix B (though one should really follow the
 * descriptions in the standard):
 *
 * char *strncpy(s,ct,n)    copy at most n characters of string ct
 *                          to s, return s. Pad with '\0's if ct
 *                          has fewer than n characters.
 * char *strncat(s,ct,n)    concatenate at most n characters of
 *                          string ct to string s, terminate s with
 *                          '\0'; return s.
 * int  strncmp(cs,ct,n)    compare at most n characters of string
 *                          cs to string ct; return <0 if cs<ct,
 *                          0 if cs==ct, or >0 if cs>ct.
 *
 * Further note that the standard requires strncmp to compare the
 * characters using unsigned char internally.
 *
 * Implementation note: since the function names are reserved by the
 * standard, I've used the prefix `liw_'. This also allows me to check
 * the functions against the standard library versions. For each library
 * function, I've written a test function that tests a particular test
 * case. Where appropriate, the test functions use internal buffers that
 * are of size MAX_BUF; at least some of the test cases should be longer
 * to test all boundary conditions.

```

```

*
* Feel free to modify, copy, and use as you wish.
*
* Lars Wirzenius <liw@iki.fi>
*/

```

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

```

```

#define MAX_BUF 16

```

```

char *liw_strncpy(char *s, const char *ct, size_t n) {
    char *p;

    p = s;
    for (; n > 0 && *ct != '\0'; --n)
        *p++ = *ct++;
    for (; n > 0; --n)
        *p++ = '\0';
    return s;
}

```

```

char *liw_strncat(char *s, const char *ct, size_t n) {
    char *p;

    p = s;
    while (*p != '\0')
        ++p;
    for (; n > 0 && *ct != '\0'; --n)
        *p++ = *ct++;
    *p = '\0';
    return s;
}

```

```

int liw_strncmp(const char *cs, const char *ct, size_t n) {
    while (n > 0 && *cs == *ct && *cs != '\0') {
        ++cs;
        ++ct;
        --n;
    }
    if (n == 0 || *cs == *ct)
        return 0;
    if (*(unsigned char *) cs < *(unsigned char *) ct)
        return -1;
    return 1;
}

```

```

void test_ncpy(const char *str) {
    char std_buf[MAX_BUF];
    char liw_buf[MAX_BUF];

    memset(std_buf, 0x42, sizeof(std_buf));
    strncpy(std_buf, str, sizeof(std_buf));

    memset(liw_buf, 0x42, sizeof(liw_buf));
    liw_strncpy(liw_buf, str, sizeof(liw_buf));

    if (memcmp(std_buf, liw_buf, sizeof(std_buf)) != 0) {
        fprintf(stderr, "liw_strncpy failed for <%s>\n", str);
        exit(EXIT_FAILURE);
    }
}

void test_ncat(const char *first, const char *second) {
    char std_buf[MAX_BUF];
    char liw_buf[MAX_BUF];

    memset(std_buf, 0x69, sizeof(std_buf));
    strcpy(std_buf, first);
    strncat(std_buf, second, sizeof(std_buf) - strlen(std_buf) - 1);

    memset(liw_buf, 0x69, sizeof(liw_buf));
    strcpy(liw_buf, first);
    liw_strncat(liw_buf, second, sizeof(liw_buf) - strlen(liw_buf) - 1);

    if (memcmp(std_buf, liw_buf, sizeof(std_buf)) != 0) {
        fprintf(stderr, "liw_strncat failed, <%s> and <%s>\n",
            first, second);
        exit(EXIT_FAILURE);
    }
}

void test_ncmp(const char *first, const char *second) {
    size_t len;
    int std_ret, liw_ret;

    if (strlen(first) < strlen(second))
        len = strlen(second);
    else
        len = strlen(first);
    std_ret = strncmp(first, second, len);
    liw_ret = liw_strncmp(first, second, len);
    if ((std_ret < 0 && liw_ret >= 0) || (std_ret > 0 && liw_ret <= 0) ||
        (std_ret == 0 && liw_ret != 0)) {
        fprintf(stderr, "liw_strncmp failed, <%s> and <%s>\n",

```

```

        first, second);
    exit(EXIT_FAILURE);
}

int main(void) {
    test_ncpy(" ");
    test_ncpy("a");
    test_ncpy("ab");
    test_ncpy("abcdefghijklmnopqrstuvxyz"); /* longer than MAX_BUF */

    test_ncat(" ", "a");
    test_ncat("a", "bc");
    test_ncat("ab", "cde");
    test_ncat("ab", "cdefghijklmnopqrstuvxyz"); /* longer than MAX_BUF */

    test_ncmp(" ", " ");
    test_ncmp(" ", "a");
    test_ncmp("a", "a");
    test_ncmp("a", "ab");
    test_ncmp("abc", "ab");

    printf("All tests pass.\n");
    return 0;
}

```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 5-6, page 107

Solution by Gregory Pietsch

Greg supplied a fresh version of this answer (which supersedes the old answer, so I've removed it) on 29 Jan 2001.

Rewrite appropriate programs from earlier chapters and exercises with pointers instead of array indexing. Good possibilities include `getline` (Chapters 1 and 4), `atoi`, `itoa`, and their variants (Chapters 2, 3, and 4), `reverse` (Chapter 3), and `strindex` and `getop` (Chapter 4).

```
/* Gregory Pietsch ex. 5-6 dated 2001-01-29 */

#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>

/* getline: get line into s, return length */
int getline(char *s, int lim)
{
    char *p;
    int c;

    p = s;
    while (--lim > 0 && (c = getchar()) != EOF && c != '\n')
        *p++ = c;
    if (c == '\n')
        *p++ = c;
    *p = '\0';
    return (int)(p - s);
}

/* atoi: convert s to an integer
 *
 * Here's the easy way:
 * int atoi(char *s){return (int)strtoul(s, NULL, 10);}
 * But I'll behave...
 */
int atoi(char *s)
```

```

{
    int n, sign;

    while (isspace(*s))
        s++;
    sign = (*s == '+' || *s == '-') ? ((*s++ == '+') ? 1 : -1) : 1;
    for (n = 0; isdigit(*s); s++)
        n = (n * 10) + (*s - '0'); /* note to language lawyers --
                                    * the digits are in consecutive
                                    * order in the character set
                                    * C90 5.2.1
                                    */

    return sign * n;
}

```

/* Shamelessly copied from my 4-12 answer

itoa() is non-standard, but defined on p.64 as having this prototype:

```
void itoa(int n, char s[])
```

Instead of this, I thought I'd use a different prototype (one I got from the library manual of one of my compilers) since it includes all of the above:

```
char *itoa(int value, char *digits, int base);
```

Description: The itoa() function converts an integer value into an ASCII string of digits. The base argument specifies the number base for

the conversion. The base must be a value in the range [2..36], where 2 is binary, 8 is octal, 10 is decimal, and 16 is hexadecimal. The buffer

pointed to by digits must be large enough to hold the ASCII string of digits plus a terminating null character. The maximum amount of buffer space used is the precision of an int in bits + 2 (one for the sign and one for the terminating null).

Returns: digits, or NULL if error.

```
*/
```

```
char *utoa(unsigned value, char *digits, int base)
```

```

{
    char *s, *p;

    s = "0123456789abcdefghijklmnopqrstuvwxyz"; /* don't care if s is in
                                                * read-only memory
                                                */

    if (base == 0)
        base = 10;
    if (digits == NULL || base < 2 || base > 36)
        return NULL;
    if (value < (unsigned) base) {
        digits[0] = s[value];
        digits[1] = '\0';
    } else {
        for (p = utoa(value / ((unsigned)base), digits, base);
            *p;
            p++);
        utoa( value % ((unsigned)base), p, base);
    }
    return digits;
}

char *itoa(int value, char *digits, int base)
{
    char *d;
    unsigned u; /* assume unsigned is big enough to hold all the
                 * unsigned values -x could possibly be -- don't
                 * know how well this assumption holds on the
                 * DeathStation 9000, so beware of nasal demons
                 */

    d = digits;
    if (base == 0)
        base = 10;
    if (digits == NULL || base < 2 || base > 36)
        return NULL;
    if (value < 0) {
        *d++ = '-';
        u = -((unsigned)value);
    } else
        u = value;
    utoa(u, d, base);
    return digits;
}

```

```

}

/* reverse, shamelessly copied from my 4-13 answer */

static void swap(char *a, char *b, size_t n)
{
    while (n--) {
        *a ^= *b;
        *b ^= *a;
        *a ^= *b;
        a++;
        b++;
    }
}

void my_memrev(char *s, size_t n)
{
    switch (n) {
    case 0:
    case 1:
        break;
    case 2:
    case 3:
        swap(s, s + n - 1, 1);
        break;
    default:
        my_memrev(s, n / 2);
        my_memrev(s + ((n + 1) / 2), n / 2);
        swap(s, s + ((n + 1) / 2), n / 2);
        break;
    }
}

void reverse(char *s)
{
    char *p;

    for (p = s; *p; p++)
        ;
    my_memrev(s, (size_t)(p - s));
}

/* strindex: return index of t in s, -1 if not found */

```



```

/* needed strchr(), so here it is: */

static char *strchr(char *s, int c)
{
    char ch = c;

    for ( ; *s != ch; ++s)
        if (*s == '\0')
            return NULL;
    return s;
}

int strindex(char *s, char *t)
{
    char *u, *v, *w;

    if (*t == '\0')
        return 0;
    for (u = s; (u = strchr(u, *t)) != NULL; ++u) {
        for (v = u, w = t; ; )
            if (*++w == '\0')
                return (int)(u - s);
            else if (*++v != *w)
                break;
    }
    return -1;
}

/* getop */

#define NUMBER '0'          /* from Chapter 4 */

int getop(char *s)
{
    int c;

    while ((*s = c = getch()) == ' ' || c == '\t')
        ;
    *(s + 1) = '\0';
    if (!isdigit(c) && c != '.')
        return c;          /* not a number */
    if (isdigit(c))         /* collect integer part */
        while (isdigit(*++s = c = getch()))
            ;

```

```
    if (c == '.')          /* collect fraction part */
        while (isdigit(*++s = c = getch()))
            ;
    *++s = '\0';
    if (c != EOF)
        ungetch(c);
    return NUMBER;
}

/* Is there any more? */
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 5-7, page 110

Solution by Steven Huang

Rewrite readlines to store lines in an array supplied by main, rather than calling alloc to maintain storage. How much faster is the program?

```
/* K&R Exercise 5-7 */
/* Steven Huang */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>

#define TRUE      1
#define FALSE     0

#define MAXLINES 5000      /* maximum number of lines */
#define MAXLEN   1000      /* maximum length of a line */
char *lineptr[MAXLINES];
char lines[MAXLINES][MAXLEN];

/* K&R2 p29 */
int getline(char s[], int lim)
{
    int c, i;

    for (i = 0; i < lim - 1 && (c = getchar()) != EOF && c != '\n'; i++)
        s[i] = c;
    if (c == '\n') {
        s[i++] = c;
    }
    s[i] = '\0';
    return i;
}
```

```

/* K&R2 p109 */
int readlines(char *lineptr[], int maxlines)
{
    int len, nlines;
    char *p, line[MAXLEN];

    nlines = 0;
    while ((len = getline(line, MAXLEN)) > 0)
        if (nlines >= maxlines || (p = malloc(len)) == NULL)
            return -1;
        else {
            line[len - 1] = '\0'; /* delete the newline */
            strcpy(p, line);
            lineptr[nlines++] = p;
        }
    return nlines;
}

int readlines2(char lines[][MAXLEN], int maxlines)
{
    int len, nlines;

    nlines = 0;
    while ((len = getline(lines[nlines], MAXLEN)) > 0)
        if (nlines >= maxlines)
            return -1;
        else
            lines[nlines++][len - 1] = '\0'; /* delete the newline */
    return nlines;
}

int main(int argc, char *argv[])
{
    /* read things into cache, to be fair. */
    readlines2(lines, MAXLINES);

    if (argc > 1 && *argv[1] == '2') {
        puts("readlines2()");
        readlines2(lines, MAXLINES);
    } else {
        puts("readlines()");
        readlines(lineptr, MAXLINES);
    }
}

```

```
    return 0;
}
```

Steven writes: "Unfortunately, the follow-up question here on which version is faster is difficult to determine on my machine, because the difference is very small. I can call malloc() one million times in under a second - this suggests that the conventional wisdom that malloc() is slow and should be avoided may need some more adjustment."

[Editor's note: That's probably because malloc is actually taking memory requests to the system as infrequently as possible, so that most of the calls invoke little more than pointer arithmetic. This suggests that the conventional wisdom may be based on real world programs, rather than artificial "how many mallocs per second can I do" benchmarks. :-)]

[This space reserved for Steven's right of reply!]

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 5-8, page 112

Solution by Lars Wirzenius

There is no error-checking in day_of_year or month_day. Remedy this defect.

```

/*
 * A solution to exercise 5-8 in K&R2, page 112:
 *
 *     There is no error checking in day_of_year or month_day. Remedy
 *     this defect.
 *
 * The error to check for is invalid argument values. That is simple, what's
 * hard is deciding what to do in case of error. In the real world, I would
 * use the assert macro from assert.h, but in this solution I take the
 * approach of returning -1 instead. This is more work for the caller, of
 * course.
 *
 * I have selected the year 1752 as the lowest allowed year, because that
 * is when Great Britain switched to the Gregorian calendar, and the leap
 * year validation is valid only for the Gregorian calendar.
 *
 * Lars Wirzenius <liw@iki.fi>
 */

#include <stdio.h>

static char daytab[2][13] = {
    {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
    {0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
};

/* day_of_year: set day of year from month & day */
int day_of_year(int year, int month, int day)
{
    int i, leap;

    if (year < 1752 || month < 1 || month > 12 || day < 1)
        return -1;

    leap = (year%4 == 0 && year%100 != 0) || year%400 == 0;
    if (day > daytab[leap][month])

```

```
        return -1;
```

```
    for (i = 1; i < month; i++)
        day += daytab[leap][i];
    return day;
}
```

```
/* month_day: set month, day from day of year */
```

```
int month_day(int year, int yearday, int *pmonth, int *pday)
{
```

```
    int i, leap;
```

```
    if (year < 1752 || yearday < 1)
        return -1;
```

```
    leap = (year%4 == 0 && year%100 != 0) || year%400 == 0;
    if ((leap && yearday > 366) || (!leap && yearday > 365))
        return -1;
```

```
    for (i = 1; yearday > daytab[leap][i]; i++)
        yearday -= daytab[leap][i];
    *pmonth = i;
    *pday = yearday;
```

```
    return 0;
}
```

```
/* main: test day_of_year and month_day */
```

```
int main(void)
{
```

```
    int year, month, day, yearday;
```

```
    for (year = 1970; year <= 2000; ++year) {
        for (yearday = 1; yearday < 366; ++yearday) {
            if (month_day(year, yearday, &month, &day) == -1) {
                printf("month_day failed: %d %d\n",
                       year, yearday);
            } else if (day_of_year(year, month, day) != yearday) {
                printf("bad result: %d %d\n", year, yearday);
                printf("month = %d, day = %d\n", month, day);
            }
        }
    }
```

```
    return 0;
}
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 5-9, page 114

Solutions by Lars Wirzenius and Gregory Pietsch

Rewrite the routines `day_of_year` and `month_day` with pointers instead of indexing.

Here's Lars's solution: @br @br

```

/*
 * A solution to exercise 5-9 in K&R2, page 114:
 *
 * Rewrite the routines day_of_year and month_day with pointers
 * instead of indexing.
 *
 * Lars Wirzenius <liw@iki.fi>
 */

#include <stdio.h>

static char daytab[2][13] = {
    {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
    {0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
};

/* original versions, for comparison purposes */

int day_of_year(int year, int month, int day)
{
    int i, leap;

    leap = (year%4 == 0 && year%100 != 0) || year%400 == 0;
    for (i = 1; i < month; i++)
        day += daytab[leap][i];
    return day;
}

```

```

void month_day(int year, int yearday, int *pmonth, int *pday)
{
    int i, leap;

    leap = (year%4 == 0 && year%100 != 0) || year%400 == 0;
    for (i = 1; yearday > daytab[leap][i]; i++)
        yearday -= daytab[leap][i];
    *pmonth = i;
    *pday = yearday;
}

/* pointer versions */

int day_of_year_pointer(int year, int month, int day)
{
    int i, leap;
    char *p;

    leap = (year%4 == 0 && year%100 != 0) || year%400 == 0;

    /* Set `p' to point at first month in the correct row. */
    p = &daytab[leap][1];

    /* Move `p' along the row, to each successive month. */
    for (i = 1; i < month; i++) {
        day += *p;
        ++p;
    }
    return day;
}

void month_day_pointer(int year, int yearday, int *pmonth, int *pday)
{
    int i, leap;
    char *p;

    leap = (year%4 == 0 && year%100 != 0) || year%400 == 0;
    p = &daytab[leap][1];
    for (i = 1; yearday > *p; i++) {
        yearday -= *p;
        ++p;
    }
}

```

```

    *pmonth = i;
    *pday = yearday;
}

int main(void)
{
    int year, month, day, yearday;

    year = 2000;
    month = 3;
    day = 1;
    printf("The date is: %d-%02d-%02d\n", year, month, day);
    printf("day_of_year: %d\n", day_of_year(year, month, day));
    printf("day_of_year_pointer: %d\n",
           day_of_year_pointer(year, month, day));

    yearday = 61;    /* 2000-03-01 */
    month_day(year, yearday, &month, &day);
    printf("Yearday is %d\n", yearday);
    printf("month_day: %d %d\n", month, day);
    month_day_pointer(year, yearday, &month, &day);
    printf("month_day_pointer: %d %d\n", month, day);

    return 0;
}

```

And here's Greg's: @br @br

```

/* Gregory Pietsch - gkpl@flash.net */

/* Given the problem, I thought that this would be a better
 * description of daytab.
 */
static int *daytab = {
    0,
    31,
    31+28,
    31+28+31,
    31+28+31+30,

```

```

    31+28+31+30+31,
    31+28+31+30+31+30,
    31+28+31+30+31+30+31,
    31+28+31+30+31+30+31+31,
    31+28+31+30+31+30+31+31+30,
    31+28+31+30+31+30+31+31+30+31,
    31+28+31+30+31+30+31+31+30+31+30,
    0,
    31,
    31+29,
    31+29+31,
    31+29+31+30,
    31+29+31+30+31,
    31+29+31+30+31+30,
    31+29+31+30+31+30+31,
    31+29+31+30+31+30+31+31,
    31+29+31+30+31+30+31+31+30,
    31+29+31+30+31+30+31+31+30+31,
    31+29+31+30+31+30+31+31+30+31+30,
};

/* is it a leap year?  (assume it's my calendar, the Gregorian) */
int leap(int year)
{
    return ((year % 4) == 0)
        && (((year % 100) != 0)
            || (year % 400) == 0));
}

/* day_of_year:  set day of year from month & day */
int day_of_year(int year, int month, int day)
{
    return *(daytab + ((month - 1) + (leap(year) * 12))) + day;
}

/* month_day:  set month, day from day of year */
void month_day(int year, int yearday, int *pmonth, int *pday)
{
    int m, ly;

    ly = leap(year);
    if (yearday < 1 || yearday > (365 + ly))
        return; /* no real error checking */

```

```
m = leap(year) ? 23 : 11;
while (*(daytab + m) > yearday)
    m--;
if (pmonth)
    *pmonth = (m % 12) + 1;
if (pday)
    *pday = yearday - (*(daytab + m));
}
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 5-10, page 118

Solution by Lars Wirzenius

Write the program `expr`, which evaluates a reverse Polish expression from the command line, where each operator or operand is a separate argument. For example,

`expr 2 3 4 + *`

evaluates $2 \times (3 + 4)$.

Note: Lars uses `EXIT_FAILURE` on error. As far as I can tell, this is the only thing which makes this a Category 1, rather than Category 0, solution.

```
/*
 * Solution to exercise 5-10 in K&R2:
 *
 *      Write the program expr, which evaluates a reverse Polish expression
 *      from the command line, where each operator or operand is a
separate
 *      argument. For example,
 *
 *              expr 2 3 4 + *
 *
 *      evaluates 2*(3+4).
 *
 * This is very similar to the program in 4.3 (and should ideally have been
 * a modification of that).
 *
 * Feel free to modify and copy freely.
 *
 * Lars Wirzenius <liw@iki.fi>
 */

#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>

#define STACK_SIZE 1024

double stack[STACK_SIZE];
int stack_height = 0;
```

```

void panic(const char *msg) {
    fprintf(stderr, "%s\n", msg);
    exit(EXIT_FAILURE);
}

void push(double value) {
    if (stack_height == STACK_SIZE)
        panic("stack is too high!");
    stack[stack_height] = value;
    ++stack_height;
}

double pop(void) {
    if (stack_height == 0)
        panic("stack is empty!");
    return stack[--stack_height];
}

int main(int argc, char **argv) {
    int i;
    double value;

    for (i = 1; i < argc; ++i) {
        switch (argv[i][0]) {
            case '\\0':
                panic("empty command line argument");
                break;

            case '0':
            case '1':
            case '2':
            case '3':
            case '4':
            case '5':
            case '6':
            case '7':
            case '8':
            case '9':
                push(atof(argv[i]));
                break;

            case '+':
                push(pop() + pop());
                break;

            case '-':
                value = pop();
                push(pop() - value);
        }
    }
}

```

```
        break;
    case '*':
        push(pop() * pop());
        break;
    case '/':
        value = pop();
        push(pop() / value);
        break;
    default:
        panic("unknown operator");
        break;
    }
}

printf("%g\n", pop());
return 0;
}
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 5-11, page 118

Solution by Gregory Pietsch

Modify the programs `entab` and `detab` (written as exercises in Chapter 1) to accept a list of tab stops as arguments. Use the default tab settings if there are no arguments.

Here's `detab`...

```

/*****
detab.c - Source code for the detab command

AUTHOR: Gregory Pietsch

DESCRIPTION:

detab - expand tabs into spaces

*****/

/* include files */
#include <stdio.h>
#include <string.h>

/* macros */
#define NO_ARG          0
#define REQUIRED_ARG     1
#define OPTIONAL_ARG    2

/* types */

/* GETOPT_LONG_OPTION_T: The type of long option */
typedef struct GETOPT_LONG_OPTION_T {
    char *name;           /* the name of the long option */
    int has_arg;          /* one of the above macros */
    int *flag;            /* determines if getopt_long() returns a
                           * value for a long option; if it is
                           * non-NULL, 0 is returned as a function
                           * value and the value of val is stored in
                           * the area pointed to by flag. Otherwise,
                           * val is returned. */
};

```

```

    int val;                /* determines the value to return if flag is
                           * NULL. */
} GETOPT_LONG_OPTION_T;

typedef enum GETOPT_ORDERING_T {
    PERMUTE,
    RETURN_IN_ORDER,
    REQUIRE_ORDER
} GETOPT_ORDERING_T;

/* globally-defined variables */
char *optarg = NULL;
int optind = 0;
int opterr = 1;
int optopt = '?';

/* statically-defined variables */

static char *program_name;
/* if nonzero, it means tab every x characters */
static unsigned long tab_every = 8;
/* -i: only handle initial tabs/spaces */
static int flag_initial = 0;
/* expand tabs into spaces */
static int flag_expand = 1;
static unsigned long *tab_stop_list = NULL;
static size_t num_tab_stops = 0;
static size_t num_tab_stops_allocated = 0;
static int show_help = 0;
static int show_version = 0;
static char *shortopts = "it:";
static GETOPT_LONG_OPTION_T longopts[] =
{
    {"initial", NO_ARG, NULL, 'i'},
    {"tabs", REQUIRED_ARG, NULL, 't'},

    {"help", NO_ARG, &show_help, 1},
    {"version", NO_ARG, &show_version, 1},
    {NULL, 0, 0, 0}
};

/* functions */

/* reverse_argv_elements:  reverses num elements starting at argv */
static void reverse_argv_elements(char **argv, int num)
{
    int i;

```

```

char *tmp;

for (i = 0; i < (num >> 1); i++) {
    tmp = argv[i];
    argv[i] = argv[num - i - 1];
    argv[num - i - 1] = tmp;
}

/* permute: swap two blocks of argv-elements given their lengths */
static void permute(char **argv, int len1, int len2)
{
    reverse_argv_elements(argv, len1);
    reverse_argv_elements(argv, len1 + len2);
    reverse_argv_elements(argv, len2);
}

/* is_option: is this argv-element an option or the end of the option
list? */
static int is_option(char *argv_element, int only)
{
    return ((argv_element == NULL)
        || (argv_element[0] == '-')
        || (only && argv_element[0] == '+'));
}

/* getopt_internal: the function that does all the dirty work */
static int getopt_internal(int argc, char **argv, char *shortopts,
    GETOPT_LONG_OPTION_T * longopts, int *longind, int
only)
{
    GETOPT_ORDERING_T ordering = PERMUTE;
    static size_t optwhere = 0;
    size_t permute_from = 0;
    int num_nonopts = 0;
    int optindex = 0;
    size_t match_chars = 0;
    char *possible_arg = NULL;
    int longopt_match = -1;
    int has_arg = -1;
    char *cp;
    int arg_next = 0;

    /* first, deal with silly parameters and easy stuff */
    if (argc == 0 || argv == NULL || (shortopts == NULL && longopts ==
NULL))
        return (optopt = '?');

```

```

if (optind >= argc || argv[optind] == NULL)
    return EOF;
if (strcmp(argv[optind], "--") == 0) {
    optind++;
    return EOF;
}
/* if this is our first time through */
if (optind == 0)
    optind = optwhere = 1;

/* define ordering */
if (shortopts != NULL && (*shortopts == '-' || *shortopts == '+')) {
    ordering = (*shortopts == '-') ? RETURN_IN_ORDER :
REQUIRE_ORDER;
    shortopts++;
}
else
    ordering = (getenv("POSIXLY_CORRECT") != NULL) ? REQUIRE_ORDER :
    PERMUTE;

/*
 * based on ordering, find our next option, if we're at the
beginning of
 * one
 */
if (optwhere == 1) {
    switch (ordering) {
    case PERMUTE:
        permute_from = optind;
        num_nonopts = 0;
        while (!is_option(argv[optind], only)) {
            optind++;
            num_nonopts++;
        }
        if (argv[optind] == NULL) {
            /* no more options */
            optind = permute_from;
            return EOF;
        } else if (strcmp(argv[optind], "--") == 0) {
            /* no more options, but have to get '--' out of the way

permute(argv + permute_from, num_nonopts, 1);
            optind = permute_from + 1;
            return EOF;
        }
        break;
    case RETURN_IN_ORDER:

```

```

        if (!is_option(argv[optind], only)) {
            optarg = argv[optind++];
            return (optopt = 1);
        }
        break;
    case REQUIRE_ORDER:
        if (!is_option(argv[optind], only))
            return EOF;
        break;
    }
}
/* we've got an option, so parse it */

/* first, is it a long option? */
if (longopts != NULL
    && (memcmp(argv[optind], "--", 2) == 0
        || (only && argv[optind][0] == '+'))
    && optwhere == 1) {
    /* handle long options */
    if (memcmp(argv[optind], "--", 2) == 0)
        optwhere = 2;
    longopt_match = -1;
    possible_arg = strchr(argv[optind] + optwhere, '=');
    if (possible_arg == NULL) {
        /* no =, so next argv might be arg */
        match_chars = strlen(argv[optind]);
        possible_arg = argv[optind] + match_chars;
        match_chars = match_chars - optwhere;
    }
    else
        match_chars = (possible_arg - argv[optind]) - optwhere;
    for (optindex = 0; longopts[optindex].name != NULL; optindex++)
{
        if (memcmp(argv[optind] + optwhere,
                    longopts[optindex].name,
                    match_chars) == 0) {
            /* do we have an exact match? */
            if (match_chars == (int)
                (strlen(longopts[optindex].name))) {
                longopt_match = optindex;
                break;
            }
            /* do any characters match? */
            else {
                if (longopt_match < 0)
                    longopt_match = optindex;
                else {

```

```

        /* we have ambiguous options */
        if (opterr)
            fprintf(stderr, "%s: option `%s' is
ambiguous "
                                "(could be `--%s' or `--%s')\n",
                                argv[0],
                                argv[optind],
                                longopts[longopt_match].name,
                                longopts[optindex].name);
        return (optopt = '?');
    }
}

if (longopt_match >= 0)
    has_arg = longopts[longopt_match].has_arg;
}

/* if we didn't find a long option, is it a short option? */
if (longopt_match < 0 && shortopts != NULL) {
    cp = strchr(shortopts, argv[optind][optwhere]);
    if (cp == NULL) {
        /* couldn't find option in shortopts */
        if (opterr)
            fprintf(stderr,
                    "%s: invalid option -- `-%c'\n",
                    argv[0],
                    argv[optind][optwhere]);

        optwhere++;
        if (argv[optind][optwhere] == '\0') {
            optind++;
            optwhere = 1;
        }
        return (optopt = '?');
    }
    has_arg = ((cp[1] == ':')
               ? ((cp[2] == ':') ? OPTIONAL_ARG : REQUIRED_ARG)
               : NO_ARG);
    possible_arg = argv[optind] + optwhere + 1;
    optopt = *cp;
}

/* get argument and reset optwhere */
arg_next = 0;
switch (has_arg) {
case OPTIONAL_ARG:
    if (*possible_arg == '=')
        possible_arg++;
    if (*possible_arg != '\0') {

```

```

        optarg = possible_arg;
        optwhere = 1;
    }
    else
        optarg = NULL;
    break;
case REQUIRED_ARG:
    if (*possible_arg == '=')
        possible_arg++;
    if (*possible_arg != '\0') {
        optarg = possible_arg;
        optwhere = 1;
    }
    else if (optind + 1 >= argc) {
        if (opterr) {
            fprintf(stderr, "%s: argument required for option `%",
                    argv[0]);
            if (longopt_match >= 0)
                fprintf(stderr, "--%s'\n",
longopts[longopt_match].name);
            else
                fprintf(stderr, "-%c'\n", *cp);
        }
        optind++;
        return (optopt = ':');
    }
    else {
        optarg = argv[optind + 1];
        arg_next = 1;
        optwhere = 1;
    }
    break;
case NO_ARG:
    if (longopt_match < 0) {
        optwhere++;
        if (argv[optind][optwhere] == '\0')
            optwhere = 1;
    }
    else
        optwhere = 1;
    optarg = NULL;
    break;
}

/* do we have to permute or otherwise modify optind? */
if (ordering == PERMUTE && optwhere == 1 && num_nonopts != 0) {
    permute(argv + permute_from, num_nonopts, 1 + arg_next);
}

```

```

        optind = permute_from + 1 + arg_next;
    }
    else if (optwhere == 1)
        optind = optind + 1 + arg_next;

    /* finally return */
    if (longopt_match >= 0) {
        if (longind != NULL)
            *longind = longopt_match;
        if (longopts[longopt_match].flag != NULL) {
            *(longopts[longopt_match].flag) =
longopts[longopt_match].val;
            return 0;
        }
        else
            return longopts[longopt_match].val;
    }
    else
        return optopt;
}

int getopt_long(int argc, char **argv, char *shortopts,
                GETOPT_LONG_OPTION_T * longopts, int *longind)
{
    return getopt_internal(argc, argv, shortopts, longopts, longind, 0);
}

void help(void)
{
    puts( "OPTIONS" );
    puts( "" );
    puts( "-i, --initial    When shrinking, make"
          " initial spaces/tabs on a line tabs" );
    puts( "                and expand every other"
          " tab on the line into spaces." );
    puts( "-t=tablist,      "
          "Specify list of tab stops.  "
          "Default is every 8 characters." );
    puts( "--tabs=tablist, "
          "The parameter tablist is a list"
          " of tab stops separated by" );
    puts( "-tablist        "
          "commas; if no commas are present,"
          " the program will put a" );
    puts( "                "
          "tab stop every x places, "
          "with x being the number in the" );
}

```



```

    puts( "                parameter." );
    puts( "" );
    puts( "--help          Print usage message"
          " and exit successfully." );
    puts( "" );
    puts( "--version       Print version "
          "information and exit successfully." );
}

void version(void)
{
    puts( "detab - expand tabs into spaces" );
    puts( "Version 1.0" );
    puts( "Written by Gregory Pietsch" );
}

/* allocate memory, die on error */
void *xmalloc(size_t n)
{
    void *p = malloc(n);

    if (p == NULL) {
        fprintf(stderr, "%s: out of memory\n", program_name);
        exit(EXIT_FAILURE);
    }
    return p;
}

/* reallocate memory, die on error */
void *xrealloc(void *p, size_t n)
{
    void *s;

    if (n == 0) {
        if (p != NULL)
            free(p);
        return NULL;
    }
    if (p == NULL)
        return xmalloc(n);
    s = realloc(p, n);
    if (s == NULL) {
        fprintf(stderr, "%s: out of memory\n", program_name);
        exit(EXIT_FAILURE);
    }
    return s;
}

```

```

/* Determine the location of the first character in the string s1
 * that is not a character in s2. The terminating null is not
 * considered part of the string.
 */
char *xstrcpbrk(char *s1, char *s2)
{
    char *sc1;
    char *sc2;

    for (sc1 = s1; *sc1 != '\0'; sc1++)
        for (sc2 = s2;; sc2++)
            if (*sc2 == '\0')
                return sc1;
            else if (*sc1 == *sc2)
                break;

    return NULL; /* terminating nulls match */
}

/* compare function for qsort() */
int ul_cmp(const void *a, const void *b)
{
    unsigned long *ula = (unsigned long *) a;
    unsigned long *ulb = (unsigned long *) b;

    return (*ula < *ulb) ? -1 : (*ula > *ulb);
}

/* handle a tab stop list -- assumes param isn't NULL */
void handle_tab_stops(char *s)
{
    char *p;
    unsigned long ul;
    size_t len = strlen(s);

    if (xstrcpbrk(s, "0123456789,") != NULL) {
        /* funny param */
        fprintf(stderr, "%s: invalid parameter\n", program_name);
        exit(EXIT_FAILURE);
    }
    if (strchr(s, ',') == NULL) {
        tab_every = strtoul(s, NULL, 10);
        if (tab_every == 0)
            tab_every = 8;
    }
    else {
        tab_stop_list = xrealloc(tab_stop_list,

```

```

        (num_tab_stops_allocated += len) * (sizeof(unsigned
long)))));
    for (p = s; (p = strtok(p, ",")) != NULL; p = NULL) {
        ul = strtoul(p, NULL, 10);
        tab_stop_list[num_tab_stops++] = ul;
    }
    qsort(tab_stop_list, num_tab_stops, sizeof(unsigned long),
        ul_cmp);
}

void parse_args(int argc, char **argv)
{
    int opt;

    do {
        switch ((opt = getopt_long(argc, argv, shortopts, longopts,
NULL))) {
            case 'i':                /* initial */
                flag_initial = 1;
                break;
            case 't':                /* tab stops */
                handle_tab_stops(optarg);
                break;
            case '?':                /* invalid option */
                fprintf(stderr, "For help, type: \n\t%s --help\n",
program_name);
                exit(EXIT_FAILURE);
            case 1:
            case 0:
                if (show_help || show_version) {
                    if (show_help)
                        help();
                    if (show_version)
                        version();
                    exit(EXIT_SUCCESS);
                }
                break;
            default:
                break;
        }
    } while (opt != EOF);
}

/* output exactly n spaces */
void output_spaces(size_t n)
{

```

```

    int x = n;                                /* assume n is small */

    printf("%*s", x, "");
}

/* get next highest tab stop */
unsigned long get_next_tab(unsigned long x)
{
    size_t i;

    if (tab_stop_list == NULL) {
        /* use tab_every */
        x += (tab_every - (x % tab_every));
        return x;
    }
    else {
        for (i = 0; i < num_tab_stops && tab_stop_list[i] <= x; i++);
        return (i >= num_tab_stops) ? 0 : tab_stop_list[i];
    }
}

/* the function that does the dirty work */
void tab(FILE * f)
{
    unsigned long linelength = 0;
    int c;
    int in_initials = 1;
    size_t num_spaces = 0;
    unsigned long next_tab;

    while ((c = getc(f)) != EOF) {
        if (c != ' ' && c != '\t' && num_spaces > 0) {
            /* output spaces and possible tabs */
            if (flag_expand
                || (flag_initial && !in_initials)
                || num_spaces == 1) {
                /* output spaces anyway */
                output_spaces(num_spaces);
                linelength += num_spaces;
                num_spaces = 0;
            }
            else
                while (num_spaces != 0) {
                    next_tab = get_next_tab(linelength);
                    if (next_tab > 0 && next_tab <= linelength +
num_spaces) {
                        /* output a tab */

```

```

        putc('\t', stdout);
        num_spaces -= (next_tab - linelength);
        linelength = next_tab;
    }
    else {
        /* output spaces */
        output_spaces(num_spaces);
        linelength += num_spaces;
        num_spaces = 0;
    }
}

switch (c) {
case ' ':          /* space */
    num_spaces++;
    break;
case '\b':         /* backspace */
    /* preserve backspaces in output; decrement length for
    tabbing
        * purposes
        */
    putc(c, stdout);
    if (linelength > 0)
        linelength--;
    break;
case '\n':         /* newline */
    putc(c, stdout);
    in_initials = 1;
    linelength = 0;
    break;
case '\t':         /* tab */
    next_tab = get_next_tab(linelength + num_spaces);
    if (next_tab == 0) {
        while ((next_tab = get_next_tab(linelength)) != 0) {
            /* output tabs */
            putc('\t', stdout);
            num_spaces -= (next_tab - linelength);
            linelength = next_tab;
        }
        /* output spaces */
        output_spaces(num_spaces);
        num_spaces = 0;
        putc('\t', stdout);
        linelength += num_spaces + 1;
    }
    else
        num_spaces = next_tab - linelength;
}

```

```

        break;
    default:
        putc(c, stdout);
        in_initials = 0;
        linelength++;
        break;
    }
}

}

int main(int argc, char **argv)
{
    int i;
    FILE *fp;
    char *allocated_argvs = xmalloc(argc + 1);
    char **new_argv = xmalloc((argc + 1) * sizeof(char *));
    char *p;

    program_name = argv[0];
    memset(allocated_argvs, 0, argc + 1);
    for (i = 0; i < argc; i++) {
        p = argv[i];
        if (isdigit(p[1])) {
            new_argv[i] = xmalloc(strlen(p) + 2);
            sprintf(new_argv[i], "-t%s", p + 1);
            allocated_argvs[i] = 1;
        }
        else
            new_argv[i] = p;
    }
    new_argv[argc] = NULL;
    parse_args(argc, new_argv);
    if (optind == argc)
        tab(stdin);
    else {
        for (i = optind; i < argc; i++) {
            if (strcmp(argv[i], "-") == 0)
                fp = stdin;
            else {
                fp = fopen(argv[i], "r");
                if (fp == NULL) {
                    fprintf(stderr, "%s: can't open %s\n",
                        argv[0], argv[i]);
                    abort();
                }
            }
        }
        tab(fp);
    }
}

```

```

        if (fp != stdin)
            fclose(fp);
    }
}
/* free everything we can */
for (i = 0; i < argc; i++)
    if (allocated_argvs[i])
        free(new_argv[i]);
free(allocated_argvs);
if (tab_stop_list != NULL)
    free(tab_stop_list);
return EXIT_SUCCESS;
}

/* END OF FILE detab.c */

```

Here's entab...

```

/*****

entab.c - Source code for the detab command

AUTHOR: Gregory Pietsch

DESCRIPTION:

entab - shrinks spaces into tabs

*****/

/* include files */
#include <stdio.h>
#include <string.h>

/* macros */
#define NO_ARG          0
#define REQUIRED_ARG     1
#define OPTIONAL_ARG    2

/* types */

/* GETOPT_LONG_OPTION_T: The type of long option */
typedef struct GETOPT_LONG_OPTION_T {
    char *name;           /* the name of the long option */
    int has_arg;          /* one of the above macros */
    int *flag;            /* determines if getopt_long() returns a

```

```

        * value for a long option; if it is
        * non-NULL, 0 is returned as a function
        * value and the value of val is stored in
        * the area pointed to by flag.  Otherwise,
        * val is returned. */
    int val;                /* determines the value to return if flag is
                             * NULL. */
} GETOPT_LONG_OPTION_T;

typedef enum GETOPT_ORDERING_T {
    PERMUTE,
    RETURN_IN_ORDER,
    REQUIRE_ORDER
} GETOPT_ORDERING_T;

/* globally-defined variables */
char *optarg = NULL;
int optind = 0;
int opterr = 1;
int optopt = '?';

/* statically-defined variables */

static char *program_name;
/* if nonzero, it means tab every x characters */
static unsigned long tab_every = 8;
/* -i: only handle initial tabs/spaces */
static int flag_initial = 0;
/* don't expand tabs into spaces */
static int flag_expand = 0;
static unsigned long *tab_stop_list = NULL;
static size_t num_tab_stops = 0;
static size_t num_tab_stops_allocated = 0;
static int show_help = 0;
static int show_version = 0;
static char *shortopts = "it:";
static GETOPT_LONG_OPTION_T longopts[] =
{
    {"initial", NO_ARG, NULL, 'i'},
    {"tabs", REQUIRED_ARG, NULL, 't'},

    {"help", NO_ARG, &show_help, 1},
    {"version", NO_ARG, &show_version, 1},
    {NULL, 0, 0, 0}
};

/* functions */

```



```

/* reverse_argv_elements:  reverses num elements starting at argv */
static void reverse_argv_elements(char **argv, int num)
{
    int i;
    char *tmp;

    for (i = 0; i < (num >> 1); i++) {
        tmp = argv[i];
        argv[i] = argv[num - i - 1];
        argv[num - i - 1] = tmp;
    }
}

/* permute: swap two blocks of argv-elements given their lengths */
static void permute(char **argv, int len1, int len2)
{
    reverse_argv_elements(argv, len1);
    reverse_argv_elements(argv, len1 + len2);
    reverse_argv_elements(argv, len2);
}

/* is_option: is this argv-element an option or the end of the option
list? */
static int is_option(char *argv_element, int only)
{
    return ((argv_element == NULL)
           || (argv_element[0] == '-')
           || (only && argv_element[0] == '+'));
}

/* getopt_internal:  the function that does all the dirty work */
static int getopt_internal(int argc, char **argv, char *shortopts,
                           GETOPT_LONG_OPTION_T * longopts, int *longind, int
only)
{
    GETOPT_ORDERING_T ordering = PERMUTE;
    static size_t optwhere = 0;
    size_t permute_from = 0;
    int num_nonopts = 0;
    int optindex = 0;
    size_t match_chars = 0;
    char *possible_arg = NULL;
    int longopt_match = -1;
    int has_arg = -1;
    char *cp;
    int arg_next = 0;

```

```

/* first, deal with silly parameters and easy stuff */
if (argc == 0 || argv == NULL || (shortopts == NULL && longopts ==
NULL))
    return (optopt = '?');
if (optind >= argc || argv[optind] == NULL)
    return EOF;
if (strcmp(argv[optind], "--") == 0) {
    optind++;
    return EOF;
}
/* if this is our first time through */
if (optind == 0)
    optind = optwhere = 1;

/* define ordering */
if (shortopts != NULL && (*shortopts == '-' || *shortopts == '+')) {
    ordering = (*shortopts == '-') ? RETURN_IN_ORDER :
REQUIRE_ORDER;
    shortopts++;
}
else
    ordering = (getenv("POSIXLY_CORRECT") != NULL) ? REQUIRE_ORDER :
    PERMUTE;

/*
 * based on ordering, find our next option, if we're at the
beginning of
 * one
 */
if (optwhere == 1) {
    switch (ordering) {
    case PERMUTE:
        permute_from = optind;
        num_nonopts = 0;
        while (!is_option(argv[optind], only)) {
            optind++;
            num_nonopts++;
        }
        if (argv[optind] == NULL) {
            /* no more options */
            optind = permute_from;
            return EOF;
        } else if (strcmp(argv[optind], "--") == 0) {
            /* no more options, but have to get '--' out of the way
*/

            permute(argv + permute_from, num_nonopts, 1);

```

```

        optind = permute_from + 1;
        return EOF;
    }
    break;
case RETURN_IN_ORDER:
    if (!is_option(argv[optind], only)) {
        optarg = argv[optind++];
        return (optopt = 1);
    }
    break;
case REQUIRE_ORDER:
    if (!is_option(argv[optind], only))
        return EOF;
    break;
}
}
/* we've got an option, so parse it */

/* first, is it a long option? */
if (longopts != NULL
    && (memcmp(argv[optind], "--", 2) == 0
        || (only && argv[optind][0] == '+'))
    && optwhere == 1) {
    /* handle long options */
    if (memcmp(argv[optind], "--", 2) == 0)
        optwhere = 2;
    longopt_match = -1;
    possible_arg = strchr(argv[optind] + optwhere, '=');
    if (possible_arg == NULL) {
        /* no =, so next argv might be arg */
        match_chars = strlen(argv[optind]);
        possible_arg = argv[optind] + match_chars;
        match_chars = match_chars - optwhere;
    }
    else
        match_chars = (possible_arg - argv[optind]) - optwhere;
    for (optindex = 0; longopts[optindex].name != NULL; optindex++)
    {
        if (memcmp(argv[optind] + optwhere,
                    longopts[optindex].name,
                    match_chars) == 0) {
            /* do we have an exact match? */
            if (match_chars == (int)
                (strlen(longopts[optindex].name))) {
                longopt_match = optindex;
                break;
            }

```

ambiguous "

```

        /* do any characters match? */
        else {
            if (longopt_match < 0)
                longopt_match = optindex;
            else {
                /* we have ambiguous options */
                if (opterr)
                    fprintf(stderr, "%s: option `%s' is
ambiguous "
                                "(could be `--%s' or `--%s')\n",
                                argv[0],
                                argv[optind],
                                longopts[longopt_match].name,
                                longopts[optindex].name);
                return (optopt = '?');
            }
        }
    }
}

if (longopt_match >= 0)
    has_arg = longopts[longopt_match].has_arg;
}

/* if we didn't find a long option, is it a short option? */
if (longopt_match < 0 && shortopts != NULL) {
    cp = strchr(shortopts, argv[optind][optwhere]);
    if (cp == NULL) {
        /* couldn't find option in shortopts */
        if (opterr)
            fprintf(stderr,
                    "%s: invalid option -- `-%c'\n",
                    argv[0],
                    argv[optind][optwhere]);
        optwhere++;
        if (argv[optind][optwhere] == '\0') {
            optind++;
            optwhere = 1;
        }
        return (optopt = '?');
    }
    has_arg = ((cp[1] == ':')
               ? ((cp[2] == ':') ? OPTIONAL_ARG : REQUIRED_ARG)
               : NO_ARG);
    possible_arg = argv[optind] + optwhere + 1;
    optopt = *cp;
}

/* get argument and reset optwhere */
arg_next = 0;

```

```

switch (has_arg) {
case OPTIONAL_ARG:
    if (*possible_arg == '=')
        possible_arg++;
    if (*possible_arg != '\0') {
        optarg = possible_arg;
        optwhere = 1;
    }
    else
        optarg = NULL;
    break;
case REQUIRED_ARG:
    if (*possible_arg == '=')
        possible_arg++;
    if (*possible_arg != '\0') {
        optarg = possible_arg;
        optwhere = 1;
    }
    else if (optind + 1 >= argc) {
        if (opterr) {
            fprintf(stderr, "%s: argument required for option `%",
                    argv[0]);
            if (longopt_match >= 0)
                fprintf(stderr, "--%s'\n",
longopts[longopt_match].name);
            else
                fprintf(stderr, "-%c'\n", *cp);
        }
        optind++;
        return (optopt = ':');
    }
    else {
        optarg = argv[optind + 1];
        arg_next = 1;
        optwhere = 1;
    }
    break;
case NO_ARG:
    if (longopt_match < 0) {
        optwhere++;
        if (argv[optind][optwhere] == '\0')
            optwhere = 1;
    }
    else
        optwhere = 1;
    optarg = NULL;
    break;

```

```

    }

    /* do we have to permute or otherwise modify optind? */
    if (ordering == PERMUTE && optwhere == 1 && num_nonopts != 0) {
        permute(argv + permute_from, num_nonopts, 1 + arg_next);
        optind = permute_from + 1 + arg_next;
    }
    else if (optwhere == 1)
        optind = optind + 1 + arg_next;

    /* finally return */
    if (longopt_match >= 0) {
        if (longind != NULL)
            *longind = longopt_match;
        if (longopts[longopt_match].flag != NULL) {
            *(longopts[longopt_match].flag) =
longopts[longopt_match].val;
            return 0;
        }
        else
            return longopts[longopt_match].val;
    }
    else
        return optopt;
}

int getopt_long(int argc, char **argv, char *shortopts,
                GETOPT_LONG_OPTION_T * longopts, int *longind)
{
    return getopt_internal(argc, argv, shortopts, longopts, longind, 0);
}

void help(void)
{
    puts( "OPTIONS" );
    puts( "" );
    puts( "-i, --initial    When shrinking, make"
          " initial spaces/tabs on a line tabs" );
    puts( "                and expand every other"
          " tab on the line into spaces." );
    puts( "-t=tablist,      "
          "Specify list of tab stops.  "
          "Default is every 8 characters." );
    puts( "--tabs=tablist, "
          "The parameter tablist is a list"
          " of tab stops separated by" );
    puts( "-tablist        "

```

```

        "commas; if no commas are present,"
        " the program will put a" );
puts( "
        "
        "tab stop every x places, "
        "with x being the number in the" );
puts( "
        parameter." );
puts( "" );
puts( "--help          Print usage message"
        " and exit successfully." );
puts( "" );
puts( "--version       Print version "
        "information and exit successfully." );
}

void version(void)
{
    puts( "detab - expand tabs into spaces" );
    puts( "Version 1.0" );
    puts( "Written by Gregory Pietsch" );
}

/* allocate memory, die on error */
void *xmalloc(size_t n)
{
    void *p = malloc(n);

    if (p == NULL) {
        fprintf(stderr, "%s: out of memory\n", program_name);
        exit(EXIT_FAILURE);
    }
    return p;
}

/* reallocate memory, die on error */
void *xrealloc(void *p, size_t n)
{
    void *s;

    if (n == 0) {
        if (p != NULL)
            free(p);
        return NULL;
    }
    if (p == NULL)
        return xmalloc(n);
    s = realloc(p, n);
    if (s == NULL) {

```

```

        fprintf(stderr, "%s: out of memory\n", program_name);
        exit(EXIT_FAILURE);
    }
    return s;
}

/* Determine the location of the first character in the string s1
 * that is not a character in s2.  The terminating null is not
 * considered part of the string.
 */
char *xstrcpbrk(char *s1, char *s2)
{
    char *sc1;
    char *sc2;

    for (sc1 = s1; *sc1 != '\0'; sc1++)
        for (sc2 = s2;; sc2++)
            if (*sc2 == '\0')
                return sc1;
            else if (*sc1 == *sc2)
                break;
    return NULL;          /* terminating nulls match */
}

/* compare function for qsort() */
int ul_cmp(const void *a, const void *b)
{
    unsigned long *ula = (unsigned long *) a;
    unsigned long *ulb = (unsigned long *) b;

    return (*ula < *ulb) ? -1 : (*ula > *ulb);
}

/* handle a tab stop list -- assumes param isn't NULL */
void handle_tab_stops(char *s)
{
    char *p;
    unsigned long ul;
    size_t len = strlen(s);

    if (xstrcpbrk(s, "0123456789,") != NULL) {
        /* funny param */
        fprintf(stderr, "%s: invalid parameter\n", program_name);
        exit(EXIT_FAILURE);
    }
    if (strchr(s, ',') == NULL) {
        tab_every = strtoul(s, NULL, 10);
    }
}

```



```

        if (tab_every == 0)
            tab_every = 8;
    }
    else {
        tab_stop_list = xrealloc(tab_stop_list,
                                (num_tab_stops_allocated += len) * (sizeof(unsigned
long)));
        for (p = s; (p = strtok(p, ",")) != NULL; p = NULL) {
            ul = strtoul(p, NULL, 10);
            tab_stop_list[num_tab_stops++] = ul;
        }
        qsort(tab_stop_list, num_tab_stops, sizeof(unsigned long),
              ul_cmp);
    }
}

void parse_args(int argc, char **argv)
{
    int opt;

    do {
        switch ((opt = getopt_long(argc, argv, shortopts, longopts,
NULL))) {
            case 'i':                /* initial */
                flag_initial = 1;
                break;
            case 't':                /* tab stops */
                handle_tab_stops(optarg);
                break;
            case '?':                /* invalid option */
                fprintf(stderr, "For help, type: \n\t%s --help\n",
program_name);
                exit(EXIT_FAILURE);
            case 1:
            case 0:
                if (show_help || show_version) {
                    if (show_help)
                        help();
                    if (show_version)
                        version();
                    exit(EXIT_SUCCESS);
                }
                break;
            default:
                break;
        }
    } while (opt != EOF);
}

```

```

}

/* output exactly n spaces */
void output_spaces(size_t n)
{
    int x = n;                /* assume n is small */

    printf("%*s", x, "");
}

/* get next highest tab stop */
unsigned long get_next_tab(unsigned long x)
{
    size_t i;

    if (tab_stop_list == NULL) {
        /* use tab_every */
        x += (tab_every - (x % tab_every));
        return x;
    }
    else {
        for (i = 0; i < num_tab_stops && tab_stop_list[i] <= x; i++);
        return (i >= num_tab_stops) ? 0 : tab_stop_list[i];
    }
}

/* the function that does the dirty work */
void tab(FILE * f)
{
    unsigned long linelength = 0;
    int c;
    int in_initials = 1;
    size_t num_spaces = 0;
    unsigned long next_tab;

    while ((c = getc(f)) != EOF) {
        if (c != ' ' && c != '\t' && num_spaces > 0) {
            /* output spaces and possible tabs */
            if (flag_expand
                || (flag_initial && !in_initials)
                || num_spaces == 1) {
                /* output spaces anyway */
                output_spaces(num_spaces);
                linelength += num_spaces;
                num_spaces = 0;
            }
            else

```

```

        while (num_spaces != 0) {
            next_tab = get_next_tab(linlength);
            if (next_tab > 0 && next_tab <= linlength +
num_spaces) {

                /* output a tab */
                putc('\t', stdout);
                num_spaces -= (next_tab - linlength);
                linlength = next_tab;
            }
            else {
                /* output spaces */
                output_spaces(num_spaces);
                linlength += num_spaces;
                num_spaces = 0;
            }
        }
    }
    switch (c) {
    case ' ':
        /* space */
        num_spaces++;
        break;
    case '\b':
        /* backspace */
        /* preserve backspaces in output; decrement length for
tabbing
        * purposes
        */
        putc(c, stdout);
        if (linlength > 0)
            linlength--;
        break;
    case '\n':
        /* newline */
        putc(c, stdout);
        in_initials = 1;
        linlength = 0;
        break;
    case '\t':
        /* tab */
        next_tab = get_next_tab(linlength + num_spaces);
        if (next_tab == 0) {
            while ((next_tab = get_next_tab(linlength)) != 0) {
                /* output tabs */
                putc('\t', stdout);
                num_spaces -= (next_tab - linlength);
                linlength = next_tab;
            }
            /* output spaces */
            output_spaces(num_spaces);
            num_spaces = 0;
        }
    }
}

```

```

        putc('\t', stdout);
        linelength += num_spaces + 1;
    }
    else
        num_spaces = next_tab - linelength;
    break;
default:
    putc(c, stdout);
    in_initials = 0;
    linelength++;
    break;
}
}

```

```

int main(int argc, char **argv)
{
    int i;
    FILE *fp;
    char *allocated_argvs = xmalloc(argc + 1);
    char **new_argv = xmalloc((argc + 1) * sizeof(char *));
    char *p;

    program_name = argv[0];
    memset(allocated_argvs, 0, argc + 1);
    for (i = 0; i < argc; i++) {
        p = argv[i];
        if (isdigit(p[1])) {
            new_argv[i] = xmalloc(strlen(p) + 2);
            sprintf(new_argv[i], "-t%s", p + 1);
            allocated_argvs[i] = 1;
        }
        else
            new_argv[i] = p;
    }
    new_argv[argc] = NULL;
    parse_args(argc, new_argv);
    if (optind == argc)
        tab(stdin);
    else {
        for (i = optind; i < argc; i++) {
            if (strcmp(argv[i], "-") == 0)
                fp = stdin;
            else {
                fp = fopen(argv[i], "r");
                if (fp == NULL) {
                    fprintf(stderr, "%s: can't open %s\n",

```

```
                argv[0], argv[i]);
            abort();
        }
    }
    tab(fp);
    if (fp != stdin)
        fclose(fp);
}

/* free everything we can */
for (i = 0; i < argc; i++)
    if (allocated_argvs[i])
        free(new_argv[i]);
free(allocated_argvs);
if (tab_stop_list != NULL)
    free(tab_stop_list);
return EXIT_SUCCESS;
}

/* END OF FILE entab.c */
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 5-13, page 118

Solutions by Gregory Pietsch and Steven Huang

Write the program `tail`, which prints the last `n` lines of its input. By default, `n` is 10, say, but it can be changed by an optional argument, so that

```
tail -n
```

prints the last `n` lines. The program should behave rationally no matter how unreasonable the input or the value of `n`. Write the program so it makes the best use of available storage; lines should be stored as in the sorting program of Section 5.6, not in a two-dimensional array of fixed size.

Gregory Pietsch's solution

```
/******
```

```
tail.c - Source code for the tail command
```

```
AUTHOR: Gregory Pietsch <gkpl@flash.net>
```

```
DESCRIPTION:
```

```
tail prints the last part of each file on the command line (10 lines by
default); it reads from standard input if no files are given or when a
filename of '-' is encountered. If more than one file is given, it
prints a header consisting of the file's name enclosed in '==>' and '<==' before
the output for each file.
```

```
There are two option formats for tail: the new one, in which numbers are
arguments to the option letters; and the old one, in which the number
precedes any option letters. In this version, the old format is barely
supported. Supporting it fully is left as an exercise to the reader ;-).
```

```
GNU's -f (or --follow) option is not supported. With that option, the
program loops forever on the assumption that the file being tailed is
growing. I couldn't figure out how to determine if the program is reading
from a pipe in ANSI C; this option is ignored if reading from a pipe.
```

```
*****/
```

```
/* include files */
#include <ctype.h>
#include <stdio.h>
```

```

#include <stdlib.h>
#include <string.h>

/* macros */

#define NO_ARG          0
#define REQUIRED_ARG    1
#define OPTIONAL_ARG    2

/* how many characters will fill one's tail (literally) */
#define TAIL_BUFFER_SIZE 16384

/* how much for a string buffer */
#define TAIL_STRING_BUFFER_SIZE 256

/* need MIN */
#ifndef MIN
#define MIN(x,y) ((x)<(y)?(x):(y))
#endif

/* types */

typedef enum VERBOSITY_T {
    NEVER,
    SOMETIMES,
    ALWAYS
} VERBOSITY_T;

typedef struct LINE_QUEUE_EL_T {
    char *s;
    struct LINE_QUEUE_EL_T *next;
} LINE_QUEUE_EL_T;

typedef struct LINE_QUEUE_T {
    struct LINE_QUEUE_EL_T *first;
    struct LINE_QUEUE_EL_T *last;
    unsigned long num_elements;
} LINE_QUEUE_T;

/* GETOPT_LONG_OPTION_T: The type of long option */
typedef struct GETOPT_LONG_OPTION_T {
    char *name;           /* the name of the long option */
    int has_arg;          /* one of the above macros */
    int *flag;            /* determines if getopt_long() returns a
                           * value for a long option; if it is
                           * non-NULL, 0 is returned as a function
                           * value and the value of val is stored in
                           * the area pointed to by flag. Otherwise,
                           * val is returned. */

```

```

    int val;                                /* determines the value to return if flag is
                                           * NULL. */
} GETOPT_LONG_OPTION_T;

typedef enum GETOPT_ORDERING_T {
    PERMUTE,
    RETURN_IN_ORDER,
    REQUIRE_ORDER
} GETOPT_ORDERING_T;

/* globally-defined variables */
char *optarg = NULL;
int optind = 0;
int opterr = 1;
int optopt = '?';

/* statically-defined variables */

static int show_help = 0;
static int show_version = 0;
static char *shortopts = "c:ln:qv";
static GETOPT_LONG_OPTION_T longopts[] =
{
    {"bytes", REQUIRED_ARG, NULL, 'c'},
    {"lines", REQUIRED_ARG, NULL, 'n'},
    {"quiet", NO_ARG, NULL, 'q'},
    {"silent", NO_ARG, NULL, 'q'},
    {"verbose", NO_ARG, NULL, 'v'},

    {"help", NO_ARG, &show_help, 1},
    {"version", NO_ARG, &show_version, 1},
    {NULL, 0, 0, 0}
};

static char *program_name;
static int flag_bytes = 0;
static VERBOSITY_T flag_verbosity = SOMETIMES;
static unsigned long number = 0;
static int flag_skip = 0;

/* functions */

/* reverse_argv_elements: reverses num elements starting at argv */
static void reverse_argv_elements(char **argv, int num)
{
    int i;
    char *tmp;

    for (i = 0; i < (num >> 1); i++) {
        tmp = argv[i];

```



```

        argv[i] = argv[num - i - 1];
        argv[num - i - 1] = tmp;
    }
}

/* permute: swap two blocks of argv-elements given their lengths */
static void permute(char **argv, int len1, int len2)
{
    reverse_argv_elements(argv, len1);
    reverse_argv_elements(argv, len1 + len2);
    reverse_argv_elements(argv, len2);
}

/* is_option: is this argv-element an option or the end of the option list? */
static int is_option(char *argv_element, int only)
{
    return ((argv_element == NULL)
           || (argv_element[0] == '-')
           || (only && argv_element[0] == '+'));
}

/* getopt_internal: the function that does all the dirty work */
static int getopt_internal(int argc, char **argv, char *shortopts,
                          GETOPT_LONG_OPTION_T * longopts, int *longind, int only)
{
    GETOPT_ORDERING_T ordering = PERMUTE;
    static size_t optwhere = 0;
    size_t permute_from = 0;
    int num_nonopts = 0;
    int optindex = 0;
    size_t match_chars = 0;
    char *possible_arg = NULL;
    int longopt_match = -1;
    int has_arg = -1;
    char *cp;
    int arg_next = 0;

    /* first, deal with silly parameters and easy stuff */
    if (argc == 0 || argv == NULL || (shortopts == NULL && longopts == NULL))
        return (optopt = '?');
    if (optind >= argc || argv[optind] == NULL)
        return EOF;
    if (strcmp(argv[optind], "--") == 0) {
        optind++;
        return EOF;
    }
    /* if this is our first time through */
    if (optind == 0)
        optind = optwhere = 1;

```

```

/* define ordering */
if (shortopts != NULL && (*shortopts == '-' || *shortopts == '+')) {
    ordering = (*shortopts == '-') ? RETURN_IN_ORDER : REQUIRE_ORDER;
    shortopts++;
}
else
    ordering = (getenv("POSIXLY_CORRECT") != NULL) ? REQUIRE_ORDER :
        PERMUTE;

/*
 * based on ordering, find our next option, if we're at the beginning of
 * one
 */
if (optwhere == 1) {
    switch (ordering) {
    case PERMUTE:
        permute_from = optind;
        num_nonopts = 0;
        while (!is_option(argv[optind], only)) {
            optind++;
            num_nonopts++;
        }
        if (argv[optind] == NULL) {
            /* no more options */
            optind = permute_from;
            return EOF;
        } else if (strcmp(argv[optind], "--") == 0) {
            /* no more options, but have to get '--' out of the way */
            permute(argv + permute_from, num_nonopts, 1);
            optind = permute_from + 1;
            return EOF;
        }
        break;
    case RETURN_IN_ORDER:
        if (!is_option(argv[optind], only)) {
            optarg = argv[optind++];
            return (optopt = 1);
        }
        break;
    case REQUIRE_ORDER:
        if (!is_option(argv[optind], only))
            return EOF;
        break;
    }
}
/* we've got an option, so parse it */

/* first, is it a long option? */

```

```

if (longopts != NULL
    && (memcmp(argv[optind], "--", 2) == 0
        || (only && argv[optind][0] == '+'))
    && optwhere == 1) {
    /* handle long options */
    if (memcmp(argv[optind], "--", 2) == 0)
        optwhere = 2;
    longopt_match = -1;
    possible_arg = strchr(argv[optind] + optwhere, '=');
    if (possible_arg == NULL) {
        /* no =, so next argv might be arg */
        match_chars = strlen(argv[optind]);
        possible_arg = argv[optind] + match_chars;
        match_chars = match_chars - optwhere;
    }
    else
        match_chars = (possible_arg - argv[optind]) - optwhere;
    for (optindex = 0; longopts[optindex].name != NULL; optindex++) {
        if (memcmp(argv[optind] + optwhere,
                    longopts[optindex].name,
                    match_chars) == 0) {
            /* do we have an exact match? */
            if (match_chars == (int)(strlen(longopts[optindex].name))) {
                longopt_match = optindex;
                break;
            }
            /* do any characters match? */
            else {
                if (longopt_match < 0)
                    longopt_match = optindex;
                else {
                    /* we have ambiguous options */
                    if (opterr)
                        fprintf(stderr, "%s: option '%s' is ambiguous "
                                "(could be '--%s' or '--%s')\n",
                                argv[0],
                                argv[optind],
                                longopts[longopt_match].name,
                                longopts[optindex].name);
                    return (optopt = '?');
                }
            }
        }
    }
}

if (longopt_match >= 0)
    has_arg = longopts[longopt_match].has_arg;
}

/* if we didn't find a long option, is it a short option? */
if (longopt_match < 0 && shortopts != NULL) {

```

```

    cp = strchr(shortopts, argv[optind][optwhere]);
    if (cp == NULL) {
        /* couldn't find option in shortopts */
        if (opterr)
            fprintf(stderr,
                    "%s: invalid option -- `-%c'\n",
                    argv[0],
                    argv[optind][optwhere]);
        optwhere++;
        if (argv[optind][optwhere] == '\0') {
            optind++;
            optwhere = 1;
        }
        return (optopt = '?');
    }
    has_arg = ((cp[1] == ':')
               ? ((cp[2] == ':') ? OPTIONAL_ARG : REQUIRED_ARG)
               : NO_ARG);
    possible_arg = argv[optind] + optwhere + 1;
    optopt = *cp;
}
/* get argument and reset optwhere */
arg_next = 0;
switch (has_arg) {
case OPTIONAL_ARG:
    if (*possible_arg == '=')
        possible_arg++;
    if (*possible_arg != '\0') {
        optarg = possible_arg;
        optwhere = 1;
    }
    else
        optarg = NULL;
    break;
case REQUIRED_ARG:
    if (*possible_arg == '=')
        possible_arg++;
    if (*possible_arg != '\0') {
        optarg = possible_arg;
        optwhere = 1;
    }
}
else if (optind + 1 >= argc) {
    if (opterr) {
        fprintf(stderr, "%s: argument required for option `",
                argv[0]);
        if (longopt_match >= 0)
            fprintf(stderr, "--%s'\n", longopts[longopt_match].name);
        else
            fprintf(stderr, "-%c'\n", *cp);
    }
}

```

```

    }
    optind++;
    return (optopt = ':');
}
else {
    optarg = argv[optind + 1];
    arg_next = 1;
    optwhere = 1;
}
break;
case NO_ARG:
    if (longopt_match < 0) {
        optwhere++;
        if (argv[optind][optwhere] == '\0')
            optwhere = 1;
    }
    else
        optwhere = 1;
    optarg = NULL;
    break;
}

/* do we have to permute or otherwise modify optind? */
if (ordering == PERMUTE && optwhere == 1 && num_nonopts != 0) {
    permute(argv + permute_from, num_nonopts, 1 + arg_next);
    optind = permute_from + 1 + arg_next;
}
else if (optwhere == 1)
    optind = optind + 1 + arg_next;

/* finally return */
if (longopt_match >= 0) {
    if (longind != NULL)
        *longind = longopt_match;
    if (longopts[longopt_match].flag != NULL) {
        *(longopts[longopt_match].flag) = longopts[longopt_match].val;
        return 0;
    }
    else
        return longopts[longopt_match].val;
}
else
    return optopt;
}

```

```

int getopt_long(int argc, char **argv, char *shortopts,
                GETOPT_LONG_OPTION_T * longopts, int *longind)
{
    return getopt_internal(argc, argv, shortopts, longopts, longind, 0);
}

```

```

}

void help(void)
{
    puts( "OPTIONS" );
    puts( " " );
    puts( "-c N, --bytes N          Print last N bytes.  "
        "N is a nonzero integer," );
    puts( "                                optionally followed by one of "
        "the following" );
    puts( "                                characters:" );
    puts( "                                b          512-byte blocks." );
    puts( "                                k          1-kilobyte blocks." );
    puts( "                                m          1-megabyte blocks." );
    puts( " " );
    puts( "-N, -l N, -n N,          Print last N lines." );
    puts( "--lines N" );
    puts( " " );
    puts( "-q, --quiet,          Never print filename headers.  "
        "Normally, filename " );
    puts( "--silent        headers are printed if and only"
        " if more than one file" );
    puts( "                                is given on the command line." );
    puts( " " );
    puts( "-v, --verbose        Always print filename headers." );
    puts( " " );
    puts( "--help            Print usage message and exit successfully.");
    puts( " " );
    puts( "--version        Print version"
        " information and exit successfully." );
}

void version(void)
{
    puts( "tail - output the last part of files" );
    puts( "Version 1.0" );
    puts( "Written by Gregory Pietsch" );
}

/* allocate memory, die on error */
void *xmalloc(size_t n)
{
    void *p = malloc(n);

    if (p == NULL) {
        fprintf(stderr, "%s: out of memory\n", program_name);
        exit(EXIT_FAILURE);
    }
}

```

```

    return p;
}

/* reallocate memory, die on error */
void *xrealloc(void *p, size_t n)
{
    void *s;

    if (n == 0) {
        if (p != NULL)
            free(p);
        return NULL;
    }
    if (p == NULL)
        return xmalloc(n);
    s = realloc(p, n);
    if (s == NULL) {
        fprintf(stderr, "%s: out of memory\n", program_name);
        exit(EXIT_FAILURE);
    }
    return s;
}

/* get string duplicate */
char *xstrdup(char *s)
{
    char *p = xmalloc(strlen(s) + 1);

    strcpy(p, s);
    return p;
}

/* queue stuff - get fresh queue */
LINE_QUEUE_T *lq_create(void)
{
    LINE_QUEUE_T *lq = xmalloc(sizeof LINE_QUEUE_T);

    lq->first = NULL;
    lq->last = NULL;
    lq->num_elements = 0;
    return lq;
}

/* put an item onto the queue */
void lq_enq(LINE_QUEUE_T * lq, char *s)
{
    LINE_QUEUE_EL_T *lq_el = xmalloc(sizeof LINE_QUEUE_EL_T);

    lq_el->s = xstrdup(s);

```

```

    lq_el->next = NULL;
    if (lq->first == NULL && lq->last == NULL) {
        /* first element */
        lq->first = lq->last = lq_el;
        lq->num_elements = 1;
    }
    else {
        /* tack onto end */
        lq->last->next = lq_el;
        lq->last = lq_el;
        lq->num_elements++;
    }
}

/* take an item off the queue */
char *lq_deq(LINE_QUEUE_T * lq)
{
    char *s;
    LINE_QUEUE_EL_T *lq_el;

    if (lq->first == NULL)
        return NULL;
    lq_el = lq->first;
    s = lq_el->s;
    if (lq->first == lq->last)
        lq->first = lq->last = NULL;
    else
        lq->first = lq->first->next;
    free(lq_el);
    lq->num_elements--;
    return s;
}

/* output number lines -- this function is tough because I can only
 * use fseek() to rewind a text stream (See ISO C 7.9.9.2 if you don't
 * believe me).
 */
void tail_lines(FILE * f)
{
    char buffer[TAIL_BUFFER_SIZE];
    size_t num_read;
    int last_is_nl = 0;
    unsigned long num_skipped = 0;
    int c;
    LINE_QUEUE_T *lq = NULL;
    char *s;
    size_t s_size = 0;
    size_t s_allocated = 0;
    char *p;

```



```

if (flag_skip) {
    /* skip a bunch of lines, output everything else */
    while ((c = getc(f)) != EOF && num_skipped < number) {
        if (c == '\n')
            num_skipped++;
    }
    while ((num_read = fread(buffer, 1, TAIL_BUFFER_SIZE, f)) != 0)
{
        fwrite(buffer, 1, num_read, stdout);
        last_is_nl = (buffer[num_read - 1] == '\n');
    }
    if (!last_is_nl)
        fputc('\n', stdout);
}
else {
    lq = lq_create();
    s = xmalloc(TAIL_STRING_BUFFER_SIZE);
    s_allocated = TAIL_STRING_BUFFER_SIZE;
    while ((c = getc(f)) != EOF) {
        /* add to s, if not at eof or end of line */
        if (c != '\n') {
            if (s_size == s_allocated - 1) {
                s_allocated += TAIL_STRING_BUFFER_SIZE;
                s = xrealloc(s, s_allocated);
            }
            s[s_size++] = c;
        }
        else {
            /* enqueue s, possibly dequeuing if we don't need a
line */
            s[s_size] = '\0';
            lq_enq(lq, s);
            if (lq->num_elements > number)
                free(lq_deq(lq));
            s_size = 0;
        }
    }
    while (lq->num_elements != 0) {
        /* print out strings */
        p = lq_deq(lq);
        puts(p);
        free(p);
    }
    free(s);
    free(lq);
}
}

```

```

/* output number characters, or skip over number characters */
void tail_chars(FILE * f)
{
    char buffer[TAIL_BUFFER_SIZE];
    size_t num_read;
    int last_is_nl = 0;
    long lnum = number;

    if (flag_skip)
        fseek(f, lnum, SEEK_SET);
    else
        fseek(f, -lnum, SEEK_END);
    while ((num_read = fread(buffer, 1, TAIL_BUFFER_SIZE, f)) != 0) {
        fwrite(buffer, 1, num_read, stdout);
        last_is_nl = (buffer[num_read - 1] == '\n');
    }
    if (!last_is_nl)
        fputc('\n', stdout);
}

void parse_args(int argc, char **argv)
{
    int opt;
    char *p;
    int flag_found_number = 0;
    int verbosity_changed = 0;

    do {
        switch ((opt = getopt_long(argc, argv, shortopts, longopts, NULL))) {
            case 'c': /* print bytes */
                if (flag_found_number) {
                    fprintf(stderr, "%s: invalid arguments%s", program_name);
                    abort();
                }
                flag_bytes = 1;
                p = optarg;
                if (*p == '+') {
                    flag_skip = 1;
                    p++;
                }
                for (number = 0;
                     isdigit(*p);
                     number = number * 10 + (*p++ - '0'));
                switch (*p) {
                    case 'b': /* 512-byte blocks */
                        number *= 512;
                        break;
                    case 'k': /* kilobyte blocks */
                        number *= 1024;

```

```

        break;
    case 'm':                /* megabyte blocks */
        number *= 1048576;
        break;
    default:
        break;
}
flag_found_number = 1;
break;
case 'l':
case 'n':                  /* lines */
    if (flag_found_number) {
        fprintf(stderr, "%s: invalid arguments%s", program_name);
        abort();
    }
    flag_bytes = 0;
    p = optarg;
    if (*p == '+') {
        flag_skip = 1;
        p++;
    }
    number = strtoul(p, NULL, 10);
    flag_found_number = 1;
    break;
case 'q':                  /* quiet */
    if (verbosity_changed) {
        fprintf(stderr, "%s: invalid arguments%s", program_name);
        abort();
    }
    verbosity_changed = 1;
    flag_verbosity = NEVER;
    break;
case 'v':                  /* verbose */
    if (verbosity_changed) {
        fprintf(stderr, "%s: invalid arguments%s", program_name);
        abort();
    }
    verbosity_changed = 1;
    flag_verbosity = ALWAYS;
    break;
case '?':                  /* invalid option */
    fprintf(stderr, "For help, type:\n\t%s --help\n", program_name);
    exit(EXIT_FAILURE);
case 1:
case 0:
    if (show_help || show_version) {
        if (show_help)
            help();
        if (show_version)

```

```

        version();
        exit(EXIT_SUCCESS);
    }
    break;
default:
    break;
}
} while (opt != EOF);
if (flag_found_number == 0 || number == 0) {
    /* didn't find anything, so set default */
    flag_bytes = 0;
    number = 10;
}
}

int main(int argc, char **argv)
{
    int i;
    int j;
    unsigned long ul;
    char **new_argv = xmalloc((argc + 1) * (sizeof(char *)));
    char *allocated_argvs = xmalloc(argc + 1);
    char *p;
    char *s;
    char *t;
    FILE *f;
    int flag_plus = 0;

    memset(allocated_argvs, 0, argc + 1);
    new_argv[0] = program_name = argv[0];
    /* deal with silly old-format arguments */
    for (i = 1, j = 1; i < argc; i++) {
        p = argv[i];
        flag_plus = 0;
        /* handle options first */
        if (*p == '-' || *p == '+') {
            if (isdigit(p[1]) || p[1] == '+' || *p == '+') {
                /* rearrange p */
                s = xmalloc(strlen(p) + 3);
                t = s;
                *t++ = '-';
                if (*p == '-')
                    p++;
                ul = 0;
                if (*p == '+') {
                    flag_plus = 1;
                    p++;
                }
                while (isdigit(*p)) {

```

```

        ul = ul * 10 + (*p - '0');
        p++;
    }
    if (strchr(p, 'q') != NULL)
        *t++ = 'q';
    if (strchr(p, 'v') != NULL)
        *t++ = 'v';
    if (strpbrk(p, "cbkm") != NULL)
        *t++ = 'c';
    if (strchr(p, 'l') != NULL)
        *t++ = 'l';
    if (strchr(p, 'n') != NULL || t[-1] == '-')
        *t++ = 'n';
    if (flag_plus)
        *t++ = '+';
    sprintf(t, "%lu", ul);
    t += strlen(t);
    if (strchr(p, 'b') != NULL)
        *t++ = 'b';
    if (strchr(p, 'k') != NULL)
        *t++ = 'k';
    if (strchr(p, 'm') != NULL)
        *t++ = 'm';
    *t = '\0';
    new_argv[j] = s;
    allocated_argvs[j++] = 1;
}
else
    new_argv[j++] = argv[i];
}
}
for (i = 1; i < argc; i++) {
    /* handle file names */
    p = argv[i];
    if (*p != '-')
        new_argv[j++] = p;
}
new_argv[argc] = NULL;
parse_args(argc, new_argv);
if (optind == argc
    || (optind == argc - 1 && strcmp(argv[optind], "-") == 0)) {
    /* no more argv-elements, tail stdin */
    if (flag_verbosity == ALWAYS)
        puts("==> standard input <==");
    flag_bytes ? tail_chars(stdin) : tail_lines(stdin);
}
else if (optind == argc - 1) {
    /* one file */
    f = fopen(new_argv[optind], flag_bytes ? "rb" : "r");

```

```

    if (f == NULL) {
        fprintf(stderr, "%s: Can't open file %s\n",
                program_name, new_argv[optind]);
        abort();
    }
    if (flag_verbosity == ALWAYS)
        printf("==> %s <==\n", new_argv[optind]);
    flag_bytes ? tail_chars(f) : tail_lines(f);
    fclose(f);
}
else {
    /* multiple files */
    for (i = optind; i < argc; i++) {
        if (strcmp(new_argv[i], "-") == 0) {
            f = stdin;
            if (flag_verbosity != NEVER)
                puts("==> standard input <==");
        }
        else {
            f = fopen(new_argv[i], flag_bytes ? "rb" : "r");
            if (f == NULL) {
                fprintf(stderr, "%s: can't open %s\n",
                        argv[0], argv[i]);
                abort();
            }
            if (flag_verbosity != NEVER)
                printf("==> %s <==\n", new_argv[i]);
        }
        flag_bytes ? tail_chars(f) : tail_lines(f);
        if (f != stdin)
            fclose(f);
    }
}
/* free all we can */
for (i = 1; i <= argc; i++)
    if (allocated_argvs[i])
        free(new_argv[i]);
free(allocated_argvs);
return EXIT_SUCCESS;
}

/* END OF FILE tail.c */

```

```

/* K&R Exercise 5-13 */
/* Steven Huang */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define DEFAULT_NUM_LINES    10
#define MAX_LINE_LEN        1000

/*
   Points of interest for a novice:

1.  atoi() has a normally annoying property of not being able to
    tell the caller conclusively whether the input was bad ("abc")
    or it was really zero ("0"), because it returns 0 for both
    cases. Here, we exploit that property, because we only want
    to accept options in the form of "-n".

2.  Try to understand how this program deals with input that
    doesn't even have as many lines as the line_ptrs[] array.
    That is, how does this program degenerate into just displaying
    everything it read? (Hint: what does it mean when line_ptrs[x]
    is NULL?)

3.  Using modulo arithmetic on an index to a circular array is
    a common and useful technique. Try to understand the range
    of values that current_line (and j, later) will take. In
    particular, why shouldn't we just do this:

    for (i = 0; i < num_lines; i++)
        if (line_ptrs[i])
            printf("%s", line_ptrs[i]);

4.  Why do we still use a "%s" to display what's inside line_ptrs,
    rather than just:

    printf(line_ptrs[i]);

5.  There is a bug in this program, where you see:

    numlines = -numlines;

    When will this break?
*/

/* K&R2 p29 */
int getline(char s[], int lim)
{

```

```

int c, i;

for (i = 0; i < lim - 1 && (c = getchar()) != EOF && c != '\n'; i++)
    s[i] = c;
if (c == '\n')
    s[i++] = c;
s[i] = '\0';
return i;
}

/* duplicates a string */
char *dupstr(const char *s)
{
    char *p = malloc(strlen(s) + 1);

    if (p)
        strcpy(p, s);
    return p;
}

int main(int argc, char *argv[])
{
    int num_lines = DEFAULT_NUM_LINES;
    char **line_ptrs;
    char buffer[MAX_LINE_LEN];
    int i;
    unsigned j, current_line;

    if (argc > 1) {
        /*
         * We use a little trick here. The command line parameter should be
         * in the form of "-n", where n is the number of lines. We don't
         * check for the "-", but just pass it to atoi() anyway, and then
         * check if atoi() returned us a negative number.
         */
        num_lines = atoi(argv[1]);
        if (num_lines >= 0) {
            fprintf(stderr, "Expected -n, where n is the number of lines\n");
            return EXIT_FAILURE;
        }
        /* Now make num_lines the positive number it's supposed to be. */
        num_lines = -num_lines;
    }

    /* First, let's get enough storage for a list of n pointers... */
    line_ptrs = malloc(sizeof *line_ptrs * num_lines);
    if (!line_ptrs) {
        fprintf(stderr, "Out of memory. Sorry.\n");
        return EXIT_FAILURE;
    }

```



```

}
/* and make them all point to NULL */
for (i = 0; i < num_lines; i++)
    line_ptrs[i] = NULL;

/* Now start reading */
current_line = 0;
do {
    getline(buffer, sizeof buffer);
    if (!feof(stdin)) {
        if (line_ptrs[current_line]) {
            /* there's already something here */
            free(line_ptrs[current_line]);
        }
        line_ptrs[current_line] = dupstr(buffer);
        if (!line_ptrs[current_line]) {
            fprintf(stderr, "Out of memory.  Sorry.\n");
            return EXIT_FAILURE;
        }
        current_line = (current_line + 1) % num_lines;
    }
} while (!feof(stdin));

/* Finished reading the file, so we are ready to print the lines */
for (i = 0; i < num_lines; i++) {
    j = (current_line + i) % num_lines;
    if (line_ptrs[j]) {
        printf("%s", line_ptrs[j]);
        free(line_ptrs[j]);
    }
}

return EXIT_SUCCESS;
}

```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 5-14, page 121

Solution by Steven Huang

Modify the sort program to handle a `-r` flag, which indicates sorting in reverse (decreasing) order. Be sure that `-r` works with `-n`.

```
/* K&R Exercise 5-14 */
/* Steven Huang */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define TRUE 1
#define FALSE 0

#define MAXLINES 5000          /* maximum number of lines */
char *lineptr[MAXLINES];

#define MAXLEN 1000            /* maximum length of a line */

int reverse = FALSE;

/* K&R2 p29 */
int getline(char s[], int lim)
{
    int c, i;

    for (i = 0; i < lim - 1 && (c = getchar()) != EOF && c != '\n'; i++)
        s[i] = c;
    if (c == '\n') {
        s[i++] = c;
    }
    s[i] = '\0';
    return i;
}

/* K&R2 p109 */
```

```

int readlines(char *lineptr[], int maxlines)
{
    int len, nlines;
    char *p, line[MAXLEN];

    nlines = 0;
    while ((len = getline(line, MAXLEN)) > 0)
        if (nlines >= maxlines || (p = malloc(len)) == NULL)
            return -1;
        else {
            line[len - 1] = '\0'; /* delete the newline */
            strcpy(p, line);
            lineptr[nlines++] = p;
        }
    return nlines;
}

/* K&R2 p109 */
void writelines(char *lineptr[], int nlines)
{
    int i;

    for (i = 0; i < nlines; i++)
        printf("%s\n", lineptr[i]);
}

int pstrcmp(const void *p1, const void *p2)
{
    char * const *s1 = reverse ? p2 : p1;
    char * const *s2 = reverse ? p1 : p2;

    return strcmp(*s1, *s2);
}

int numcmp(const void *p1, const void *p2)
{
    char * const *s1 = reverse ? p2 : p1;
    char * const *s2 = reverse ? p1 : p2;
    double v1, v2;

    v1 = atof(*s1);
    v2 = atof(*s2);
    if (v1 < v2)
        return -1;
    else if (v1 > v2)

```

```

        return 1;
    else
        return 0;
}

int main(int argc, char *argv[])
{
    int nlines;
    int numeric = FALSE;
    int i;

    for (i = 1; i < argc; i++) {
        if (*argv[i] == '-') {
            switch (*(argv[i] + 1)) {
                case 'n':    numeric = TRUE;    break;
                case 'r':    reverse = TRUE;    break;
                default:
                    fprintf(stderr, "invalid switch '%s'\n", argv[i]);
                    return EXIT_FAILURE;
            }
        }
    }

    if ((nlines = readlines(lineptr, MAXLINES)) >= 0) {
        qsort(lineptr, nlines, sizeof(*lineptr), numeric ? numcmp : pstrcmp);
        writelines(lineptr, nlines);
        return EXIT_SUCCESS;
    } else {
        fputs("input too big to sort\n", stderr);
        return EXIT_FAILURE;
    }
}

```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 6-1, page 136

Solution by Ben Pfaff

Regis (a comp.lang.c regular) also provided a solution, which was far too big to post here. It may still be available from [his website](#).

Our version of getword does not properly handle underscores, string constants, comments, or preprocessor control lines. Write a better version.

```
/* K&R 6-1: "Our version of getword() does not properly handle
underscores, string constants, or preprocessor control lines.
Write a better version."
```

This is intended to be a solution to K&R 6-1 in "category 0" as defined by the official rules given on Richard Heathfield's "The C Programming Language Answers To Exercises" page, found at <http://users.powernet.co.uk/eton/kandr2/index.html>.

For more information on the language for which this is a lexical analyzer, please see the comment preceding getword() below.

Note that there is a small modification to ungetch() as defined by K&R. Hopefully this lies within the rules. */

```
/* knr61.c - answer to K&R2 exercise 6-1.
Copyright (C) 2000 Ben Pfaff <blp@gnu.org>.
```

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

```

    You should have received a copy of the GNU General Public License
    along with this program; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA
    02111-1307, USA. */
#include <ctype.h>
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* Tokens.  Other non-whitespace characters self-represent themselves
   as tokens. */
enum token
{
    TOK_ID = UCHAR_MAX + 1,      /* Identifier. */
    TOK_STRING,                  /* String constant. */
    TOK_CHAR,                    /* Character constant. */
    TOK_EOF                      /* End of file. */
};

enum token getword (char *word, int lim);

static int skipws (void);
static int getstelem (char **, int *, int);

static int getch (void);
static void ungetch (int);
static void putch (char **, int *, int);

/* Main program for testing. */
int
main (void)
{
    ungetch ('\n');

    for (;;)
    {
        char word[64];
        enum token token;

        /* Get token. */
        token = getword (word, sizeof word);

        /* Print token type. */
    }
}

```

```

switch (token)
{
case TOK_ID:
    printf ("id");
    break;

case TOK_STRING:
    printf ("string");
    break;

case TOK_CHAR:
    printf ("char");
    break;

case TOK_EOF:
    printf ("eof\n");
    return 0;

default:
    printf ("other");
    word[0] = token;
    word[1] = '\0';
    break;
}

/* Print token value more or less unambiguously. */
{
    const char *s;

    printf ("\t");
    for (s = word; *s != '\0'; s++)
        if (isprint (*s) && *s != '\\')
            putchar (*s);
        else if (*s == '\\')
            printf ("\\");
        else
            /* Potentially wrong. */
            printf ("\\x%02x", *s);
    printf ("'\n");
}
}

```

```

/* Parses C-like tokens from stdin:

```

- Parses C identifiers and string and character constants.
- Other characters, such as operators, punctuation, and digits not part of identifiers are considered as tokens in themselves.
- Skip comments and preprocessor control lines.

Does not handle trigraphs, line continuation with \, or numerous other special C features.

Returns a token type. This is either one of TOK_* above, or a single character in the range 0...UCHAR_MAX.

If TOK_ID, TOK_STRING, or TOK_CHAR is returned, WORD[] is filled with the identifier or string value, truncated at LIM - 1 characters and terminated with '\0'.

For other returned token types, WORD[] is indeterminate. */

```
enum token
getword (char *word, int lim)
{
    int beg_line, c;

    for (;;)
    {
        beg_line = skipws ();
        c = getch ();

        if (!beg_line || c != '#')
            break;

        /* Skip preprocessor directive. */
        do
        {
            c = getch ();
            if (c == EOF)
                return TOK_EOF;
        }
        while (c != '\n');
        ungetch ('\n');
    }

    if (c == EOF)
        return TOK_EOF;
```



```

else if (c == '_' || isalpha ((unsigned char) c))
{
    do
    {
        putchar (&word, &lim, c);
        c = getch ();
    }
    while (isalnum ((unsigned char) c) || c == '_');

    ungetch (c);
    return TOK_ID;
}
else if (c == '\\' || c == '"')
{
    int quote = c;
    word[0] = '\\0';
    while (getstelem (&word, &lim, quote))
        ;
    return quote == '\\' ? TOK_CHAR : TOK_STRING;
}
else
    return (unsigned char) c;
}

/* Skips whitespace and comments read from stdin.
   Returns nonzero if a newline was encountered, indicating that we're
   at the beginning of a line. */
static int
skipws (void)
{
    /* Classification of an input character. */
    enum class
    {
        CLS_WS = 0,                /* Whitespace. */
        CLS_BEG_CMT,              /* Slash-star beginning a comment. */
        CLS_END_CMT,              /* Star-slash ending a comment. */
        CLS_OTHER,                /* None of the above. */

        CLS_IN_CMT = 4            /* Combined with one of the above,
                                   indicates we're inside a comment. */
    };

    /* Either 0, if we're not inside a comment,
       or CLS_IN_CMT, if we are inside a comment. */
    enum class in_comment = 0;

```

```

/* Have we encountered a newline outside a comment? */
int beg_line = 0;

for (;;)
{
    int c;                /* Input character. */
    enum class class;     /* Classification of `c'. */

    /* Get an input character and determine its classification. */
    c = getch ();
    switch (c)
    {
        case '\n':
            if (!in_comment)
                beg_line = 1;
            /* Fall through. */

        case ' ': case '\f': case '\r': case '\t': case '\v':
            class = CLS_WS;
            break;

        case '/':
            /* Outside a comment, slash-star begins a comment. */
            if (!in_comment)
            {
                c = getch ();
                if (c == '*')
                    class = CLS_BEG_CMT;
                else
                {
                    ungetch (c);
                    c = '/';
                    class = CLS_OTHER;
                }
            }
            else
                class = CLS_OTHER;
            break;

        case '*':
            /* Inside a comment, star-slash ends the comment. */
            if (in_comment)
            {
                c = getch ();

```

```

        if (c == '/')
            class = CLS_END_CMT;
        else
        {
            ungetch (c);
            class = CLS_OTHER;
        }
    }
else
    class = CLS_OTHER;
break;

default:
    /* Other characters. */
    if (c == EOF)
        return 0;
    class = CLS_OTHER;
}

/* Handle character `c' according to its classification
   and whether we're inside a comment. */
switch (class | in_comment)
{
case CLS_WS:
case CLS_WS | CLS_IN_CMT:
case CLS_OTHER | CLS_IN_CMT:
    break;

case CLS_BEG_CMT:
    in_comment = CLS_IN_CMT;
    break;

case CLS_OTHER:
    ungetch (c);
    return beg_line;

case CLS_END_CMT | CLS_IN_CMT:
    in_comment = 0;
    break;

case CLS_BEG_CMT | CLS_IN_CMT:
case CLS_END_CMT:
default:
    printf ("can't happen\n");
    break;
}

```

```

    }
}

/* Get a character inside a quoted string or character constant.
   QUOTE is ' for a character constant or " for a quoted string.
   *WORDP points to a string being constructed that has *LIMP bytes
   available. */
static int
getstelem (char **wordp, int *limp, int quote)
{
    int c;

    /* Handle end-of-quote and EOF. */
    c = getch ();
    if (c == quote || c == EOF)
        return 0;

    /* Handle ordinary string characters. */
    if (c != '\\')
    {
        putchar (wordp, limp, c);
        return 1;
    }

    /* We're in a \ escape sequence.
       Get the second character. */
    c = getch ();
    if (c == EOF)
        return 0;

    /* Handle simple single-character escapes. */
    {
        static const char escapes[] = {"' '??\"\\a\\b\\f\\n\\r\\t\\v"};
        const char *cp = strchr (escapes, c);
        if (cp != NULL)
        {
            putchar (wordp, limp, cp[1]);
            return 1;
        }
    }

    /* Handle hexadecimal and octal escapes.
       This also handles invalid escapes by default,
       doing nothing useful with them.

```

```

    That's okay because invalid escapes generate undefined behavior. */
{
    unsigned char v = 0;

    if (c == 'x' || c == 'X')
        for (;;)
        {
            static const char hexits[] = "0123456789abcdef";
            const char *p;

            c = getch ();
            p = strchr (hexits, tolower ((unsigned char) c));
            if (p == NULL)
                break;
            v = v * 16 + (p - hexits);
        }
    else
    {
        int i;

        for (i = 0; i < 3; i++)
        {
            v = v * 8 + (c - '0');
            c = getch ();
            if (c < '0' || c > '7')
                break;
        }
    }

    putch (wordp, limp, v);
    ungetch (c);
}

return 1;
}

/* Capacity of putback buffer. */
#define BUFSIZE 100

/* Putback buffer. */
char buf[BUFSIZE];

/* Number of characters in putback buffer. */
int bufp = 0;

```

```

/* Retrieves and returns a character from stdin or from the putback
   buffer.
   Returns EOF if end of file is encountered. */
int
getch (void)
{
    return bufp > 0 ? buf[--bufp] : getchar ();
}

/* Stuffs character C into the putback buffer.
   From the caller's perspective, fails silently if the putback buffer
   is full. */
void
ungetch (int c)
{
    if (c == EOF)
        return;

    if (bufp >= BUFSIZE)
        printf ("ungetch: too many characters\n");
    else
        buf[bufp++] = c;
}

/* Stuffs character C into buffer *WORDP, which has *LIMP bytes
   available.
   Advances *WORDP and reduces *LIMP as appropriate.
   Drops the character on the floor if it would overflow the buffer.
   Ensures that *WORDP is null terminated if possible. */
static void
putch (char **wordp, int *limp, int c)
{
    if (*limp > 1)
    {
        *(*wordp)++ = c;
        (*limp)--;
    }
    if (*limp > 0)
        **wordp = '\0';
}

/*
   Local variables:
   compile-command: "checkergcc -W -Wall -ansi -pedantic knr61.c -o knr61"
   End:

```

* /

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 6-3, page 143

Solution by Richard Heathfield

Bug (noticed by John W Krahn) fixed 11 June 2002. The noise word list was broken because it contained out-of-order data. I fixed this, and made the program more generally useful, by performing all string comparisons without regard to case.

Write a cross-referencer that prints a list of all words in a document, and, for each word, a list of the line numbers on which it occurs. Remove noise words like "the", "and," and so on.

```
/* Write a cross-referencer program that prints a list of all words in a
 * document, and, for each word, a list of the line numbers on which it
 * occurs. Remove noise words like "the", "and," and so on.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

/* no such thing as strdup, so let's write one
 *
 * supplementary question: why did I call this function dupstr,
 * rather than strdup?
 */

char *dupstr(char *s)
{
    char *p = NULL;

    if(s != NULL)
    {
        p = malloc(strlen(s) + 1);
        if(p)
```



```

    {
        strcpy(p, s);
    }
}

return p;
}

/* case-insensitive string comparison */
int i_strcmp(const char *s, const char *t)
{
    int diff = 0;
    char cs = 0;
    char ct = 0;

    while(diff == 0 && *s != '\0' && *t != '\0')
    {
        cs = tolower((unsigned char)*s);
        ct = tolower((unsigned char)*t);
        if(cs < ct)
        {
            diff = -1;
        }
        else if(cs > ct)
        {
            diff = 1;
        }
        ++s;
        ++t;
    }

    if(diff == 0 && *s != *t)
    {
        /* the shorter string comes lexicographically sooner */
        if(*s == '\0')
        {
            diff = -1;
        }
        else
        {
            diff = 1;
        }
    }
}

```

```

    return diff;
}

struct linelist
{
    struct linelist *next;
    int line;
};

struct wordtree
{
    char *word;
    struct linelist *firstline;
    struct wordtree *left;
    struct wordtree *right;
};

void printlist(struct linelist *list)
{
    if(list != NULL)
    {
        printlist(list->next);
        printf("%6d ", list->line);
    }
}

void printtree(struct wordtree *node)
{
    if(node != NULL)
    {
        printtree(node->left);
        printf("%18s ", node->word);
        printlist(node->firstline);
        printf("\n");
        printtree(node->right);
    }
}

struct linelist *addlink(int line)
{
    struct linelist *new = malloc(sizeof *new);
    if(new != NULL)
    {

```

```

    new->line = line;
    new->next = NULL;
}

return new;
}

void deletelist(struct linelist *listnode)
{
    if(listnode != NULL)
    {
        deletelist(listnode->next);
        free(listnode);
    }
}

void deleteword(struct wordtree **node)
{
    struct wordtree *temp = NULL;
    if(node != NULL)
    {
        if(*node != '\0')
        {
            if((*node)->right != NULL)
            {
                temp = *node;
                deleteword(&temp->right);
            }
            if((*node)->left != NULL)
            {
                temp = *node;
                deleteword(&temp->left);
            }
            if((*node)->word != NULL)
            {
                free((*node)->word);
            }
            if((*node)->firstline != NULL)
            {
                deletelist((*node)->firstline);
            }
            free(*node);
            *node = NULL;
        }
    }
}

```

```

    }
}

struct wordtree *addword(struct wordtree **node, char *word, int line)
{
    struct wordtree *wordloc = NULL;
    struct linelist *newline = NULL;
    struct wordtree *temp = NULL;
    int diff = 0;

    if(node != NULL && word != NULL)
    {
        if(NULL == *node)
        {
            *node = malloc(sizeof **node);
            if(NULL != *node)
            {
                (*node)->left = NULL;
                (*node)->right = NULL;
                (*node)->word = dupstr(word);
                if((*node)->word != NULL)
                {
                    (*node)->firstline = addlink(line);
                    if((*node)->firstline != NULL)
                    {
                        wordloc = *node;
                    }
                }
            }
        }
        else
        {
            diff = i_strcmp((*node)->word, word);
            if(0 == diff)
            {
                /* we have seen this word before! add this line number to
                 * the front of the line number list. Adding to the end
                 * would keep them in the right order, but would take
                 * longer. By continually adding them to the front, we
                 * take less time, but we pay for it at the end by having
                 * to go to the end of the list and working backwards.
                 * Recursion makes this less painful than it might have been.
                 */
                newline = addlink(line);
            }
        }
    }
}

```

```

        if(newline != NULL)
        {
            wordloc = *node;
            newline->next = (*node)->firstline;
            (*node)->firstline = newline;
        }
    }
    else if(0 < diff)
    {
        temp = *node;
        wordloc = addword(&temp->left, word, line);
    }
    else
    {
        temp = *node;
        wordloc = addword(&temp->right, word, line);
    }
}

if(wordloc == NULL)
{
    deleteword(node);
}

return wordloc;
}

/* We can't use strchr because it's not yet been discussed, so we'll
 * write our own instead.
 */
char *char_in_string(char *s, int c)
{
    char *p = NULL;

    /* if there's no data, we'll stop */
    if(s != NULL)
    {
        if(c != '\0')
        {
            while(*s != '\0' && *s != c)
            {
                ++s;
            }
        }
    }
}

```

```

        if(*s == c)
        {
            p = s;
        }
    }

    return p;
}

/* We can't use strtok because it hasn't been discussed in the text
 * yet, so we'll write our own.
 * To minimise hassle at the user end, let's modify the user's pointer
 * to s, so that we can just call this thing in a simple loop.
 */
char *tokenise(char **s, char *delims)
{
    char *p = NULL;
    char *q = NULL;

    if(s != NULL && *s != '\0' && delims != NULL)
    {
        /* pass over leading delimiters */
        while(NULL != char_in_string(delims, **s))
        {
            ++*s;
        }
        if(**s != '\0')
        {
            q = *s + 1;
            p = *s;
            while(*q != '\0' && NULL == char_in_string(delims, *q))
            {
                ++q;
            }

            *s = q + (*q != '\0');
            *q = '\0';
        }
    }

    return p;
}

```

```

/* return zero if this word is not a noise word,
 * or non-zero if it is a noise word
 */

```

```

int NoiseWord(char *s)
{
    int found = 0;
    int giveup = 0;

    char *list[] =
    {
        "a",
        "an",
        "and",
        "be",
        "but",
        "by",
        "he",
        "I",
        "is",
        "it",
        "off",
        "on",
        "she",
        "so",
        "the",
        "they",
        "you"
    };
    int top = sizeof list / sizeof list[0] - 1;

    int bottom = 0;

    int guess = top / 2;

    int diff = 0;

    if(s != NULL)
    {
        while(!found && !giveup)
        {
            diff = i_strcmp(list[guess], s);
            if(0 == diff)

```

```

        {
            found = 1;
        }
    else if(0 < diff)
    {
        top = guess - 1;
    }
    else
    {
        bottom = guess + 1;
    }
    if(top < bottom)
    {
        giveup = 1;
    }
    else
    {
        guess = (top + bottom) / 2;
    }
}

return found;
}

/*
 * Argh! We can't use fgets()! It's not discussed until page 164.
 * Oh well... time to roll our own again...
 */

char *GetLine(char *s, int n, FILE *fp)
{
    int c = 0;
    int done = 0;
    char *p = s;

    while(!done && --n > 0 && (c = getc(fp)) != EOF)
    {
        if((*p++ = c) == '\n')
        {
            done = 1;
        }
    }
}

```



```

    *p = '\0';

    if(EOF == c && p == s)
    {
        p = NULL;
    }
    else
    {
        p = s;
    }

    return p;
}

/*
 * Ideally, we'd use a clever GetLine function which expanded its
 * buffer dynamically to cope with large lines. Since we can't use
 * realloc, and because other solutions would require quite hefty
 * engineering, we'll adopt a simple solution - a big buffer.
 *
 * Note: making the buffer static will help matters on some
 * primitive systems which don't reserve much storage for
 * automatic variables, and shouldn't break anything anywhere.
 */

#define MAXLINE 8192

int main(void)
{
    static char buffer[MAXLINE] = {0};
    char *s = NULL;
    char *word = NULL;
    int line = 0;
    int giveup = 0;
    struct wordtree *tree = NULL;

    char *delims = " \t\n\r\a\f\v!\"%^&*()_+={}[]\\|/,<>:;#~?";

    while(!giveup && GetLine(buffer, sizeof buffer, stdin) != NULL)
    {
        ++line;
        s = buffer;
        while(!giveup && (word = tokenise(&s, delims)) != NULL)

```

```
{
    if(!NoiseWord(word))
    {
        if(NULL == addword(&tree, word, line))
        {
            printf("Error adding data into memory. Giving up.\n");
            giveup = 1;
        }
    }
}

if(!giveup)
{
    printf("%18s   Line Numbers\n", "Word");
    printtree(tree);
}

deleteword(&tree);

return 0;
}
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 6-4, page 143

Solution by Bryan Williams

Write a program that prints the distinct words in its input sorted into decreasing order of frequency of occurrence. Precede each word by its count.

Bryan's solution is, as far as I can tell, Category 1 only because he uses EXIT_SUCCESS and EXIT_FAILURE .

```

/*

Chapter 6. Structures

    Write a program that prints out the distinct words in its
    input sorted into decreasing order of frequency of occurrence.
    Precede each word by its count.

Author: Bryan Williams

*/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include <assert.h>

typedef struct WORD
{
    char *Word;
    size_t Count;
    struct WORD *Left;
    struct WORD *Right;
} WORD;

/*
Assumptions: input is on stdin, output to stdout.

Plan: read the words into a tree, keeping a count of how many we have,
allocate an array big enough to hold Treecount (WORD *)'s
walk the tree to populate the array.
qsort the array, based on size.
printf the array
free the array
free the tree
free tibet (optional)

```

```

        free international shipping!
*/

#define SUCCESS                                0
#define CANNOT_MALLOC_WORDARRAY              1
#define NO_WORDS_ON_INPUT                     2
#define NO_MEMORY_FOR_WORDNODE                3
#define NO_MEMORY_FOR_WORD                    4

#define NONALPHA "1234567890 \v\f\n\t\r+=-*/\\,.;:'#~?<>|{}[]`!\\"$%^&()"

int ReadInputToTree(WORD **DestTree, size_t *Treecount, FILE *Input);
int AddToTree(WORD **DestTree, size_t *Treecount, char *Word);
int WalkTree(WORD **DestArray, WORD *Word);
int CompareCounts(const void *vWord1, const void *vWord2);
int OutputWords(FILE *Dest, size_t Count, WORD **WordArray);
void FreeTree(WORD *W);
char *dupstr(char *s);

int main(void)
{
    int Status = SUCCESS;
    WORD *Words = NULL;
    size_t Treecount = 0;
    WORD **WordArray = NULL;

    /* Read the words on stdin into a tree */
    if(SUCCESS == Status)
    {
        Status = ReadInputToTree(&Words, &Treecount, stdin);
    }

    /* Sanity check for no sensible input */
    if(SUCCESS == Status)
    {
        if(0 == Treecount)
        {
            Status = NO_WORDS_ON_INPUT;
        }
    }

    /* allocate a sufficiently large array */
    if(SUCCESS == Status)
    {
        WordArray = malloc(Treecount * sizeof *WordArray);
        if(NULL == WordArray)
        {
            Status = CANNOT_MALLOC_WORDARRAY;
        }
    }
}

```

```

/* Walk the tree into the array */
if(SUCCESS == Status)
{
    Status = WalkTree(WordArray, Words);
}

/* qsort the array */
if(SUCCESS == Status)
{
    qsort(WordArray, Treecount, sizeof *WordArray, CompareCounts);
}

/* walk down the WordArray outputting the values */
if(SUCCESS == Status)
{
    Status = OutputWords(stdout, Treecount, WordArray);
}

/* free the word array */
if(NULL != WordArray)
{
    free(WordArray);
    WordArray = NULL;
}

/* and free the tree memory */
if(NULL != Words)
{
    FreeTree(Words);
    Words = NULL;
}

/* Error report and we are finshed */
if(SUCCESS != Status)
{
    fprintf(stderr, "Program failed with code %d\n", Status);
}
return (SUCCESS == Status ? EXIT_SUCCESS : EXIT_FAILURE);
}

```

```

void FreeTree(WORD *W)
{
    if(NULL != W)
    {
        if(NULL != W->Word)
        {
            free(W->Word);
            W->Word = NULL;
        }
    }
}

```

```

    }
    if(NULL != W->Left)
    {
        FreeTree(W->Left);
        W->Left = NULL;
    }
    if(NULL != W->Right)
    {
        FreeTree(W->Right);
        W->Right = NULL;
    }
}
}

int AddToTree(WORD **DestTree, size_t *Treecount, char *Word)
{
    int Status = SUCCESS;
    int CompResult = 0;

    /* safety check */
    assert(NULL != DestTree);
    assert(NULL != Treecount);
    assert(NULL != Word);

    /* ok, either *DestTree is NULL or it isn't (deep huh?) */
    if(NULL == *DestTree) /* this is the place to add it then */
    {
        *DestTree = malloc(sizeof **DestTree);
        if(NULL == *DestTree)
        {
            /* horrible - we're out of memory */
            Status = NO_MEMORY_FOR_WORDNODE;
        }
        else
        {
            (*DestTree)->Left = NULL;
            (*DestTree)->Right = NULL;
            (*DestTree)->Count = 1;
            (*DestTree)->Word = dupstr(Word);
            if(NULL == (*DestTree)->Word)
            {
                /* even more horrible - we've run out of memory in the middle */
                Status = NO_MEMORY_FOR_WORD;
                free(*DestTree);
                *DestTree = NULL;
            }
            else
            {
                /* everything was successful, add one to the tree nodes count */
                ++*Treecount;
            }
        }
    }
}

```

```

    }
}
else /* we need to make a decision */
{
    CompResult = strcmp(Word, (*DestTree)->Word);
    if(0 < CompResult)
    {
        Status = AddToTree(&(*DestTree)->Left, Treecount, Word);
    }
    else if(0 > CompResult)
    {
        Status = AddToTree(&(*DestTree)->Left, Treecount, Word);
    }
    else
    {
        /* add one to the count - this is the same node */
        ++(*DestTree)->Count;
    }
} /* end of else we need to make a decision */

return Status;
}

int ReadInputToTree(WORD **DestTree, size_t *Treecount, FILE *Input)
{
    int Status = SUCCESS;
    char Buf[8192] = {0};
    char *Word = NULL;

    /* safety check */
    assert(NULL != DestTree);
    assert(NULL != Treecount);
    assert(NULL != Input);

    /* for every line */
    while(NULL != fgets(Buf, sizeof Buf, Input))
    {
        /* strtok the input to get only alpha character words */
        Word = strtok(Buf, NONALPHA);
        while(SUCCESS == Status && NULL != Word)
        {
            /* deal with this word by adding it to the tree */
            Status = AddToTree(DestTree, Treecount, Word);

            /* next word */
            if(SUCCESS == Status)
            {
                Word = strtok(NULL, NONALPHA);
            }
        }
    }
}

```

```

    return Status;
}

int WalkTree(WORD **DestArray, WORD *Word)
{
    int Status = SUCCESS;
    static WORD **Write = NULL;

    /* safety check */
    assert(NULL != Word);

    /* store the starting point if this is the first call */
    if(NULL != DestArray)
    {
        Write = DestArray;
    }

    /* Now add this node and it's kids */
    if(NULL != Word)
    {
        *Write = Word;
        ++Write;
        if(NULL != Word->Left)
        {
            Status = WalkTree(NULL, Word->Left);
        }
        if(NULL != Word->Right)
        {
            Status = WalkTree(NULL, Word->Right);
        }
    }

    return Status;
}

/*
    CompareCounts is called by qsort. This means that it gets pointers to the
    data items being compared. In this case the data items are pointers too.
*/
int CompareCounts(const void *vWord1, const void *vWord2)
{
    int Result = 0;
    WORD * const *Word1 = vWord1;
    WORD * const *Word2 = vWord2;

    assert(NULL != vWord1);
    assert(NULL != vWord2);

```



```

/* ensure the result is either 1, 0 or -1 */
if((*Word1)->Count < (*Word2)->Count)
{
    Result = 1;
}
else if((*Word1)->Count > (*Word2)->Count)
{
    Result = -1;
}
else
{
    Result = 0;
}

return Result;
}

int OutputWords(FILE *Dest, size_t Count, WORD **WordArray)
{
    int Status = SUCCESS;
    size_t Pos = 0;

    /* safety check */
    assert(NULL != Dest);
    assert(NULL != WordArray);

    /* Print a header */
    fprintf(Dest, "Total Words : %lu\n", (unsigned long)Count);

    /* Print the words in descending order */
    while(SUCCESS == Status && Pos < Count)
    {
        fprintf(Dest, "%10lu %s\n", (unsigned long)WordArray[Pos]->Count, WordArray[Pos]-
>Word);
        ++Pos;
    }

    return Status;
}

/*
    dupstr: duplicate a string
*/
char *dupstr(char *s)
{
    char *Result = NULL;
    size_t slen = 0;

    /* sanity check */

```

```
assert(NULL != s);

/* get string length */
slen = strlen(s);

/* allocate enough storage */
Result = malloc(slen + 1);

/* populate string */
if(NULL != Result)
{
    memcpy(Result, s, slen);
    *(Result + slen) = '\0';
}

return Result;
}
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 6-5, page 145

Solutions by Paul Griffiths and Gregory Pietsch

Write a function undef that will remove a name and definition from the table maintained by lookup and install.

```
int undef(char * name) {
    struct nlist * np1, * np2;

    if ((np1 = lookup(name)) == NULL) /* name not found */
        return 1;

    for ( np1 = np2 = hashtab[hash(name)]; np1 != NULL;
          np2 = np1, np1 = np1->next ) {
        if ( strcmp(name, np1->name) == 0 ) { /* name found */

            /* Remove node from list */

            if ( np1 == np2 )
                hashtab[hash(name)] = np1->next;
            else
                np2->next = np1->next;

            /* Free memory */

            free(np1->name);
            free(np1->defn);
            free(np1);

            return 0;
        }
    }

    return 1; /* name not found */
}
```

Gregory Pietsch's solution

```
void undef(char *s)
{
    struct nlist *np1, *np2;
    unsigned hashval = hash(s);

    for (np1 = hashtab[hashval], np2 = NULL;
         np1 != NULL;
         np2 = np1, np1 = np1->next)
        if (strcmp(s, np1->name) == 0) {
            /* found a match */
            free(np1->name);
            free(np1->defn);
            if (np2 == NULL)
                /* at the beginning? */
                hashtab[hashval] = np1->next;
            else
                /* in the middle or at the end? */
                np2->next = np1->next;
            free(np1);
            return;
        }
}
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 7-1, page 153

Solution by [Richard Heathfield](#)

Write a program that converts upper case to lower or lower case to upper, depending on the name it is invoked with, as found in `argv[0]`.

```

/* This program converts its input to upper case
 * (if argv[0] begins with U or u) or lower case.
 * If argc is 0, it prints an error and quits.
 */

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

int main(int argc, char **argv)
{
    int (*convcase[2])(int) = {toupper, tolower};
    int func;
    int result = EXIT_SUCCESS;

    int ch;

    if(argc > 0)
    {
        if(toupper((unsigned char)argv[0][0]) == 'U')
        {
            func = 0;
        }
        else
        {
            func = 1;
        }

        while((ch = getchar()) != EOF)
        {
            ch = (*convcase[func])((unsigned char)ch);
            putchar(ch);
        }
    }
    else
    {
        fprintf(stderr, "Unknown name. Can't decide what to do.\n");
    }
}

```

```

    result = EXIT_FAILURE;
}

return result;
}

```

Here's a category 1 solution from Bryan Williams...

```

/*

Exercise 7-1. Write a program that converts upper case to lower case or lower case
to upper,
            depending on the name it is invoked with, as found in argv[0].

Assumptions: The program should read from stdin, until EOF, converting the output
to stdout
            appropriately.

            The correct outputs should be :

            Program Name          Output
            lower                 stdin with all caps converted to lower
case
            upper                 stdin with all lowercase characters
converted to uppercase
            [anything else]      helpful message explaining how to use this

Author : Bryan Williams

*/

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

#define SUCCESS          0
#define NO_ARGV0        1
#define BAD_NAME        2

int main(int argc, char *argv[])
{
    int ErrorStatus = SUCCESS;
    int (*convert)(int c) = NULL;
    int c = 0;

```

```

/* check that there were any arguments */
if(SUCCESS == ErrorStatus)
{
    if(0 >= argc)
    {
        printf("Your environment has not provided a single argument for the program
name.\n");
        ErrorStatus = NO_ARGV0;
    }
}

/* check for valid names in the argv[0] string */
if(SUCCESS == ErrorStatus)
{
    if(0 == strcmp(argv[0], "lower"))
    {
        convert = tolower;
    }
    else if(0 == strcmp(argv[0], "upper"))
    {
        convert = toupper;
    }
    else
    {
        printf("This program performs two functions.\n");
        printf("If the executable is named lower then it converts all the input on
stdin to lowercase.\n");
        printf("If the executable is named upper then it converts all the input on
stdin to uppercase.\n");
        printf("As you have named it %s it prints this message.\n", argv[0]);
        ErrorStatus = BAD_NAME;
    }
}

/* ok so far, keep looping until EOF is encountered */
if(SUCCESS == ErrorStatus)
{
    while(EOF != (c = getchar()))
    {
        putchar((*convert)(c));
    }
}

/* and return what happened */
return SUCCESS == ErrorStatus ? EXIT_SUCCESS : EXIT_FAILURE;
}

```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 7-2, page 155

Solution by Richard Heathfield

Write a program that will print arbitrary input in a sensible way. As a minimum, it should print non-graphic characters in octal or hexadecimal according to local custom, and break long text lines.

```

/* Use -o for octal output, -x for hexadecimal
 */

#include <stdio.h>

#define OCTAL      8
#define HEXADECIMAL 16

void ProcessArgs(int argc, char *argv[], int *output)
{
    int i = 0;
    while(argc > 1)
    {
        --argc;
        if(argv[argc][0] == '-')
        {
            i = 1;
            while(argv[argc][i] != '\0')
            {
                if(argv[argc][i] == 'o')
                {
                    *output = OCTAL;
                }
                else if(argv[argc][i] == 'x')
                {
                    *output = HEXADECIMAL;
                }
                else
                {
                    /* Quietly ignore unknown switches, because we don't want to
                     * interfere with the program's output. Later on in the
                     * chapter, the delights of fprintf(stderr, "yadayadayada\n")
                     * are revealed, just too late for this exercise.
                     */
                }
            }
            ++i;
        }
    }
}

```

```

    }
}

int can_print(int ch)
{
    char *printable = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890 !
\"#%&'()*+,-./:;<=>?[\\]^_`{|}~\\t\\f\\v\\r\\n";

    char *s;
    int found = 0;

    for(s = printable; !found && *s; s++)
    {
        if(*s == ch)
        {
            found = 1;
        }
    }

    return found;
}

int main(int argc, char *argv[])
{
    int split = 80;
    int output = HEXADEDECIMAL;
    int ch;
    int textrun = 0;
    int binaryrun = 0;
    char *format;
    int width = 0;

    ProcessArgs(argc, argv, &output);

    if(output == HEXADEDECIMAL)
    {
        format = "%02X ";
        width = 4;
    }
    else
    {
        format = "%3o ";
        width = 4;
    }

    while((ch = getchar()) != EOF)
    {
        if(can_print(ch))
        {
            if(binaryrun > 0)
            {

```

```
    putchar( '\\n' );
    binaryrun = 0;
    textrun = 0;
}
putchar(ch);
++textrun;
if(ch == '\\n')
{
    textrun = 0;
}

if(textrun == split)
{
    putchar( '\\n' );
    textrun = 0;
}
}
else
{
    if(textrun > 0 || binaryrun + width >= split)
    {
        printf("\\nBinary stream: ");
        textrun = 0;
        binaryrun = 15;
    }
    printf(format, ch);
    binaryrun += width;
}
}

putchar( '\\n' );

return 0;
}
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 7-3, page 156

Solution by Gregory Pietsch

Revise minprintf to handle more of the other facilities of printf.

```
/* Gregory Pietsch - K&R2 Exercise 7-3 - 2001-01-28 gkp@flash.net */

#include <stdarg.h>
#include <stdio.h>

/* minprintf:  minimal printf with variable argument list */
void minprintf(char *fmt, ...)
{
    va_list ap;
    char *p, *sval;
    int ival;
    double dval;
    unsigned uval;

    va_start(ap, fmt);    /* make ap point to the first unnamed arg */
    for (p = fmt; *p; p++) {
        if (*p != '%') {
            putchar(*p);
            continue;
        }
        switch (*++p) {
            case 'd':
            case 'i':
                ival = va_arg(ap, int);
                printf("%d", ival);
                break;
            case 'c':
                ival = va_arg(ap, int);
                putchar(ival);
                break;
            case 'u':
```

```

        uval = va_arg(ap, unsigned int);
        printf("%u", uval);
        break;
    case 'o':
        uval = va_arg(ap, unsigned int);
        printf("%o", uval);
        break;
    case 'x':
        uval = va_arg(ap, unsigned int);
        printf("%x", uval);
        break;
    case 'X':
        uval = va_arg(ap, unsigned int);
        printf("%X", uval);
        break;
    case 'e':
        dval = va_arg(ap, double);
        printf("%e", dval);
        break;
    case 'f':
        dval = va_arg(ap, double);
        printf("%f", dval);
        break;
    case 'g':
        dval = va_arg(ap, double);
        printf("%g", dval);
        break;
    case 's':
        for (sval = va_arg(ap, char *); *sval; sval++)
            putchar(*sval);
        break;
    default:
        putchar(*p);
        break;
}
va_end(ap);
}

/* end of function */

```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 7-6, page 165

Solutions by Rick Dearman and "Flippant Squirrel".

Write a program to compare two files, printing the first line where they differ.

Here's Rick's solution:

```

/*****
KnR 7-6
-----
Write a program to compare two files and print the
first line where they differ.

Author: Rick Dearman
email: rick@ricken.demon.co.uk

Note: This program prints ALL the lines that are
different using the <> indicators used by
the unix diff command. However this program
will not cope with something as simple as a
line being removed.

In reality the program would be more useful
if it searched forward for matching lines.
This would be a better indicator of the simple
removal of some lines.

This has lead me to track down a version of the
"diff" command available on GNU/Linux systems.
for more information go to the web site at:
www.gnu.org

*****/
#include <stdio.h>
#include <string.h>
#define MAXLINE 1000

void diff_line( char *lineone, char *linetwo, int linenumber )
{
    if(strcmp (lineone, linetwo) < 0 || strcmp (lineone, linetwo) > 0)
        printf( "%d<%s\n%d>%s\n", linenumber, lineone, linenumber, linetwo);
}

```

```

int main(int argc, char *argv[] )
{
    FILE *fp1, *fp2;
    char fp1_line[MAXLINE], fp2_line[MAXLINE];
    int i;

    if ( argc != 3 )
    {
        printf("differ fileone filetwo\n");
        exit(0);
    }

    fp1 = fopen( argv[1], "r" );
    if ( ! fp1 )
    {
        printf("Error opening file %s\n", argv[1]);
    }

    fp2 = fopen( argv[2], "r" );
    if ( ! fp2 )
    {
        printf("Error opening file %s\n", argv[2]);
    }

    i = 0;
    while ( (fgets(fp1_line, MAXLINE, fp1) != NULL)
        && (fgets(fp2_line, MAXLINE, fp2) != NULL))
    {
        diff_line( fp1_line, fp2_line, i );
        i++;
    }

    return 0;
}

```

and here's "flippant squirrel"'s solution:

```

/* Exercise 7-6 - write a program to compare two files, printing the first line
 * where they differ
 *
 * Note : I amended this a bit...if a file is shorter than the other, but is
identical
 * up to that point, the program prints out "EOF" as the string that's not
equal.
 *

```



```

*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define BUFF_SIZE 1000

/* uses fgets, removes the '\n' at the end of the string if it exists */
char *safegets(char *buffer, int length, FILE *file)
{
    char *ptr;
    int len;

    if (buffer != NULL)
    {
        ptr = fgets(buffer, length, file);

        if (ptr != NULL)
        {
            len = strlen(buffer);

            if (len > 0)
            {
                if (buffer[len - 1] == '\n')
                {
                    buffer[len - 1] = '\0';
                }
            }

            return ptr;
        }

        return NULL;
    }
}

int main(int argc, char *argv[])
{
    FILE *leftFile, *rightFile;
    char buff1[BUFF_SIZE], buff2[BUFF_SIZE];
    char *ptr1, *ptr2;
    unsigned long lineNum = 0;

    if (argc < 3)
    {
        fprintf(stderr, "Usage : 7_6 <path to file> <path to file>\n");
        return 0;
    }

```

```

if (!(leftFile = fopen(argv[1], "r")))
{
    fprintf(stderr, "Couldn't open %s for reading\n", argv[1]);
    return 0;
}

if (!(rightFile = fopen(argv[2], "r")))
{
    fprintf(stderr, "Couldn't open %s for reading\n", argv[2]);
    fclose(leftFile); /* RJH 10 Jul 2000 */
    return 0;
}

/* read through each file, line by line */
ptr1 = safegets(buff1, BUFF_SIZE, leftFile);
ptr2 = safegets(buff2, BUFF_SIZE, rightFile);
++lineNum;

/* stop when either we've exhausted either file's data */
while (ptr1 != NULL && ptr2 != NULL)
{
    /* compare the two lines */
    if (strcmp(buff1, buff2) != 0)
    {
        printf("Difference:\n");
        printf("%lu\t\t\"%s\" != \"%s\"\n", lineNum, buff1, buff2);
        goto CleanUp;
    }

    ptr1 = safegets(buff1, BUFF_SIZE, leftFile);
    ptr2 = safegets(buff2, BUFF_SIZE, rightFile);
    ++lineNum;
}

/*
 * if one of the files ended prematurely, it definitely
 * isn't equivalent to the other
 */
if (ptr1 != NULL && ptr2 == NULL)
{
    printf("Difference:\n");
    printf("%lu\t\t\"%s\" != \"EOF\"\n", lineNum, buff1);
}
else if (ptr1 == NULL && ptr2 != NULL)
{
    printf("Difference:\n");
    printf("%lu\t\t\"EOF\" != \"%s\"\n", lineNum, buff2);
}
else
{

```

```
        printf("No differences\n");  
    }
```

CleanUp:

```
    fclose(leftFile);  
    fclose(rightFile);  
    return EXIT_SUCCESS;  
}
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 7-8, page 165

Solution by Steven Huang.

Write a program to print a set of files, starting each new one on a new page, with a title and a running page count for each file.

```

/* K&R Exercise 7-8 */
/* Steven Huang */

/*
   Limitation:  This program doesn't wrap long lines.
*/

#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

#define LINES_PER_PAGE 10
#define TRUE          1
#define FALSE         0

void print_file(char *file_name)
{
    FILE *f;
    int page_number = 1;
    int line_count;
    int c;
    int new_page = TRUE;

    assert(file_name != NULL);

    if ((f = fopen(file_name, "r")) != NULL) {
        while ((c = fgetc(f)) != EOF) {

```

```

    if (new_page) {
        /* print out the header */
        printf("[%s] page %d starts\n", file_name, page_number);
        new_page = FALSE;
        line_count = 1;
    }
    putchar(c);
    if (c == '\n' && ++line_count > LINES_PER_PAGE) {
        /* print out the footer */
        printf("[%s] page %d ends\n", file_name, page_number);
        /* skip another line so we can see it on screen */
        putchar('\n');
        new_page = TRUE;
        page_number++;
    }
}
if (!new_page) {
    /* file ended in the middle of a page, so we still need to
       print a footer */
    printf("[%s] page %d ends\n", file_name, page_number);
}
/* skip another line so we can see it on screen */
putchar('\n');
fclose(f);
}
}

int main(int argc, char *argv[])
{
    int i;

    if (argc < 2) {
        fputs("no files specified\n", stderr);
        return EXIT_FAILURE;
    }
    for (i = 1; i < argc; i++) {
        print_file(argv[i]);
    }
    return EXIT_SUCCESS;
}

```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 7-9, page 168

Solution by Gregory Pietsch

Functions like `isupper` can be implemented to save space or to save time. Explore both possibilities.

This question is best left to an essay rather than code, so here's my take: The easiest way to implement the eleven `is()` functions in C90's version of `<ctype.h>` is via a table lookup. If `UCHAR_MAX` is 255, then a table would take up around 514 8-bit bytes and still have room for five more `is()` functions. In modern programs, this is a miniscule expense of both space and time since a mere table lookup doesn't cost a whole lot (although space may be a priority for embedded systems). Additionally, since the `is()` functions of `<ctype.h>` are locale-dependent and therefore subject to locale-specific whims, a table could more easily be modified than modifying hard calculations. Consider the following three implementations of `isupper()`: Implementation #1:

```
int isupper(int c)
{
    return (c >= 'A' && c <= 'Z');
```

Implementation #2:

```
int isupper(int c)
{
    return (strchr("ABCDEFGHIJKLMNOPQRSTUVWXYZ", c) != NULL);
```

Implementation #3:

```
/* Presumably, _UP is a power of 2 and
 * _Ctype is a table
 */

int isupper(int c)
{
    return ((_Ctype[(unsigned char)c] & _UP) != 0);
```

Implementation #1 fails in EBCDIC and implementation #2 fails in a locale that adds more upperspace characters than the ones mentioned. Implementation #3, however, suggests that `_Ctype[]` can be modified to accommodate new uppercase characters.

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 8-1, page 174

Solution by Andrew Tesker

Ron Scott has also sent me a solution to this exercise. Once he has granted me permission to display it here, I will post it on this site.

Rewrite the program cat from Chapter 7 using read, write, open and close instead of their standard library equivalents. Perform experiments to determine the relative speeds of the two versions.

```

/*
  Andrew Tesker
  ucat.c
  a version of cat using UNIX system access
*/

#include <stdio.h>
#include <fcntl.h>
#define BUFSIZE 1024

int main(int argc, char *argv[])
{
    int fd1;
    void filecopy(int f, int t);

    if(argc == 1)
        filecopy(0, 1);

    else {
        while(--argc > 0)
            if((fd1 = open(*++argv, O_RDONLY, 0)) == -1) {
                printf("unix cat: can't open %s\n", *argv);
                return 1;
            }
            else {
                filecopy(fd1, 1);
                close(fd1);
            }
    }
}

```

```
    }  
}  
  
return 0;  
  
}  
  
void filecopy(int from, int to)  
{  
    int n;  
    char buf[BUFSIZE];  
  
    while((n=read(from, buf, BUFSIZE)) > 0 )  
        write(to, buf, n);  
}
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 8-3, page 179

Solution by Gregory Pietsch

Design and write `_flushbuf`, `fflush`, and `fclose`.

```

/* Editor's note: Gregory didn't supply a main() for this. Normally, in these
situations,
 * I'd supply one myself, so that you can easily run and test the code. But, in this
case,
 * I wouldn't know where to start! If anyone wants to fill the gap, please let me
know.
 * Thanks.
 *          RJH, 28 June 2000
 */

#include <stdio.h>
/* on p.176 */
#include "syscalls.h"
/* or stdlib.h */

/* _flushbuf - flush a buffer
 * According to the code on p. 176, _flushbuf
 * is what putc calls when the buffer is full.
 * EOF as the character causes everything to
 * be written -- I don't tack on the EOF.
 */
int _flushbuf(int c, FILE *f)
{
    int num_written, bufsize;
    unsigned char uc = c;

    if ((f->flag & (_WRITE|_EOF|_ERR)) != _WRITE)
        return EOF;
    if (f->base == NULL && ((f->flag & _UNBUF) == 0)) {
        /* no buffer yet */
        if ((f->base = malloc(BUFSIZ)) == NULL)
            /* couldn't allocate a buffer, so try unbuffered */
            f->flag |= _UNBUF;
        else {
            f->ptr = f->base;
            f->cnt = BUFSIZ - 1;
        }
    }
    if (f->flag & _UNBUF) {
        /* unbuffered write */
        f->ptr = f->base = NULL;
        f->cnt = 0;
    }
}

```

```

    if (c == EOF)
        return EOF;
    num_written = write(f->fd, &uc, 1);
    bufsize = 1;
} else {
    /* buffered write */
    if (c != EOF)
        f->ptr++ = uc;
    bufsize = (int)(f->ptr - f->base);
    num_written = write(f->fd, f->base, bufsize);
    f->ptr = f->base;
    f->cnt = BUFSIZ - 1;
}
if (num_written == bufsize)
    return c;
else {
    f->flag |= _ERR;
    return EOF;
}
}

/* fflush */
int fflush(FILE *f)
{
    int retval;
    int i;

    retval = 0;
    if (f == NULL) {
        /* flush all output streams */
        for (i = 0; i < OPEN_MAX; i++) {
            if ((_iob[i]->flag & _WRITE) && (fflush(_iob[i]) == -1))
                retval = -1;
        }
    } else {
        if ((f->flag & _WRITE) == 0)
            return -1;
        _flushbuf(EOF, f);
        if (f->flag & _ERR)
            retval = -1;
    }
    return retval;
}

/* fclose */
int fclose(FILE *f)
{
    int fd;

    if (f == NULL)
        return -1;
    fd = f->fd;

```

```
fflush(f);  
f->cnt = 0;  
f->ptr = NULL;  
if (f->base != NULL)  
    free(f->base);  
f->base = NULL;  
f->flag = 0;  
f->fd = -1;  
return close(fd);  
}
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 8-4, page 179

Solution by Gregory Pietsch

The standard library function

*int fseek(FILE *fp, long offset, int origin)*

is identical to lseek except that fp is a file pointer instead of a file descriptor and the return value is an int status, not a position. Write fseek. Make sure that your fseek coordinates properly with the buffering done for the other functions of the library.

Here's Gregory's first solution:

```
/* Gregory Pietsch -- My category 0 solution to 8-4 */

int fseek(FILE *f, long offset, int whence)
{
    if ((f->flag & _UNBUF) == 0 && base != NULL) {
        /* deal with buffering */
        if (f->flag & _WRITE) {
            /* writing, so flush buffer */
            fflush(f); /* from 8-3 */
        } else if (f->flag & _READ) {
            /* reading, so trash buffer */
            f->cnt = 0;
            f->ptr = f->base;
        }
    }
    return (lseek(f->fd, offset, whence) < 0);
}
```

...and here's his second, which is considerably more comprehensive:

```
/*
```

[The following solution is in the zip file as krx80401.c - RJH (ed.)]

EXERCISE 8-4

I thought I'd improve 8-4 too. I'm trying my best to get this as close to ISO C as possible given the restrictions that I'm under. (A real implementation would have `fsetpos()` borrow some of the same code.)

```

*/

/* Gregory Pietsch -- My category 0 solution to 8-4 */

#define SEEK_SET 0
#define SEEK_CUR 1
#define SEEK_END 2

int fseek(FILE *f, long offset, int whence)
{
    int result;

    if ((f->flag & _UNBUF) == 0 && base != NULL) {
        /* deal with buffering */
        if (f->flag & _WRITE) {
            /* writing, so flush buffer */
            if (fflush(f))
                return EOF; /* from 8-3 */
        } else if (f->flag & _READ) {
            /* reading, so trash buffer --
             * but I have to do some housekeeping first
             */
            if (whence == SEEK_CUR) {
                /* fix offset so that it's from the last
                 * character the user read (not the last
                 * character that was actually read)
                 */
                if (offset >= 0 && offset <= f->cnt) {
                    /* easy shortcut */
                    f->cnt -= offset;
                    f->ptr += offset;
                    f->flags &= ~_EOF; /* see below */
                    return 0;
                } else
                    offset -= f->cnt;
            }
            f->cnt = 0;
        }
    }
}

```

```
        f->ptr = f->base;
    }
}
result = (lseek(f->fd, offset, whence) < 0);
if (result == 0)
    f->flags &= ~_EOF; /* if successful, clear EOF flag */
return result;
}
```

[Back to index](#)

You are visitor number **456855** - call again soon!

"The C Programming Language", 2nd edition, Kernighan and Ritchie

Answer to Exercise 8-6, page 189

Solution by Bryan Williams

The standard library function `calloc(n, size)` returns a pointer to n objects of size $size$, with the storage initialized to zero. Write `calloc`, by calling `malloc` or by modifying it.

```

/*
    Exercise 8.6. The standard library function calloc(n, size) returns a pointer to
     $n$  objects
                    of size  $size$ , with the storage initialised to zero. Write calloc,
    by calling
                    malloc or by modifying it.

    Author: Bryan Williams
*/

#include <stdlib.h>
#include <string.h>

/*
    Decided to re-use malloc for this because :
    1) If the implementation of malloc and the memory management layer changes,
    this will be ok.
    2) Code re-use is great.
*/

void *mycalloc(size_t nmemb, size_t size)
{
    void *Result = NULL;

    /* use malloc to get the memory */
    Result = malloc(nmemb * size);

    /* and clear the memory on successful allocation */
    if(NULL != Result)
    {
        memset(Result, 0x00, nmemb * size);
    }

    /* and return the result */
    return Result;
}

/* simple test driver, by RJH */

#include <stdio.h>

```

```
int main(void)
{
    int *p = NULL;
    int i = 0;

    p = mycalloc(100, sizeof *p);
    if(NULL == p)
    {
        printf("mycalloc returned NULL.\n");
    }
    else
    {
        for(i = 0; i < 100; i++)
        {
            printf("%08X ", p[i]);
            if(i % 8 == 7)
            {
                printf("\n");
            }
        }
        printf("\n");
        free(p);
    }

    return 0;
}
```

[Back to index](#)

You are visitor number **456855** - call again soon!



Petition for a Software Patent Free Europe



Petition

[Form](#)

[Signatures](#)

Reference

Press

Statements

Sponsors

Defend the European Parliament's Directive!

This petition is directed to the European Parliament and to the European Council. Its goal is to warn European Authorities against the dangers of software patents. This petition is supported by the [EuroLinux Alliance](#) together with European companies and non-profit associations. Please make this petition well known to everybody concerned and do whatever else you can to [help our ongoing struggle](#).

I am concerned by [current plans](#) to legalise software patents in Europe, considering their [damaging effect on innovation and competition](#).

I am concerned by the [possible use of software patents](#) to patent business methods, education methods, health methods, etc.

I am concerned by the current [track record of abuses](#) from the European Patent Office, especially by their tendency to abuse their judicial power to extend the scope of patentability.

I am surprised that no economic report has ever been published by European Authorities to study the impact of software patents on innovation and competition.

I urge decisionmakers at all levels in Europe to enforce the Law, which [clearly prohibits patenting pure computer programs](#), instead of changing it.

I urge decisionmakers at all levels in Europe to reconsider their current plans and to make sure patents are not abused to prohibit or restrict the dissemination of computer programs and intellectual methods.

First Name:

Last Name:

Email:



Home Page:

Position or job:

Company:

Country:

Please enter your email so that we can send you a confirmation request and notify you of forthcoming actions

The countries explicitly listed in this form are all members of associated members of the European Union or the European Patent Convention because important decisions will be taken in Europe in the next 6 months. However, we truly welcome support and signatures from other regions in the World.

The current total number of signatures is:
436753

To support us with your company logo, to suggest missing European countries, to provide translation or for any comments, send an email to: petition@eurolinux.org

Privacy Statement

This web site has been declared at the [CNIL](#), the French national organisation in charge of protecting privacy and individual liberties, and to control the use of personal data in databases. This web site intends to fully implement European privacy Law which is currently one of the strongest in the world.

Only the first name, last name, position, company and country will be published on this site. Emails will only be provided to the European Parliament as a kind of proof of signature

Any user who submits information to this petition is entitled to access it, modify it or to remove it. Requests should be sent to privacy@eurolinux.org. It is our policy not to provide or sell the data you submit to a third party. Your email may be used to confirm your signature or to send to you information about future actions strictly related to this petition. In case we do not receive any different notification from you, your personal data will be erased from this site one year after your last submission.

Some sponsors (randomly chosen)



More sponsors

- [7bulls.com](#) - [@New Media](#) - [@Software PLC](#) - [A.C.T. Germany](#) - [ABC SYSTEM Systemy Komputerowe s.c](#) - [ABN Concept](#) - [AFAS \(Association Française des Auteurs de Shareware\)](#) - [AFUL](#) - [AMLA Informatique](#) - [AP2C](#) - [APR Job](#) - [APRIL](#) - [ASSOLI \(Associazione Software Libero\)](#) - [AT.UG](#) - [Absolute 3d](#) - [Accelance](#) - [Adequat](#) - [Agi Bilgisayar Ltd. Sti.](#) - [Albedo Systems Ltd](#) - [Alcove SA](#) - [Alef Multimedia](#) - [Alpheus Solutions GmbH](#) - [Anacronia](#) - [Andersen and Co](#) - [Anfora](#) - [Anneau Français du Shareware](#) - [Arkam](#) - [Arkane Media](#) - [Asociacion de Internautas](#) - [Association Anima](#) - [Atrid](#) - [Barrapunto](#) - [Beautiful Code B.V.](#) - [Belval SA](#) - [Beyond Bishounen](#) - [BitWizard B.V.](#) - [Bitnetics](#) - [Blorange](#) - [Borbala Software & Services](#) - [Bullet Online Ltd](#) - [Bèta Research](#) - [Caliu \(Catalan Linux Users\)](#) - [Catapult](#) - [Coherent Light](#) - [Colaço & Meideros](#) - [Cookie](#) - [Core Networks](#) - [Cory-Net Ltd.](#) - [CustoM srl](#) - [Cyberbotics](#) - [D-Tek](#) - [DIGIWEB France](#) - [Data Ltd \(London\)](#), [Tresor Data \(Vancouver\)](#), [Tresor Data GmbH \(Munich\)](#) and [Tresor Communication GmbH \(Munich\)](#) - [DemoLinux](#) - [DevTrend Solutions](#) - [Digilex Webdesign & webhosting](#) - [Digital Networks](#) - [Domain Partner](#) - [E-Suite](#) - [E/AS](#) - [ESN](#) - [EDV-Beratung, Sicherheit, Netzbetreuung](#) - [ESR Pollmeier GmbH](#) - [Easter Eggs SARL](#) - [Eazel](#) - [Elfring Fonts](#) - [Europe-Shareware](#) - [FSF Europe](#) - [Fen Systems Ltd.](#) - [Firenze LUG](#) - [Fluxus](#) - [Free & Alter](#) - [Freepatents](#) - [FrontBase](#) - [Frontsite AG](#) - [G-N-U GmbH](#) - [GL7](#) - [GNU Linux Jobs](#) - [GULDEI-ISEP](#) - [Gavnholt Communications ApS](#) - [German UNIX User Group \(GUUG\) e.V.](#) - [GkWare](#) - [Global ServerMasters](#) - [GraphMed](#) - [HS Development BV](#) - [Hammernode Internet Services, Int'l](#) - [Helide SA](#) - [Hellea sprl](#) - [Henzai Inc.](#) - [HiFish](#) - [HispaFuente SL](#) - [HispaLinux](#) - [I+D Agora S.L.](#) - [I-Biznes](#) - [ID-Alizes](#) - [IDP](#) - [Inter Diseño y Publicidad](#) - [ISI Argonauta S.L.](#) - [Ibgames](#) - [Igiro](#) - [InSecTeam](#) - [Inlab Software GmbH](#) - [Innominate](#) - [Internet Info](#) - [Internet Intelligence Ltd](#) - [Intevation GmbH](#) - [Intradat GmbH](#) - [Introtexx](#) - [IoS GmbH](#) - [Ionix Services](#) - [Janvier Consultants](#) - [Jörg Bemmé Datensysteme](#) - [Jürgen Habich Netzwerktechnik / Kryptotechnik](#) - [KDE](#) - [KTK Software](#) - [KW Consulting GmbH](#) - [Klarälvdalens Datakonsult AB](#) - [Koen Electronic Media Agency](#) - [LF.net Netzwerksysteme GmbH](#) - [LUGA](#) - [Lamer s.c](#) - [Landcroft Computing](#) - [Leader IT](#) - [Lemke Software](#) - [LiHaS \(Datentechnik GbR\)](#) - [Lifix Systems Oy](#) - [Linbox SA](#) - [Linux Bazaar](#) - [Linux Impresa](#) - [Linux Netshop](#) - [LinuxIT](#) - [LinuxTag](#) - [LinuxTest](#) - [Linuxcare](#) - [LiveChatNow](#) - [Logidee SARL](#) - [Lolix SA](#) - [Lollibomb](#) - [Loster](#) - [LuCAS](#) - [Luno Networks Ltd.](#) - [Magic Entertainment](#) - [Magic Vision Multimedia](#) - [Main Group Inc.](#) - [Manchester Multimedia Centre](#) - [MandrakeSoft SA](#) - [Manticore](#) - [Metaweb](#) - [Michael Krauß Datentechnik GmbH](#) - [Micro-Matic Research](#) - [Micrologica Systems Limited](#) - [Murphy Software bv](#) - [MySQL](#) - [NF Consulting](#) - [NLC \(Norwegian Linux Community\)](#) - [Neopoly AG](#) - [NetGallery](#) - [Netpresenter](#) - [Nevrax](#) - [NewsNow Publishing Ltd](#) - [Nexedi SARL](#) - [Nexus Data Solutions, UK](#) - [Nichols Computer Consultancy GmbH](#) - [Nijweide BV](#) - [No Nonsense Software](#) - [O3 Software GmbH und Co.](#) -

[OKDesign oHG](#) - [OSI Online](#) - [OST](#) - [Objectweb](#) - [Octant Informatique](#) - [Open CASCADE SA](#) - [Open Logic Systems](#) - [Open Technology Assembly](#) - [OpenIT](#) - [Opencare SA](#) - [Orange Interactive GmbH](#) - [Orbeon](#) - [PROSA Forbundet af IT-professionelle](#) - [Paralline](#) - [Passys](#) - [Phaidros Software AG](#) - [PlanetLONE Marketing](#) - [Pluto](#) - [Ponsit AG](#) - [Principia Systems](#) - [Progetto Sistemi S.R.L.](#) - [Prosa](#) - [Provisio Digital Media](#) - [Quality Research](#) - [Quantis](#) - [R.U.Sh. \(Regroupement des Utilisateurs du Shareware\)](#) - [RAQPM](#) - [RUS-CERT](#) - [Red Hat](#) - [Ridgecrop Consultants Limited](#) - [SDC](#) - [SIUG \(Swiss Internet User Group\)](#) - [SMO GmbH](#) - [SPiN GmbH](#) - [SafetyCable AS](#) - [SciFace Software GmbH & Co. KG](#) - [Secome](#) - [Seminar für freiheitliche Ordnung](#) - [SerNet Service Network GmbH](#) - [Service-Partner](#) - [Servus.at](#) - [Siena LUG](#) - [Sistema](#) - [Siteck / SIDD SARL](#) - [Skygate Technology](#) - [Skyrix \(MDlink GmbH\)](#) - [Smart Agora AS](#) - [SoftENGINE](#) - [Solidcode](#) - [Solutionworks AG](#) - [St. John's School](#) - [Starfish Sailing Tours Co. Ltd.](#) - [Stensborg APS](#) - [Stichting NLnet](#) - [Stim GmbH](#) - [SuSE AG](#) - [Swiss Open Systems User Group](#) - [Synapse](#) - [Sysgo Real-Time Solutions GmbH](#) - [TGMedien](#) - [TRI-EDRE](#) - [Taz \(Contrapress media GmbH\)](#) - [TechWorks](#) - [TechnoChroniqueurs Associés](#) - [Telecomsoft s.l.](#) - [Tension](#) - [The Gnome Foundation](#) - [The Logical Language Group, Inc.](#) - [Thiesen GmbH](#) - [Ticcon AG](#) - [TigrosNet](#) - [Timefields Ltd](#) - [Toolinux](#) - [TrollTech](#) - [UIT Cambridge Ltd.](#) - [UZCOM](#) - [Uni Rostock Lehrstuhl IuK](#) - [Univers Sante](#) - [VShop](#) - [Velic](#) - [Vibe](#) - [W@ppup](#) - [WMP GmbH](#) - [WORLDNET S. A.S.](#) - [WSKA International](#) - [Wave s.r.l.](#) - [Web in Space e.K.](#) - [Weber Kom](#) - [Wedeco AG](#) - [Winbabel](#) - [XmlBlaster](#) - [Xsec Srl](#) - [Xtelligent IT Consulting GmbH](#) - [Yalbi](#) - [ZCL Information Systems](#) - [ZONE.PL](#) - [a2e](#) - [atsec information security GmbH](#) - [axiel \(Groupe Indigo Développement\)](#) - [catWorkX](#) - [cjt Systemsoftware AG](#) - [dai.at](#) - [daudi](#) - [dbolliger](#) - [eCONNEX AG](#) - [eCreaSol](#) - [electrofante.com](#) - [envicon](#) - [erde3net](#) - [Internetagentur](#) - [euroAstra](#) - [g10 Code GmbH](#) - [i-3, Integración de Información en Internet, S.L.](#) - [i-Orange](#) - [ines GmbH](#) - [iuveno](#) - [jTe@m IT-Solutions GmbH](#) - [kernel concepts / Engel & Faerber GbR](#) - [linux.se](#) - [mauin.com](#) - [media lights](#) - [mirrormill](#) - [montreal.com](#) - [nOnline](#) - [netzwerk neue medien e.V.](#) - [noLimits Technologies](#) - [quintessenz.at](#) - [roCon](#) - [Rost+Oberndörfer Consulting GbR](#) - [tuxline](#) - [vov.de](#) - [webXXL](#) - [weigl interservice](#) - [wjK-Software, Inc.](#) - [Representation Europe](#) - [www.smr.co.uk](#) - [zeitwerk GmbH](#) -

C Bits and Bobs

Richard Heathfield

Last updated: 31 October 2003

[bitpal.c](#) - Binary palindromes

[clint.c](#) - ASCII art in C

[clint.txt](#) - Output of clint.c

[clint.jpg](#) - The source photograph from which clint.c was generated

[convnum.c](#) - An ad hoc demo that turns, say, 1034923 into one million thirty four thousand nine hundred and twenty three

[dirparse.c](#) - Traversing a Windows directory tree. Caution: Windows only!

[dlldemo.zip](#) - A really simple Windows DLL project

[emgen](#) - An error message utility

[fgetdata](#) - Routines to read an entire word or line from a text stream

[foofun.c](#) - See the comp.lang.c FAQ, Question 1.22

[gcg8.c](#) - A code generator generator generator generator generator generator generator generator generator generator!! If your compiler refuses to touch it, try the next one down...

[gcg6.c](#) - A code generator generator generator generator generator generator generator generator generator. If your compiler still doesn't like it, try the next one down...

[gcg5.c](#) - A code generator generator generator generator generator generator generator generator. If your compiler still doesn't like it, try the next one down...

[gcg4.c](#) - A code generator generator generator generator generator generator. If your compiler still doesn't like it, try getting a better compiler.

[getch.c](#) - A simple demonstration of how to get extended keycodes in an MS-DOS or Windows console application.

[hello.c](#) - A very simple demonstration of how to open a file and write some text into it, with proper error handling.

[Using C libraries](#) - A discussion of code libraries in C - why to use them, how to make them, and how to use them

[msrecurse.c](#) - A quotation from Micros~1's Visual C++ documentation, reproduced here as a classic example of - well, read it for yourself. I have not spoiled your fun by annotating it.

[piglatin.c](#) - A rather naive implementation of the Pig Latin language. Use argv for input!

[primes.c](#) - A prime number counter. This program uses the Sieve of Eratosthenes to count how many prime numbers there are in the range 1 to 1000000 (one million). It demonstrates one or two handy performance tips - not least how to store the sieve in under 64KB (so that the program will even work in MS-DOS)

[proof.c](#) - Another take on FAQ 1.22

[rand.c](#) - A simple demonstration of pseudorandom number manipulation.

[safesrand.c](#) - Lawrence Kirby's time_seed routine. Usage: srand(time_seed());

[split.c](#) - Splitting a text line into tokens

[strarr.c](#) - Simple string array management in C (from the book [C Unleashed](#) by Heathfield and Kirby)

[strarr.h](#) - Header for strarr.c

[strarrdemo.c](#) - Demonstration code for string arrays

[ttt.c](#) - A noughts and crosses game - worth a look if you're into permutation processing

[Visual C++ console projects](#) - How to create them.

[Vigenère](#) - Strictly speaking, *not* a Vigenère cipher. This program applies a repeated key to a plaintext using XOR.

[wbods50.c](#) - An MS-DOS implementation of John Conway's "Game of Life" using no headers, no libraries, no assembly language, no nothing - it's just raw C code.

That's it for now. Suggestions for improvements to this page are most welcome.

You are visitor number - call again soon!

Recommended C Books

(Return to [Home](#) page)

Last updated 1 August 2001

#include <disclaimer.h>

Rick Dearman first posted this list (to the best of my knowledge and belief) in comp.lang.c some time in 1998 or 1999. At some point I thought "that's a good list - I'll pinch it", so I did, and occasionally, when Rick wasn't around to post it himself when people asked about good books on C, I would post it instead. At some point, Rick then claimed that he'd never heard of this list, and was glad I'd come up with it, and could he pinch it?

Conclusion: this list was written by nobody^[1]. But I seem to have been landed with the job of maintaining it.

Shut up, Richard. Here's the list:

General introduction/tutorial:

(1) For real beginners looking for a solid introduction:

C Programming: A Modern Approach, K.N.King, W.W.Norton & Company, 1996. ISBN 0-393-96945-2

(2) For somewhat more experienced users looking for a solid introduction:

The C Programming Language, 2nd Ed. Kernighan & Ritchie. Prentice Hall, 1988. ISBN 0-13-110362-8

(3) Other recommended introductory books:

C: How to Program, 2nd Ed. Deitel, H.M. & Deitel, P.J. Prentice Hall, 1994. ISBN: 0-13-226119-7

Reference books:

C: A Reference Manual, 4th Ed. Harbison & Steele. Prentice Hall, 1995. ISBN 0-13-326224-3

The Standard C Library. P.J.Plauger. Prentice Hall, 1992. ISBN 0-13-131509-9

C Programming FAQs Steve Summit. Addison-Wesley, 1996. ISBN 0-201-84519-9

ISO/IEC 9899:1999 ISO. Available from various national standards bodies.

Advanced topics / further exploration:

C Traps and Pitfalls. Andrew Koenig. Addison-Wesley, 1989. ISBN 0-201-17928-8

Expert C Programming: Deep C Secrets, Peter van der Linden, Prentice Hall, 1994. ISBN 0-13-177429-8

Practical C Programming. Steve Oualline. O'Reilly & Associates, 1993. ISBN 1-56592-035-X

Problem Solving And Program Design In C, 2nd Ed. Hanly & Koffman. Addison-Wesley, 1996. ISBN 0-201-59063-8

Algorithms in C, 3rd Ed. Robert Sedgewick Addison-Wesley, 1998. ISBN 0-201-31452-5

*C Unleashed **, Heathfield, Kirby et al, Sams Publishing, 2000. ISBN 0-672-31896-2

*Disclaimer: for a long time I left *C Unleashed* off the list, because I was waiting to see whether it would become well-regarded within comp.lang.c; I considered it irresponsible to regularly post, in that newsgroup, a URL which advertised my own book, without having some indication from the newsgroup that it was acceptable so to do. In a thread at the end of July 2001, however, several regulars within the newsgroup convinced me that it was time to add the book to the list (and one or two evinced surprise that I had not done so already). Of course, I should add that several of the regulars were contributors to the book! Naming and blaming time: apart from myself and Lawrence, other clc contributors to the book include:

- Dann "Pole-Axe" Corbit
- William "Brett" Fishburne
- Scott "Mad Scientist" Fluhrer
- Sam "Determinant" Hobbs
- Jack "Nicholson" Klein
- Mikey "Optimum" Lee
- Ben "The Fence" Pfaff
- Peter "Send Me Money" Seebach
- Steve "MacMan" Summit
- Stephan "Nice Guy" Wilms
- Ian "Hairy" Woods

(Some of these haven't been seen around for a while, more's the pity.)

It occurs to me that this disclaimer draws more attention to the book than is perhaps fair on the other books! This can be rectified easily, if you wish it. Simply write a short summary of any of the other books, and I'll post it here. Please try to keep it to fewer than two thousand words. :-)

Jack Klein offers a [more detailed discussion](#) of some of these books on his site. Also, you might want to check reviews by [the Association of C and C++ Users](#) before buying a book of whose value you are unsure.

[1] ...or so I thought. On 3 August 2001, I received an email from [Ian Hay](#), most of which is reproduced below.

Hi Richard -

I haven't lurked in c.l.c. for a looooong time, but I'm hoping you'll remember me vaguely. I happened to take a look at the c.l.c. group for the first time in a while, and found a discussion about book recommendations. I saw your link to this page (I believe maintained by you) [URL snipped] at the beginning of which you say [Elided. So read up! - RJH]

To solve a minor (and totally insignificant) mystery, it was originally compiled by none other than myself (though it has been modified and augmented a bit - presumably for the better.) It was first posted on April 17, 1999, and then a few times afterwards. I recall you specifically thinking it was a good list. The first appearance (I believe) is linked below:

[\(Click here to link to the article in groups.google.com\)](#)

Anyways, this message is [...] I guess to maintain the integrity of c.l.c.'s institutional and collective memory. I think it's really cool that a complete C amateur (who hasn't written a line of code in three years - I've been in law school) compiled a book list that's still maintained and referred to.

Greetings from Toronto, and glad to see that all the regular posters are still here.

I.

You are visitor number **456855** - come back soon!



[See enlarged version of picture](#)

Introduction

C Unleashed is a book about C programming. Not Microsoft (R) (TM) (etc) C, not Borland C, not Unix C, just plain ol' C, as defined by ISO.

The book covers a wide range of topics, but restricts itself (except in one or two more excitable moments) to the ISO C language, which is portable across a huge variety of architectures, operating systems, and compilers.

If you're looking for the Errata page, [click here](#).

If you came to this page by mistake, and want to go back to my splendiferous (cough) Home Page, [click here](#) instead.

If you want to buy the book from www.amazon.co.uk right now (and who could blame you?), [click here](#) to order it.

Author List

C Unleashed was written between October 1999 and April 2000, by a somewhat large group of people. Here they all are:

- Richard Heathfield
- Lawrence Kirby
- Dann Corbit
- Chad Dixon
- Brett Fishburne
- Scott Fluhrer
- Sam Hobbs
- Ian Kelly
- Jack Klein
- Mike Lee
- Ben Pfaff
- Peter Seebach
- Steve Summit
- Mathew Watson
- Stephan Wilms
- Ian Woods
- Mike Wright

Many thanks to the Sams Editorial Staff (and I've probably missed some people out - my apologies to them):

- Carol Ackerman
- Gus Miklos
- Karen Walsh
- Kim Cofer
- Katie Robinson

Technical Editors

- Peter Seebach
- Steve Krattiger
- Vinay Pai

Chapter Summary

Part I - The C Language Revisited

- 1 C Programmer Unleashed (Richard Heathfield)
- 2 Holy Wars: Programming Standards - Causes and Cures (Richard Heathfield)
- 3 Optimization (Mike Lee)
- 4 Dealing with Dates (Lawrence Kirby)
- 5 Playing with Bits and Bytes (Lawrence Kirby)
- 6 Offline Data Storage and Retrieval (Steve Summit)
- 7 When Things Go Wrong: Code Mending (Richard Heathfield)
- 8 Managing Memory (Richard Heathfield)
- 9 Simulations and Controllers (Mathew Watson)
- 10 Recursion (Peter Seebach)

Part II - Data Organization

- 11 Simple Abstract Data Structures (Richard Heathfield)
- 12 Binary Search Trees (Ben Pfaff)
- 13 Rapid Sorting Techniques (Dann Corbit)
- 14 Tries (Scott Fluhner)
- 15 Sparse Matrix (Brett Fishburne)
- 16 Working with Graphs (Ian Woods)

Part III - Advanced Topics

- 17 Matrix Arithmetic (Sam Hobbs)
- 18 Digital Signal Processing (Jack Klein)
- 19 Expression Parsing and Evaluation (Ian Kelly)
- 20 Making Software Tools (Richard Heathfield)
- 21 Genetic Algorithms (Mike Wright)
- 22 Cross-Platform Development: Communications Programming (Chad Dixon)
- 23 Writing Common Gateway Interface (CGI) Applications in C (Chad Dixon)
- 24 Arbitrary Precision Arithmetic (Ian Kelly)
- 25 Natural Language Processing (Ian Kelly)
- 26 Encryption (Mike Wright)
- 27 Embedded Systems (Stephan Wilms)
- 28 Parallel Processing (Stephan Wilms)
- 29 Looking to the Future: C99 (Peter Seebach)

Note to Chinese readers: I'm sorry, but the sheer volume of spam from Chinese sites has forced me to block emails from .com.cn addresses. If you send me an email from such an address, I won't get to see it.

If you can persuade your countrymen not to send spam any more, I may reconsider the situation next year.

You are visitor number **456855** - call again soon!

Cryptology Programs

Disclaimer: I am interested in cryptology, but by no means an expert.

[Source code](#)

What's in the zip file?

- `msc1.c` - a program to crack monoalphabetic substitution ciphers
- `cdx.c` - encryption/decryption using CDX-2 (needs `sb.c`)
- `sb.c` - S-boxes for CDX-2 (needs `cdx.c`)
- `sbgen.c` - generates a new version of `sb.c` (S-boxes for CDX-2)
- `keygen.c` - generates a moderately random key

Note: `keygen.c` is not portable - it requires console functions `kbhit()` and `getch()`.

Monoalphabetic Substitution Cipher Cracking

`msc1.c` - MSC1 is a program for cracking a monoalphabetic substitution cipher. Click [here](#) for a demonstration run.

CDX-2

The CDX-2 cryptographic algorithm works as follows:

- Load entire plaintext into one buffer
- For each byte in the key (note: therefore, there are as many rounds as the key is long):
 - For each byte in the plaintext
 - Substitute this byte with the S-box value corresponding to this byte's value - use the box corresponding to the current key byte (modulo the number of S-boxes)
 - Rotate the entire plaintext by a number of bits corresponding to a lookup value based on the current key byte
 - Perform a Vigenere-style XOR of the key against the buffer.

`sb.c` contains the S-boxes for CDX-2.

`sbgen.c` will generate a set of random S-boxes (i.e. a new version of `sb.c`). Output is to stdout, so you'll need to redirect to a shiny new C file.

Key Generation

keygen.c generates a one-megabyte random(ish!) bitstream. It's not portable, I'm afraid. It was written for Borland C. The idea is to compile it, run it, and hit the keyboard like crazy. Just keep typing. Anything. Lots. Eventually, it'll come out the other end with this one meg file. Use whichever bits you like from that file. Don't feel you need to keep the rest of them, because you can generate a new file whenever you like.

If anyone feels like calculating the entropy involved, do feel free (and I'd be glad to hear from you).

A Simple Authentication Protocol

[Authenticating yourself](#) can be a real challenge. Here's one suggested solution (which has *NOT* been peer reviewed yet, as at 12 Jan 2001).

Cracking a substitution cipher - a case study

[Cracking substitution ciphers like this one](#), however, is no challenge at all.

You are visitor number **456855** - call again soon!

CDX-2 Encryption Program

I promised you an update to CDX-1, the first implementation of the CDX encryption algorithm, and here it is.

This version is considerably faster (about 16 times faster than CDX-1) and uses S-boxes to enhance security. The S-boxes need a lot more work, but they're better in than out.

Please note that you should not rely on CDX-2 for securing important data. This program is merely illustrative, and I make no great claims for it. Having said that, it hasn't been cracked yet (unless GCHQ or the NSA were feeling particularly bored recently). If you want true security, use a more widely published algorithm. I recommend TwoFish, which claims to be patent-free, and which will *probably* win the competition to succeed DES. You can find out more about TwoFish [here](#) - it's received considerable public scrutiny and, so far, has defeated all known attempts to crack it. (The NSA and/or GCHQ may have cracked it - who knows?)

The above paragraph may be a bit of a let-down, if you were expecting this algorithm to be "unbreakable". I have news for you. Unbreakable algorithms don't exist (except for a correctly-used One Time Pad, which is impractical in large-scale encryption applications). Some people claim their algorithms are unbreakable; and they don't publish those algorithms. (If their algorithms are truly unbreakable, what are they scared of?) Or they do publish the algorithms, only to find that those algorithms are cracked within a week. Or a day. I myself have cracked one unbreakable algorithm within two hours, and another within six hours, despite laying no claims to expertise in cryptanalysis. The best you can hope for is an algorithm which (a) places all the security in the key and (b) cannot be attacked by any method short of brute force. I don't know of any attacks for CDX-2, which is one of the reasons I've published it. If there are such attacks, I want to know about them, and showing the world my algorithm really is the best way to do this, counter-intuitive as it may seem. Much of cryptology is counter-intuitive (sigh).

Now, whilst I don't *know* of any attacks on CDX-2, I do know of one *possible* attack. If all the bytes in the plaintext are the same, the ciphertext displays considerable regularity. I don't know how much of a weakness this is, since in "real life", people don't encode this kind of plaintext. Still, it's a possible weakness, which is why I mention it here.

Update 27 November 2003: For details of the CDX-2 challenge, follow [this link](#).

This program comprises two source files. One contains the expression in ANSI C code of the algorithm itself, and the other contains the S-boxes. The S-box source file is around 140 kilobytes in size(!), so it's not displayed here. (It's just loads of numbers anyway.) But you can download it from [here](#) together with the other two files you need.

You're about to see two source files. The first is not part of the encryption program itself, but is a preliminary step in the build process. I've already performed this step for you, but you can do it yourself if you like. You should be aware, however, that the decryption process must use exactly the same S-boxes as the encryption process! Running "sbgen" re-creates sb.c with completely new S-boxes each time. So you'll either want to use the existing sb.c (for compatibility with everyone else using the same S-boxes) or generate a new sb.c to provide you with your own personal encryption algorithm. Note: don't rely on your own S-boxes for security. You should assume that an adversary has access to your source code!

One further note about S-boxes: CDX-2 uses randomly generated S-boxes, with no selection at all. This is not a good thing. Decent S-boxes are a bit cleverer than that. When I fully understand the clever bit, I'll incorporate it (presumably into CDX-3!). Until then, these will do.

The other file you'll see contains the encryption code itself. But first, the S-box generator:

```
#include <stdio.h>
```

```

#include <stdlib.h>
#include <time.h>
#include <limits.h>

#define DEFAULT_NUM_BOXES 8
#define MAX_BOXES          256

int main(int argc, char **argv)
{
    unsigned int i;
    unsigned int temp;
    unsigned int j;
    unsigned int S_box[UCHAR_MAX + 1];
    unsigned int S_box_reverse[UCHAR_MAX + 1];
    int box = 0;
    unsigned int seed = 0;

    int numboxes = DEFAULT_NUM_BOXES;

    if(argc > 1)
    {
        numboxes = atoi(argv[1]);
        if(numboxes <= 0 || numboxes > MAX_BOXES)
            numboxes = DEFAULT_NUM_BOXES;
    }

    seed = (unsigned)time(NULL);

    srand(seed);

    printf("#include <limits.h>\n\n");
    printf("#include <assert.h>\n\n");
    printf("int GetSBoxCount(void)\n");
    printf("{\n    return %d;\n}\n\n", numboxes);

    printf("unsigned char Subst(int box, unsigned char ch)\n{\n");

    printf("    static unsigned int S_box[%d][UCHAR_MAX + 1] =\n    {\n", numboxes);

    for(box = 0; box < numboxes; box++)
    {
        /* initialise box */
        for(i = 0; i <= UCHAR_MAX; i++)
        {
            S_box[i] = i;
        }
        /* shuffle box */
        for(i = 0; i <= UCHAR_MAX; i++)
        {
            j = (unsigned int)((UCHAR_MAX + 1) * (rand() / ((double)RAND_MAX + 1.0)));
            temp = S_box[i];

```

```

        S_box[i] = S_box[j];
        S_box[j] = temp;
    }
    /* calculate reverse of sbox */
    for(i = 0; i <= UCHAR_MAX; i++)
    {
        S_box_reverse[S_box[i]] = i;
    }

    printf("    {\n    ");

    for(i = 0; i < UCHAR_MAX; i++)
    {
        printf("    %3d,", S_box[i]);
        if(i % 8 == 7)
        {
            printf("\n    ");
        }
    }
    printf("    %3d\n    }%s\n", S_box[i], box == numboxes - 1 ? " " : ",");
}

printf("    };\n");

printf("\n    assert(box >= 0 && box < GetSBoxCount());\n");

printf("\n    return S_box[box][ch];\n}\n\n");

srand(seed);

printf("unsigned char UnSubst(int box, unsigned char ch)\n{\n");
printf("    static unsigned int S_box_reverse[%d][UCHAR_MAX + 1] =\n    {\n",
numboxes);

for(box = 0; box < numboxes; box++)
{
    /* initialise box */
    for(i = 0; i <= UCHAR_MAX; i++)
    {
        S_box[i] = i;
    }
    /* shuffle box */
    for(i = 0; i <= UCHAR_MAX; i++)
    {
        j = (unsigned int)((UCHAR_MAX + 1) * (rand() / ((double)RAND_MAX + 1.0)));
        temp = S_box[i];
        S_box[i] = S_box[j];
        S_box[j] = temp;
    }
    /* calculate reverse of sbox */
    for(i = 0; i <= UCHAR_MAX; i++)

```

```

    {
        S_box_reverse[S_box[i]] = i;
    }
    printf("    {\n    ");

    for(i = 0; i < UCHAR_MAX; i++)
    {
        printf("    %3d,", S_box_reverse[i]);
        if(i % 8 == 7)
        {
            printf("\n    ");
        }
    }

    printf("    %3d\n    }%s\n", S_box_reverse[i], box == numboxes - 1 ? "" : ",");

}
printf("    };\n\n    assert(box >= 0 && box < GetSBoxCount());\n    return S_box_reverse
[box][ch];\n}\n\n");

return 0;
}

```

Once you've run this program (or just unzipped sb.c if you prefer), you are in a position to compile cdx2.c - I've not included a makefile because I don't know what operating system or compiler you're using, but it's easy enough to build cdx2 - just compile cdx2.c and sb.c, then link them together. If you're using Linux:

```
gcc -W -Wall -O2 -ansi -pedantic -o cdx cdx2.c sb.c
```

Borland compiler:

```
bcc32 -A cdx2.c sb.c
```

And without further ado, here's CDX-2 itself:

```

/*
    CDX-2

    ---- A l g o r i t h m   D e t a i l s -----

```

Here's a summary of the encryption algorithm:

Generate a block of 256 integers, each containing a different prime number. (Actually, they don't need to be prime, but most of them should at least be odd, and greater than 50 on average).

Read the plaintext P into a buffer PB.

Read the key K into a buffer KB.

For each byte in the key

Replace each byte in the plaintext with its corresponding S-box value, each time using the next unused S-box (and starting over when you run out).

Locate the prime number PN corresponding to this byte of the key

Rotate all the bits in the plaintext by PN bits (left)

For each byte B in the buffer PB

$P[B] = P[B] \oplus K[B \% \text{length}(KB)]$

roF

roF

Write the plaintext, which is now really ciphertext, to the output file.

The decryption process is of course the reverse. For bit rotation purposes, the key is read starting at the back.

Complete C source code is provided, which should be taken as definitive. The text description above is a simple and thus perhaps inaccurate guide to the algorithm.

```

*/

/***** Source code starts here *****/

/*
 * cdx2.c
 *
 * Cryptographic algorithm copyright 1999-2004 Richard Heathfield
 *
 * Source code copyright 1999-2004 Richard Heathfield
 *
 * All rights reserved.
 *
 * You are hereby granted permission to copy this file onto
 * one computer only, for the purposes of study and compilation
 * and for no other purpose of any kind whatsoever. You are
 * not granted permission to amend this source code in any way.
 * You may not give copies of this file away, and you may not
 * sell copies of this file.
 *
 * Description:
 *   This program implements the CDX-2 encryption algorithm.
 *   (CDX-0 was an internal algorithm which has been discarded
 *   for being insecure. CDX-1 just sucked.)
 *
 * Usage:
 *   Encryption:
 *   enc plaintextfile ciphertextfile keyfile
 *   Decryption:
 *   enc ciphertextfile plaintextfile keyfile -d
 *
 * Assumptions:
 *   I'm not sure how this program would fare with sizeof(int) < 4

```

```

*
* Portability:
*   Code tested on Microsoft Visual C++ 5.0 Pro, Borland C++ 5.02,
*   and Delorie C (gcc), all under Windows 95 DOS box.
*   Also under Linux/GNU gcc.
*
*   I see no reason why it shouldn't work under any platform or
*   operating system. If you have problems, let me know the circumstances
*   (which OS (name and version), which compiler (name and version)). Thanks.
*
* Credits:
*   Bryan Williams (speed!)
*   Mathew Watson (debugging!)
*/

```

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <limits.h>

```

```

#define RETAINMASK(n) ((UCHAR_MAX & ~((unsigned char)0)) >> (n))
#define TOPMASK(n) (~RETAINMASK(n))
#define LOWMASK(n) (~((UCHAR_MAX & ~0) << (n)))

```

```

extern unsigned char Subst(int, unsigned char);
extern unsigned char UnSubst(int, unsigned char);
extern int GetSBoxCount(void);

```

```

int GetFileLength(FILE *fp)
{
    int len = 0;

    rewind(fp);
    while(fgetc(fp) != EOF)
    {
        len++;
    }
    rewind(fp);

    return len;
}

```

```

void XORBuffer(unsigned char *buffer, int DataLen, unsigned char *Key, int KeyLen)
{
    int i;

    for(i = 0; i < DataLen; i++)
    {
        buffer[i] ^= Key[i % KeyLen];
    }
}

```



```

void RotateBufferLeft(unsigned char *buffer, size_t size, int n, unsigned char
*spare)
{
    int i;
    size_t numbytes;
    int numbits;
    unsigned char carriedforward = 0;
    unsigned char broughtforward = 0;
    unsigned char topmask = 0;

    n %= (size * CHAR_BIT);

    numbytes = n / CHAR_BIT;
    memcpy(spare, buffer, numbytes);
    memmove(buffer, buffer + numbytes, size - numbytes);
    memcpy(buffer + size - numbytes, spare, numbytes);
    numbits = n % CHAR_BIT;

    topmask = TOPMASK(numbits);

    /* erm, get the one mad carry to begin with */
    carriedforward = (buffer[0] & topmask) >> (CHAR_BIT - numbits);
    for(i = size - 1; i >= 0; --i)
    {
        /* 1) Store top numbits bits */
        broughtforward = (buffer[i] & topmask) >> (CHAR_BIT - numbits);
        /* 2) Shift other bits by numbits */
        buffer[i] <=< numbits;
        /* 3) and in the previously shifted numbits from byte beforehand */
        buffer[i] |= carriedforward;
        /* 4) carry forward */
        carriedforward = broughtforward;
    }
}

void RotateBufferRight(unsigned char *buffer, size_t size, int n, unsigned char
*spare)
{
    size_t i;
    size_t numbytes;
    int numbits;
    unsigned char carriedforward = 0;
    unsigned char broughtforward = 0;
    unsigned char lowmask = 0;

    n %= (size * CHAR_BIT);

    numbytes = n / CHAR_BIT;

```

```

memcpy(spare, buffer + size - numbytes, numbytes);
memmove(buffer + numbytes, buffer, size - numbytes);
memcpy(buffer, spare, numbytes);

numbits = n % CHAR_BIT;

lowmask = LOWMASK(numbits);

/* erm, get the one mad carry to begin with */
carriedforward = (buffer[size - 1] & lowmask) << (CHAR_BIT - numbits);
for(i = 0; i < size; ++i)
{
    /* 1) Store top numbits bits */
    broughtforward = (buffer[i] & lowmask) << (CHAR_BIT - numbits);
    /* 2) Shift other bits by numbits */
    buffer[i] >>= numbits;
    /* 3) and in the previously shifted numbits from byte beforehand */
    buffer[i] |= carriedforward;
    /* 4) carry forward */
    carriedforward = broughtforward;
}
}

```

```

void Help(char *s)
{
    if(!s || *s == 0)
        s = "enc";
    printf("Usage:\n");
    printf("%s inputfile outputfile keyfile [-d]\n", s);
    printf("-d specifies decryption\n");
}

```

```

unsigned char *GetPass(char *Filename, unsigned int *n)
{
    unsigned char *buff = NULL;
    FILE *fp;

    fp = fopen(Filename, "rb");
    if(fp != NULL)
    {
        *n = (unsigned int)GetFileLength(fp);
        buff = malloc(*n);
        if(buff != NULL)
        {
            if(!fread(buff, *n, 1, fp))
            {
                printf("Read error on key.\n");
                exit(EXIT_FAILURE);
            }
        }
        else

```

```

    {
        printf("Memory error - key too big.\n");
    }
    fclose(fp);
}
return buff;
}

int CheckArgs(int argc, char *argv[])
{
    if(argc < 4)
    {
        Help(argv[0]);
        return 0;
    }
    if(argc > 4 && strcmp(argv[4], "-d"))
    {
        printf("Syntax error.\n");
        return 0;
    }
    return 1;
}

int encrypt(char *infile, char *outfile, char *keyfile, int decrypt)
{
    static unsigned int PN_BLOCK[] =
    {
        2,    3,    5,    7,    11,    13,    17,    19,
        23,    29,    31,    37,    41,    43,    47,    53,
        59,    61,    67,    71,    73,    79,    83,    89,
        97,    101,    103,    107,    109,    113,    127,    131,
        137,    139,    149,    151,    157,    163,    167,    173,
        179,    181,    191,    193,    197,    199,    211,    223,
        227,    229,    233,    239,    241,    251,    257,    263,
        269,    271,    277,    281,    283,    293,    307,    311,
        313,    317,    331,    337,    347,    349,    353,    359,
        367,    373,    379,    383,    389,    397,    401,    409,
        419,    421,    431,    433,    439,    443,    449,    457,
        461,    463,    467,    479,    487,    491,    499,    503,
        509,    521,    523,    541,    547,    557,    563,    569,
        571,    577,    587,    593,    599,    601,    607,    613,
        617,    619,    631,    641,    643,    647,    653,    659,
        661,    673,    677,    683,    691,    701,    709,    719,
        727,    733,    739,    743,    751,    757,    761,    769,
        773,    787,    797,    809,    811,    821,    823,    827,
        829,    839,    853,    857,    859,    863,    877,    881,
        883,    887,    907,    911,    919,    929,    937,    941,
        947,    953,    967,    971,    977,    983,    991,    997,
        1009,    1013,    1019,    1021,    1031,    1033,    1039,    1049,
        1051,    1061,    1063,    1069,    1087,    1091,    1093,    1097,
        1103,    1109,    1117,    1123,    1129,    1151,    1153,    1163,
        1171,    1181,    1187,    1193,    1201,    1213,    1217,    1223,
    }

```

```

    1229, 1231, 1237, 1249, 1259, 1277, 1279, 1283,
    1289, 1291, 1297, 1301, 1303, 1307, 1319, 1321,
    1327, 1361, 1367, 1373, 1381, 1399, 1409, 1423,
    1427, 1429, 1433, 1439, 1447, 1451, 1453, 1459,
    1471, 1481, 1483, 1487, 1489, 1493, 1499, 1511,
    1523, 1531, 1543, 1549, 1553, 1559, 1567, 1571,
    1579, 1583, 1597, 1601, 1607, 1609, 1613, 1619
};

```

```

unsigned char *Spare = NULL;

```

```

unsigned char *Key = NULL;

```

```

unsigned int KeyLen = 0;

```

```

FILE *fpIn = NULL;

```

```

FILE *fpOut = NULL;

```

```

int DataLen = 0;

```

```

unsigned char *buffer = NULL;

```

```

int i = 0;

```

```

int j = 0;

```

```

int numboxes = 0;

```

```

int box = 0;

```

```

numboxes = GetSBoxCount();

```

```

fpIn = fopen(infile, "rb");

```

```

if(fpIn == NULL)

```

```

{

```

```

    printf("Can't open input file.\n");

```

```

    return EXIT_FAILURE;

```

```

}

```

```

fpOut = fopen(outfile, "wb");

```

```

if(fpOut == NULL)

```

```

{

```

```

    fclose(fpIn);

```

```

    printf("Can't open output file.\n");

```

```

    return EXIT_FAILURE;

```

```

}

```

```

DataLen = GetFileLength(fpIn);

```

```

buffer = malloc(DataLen);

```

```

if(buffer == NULL)

```

```

{

```

```

    fclose(fpIn), fclose(fpOut);

```

```

    printf("Insufficient memory.\n");

```

```

    return EXIT_FAILURE;

```

```

}

```

```

Spare = malloc(DataLen);

```

```

if(NULL == Spare)

```

```

{

```

```

    free(buffer);

```

```

    fclose(fpIn), fclose(fpOut);

```

```

    printf("Insufficient memory.\n");
    return EXIT_FAILURE;
}

if(fread(buffer, DataLen, 1, fpIn) != 1)
{
    fclose(fpIn), fclose(fpOut), free(buffer), free(Spare);
    printf("Read error.\n");
    return EXIT_FAILURE;
}

Key = GetPass(keyfile, &KeyLen);
if(Key == NULL)
{
    fclose(fpIn), fclose(fpOut), free(buffer), free(Spare);
    printf("Key error.\n");
    return EXIT_FAILURE;
}
printf("\nProcessing");
fflush(stdout);

if(decrypt)
{
    for(i = 0; i < (int)KeyLen; i++)
    {
        box = (KeyLen - i - 1) % numboxes;

        printf(".");
        fflush(stdout);
        XORBuffer(buffer, DataLen, Key, KeyLen);
        RotateBufferRight(buffer, DataLen, PN_BLOCK[Key[KeyLen - (i + 1)]], Spare);
        for(j = 0; j < DataLen; j++)
        {
            buffer[j] = UnSubst(box, buffer[j]);
        }
    }
}
else
{
    for(i = 0; i < (int)KeyLen; i++)
    {
        box = i % numboxes;
        printf(".");
        fflush(stdout);
        for(j = 0; j < DataLen; j++)
        {
            buffer[j] = Subst(box, buffer[j]);
        }
        RotateBufferLeft(buffer, DataLen, PN_BLOCK[Key[i]], Spare);
        XORBuffer(buffer, DataLen, Key, KeyLen);
    }
}

```

```
    }  
}  
  
if(fwrite(buffer, DataLen, 1, fpOut) != 1)  
{  
    printf("Write error.\n");  
}  
  
fclose(fpIn), fclose(fpOut), free(buffer), free(Spare);  
  
return EXIT_SUCCESS;  
}  
  
int main(int argc, char *argv[])  
{  
    int rc;  
    int decrypt = 0;  
    if(!CheckArgs(argc, argv))  
    {  
        rc = EXIT_FAILURE;  
    }  
    else  
    {  
        if(argc > 4 && strcmp(argv[4], "-d") == 0)  
        {  
            decrypt = 1;  
        }  
        rc = encrypt(argv[1], argv[2], argv[3], decrypt);  
    }  
    printf("\n");  
    return rc;  
}
```

[Back to index](#)

You are visitor number **456855** - call again soon!

/*

This file contains the source code for a cryptographic algorithm. If anyone feels like trying to break it, be my guest. To this end, I've included a file named CIPHER in the same place as this text file.

Note: this implementation is slow. If anyone has any portable ideas for speeding it up, do feel free to drop me a line. The changes must not affect the output file's contents in any way.

---- Algorithm Details -----

Here's a summary of the encryption algorithm:

Generate a block of 256 integers, each containing a different prime number. (Actually, they don't need to be prime, but most of them should at least be odd, and greater than 50 on average).

Read the plaintext P into a buffer PB.

Read the key K into a buffer KB.

For each byte in the key

 Locate the prime number PN corresponding to this byte of the key

 Rotate all the bits in the plaintext by PN bits (left)

For each byte B in the buffer PB

$P[B] = P[B] \wedge K[B \% \text{length}(KB)]$

roF

roF

Write the plaintext, which is now really ciphertext, to the output file.

The decryption process is of course the reverse. For bit rotation purposes, the key is read starting at the back.

Complete C source code is provided, which should be taken as definitive. The text description above is a simple and thus perhaps inaccurate guide to the algorithm.

The accompanying file, CIPHER, has been encrypted by this algorithm.

*/

/***** Source code starts here *****/

/*

* cdx.c

*

* Cryptographic algorithm copyright 1999-2000 Richard Heathfield

*

* Source code copyright 1999-2000 Richard Heathfield

*

* All rights reserved.

*

* You are hereby granted permission to copy this file onto
* one computer only, for the purposes of study and compilation
* and for no other purpose of any kind whatsoever. You are
* not granted permission to amend this source code in any way.
* You may not give copies of this file away, and you may not
* sell copies of this file.

*

* Description:

* This program implements the CDX-1 encryption algorithm.
* (CDX-0 was an internal algorithm which has been discarded
* for being insecure.)

*

* Usage:

* Encryption:

* enc plaintextfile ciphertextfile keyfile

* Decryption:

* enc ciphertextfile plaintextfile keyfile -d

*

* Assumptions:

* CHAR_BIT must be 8, 16, or 32.

* I'm not sure how this program would fare with sizeof(int) < 4

*

* Portability:

* Code tested on Microsoft Visual C++ 5.0 Pro, Borland C++ 5.02,
* and Delorie C (gcc), all under Windows 95 DOS box.

*

* I see no reason why it shouldn't work under Linux/Unix, or indeed
* under any platform or operating system. If you have problems,
* let me know the circumstances (which OS (name and version), which
* compiler (name and version)). Thanks.

*

* Credits:

* Mathew Watson (debugging!)
* Bob Stout's Snippets Collection (bit macros)
*/

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <limits.h>

#define MASK (CHAR_BIT-1)
#define SHIFT ((CHAR_BIT==8)?3:((CHAR_BIT==16)?4:5))
#define BitOff(a,x) ((void)((a)[(x)>>SHIFT] &= ~(1 << ((x)&MASK))))
#define BitOn(a,x) ((void)((a)[(x)>>SHIFT] |= (1 << ((x)&MASK))))
#define BitFlip(a,x) ((void)((a)[(x)>>SHIFT] ^= (1 << ((x)&MASK))))
#define IsBit(a,x) ((a)[(x)>>SHIFT] & (1 << ((x)&MASK)))
```

```
int GetFileLength(FILE *fp)
{
    int len = 0;

    rewind(fp);
    while(fgetc(fp) != EOF)
    {
        len++;
    }
    rewind(fp);

    return len;
}
```

```
void XORBuffer(unsigned char *buffer, int DataLen, unsigned char *Key, int KeyLen)
{
    int i;

    for(i = 0; i < DataLen; i++)
    {
        buffer[i] ^= Key[i % KeyLen];
    }
}
```

```
void RotateBufferLeft(unsigned char *buffer, size_t size, int n)
{

```

```
int LeftBit, ThisBit, NewBit;
size_t i;
int iter;
size_t numbytes;
int numbits;
unsigned char byte;
```

```
n %= (size * CHAR_BIT);
```

```
numbytes = n / CHAR_BIT;
for(i = 0; i < numbytes; i++)
{
    byte = buffer[0];
    memmove(buffer, buffer + 1, size - 1);
    buffer[size - 1] = byte;
}
numbits = n % CHAR_BIT;
```

```
for(iter = 0; iter < numbits; iter++)
{
    LeftBit = IsBit(buffer, 0) ? 1 : 0;
    ThisBit = LeftBit;
    for(i = 0; i < (CHAR_BIT * size) - 1; i++)
    {
        NewBit = IsBit(buffer, (i + 1)) ? 1 : 0;
        if(NewBit != ThisBit)
        {
            BitFlip(buffer, i);
        }
        ThisBit = NewBit;
    }

    if(NewBit != LeftBit)
        BitFlip(buffer, i);
}
}
```

```
void RotateBufferRight(unsigned char *buffer, size_t size, int n)
{
    int RightBit, ThisBit, NewBit;
    size_t i;
    int iter;
    size_t numbytes;
```

```
int numbits;
unsigned char byte;

n %= (size * CHAR_BIT);

numbytes = n / CHAR_BIT;

for(i = 0; i < numbytes; i++)
{
    byte = buffer[size - 1];
    memmove(buffer + 1, buffer, size - 1);
    buffer[0] = byte;
}
numbits = n % CHAR_BIT;

for(iter = 0; iter < numbits; iter++)
{
    RightBit = IsBit(buffer, (CHAR_BIT * size) - 1) ? 1 : 0;
    ThisBit = RightBit;
    for(i = (CHAR_BIT * size) - 1; i > 0; i--)
    {
        NewBit = IsBit(buffer, (i - 1)) ? 1 : 0;
        if(NewBit != ThisBit)
        {
            BitFlip(buffer, i);
        }
        ThisBit = NewBit;
    }

    if(NewBit != RightBit)
        BitFlip(buffer, 0);
}

void Help(char *s)
{
    if(!s || *s == 0)
        s = "enc";
    printf("Usage:\n");
    printf("%s inputfile outputfile keyfile [-d]\n", s);
    printf("-d specifies decryption\n");
}
```

```
unsigned char *GetPass(char *Filename, unsigned int *n)
{
    unsigned char *buff = NULL;
    FILE *fp;

    fp = fopen(Filename, "rb");
    if(fp != NULL)
    {
        *n = (unsigned int)GetFileLength(fp);
        buff = malloc(*n);
        if(buff != NULL)
        {
            if(!fread(buff, *n, 1, fp))
            {
                printf("Read error on key.\n");
                exit(EXIT_FAILURE);
            }
        }
        else
        {
            printf("Memory error - key too big.\n");
        }
        fclose(fp);
    }
    return buff;
}
```

```
int CheckArgs(int argc, char *argv[])
{
    if(argc < 4)
    {
        Help(argv[0]);
        return 0;
    }
    if(argc > 4 && strcmp(argv[4], "-d"))
    {
        printf("Syntax error.\n");
        return 0;
    }
    return 1;
}
```

```
int encrypt(char *infile, char *outfile, char *keyfile, int decrypt)
```

```
{
static unsigned int PN_BLOCK[] =
{
    2,  3,  5,  7, 11, 13, 17, 19,
    23, 29, 31, 37, 41, 43, 47, 53,
    59, 61, 67, 71, 73, 79, 83, 89,
    97, 101, 103, 107, 109, 113, 127, 131,
    137, 139, 149, 151, 157, 163, 167, 173,
    179, 181, 191, 193, 197, 199, 211, 223,
    227, 229, 233, 239, 241, 251, 257, 263,
    269, 271, 277, 281, 283, 293, 307, 311,
    313, 317, 331, 337, 347, 349, 353, 359,
    367, 373, 379, 383, 389, 397, 401, 409,
    419, 421, 431, 433, 439, 443, 449, 457,
    461, 463, 467, 479, 487, 491, 499, 503,
    509, 521, 523, 541, 547, 557, 563, 569,
    571, 577, 587, 593, 599, 601, 607, 613,
    617, 619, 631, 641, 643, 647, 653, 659,
    661, 673, 677, 683, 691, 701, 709, 719,
    727, 733, 739, 743, 751, 757, 761, 769,
    773, 787, 797, 809, 811, 821, 823, 827,
    829, 839, 853, 857, 859, 863, 877, 881,
    883, 887, 907, 911, 919, 929, 937, 941,
    947, 953, 967, 971, 977, 983, 991, 997,
    1009, 1013, 1019, 1021, 1031, 1033, 1039, 1049,
    1051, 1061, 1063, 1069, 1087, 1091, 1093, 1097,
    1103, 1109, 1117, 1123, 1129, 1151, 1153, 1163,
    1171, 1181, 1187, 1193, 1201, 1213, 1217, 1223,
    1229, 1231, 1237, 1249, 1259, 1277, 1279, 1283,
    1289, 1291, 1297, 1301, 1303, 1307, 1319, 1321,
    1327, 1361, 1367, 1373, 1381, 1399, 1409, 1423,
    1427, 1429, 1433, 1439, 1447, 1451, 1453, 1459,
    1471, 1481, 1483, 1487, 1489, 1493, 1499, 1511,
    1523, 1531, 1543, 1549, 1553, 1559, 1567, 1571,
    1579, 1583, 1597, 1601, 1607, 1609, 1613, 1619
};

unsigned char *Key;
unsigned int KeyLen;
FILE *fpIn, *fpOut;
int DataLen;
unsigned char *buffer;
int i;
```

```
fpIn = fopen(infile, "rb");
if(fpIn == NULL)
{
    printf("Can't open input file.\n");
    return EXIT_FAILURE;
}
fpOut = fopen(outfile, "wb");
if(fpOut == NULL)
{
    fclose(fpIn);
    printf("Can't open output file.\n");
    return EXIT_FAILURE;
}

DataLen = GetFileLength(fpIn);
buffer = malloc(DataLen);
if(buffer == NULL)
{
    fclose(fpIn), fclose(fpOut);
    printf("Insufficient memory.\n");
    return EXIT_FAILURE;
}

if(fread(buffer, DataLen, 1, fpIn) != 1)
{
    fclose(fpIn), fclose(fpOut), free(buffer);
    printf("Read error.\n");
    return EXIT_FAILURE;
}

Key = GetPass(keyfile, &KeyLen);
if(Key == NULL)
{
    fclose(fpIn), fclose(fpOut), free(buffer);
    printf("Key error.\n");
    return EXIT_FAILURE;
}
printf("\nProcessing");
fflush(stdout);

if(decrypt)
```

```
{
    for(i = 0; i < (int)KeyLen; i++)
    {
        printf(".");
        fflush(stdout);
        XORBuffer(buffer, DataLen, Key, KeyLen);
        RotateBufferRight(buffer, DataLen, PN_BLOCK[Key[KeyLen - (i + 1)]]);
    }
}
else
{
    for(i = 0; i < (int)KeyLen; i++)
    {
        printf(".");
        fflush(stdout);
        RotateBufferLeft(buffer, DataLen, PN_BLOCK[Key[i]]);
        XORBuffer(buffer, DataLen, Key, KeyLen);
    }
}

if(fwrite(buffer, DataLen, 1, fpOut) != 1)
{
    printf("Write error.\n");
}

fclose(fpIn), fclose(fpOut), free(buffer);

return EXIT_SUCCESS;
}

int main(int argc, char *argv[])
{
    int rc;
    int decrypt = 0;
    if(!CheckArgs(argc, argv))
    {
        rc = EXIT_FAILURE;
    }
    else
    {
        if(argc > 4 && strcmp(argv[4], "-d") == 0)
        {
```

```
    decrypt = 1;
}
rc = encrypt(argv[1], argv[2], argv[3], decrypt);
}
return rc;
}
```


Visual Basic UDP Programming demo

No doubt a fuller explanation of this code will appear in due course. No doubt.

Disclaimer: I didn't write this stuff. A guy called Mark Pickering did, at my request. (Hi Mark!)

I asked him if I could distribute it and he said fine, no problem.

If you find any errors in this code, it's your problem, not mine. It's unsupported.

[Give me the code!](#)

(Return to [Home](#) page)

456855

Simple database manipulation in Visual Basic 5 Pro

C_Dreamer

26/10/1999

Return to [Home](#) page.

Okay, let's get the sarcasm out of the way first. You're right - I'm /not/ a Visual Basic programmer. I do C, and a bit of C++. My Visual Basic skills are not really something I shout about, and with good reason. Nevertheless, I do use it occasionally - not as much nowadays as I used to. Anyway, this morning (26/10/1999) I was asked how to access an Access (if you'll pardon the pun) database from Visual Basic. Having written several programs along these lines a year or two ago, I wittered on briefly about recordsets and workspaces but rapidly had to admit that I'd completely forgotten how to do this! So it was time for a voyage of re-discovery. Thus this article which, I suspect, is about to trace the effects of a strange combination of experience and amnesia.

Without further ado, let's fire up Visual Basic. You might care to pour yourself a coffee while you wait...

Well, that didn't take too long, did it? Okay, the first thing I need to do is to create a database. Normally, one would use Access to do this. It is *possible* in VB, but I seem to recall that it's awkward, so Access would seem to be the Right Thing (ahem - I'll keep my opinions of Access to myself). Since I don't *have* Access, however (at least, I have an Office CD lurking in the depths, but I'm not in *that* much of an exploratory mood), I'm going to have to come up with some other way.

Fortunately, Visual Basic comes complete with a little application called VisData. You will find it skulking in the Samples directory.

In a moment we're going to use it to create a database. But first, what kind of database do we want to create? Just for a moment, there was a chance I'd choose an interesting example, but I'm afraid I'm going to wimp out and choose an old favourite; college grades. Here's my first cut at a design:

Table 1: Course

Each record contains

Field: Course Name

Field: Number of Students

Field: Tutor name

Table 2: Student

Each record contains

Field: Student Name

Field: Cour...

Oh dear, this isn't going to work, is it? Students enrol on more than one course, and more than one student enrolls on any one course (except for the really scary or dull courses, I suppose), so we have the dreaded many-to-many relationship - something to be avoided in database design. Now, I suspect that Access can sort this all out by making an invisible intermediate table - but I'm not sure, and anyway we can write a much more useful table of this kind ourselves. After all, if there's one thing that links a student to a course, it's a grade!

So, here's our second cut at the design:

Table 1: Course

Each record contains

Field: Course Number (key field)

Field: Course Name

Field: Tutor name

Table 2: Student

Each record contains

Field: Student ID (key field)

Field: Student Name

Table 3: Grade

Each record contains

Field: Student ID

Field: Course ID

Field: Grade

This isn't very sophisticated, I know. We'll keep it simple for now. If you are an experienced DBA, you've probably already stopped reading this article anyway, and if you aren't an experienced DBA, you probably don't want too much sophistication. If you don't even know what a DBA is, don't worry about it.

Now, let's fire up VisData (it's a Visual Basic app in the Samples\VisData directory of your VB installation) and create that table.

The first question VisData asked me when I started it was something to do with an ID file. I didn't understand the question, so I clicked No and that seemed to satisfy it. (When in doubt, say No to anything a computer asks you...)

Next, I moused through File/New... / Database/Access 7.0 mdb, which gave me a dialog box asking me where I wanted my new database to be put. Good question. I created a new directory for it. I chose the marvellously original name "VBDBDemo" for both the directory and the database.

I was now presented with a Database Window and an SQL Statement window, neither of which looked particularly friendly. Still, I clicked on the SQL Statement window and it looked like it wanted me to type in it, so I did. I don't know about you, but I'm not that hot on SQL, so I cheated and looked up CREATE TABLE in online help. Hey, at least I knew what to look up, and that's half the battle, right?

Here's what we want to do:

```
CREATE TABLE Course (ID TEXT CONSTRAINT COURSEID PRIMARY KEY, NAME TEXT, TUTOR TEXT);
```

```
CREATE TABLE Student (ID TEXT CONSTRAINT STUDENTID PRIMARY KEY, FIRSTNAME TEXT, LASTNAME TEXT);
```

```
CREATE TABLE Grade (COURSEID TEXT, STUDENTID TEXT, GRADE TEXT, MARK INTEGER);
```

Well, that's my guess anyway. I've included a new field in the Grade table, on a whim - the MARK field, which I reckon will be handy for recording the actual percentage scored on a course, and hence will be useful in later calculations, should we choose to make them.

(Note that the Course ID and Student ID would, in a *real* program, almost certainly be selected by the program, not typed in by the user, using an incremental counter. You can actually tell Access to do this when you create a table - but I couldn't work out how to tell VisData. As long as you know how it *should* be done, we can carry on and do it the lame way for now.)

Okay, let's try that. Copypaste rocks... click Execute...

Well, I was asked if this was a passthrough query. In accordance with IIDSN (If In Doubt, Say No), I said No. I got an error message. So I tried again, and this time said Yes. This time, the error I got was "table already exists" which is cool by me.

I repeated the whole thing, complete with error messages, for the other two tables.

Now, in theory, I have a database - with no records in it, but at least it should have some tables - unless those ominous error messages portended some dire problem. To test my theory, I closed the database and re-opened it to see if I could work out a way of examining the tables. I wasn't hopeful but, hey!, it worked!

So, I'm happy now. I have a database. (The only minor cloud on the horizon is that, when I clicked on the field names, I observed that they had a DataUpdatable property which, in every case, was set to False. But we'll cross that bridge when we come to it.)

I've just leapt forward in time, and discovered that this wasn't in fact a problem - or at least didn't appear to be. Then I went back in time so that I could report this wonderful news here and now. Sorry I can't stop - I have to go back to the future.

(Did you just hear a noise? No? Oh well, maybe it's just me [shudder])

So, now let's add some records to our database. For this, I need to write a simple VB application (at last, I hear you cry!). If I can just get some courses into the database programmatically I'll be happy enough (for now), so I just want three text boxes and two buttons. The text boxes are for ID, Course Name, and Tutor, and the buttons are for "Add" and "Exit". Traditionally, these boxes should be called Text1, Text2 and Text3, but I'm going to break with this long and honourable history for the sake of sheer clarity - I'm instead going to call them edCourseID, edCourseName, edCourseTutor. The ed prefix reminds me that I'm dealing with an edit box, and Course (which may seem redundant) simply reminds me which table I'm dealing with. I don't intend to explore data-bound controls here; apparently they're really easy, so that would feel like ducking out - so I'm just going to use these edit controls to get user data into memory. It'll be our job to get it from memory into the database, using real code.

Please note: unless you have built the sample database using the same field and table names as I did, the following code won't work for you. That may not be a problem, if you are only reading this to get an idea of how to do similar things to your own database - which is fine by me; after all, that's the whole point, isn't it?

So, I close VisData and start a new project. I'm going to put it in the same directory as the database, to make life as simple as possible. I won't insult your intelligence by describing how I put three edit boxes and two buttons onto a form. Give me a moment, and then we'll take a look at the code I put on the button. (Please note, I'm not going to mess about with validation. I'll leave error checking and stuff as an exercise for the motivated reader. Of course, any production quality application must check pretty well everything for errors at all times.)

One minor point which will have you scratching your hair out if I don't tell you about it: you need to add a new reference to your project before we go much further. In the Project menu, click References... and find Microsoft DAO 3.51 Object Library. Check it (click the little square so you get a tick mark or check mark or whatever you call them in whatever country you find yourself in right now), and click OK. (I was scratching around in Project/Components for quite a while and couldn't find it anywhere - so I phoned a friend (hi MasterW!) who put me straight and told me to look in References, whilst roundly cursing my name for daring to remind him of the existence of Microsoft Access).

Go have some coffee. I won't be long.

Okay, I'm back, and it worked beautifully. I just checked with VisData, and it showed me that I have successfully added a record to the Course table. So, here's the code that achieved it:

```
Private Sub cmdAddCourse_Click()  
  
    Dim ws As Workspace  
  
    Dim db As Database  
  
    Dim rs As Recordset  
  
  
    Set ws = CreateWorkspace("VBDemoWorkspace", "admin", "", dbUseJet)  
  
    Set db = ws.OpenDatabase("vbdbdemo.mdb", , False)
```



```
Set rs = db.OpenRecordset("COURSE")
```

```
With rs
```

```
    .AddNew
```

```
    !ID = edCourseID.Text
```

```
    !Name = edCourseName.Text
```

```
    !Tutor = edCourseTutor.Text
```

```
    .Update
```

```
End With
```

```
rs.Close
```

```
Set rs = Nothing
```

```
db.Close
```

```
Set db = Nothing
```

```
ws.Close
```

```
Set ws = Nothing
```

```
End Sub
```

There, that wasn't so bad, was it?

Given this promising start, it shouldn't be too hard to repeat our success with the Student table. Here's the code:

```
Private Sub cmdAddStudent_Click()
```

```
    Dim ws As Workspace
```

```
    Dim db As Database
```

```
    Dim rs As Recordset
```

```
    Set ws = CreateWorkspace("VBDemoWorkspace", "admin", "", dbUseJet)
```

```
    Set db = ws.OpenDatabase("vbdbdemo.mdb", , False)
```

```

Set rs = db.OpenRecordset("STUDENT")

With rs

    .AddNew

    !ID = edStudentID.Text

    !FirstName = edStudentFirstName.Text

    !LastName = edStudentLastName.Text

    .Update

End With

rs.Close

Set rs = Nothing

db.Close

Set db = Nothing

ws.Close

Set ws = Nothing

```

End Sub

Does that look familiar? ;-)

I had a quick peek using VisData to ensure that I successfully added a record. In case you're wondering how I did this, I simply ran the demo to add a record, then slipped into VisData and opened the database, then typed `SELECT * FROM STUDENT;` in the SQL Statement box, and clicked on Execute.

It would be more satisfactory if we could examine the data from within our own program. We're going to do that next - we *have* to, because our next table, Grade, is based on the other two tables; it refers to both of them. The plan is to read the two tables, and put their contents into dropdown lists, so that we can pick and click. To do this, we will have to iterate through the other two tables. Note: the best time to do this is when the user selects the New Grade option. I chose to put all this functionality on a new form (in fact, I used a separate form for each 'New Foo') - so the right place for the code is the `Form_Activate` method of that form.

```
Private Sub Form_Activate()
```

```
Dim ws As Workspace
```

```
Dim db As Database
```

' rs is fine when you only have one table. With more, better names are a good idea.

```
Dim rsStudent As Recordset
```

```
Dim rsCourse As Recordset
```

```
Set ws = CreateWorkspace("VBDemoWorkspace", "admin", "", dbUseJet)
```

```
Set db = ws.OpenDatabase("vbdbdemo.mdb", , False)
```

```
Set rsStudent = db.OpenRecordset("STUDENT")
```

```
Set rsCourse = db.OpenRecordset("COURSE")
```

```
With rsCourse
```

```
    .MoveFirst
```

```
    While Not .EOF
```

```
        ' I chose to display the whole record, with @ signs as field
```

```
        ' delimiters, in case I need to yank out the data later (and I do!).
```

```
        ' The presentation is a bit naff, but this is a demo, not a finished  
product,
```

```
        ' and it's not as if I'm getting paid for it. :-)
```

```
        Temp$ = !ID & "@ " & !Name & " @ " & !Tutor
```

```
        comboCourse.AddItem (Temp$)
```

```
    .MoveNext
```


Wend

End With

' Not being a VB expert, I'm not sure if I have fallen foul of an off-by-one error

' on the record iterations. I've coded this the way I would in C - try to read a

' record, then check EOF in case it failed. I presume VB works in the same way

' but that presumption could be wrong. It seems to work as written, but I've

' not stress-tested it. Either do that yourself, or consult a VB expert.

With rsStudent

.MoveFirst

While Not .EOF

Temp\$ = !ID & "@ " & !FirstName & " " & !LastName

comboStudent.AddItem (Temp\$)

.MoveNext

Wend

End With

rsCourse.Close

Set rsCourse = Nothing

rsStudent.Close

Set rsStudent = Nothing

db.Close

Set db = Nothing

ws.Close

```
Set ws = Nothing
```

```
End Sub
```

Now that code populates our combo boxes; this enables us to select existing records from the database, so that we can link up to them. If we'd just left the user to remember, our data might well have got into a right mess.

Now that we have the combo boxes working (and they do, I checked) we can cut code to add the record to the Grade table.

```
Private Sub cmdAdd_Click()
```

```
    Dim ws As Workspace
```

```
    Dim db As Database
```

```
    Dim rs As Recordset
```

```
    Dim OkToAdd As Boolean
```

```
    Dim Course As Long
```

```
    Dim student As Long
```

```
    Set ws = CreateWorkspace("VBDemoWorkspace", "admin", "", dbUseJet)
```

```
    Set db = ws.OpenDatabase("vbdbdemo.mdb", , False)
```

```
    Set rs = db.OpenRecordset("GRADE")
```

```
    OkToAdd = False
```

```
    If comboCourse.ListIndex >= 0 Then
```

```
        If comboStudent.ListIndex >= 0 Then
```

```
            Course = InStr(comboCourse.List(comboCourse.ListIndex), "@")
```

```
            student = InStr(comboStudent.List(comboStudent.ListIndex), "@")
```

```
            If Course > 0 And student > 0 Then
```

```
        OkToAdd = True
    End If
End If
End If
```

```
If OkToAdd Then

    With rs

        .AddNew

        !StudentID = Left(comboStudent.List(comboStudent.ListIndex), student - 1)
        !CourseID = Left(comboCourse.List(comboCourse.ListIndex), student - 1)
        !Grade = edGradeGrade.Text
        !Mark = Int(Val(edGradeMark.Text))

        .Update

    End With

End If

rs.Close

Set rs = Nothing

db.Close

Set db = Nothing

ws.Close

Set ws = Nothing

End Sub
```

So we can now add records to the course table, the student table, and (subject to those tables having some data in them) the grade table. What's left?

A database traditionally requires four primitive functions to be of any real use:

Create

Retrieve

Amend

Purge

We've dealt with data creation in this tutorial. (If you have been reading closely, you will have spotted some sneaky retrieval going on too.) And that's where we'll leave it for now. I'll pick this up again in a few years, when I will discuss Data Retrieval, Amendment, and Purging in more detail.

Click [here](#) for the sample database and the full source code (as a zip file). Don't worry too much if you get a few "can't load OCX file nag nag nag" messages - I got a bit carried away putting doodads and oojimaflips into the project, but deleted them all eventually.

C_Dreamer

October 1999

456855



Notice is served...

...that this page will be either moved elsewhere or removed altogether, on or shortly after 28 February 2003. If you have bookmarked this page, or have some other reason for not wanting it to be deleted, please email ycprfaqreprieve@eton.powernet.co.uk well before that date, explaining why you want the page to be retained. If sufficient rational objections are received, I will reprieve the page. If I don't get any objections at all, I will take it as a sign that this page is no longer read or required.

C_D, 17 Dec 2002

Unofficial Yahoo! Chat Programming Room FAQ

(Return to [Home](#) page)

Last updated 25 October 2001

Acknowledgements: J E Cripps, Jerry Coffin, Loopy, merrill196884, Mike_Wisdom, MrKite1982, Scoobi_FreeBSD, WarmCupOfJava_TheFranchise

#include <disclaimer.h>

Those of us who like to chat in Yahoo!'s programming room get asked a lot of questions,

of which the most common is... well, read on for that. Since it sometimes strains the atmosphere when you are happily nattering to a friend about sockets or recursion or whatever, and somebody wanders in asking, "Can anybody teach me HTML?", I decided to write this FAQ so that we can happily point them here, knowing that they will find the answer they need. Warning: it isn't always the answer they want. Anyway, here goes:

Q: Can I ask a question?

A: Obviously. You just did. This is the *most common* and the *most irritating* of *all* questions. Please, just ask the question you want to ask! You don't need permission to ask it.

Q: May I ask a question?

A: No. Go away. (See above.)

Q: You seem very negative. Have you had a bad day?

A: Yes. But keep going - you might find your answer in here yet.

Q: Any computer whizzkids in here?

A: No. We are all yashmak salesmen from Denver*. But some of us know JavaScript and/or HTML. The only reason that we are in the programming room at all is to say "Hi" to each other, in common with most other chat rooms.

*Except for one guy who's actually a pig farmer, but who can't spell it, and that's why *he's* in the programming room.

Q: I can't get my scanner to work / how do you attach a file in Hotmail / etc.

A: This is a *programming* room. We talk about programming here. If you can't get your scanner to work, try RTFM. If that fails, contact your vendor. If that fails, ask in the Lobby or in Yahoo! Chat Help. NOT HERE!!!

Just in case you didn't get the message, this is a *programming room*. Okay?

Q: Why won't anyone answer my question?

A: Usually one of four reasons. By far the most common is that the question is, quite simply, off-topic for this room. The second most common reason is that nobody in the room knows the correct answer to

your question. It is a long-standing chatroom convention not to answer a question if you don't know the answer. This saves on bandwidth in the long run, and is much better than giving a wrong answer! Some people do say "No I don't know" but it's not required. Another reason is that the person who does know (or the people who do know) is busy right now, and will get to you in due course, if you're patient. The fourth reason people don't answer questions is because they are being repeated over and over and over and over again - a practice known as flooding. This almost *guarantees* that you won't get an answer.

Q: What's the Java market like right now?

A: About \$1600 per tonne.

Q: Where can I download VB/C/C++?

A: Visual Basic, Visual C++, etc are commercial products - the full version cannot be downloaded legally. If there are demo versions available for download, I would be very interested to hear about them so that I can add links to this page.

Q: Well is there anywhere I CAN download a legal C compiler?

A: GNU C and C++ compilers may be freely downloaded from <http://www.gnu.org/software/software.html>. I'm told there are mirror sites listed at <http://www.gnu.org/order/ftp.html>. And on the commercial front, some people may be offering versions of Visual Basic etc for sale (cheap!) at [ebay](http://ebay.com). DJGPP is a compiler with a *good* reputation and may be found at www.delorie.com/djgpp/ but, not surprisingly, it is a busy site so don't be surprised if you get a 404 - just retry later.

Or check out some early versions of Turbo C from Borland's museum site. All good (albeit pre-ANSI) stuff.

I'm not sure how long it will last, but Borland are currently offering, for free download, Borland C++ 5 !!! Go get it quick!

Or try this site where one can search an entire database of free compilers and interpreters for specific interests, languages, proficiencies, preferences, and platforms. There are probably many other similar such sites, but this is the first I have been told about.

Delorie update: the Delorie site apparently recommends that one download djgpp from Simtel Net either at <http://www.simtel.net/simtel.net/> or <ftp://ftp.simtel.net/pub/simtelnet/gnu/djgpp/>. Consult the FAQ(s) there for installation.

Another possibility is to get a UNIX shell account and use the compiler that should be available there; check with your ISP to see what is already included in the fees charged.

*Q: How do you dynamically allocate an array of $M * N$ items in C?*

A: For this and many more questions about C, please first check [Steve Summit's C Programming FAQ](#) to see if the answer is in there. Then, if it isn't, by all means ask in the room.

Q: Good tutorials on C?

A: These are the best I've found so far:

<http://www.eskimo.com/~scs/cclass/>

http://www.geocities.com/tom_torfs/cintro.html

Q: Good books on C?

GENERAL INTRODUCTION/TUTORIAL:

1) For real beginners looking for a solid introduction:

C Programming: A Modern Approach. K.N.King. W.W.Norton & Company, 1996. ISBN 0-393-96945-2

2) For somewhat more experienced users looking for a solid introduction:

The C Programming Language, 2nd Ed. Kernighan & Ritchie. Prentice Hall, 1988. ISBN 0-13-110362-8

3) Other recommended introductory books:

C: How to Program, 2nd Ed. Deitel, H.M. & Deitel, P.J. Prentice Hall, 1994. ISBN: 0-13-226119-7

REFERENCES:

C : A Reference Manual, 4th Ed. Harbison & Steele. Prentice Hall, 1995. ISBN 0-13-326224-3

The Standard C Library. P.J.Plauger. Prentice Hall, 1992. ISBN 0-13-131509-9

C Programming FAQs Steve Summit Addison-Wesley, 1996. ISBN 0-201-84519-9

ADVANCED TOPICS / FURTHER EXPLORATION:

C Traps and Pitfalls. Andrew Koenig. Addison-Wesley, 1989. ISBN 0-201-17928-8


Expert C Programming: Deep C Secrets, Peter Van Der Linden, Prentice Hall, 1994. ISBN 0-13-177429-8

Practical C Programming. Steve Oualline. O'Reilly & Associates, 1993. ISBN 1-56592-035-X

Problem Solving And Program Design In C, 2nd Ed. Hanly & Koffman. Addison-Wesley, 1996. ISBN 0-201-59063-8

Algorithms in C, 3rd Ed. Robert Sedgewick Addison-Wesley, 1998. ISBN 0-201-31452-5

(Forthcoming) C Unleashed, Heathfield & Kirby, Sams (Macmillan Computer Publishing), 2000. ISBN 0-672-31896-2

Some of you may know I've been working on a book on intermediate/advanced level C programming for a while now. Well, it's ready! I wrote a little over 25% of the book (around 320 pages). Lawrence Kirby is the other principal author. The rest of the book (i.e. most of it) was written by various contributing authors, including Peter Seebach, Steve Summit, Dann Corbit, Jack Klein, and our own Mike Wright (Mike_Wisdom), Ian Woods (hairyian), and Chad Dixon (Loopy/127001). You can order it from [Amazon](#). Fuller information on its contents is available from [The C Unleashed Home Page](#). Because I've given you that Amazon link, I think I'm obliged to put one of these things  here.

Q: Any hackerzz in hear?

A: Please learn to spell. Depends what you mean by hackers really. If you are looking for help cracking into somebody else's computer, you are not looking for a hacker - you are looking for a partner in crime. If you want to spend your life looking over your shoulder, that's your business. True hackers are those who love, and excel at, stretching the limits of what their computer, or language, or operating system, or network, can achieve. In the past, this included breaking systems for the sheer fun of it - but nowadays it's so easy that they don't bother. If you are looking for people like that, it is no good asking "Any hackers in here" because the good ones tend not to boast that title for themselves. (Imagine someone asking 'any geniuses in here?' - a real genius isn't going to answer 'yes, me', is he? The parallel with hackers is exact; hackers are *very* clever people.) So how *do* you find a hacker when you need one? Here's how. Watch the chat, and look for people whose opinion appears to be respected by many others in the room; people who offer intelligent and well-reasoned technical opinions; people who offer help freely when they can; people who speak with the authority that comes purely from knowing their subject. People with all these characteristics tend to be the true hackers.

For further information see [the entry for "Hacker" in the Jargon File](#).

Q: What do people mean when they say "l8r"?

A: l8r means "See you later". (L-EIGHT-R). While we're at it, brb means Be Right Back, afk means Away From Keyboard, PM means Private Message, wb means Welcome Back. bbl is Be Back Later. LOL means Laughing Out Loud, or variations thereof (eg Laughs Out Loud, Laughed Out Loud etc). RO (T)FL means Rolls On (The) Floor Laughing. LMAO means Laughing My Posterior Off. There are others - work it out from context and you'll be fine.

Q: Does anybody know Java?

A: Yes, I understand that Sun Microsystems has a pretty good idea of how it goes.

*Q: All right, pedant - does anybody **here** know Java?*

A: Well, that's a better question at least. (By the way, pedants make the best programmers.) If you want to learn more about Java, try these sites first, just to check that the answer to your question isn't blindingly obvious:

<http://java.sun.com/docs/books/tutorial/>

<http://www.sun.com/stb>

<http://www.jars.com>

<http://www.gamelan.com>

For Visual J++ help, check out <http://www.microsoft.com/visualj>

Q: Can anybody help me with HTML / JavaScript?

A: No. You are beyond help. Oh all right, find a page that does what you want and save that page to your local machine. In Internet Explorer, you do this using View/Source...

If you'd rather "copy and type" try the Mosaic site [here](#).

Or you could have a look at www.htmlgoodies.com. I can thoroughly recommend this site, which contains excellent tutorial material.

Also, there's a nice little book called '10 Minute Guide to HTML 4' by Tim Evans, and published by Que. ISBN: 0-7897-1491-4. In mid-1998 the book cost £13.99 UK, \$14.99 USA, \$21.95 CAN. (Have a

care, because he's rather big on "Cascading Style Sheets" which aren't portable to all browsers.) You can reach Que at www.quecorp.com.

Q: Why do people never respond to my postings?

A: Possibly because you are advertising a site, or quoting a scripture, or flooding, or spamming. For the rules of behaviour on the Net, you are well advised to visit the sites <http://www.albion.com/netiquette/book/index.html> and/or

<http://webreference.com/roadmap/map07.html> and/or

<http://www.isoc.org/internet/conduct>

Q: Does anybody here know C?

A: Yes.

Q: Can you teach me C/C++/VB/whatever?

A: Yes we probably could, if we had the time and inclination. Most of us have neither. There are plenty of excellent books available (books are big flat things with hardcopy inside), and in fact when you buy a compiler it tends to come with documentation on how to use the language for which it is the compiler. (See a previous question for an update to this answer.)

Q: Could you please email me a copy of Visual C++ or Visual Basic? Alternatively, could you burn a copy of the CD and Fedex it to me?

A: Software piracy is a crime. Sensible people don't get involved with it. Programmers do *not* advocate software piracy. And burning CDs tends to melt them. :-)

Q: Where can I download some demo Java or Visual Basic code?

A: Try <http://www.planet-source-code.com/PlanetSourceCode/>

Q: Will you do my homework for me?

A: No. Go away.

Q: Which is the best language?

A: The reason there are so many languages is that different people like, and are good at, different things, and think in different ways. Find a reasonably mainstream language that you like the look of, and learn it. You will find learning other languages simpler after that.

Q: Can you help me fix my email?

A: No. Go away. The right place for this kind of question is Yahoo! Chat Help room, or the lobby. People in there are quite accustomed and happy to answer this kind of question.

Q: How do I do those cute smiley faces?

A: Type :) but if it is the only thing on the line, precede it with a space, or Yahoo will think you want to "emote". Once you have mastered :), try ;) :x :p and one or two other combinations. Experimentation is a wonderful thing.

Q: How do I delete the history in my browser?

A: If you didn't go to that kind of site, you wouldn't need to. Again, check the Yahoo! Chat Help room or the lobby.

Q: Can you recommend a good book on <language>?

A: For C, "The C Programming Language" by Kernighan and Ritchie (and see the list above too).

For C++, "The C++ Programming Language" by Stroustrup.

Update 10 November 1999

merrill196884 adds: Everyone peppers you with questions about a good c++ reference. My recommendation is "How to Program C++", Deitel & Deitel, Prentice Hall, ISBN 0-13-528910-6. It has a lot of similarities to many other treatments, but they really seem to have a knack for understanding what it was like not to know this stuff.

The book I REALLY liked was by the same authors called "Getting Started with MVC++6", same publisher, ISBN 0-13-016147-0. I think the target audience for it was fellas like me, who can and do pay for useful products and don't begrudge sending Bill Gates another fifty bucks. This books addresses what for me were the hardest parts of learning anything about programming as it is now done.

Thanks for that, merrill196884. Any other personal recommendations for books will be considered for inclusion on this page, except of course for Dummies books or anything by Herbert Schildt.

So what's wrong with Herbert Schildt then?

His explanations are simple, clear and, all too often, wrong. Check out this site to see what Clive Feather (a member of the ANSI committee) has to say about Schildt's [Annotated ANSI C Standard](#)

Java books - sorry, no recommendations. Watch this space.

For Visual Basic, check out the Books Online info that comes on the CD.

For MIX, "The Art of Computer Programming" by Knuth. This comes in three large volumes, and is highly mathematical in content, so be warned.

Q: I NEED HELP WITH MY PROGRAM, ANYONE CAN HELP?

A: The caps lock key on most keyboards is situated between the [shift] key and the [tab] key, over to the left somewhere. Please press it exactly once, and then never press it again.

Q: How do I determine the processor speed of my Pentium programmatically?

A: Okay, I'm cheating a bit. This isn't actually a FAQ. But it was asked once, and is fun to answer. Have a look [here](#) for Microsoft C code to do it (versions 4 onwards should compile it okay). In case you don't have MSVC, [here](#) is the binary.

Q: Why does everyone who comes in here expect us to be experts on every aspect of computing and know all the answers to all their questions, and be nice enough to help them even though their attitude is extremely pushy and rude?

A: Not everyone does this. But it's true. If you are looking for help, don't be rude or pushy. The more polite (up to a certain, nauseating point) you are, the more likely it is that people will help you.

Q: I've heard that you can't do graphics programming in C. Why not? And doesn't that make C rather limited?

A: C is defined by the ISO/ANSI C Standard; the definition explains the core language, the bit of the language that must work on all computers that have ANSI C compilers. That standard makes no provision for graphics. It doesn't even make provision for clearing the screen! These things simply cannot be done portably (Java fans will say 'Oh yes it can' but they don't even know what day it is, so we won't pay too much attention to them, will we? :-). Now, that's not to say that you can't do these things

in C. Most compiler vendors supply extension libraries to allow you to do things like - well - clearing the screen!, and graphics, and stuff like that. But the trouble is, these differ from one compiler to the next. There have been attempts to standardise this non-standard behaviour - the most successful non-standard standard (or should that be standard non-standard?) is the POSIX standard.

Q: I need some ideas for programs. Has anyone got any suggestions for useful programs I could write?

A: Sure. [Click here](#) for a list of programs and other tasks that the world would thank you for completing.

Q: I have heard that there is a new chat client under development by some of the regulars here. Is this true?

A: Yes. Actually there are several people I know of who are developing clients right now. I'm writing one, in conjunction with Loopy, in C++ Builder for the Win32 platform. Others are also keeping themselves busy, and at least one Linux client is under way. I will be posting a link here shortly to a page with more information about the Builder client.

Update 26/10/1999 - I've (rather regretfully) had to stop developing Yahello! for a while, because of the book. Once the book is done and dusted, I'll get back to the client, I promise. In the meantime, Loopy may or may not be continuing work on it - I'm not sure how busy he is.

Update 07/07/2001 - Wow, has it been that long? Well, I guess I'd better either fix those bugs or shut up about releasing it.

Q: I heard MrKite1982 wrote a chat API! Where can I get KChat? A: You can get it from www.geocities.com/mrkite1982/kchat.zip. It doesn't work by the way, since Yahoo! changed their protocol, but he may well be producing a fix shortly.

Q: Is this the last question in the FAQ list?

A: Yes. For now.

General Tips On Finding Out What You Want

If you are looking for help, there are some important steps you can take to maximise your chance of getting what you need:

Firstly, do everything you can to research the problem yourself *before* asking. Often, it is quicker to find your answer in the documentation for your compiler, in supplied reference works such as Books Online, and so on. Have you done a Web search for the information you require? Have you found an appropriate newsgroup in Usenet and tried to post a question there (having first ascertained that it's on-topic for that

group and not covered in their FAQ)? (Remember: never post attachments, binaries or HTML to a text newsgroup, or they'll ignore your question and focus on your having broken the conventions.)

Secondly, don't just stride into the room and announce your question. Check the chat for a minute or two to get a feel for the current topic. This way, you can get some idea of who is likely to be able and willing to help you. Also, the conversation in the room ebbs and flows. Sometimes the regulars (who are usually the experts) are deep in conversation with each other, and are unlikely to even see your question, because they are focusing on each others' postings. Be patient, and wait until a natural lull occurs (this happens a lot), so that you can be sure of their attention. The other way of getting their attention is to flood the room, but this is futile because they'll just reach for their 'ignore' buttons and that's not going to help you one bit.

Thirdly, ask your question without first asking if you can ask! Make sure your question is more like "Could somebody please explain strcpy() to me?" rather than "Any computer experts in here?" - otherwise it can take twenty minutes to establish contact with someone who can answer your question. Bear in mind that computing is a big subject, and nobody can know everything about everything. If you ask "any [insert subject here] experts here?", the real experts are unlikely to respond, because real experts know perfectly well that nobody can know everything, and they'll look a bit silly if they say they're experts and you then happen to ask them a question in one of the few areas they haven't studied thoroughly. Conversely, honest people who know they aren't experts will also not respond to an "any experts" question - and yet it may well be that your question doesn't need an expert or a guru; it may be that someone with a passing acquaintance with your subject happens to know the answer to your question. In summary, simply ask the question you need to ask. For example, "Can someone here explain to me how to update an Access database in Visual Basic?", not just "anyone here know VB?"

Fourthly, avoid PMing (Private Messaging) people. Doing so frequently irritates them (unless of course you are already friends with them) and even if they do respond, you are restricting the advice you get to that of one person. Talking about the problem in open chat gives others who know their stuff the chance to make corrections to the advice you receive. And if you want to know why Private Messages irritate people, just try concentrating on six or seven PM windows and the main chat window at the same time and try to be coherent in all of them. It's not easy, especially as PM windows become active whenever a message is received in them, which makes it practically impossible to complete any sentence in any window! Those fortunate souls who are using early alphas of Yahello! don't mind you PMing them, for a very simple reason - they have PM filters. This means they can't even see your PMs, let alone respond to them. (Yahello! periodically replies automatically to PMs, warning you of this.)

Fifthly, have patience. I have often posted a necessarily long answer to a question, only to find that the asker has left the room just before I could get my posting onto the screen.

And sixthly, when people *do* post answers, make sure you read them! Yet another source of annoyance to add to a long list is the person who asks a question over and over again, completely ignoring the answers that are flying at him or her from all directions. I think this is because of a tendency to distrust

advice. There is a better way to handle this. If the advice seems reasonable, follow it up to the point where it doesn't seem reasonable any more. After a while, if you visit the chat room often enough, you will get a feel for who gives good advice and who doesn't.

Happy Chatting.

This page is not authorised or endorsed by Yahoo! in any way. In fact, I doubt if they even know about it.

You are visitor number **456855** - come back soon!

Binary Bookshelf



(Return to [Home](#) page)

Last updated: 11 September 2000

#include <disclaimer.h>

Please note: I've included Amazon (co.uk) links to books I think are worth buying. If there's no link, it doesn't necessarily mean I don't think it's worth buying! But if there *is* a link, you can be reasonably sure it's a good book. :-)

In these days of Web-based learning, it's all too easy to forget that there are such things as "books". These are basically printouts, bound in either cardboard or heavy paper. Many people have asked me - well, some people have asked me - okay, somebody asked me once, "What books do you recommend for [subject]?" Hence this page.

This is a list of the books on my main "computing" bookshelf. It's not a complete list by any means, for the following reasons:

- a) I've omitted all documentation that comes with applications.
- b) There are a lot of books on the shelf, so this is an incremental process; I don't intend to do the whole list in one go.
- c) I've omitted books I possess but have not had a chance to open yet!
- d) There are some books that are so esoteric or archaic that they don't really bear mentioning in today's modern computing world. Books like "Programming Graphics in BASIC", with instructions on how to drive Commodore Pets and Atari 800s and so on. ;-)

I've attempted to provide at least the following information for each book:

- Title
- Author(s)
- Publisher
- Year

- ISBN

I have deliberately *not* included price information, because this varies so much from country to country and year to year.

Adding a full review of each book as I went along proved too daunting, so I've started again, this time with a simple list. Annotations may, or may not, appear in due course. I promise to keep the "Last Updated" field up to date.

Star Guide:

```
***** - Excellent
****   - Pretty darn good
***     - Definitely worth having
**      - Might be worth a look
*       - Take care
X       - Lemon
[blank] - I haven't got round to providing a rating yet :-)
```

Without further ado, then, let's get on with the list. Note that some reviews have been sketched in already.

I don't like sites where you have to click loads of links to get to the bit you're interested in. On the other hand, I don't like sites where you have to wait ages for a page to download, either! Since I have a lot of books, it makes sense to split the bookshelf by subject, right from the start, even though I've only really made inroads into one subject. So I'll do that. Here's the massively comprehensive Contents Page:

[C Programming Books](#)

Click the link to see reviews for:

- [The C Programming Language](#)
- [C Unleashed](#)
- [Software Solutions In C](#)
- [Programs and Data Structures In C](#)
- [Numerical Recipes In C](#)

Yep, that's it for now. Enjoy!

You are visitor number **456855** - come back soon!

Obfuscated C Programs

[Home](#)

[A Man, A Plan, A Canal, Suez!](#)

[My submission to comp.lang.c 1 April 2001](#)

[Ian Kelly's reply](#)

[Here's a pretty thing...](#)

You are visitor number **456855** - call again soon!

Main Page

From SarahWiki



Sarah Thompson

Sarah Thompson's Personal Wiki

Welcome to my personal Wiki pages. I've previously had a number of web sites spread around the place, so this site is a new attempt to try to bring everything together under one virtual roof.

In the past, I had separate personal and academic web sites. From now on, this site will do double-duty.

Academic visitors may be most interested in [My Papers](#).

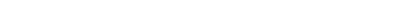
Friends, family and cyberstalkers might prefer my [Picture Gallery](#) and [my personal blog on LiveJournal](#). I also have a second, more traditional blog, [FindAtlantis](#).

Who am I?

[\[edit\]](#)

I am currently a PhD student in the CPRG group at the Computer Laboratory, University of Cambridge. My supervisor is [Alan Mycroft](#), and my second supervisor is [Martin Richards](#). My research interests are

I am currently working on a research project in conjunction with the US Air Force Research Laboratory Space Vehicles Directorate, looking at creating self-testing/self-repairing wiring harnesses (reconfigurable manifolds).

I'm also writing up. Here's the current progress on my thesis in terms of word count vs. the 60,000 word maximum: 

I intend to complete by the end of May 2006, at which point I'm beginning a postdoc in the [Intelligent Systems Division](#) at the NASA Ames Research Center in Mountain View, California.

Definitions [\[edit\]](#)

Note: This is unfinished, work-in-progress. I have now got MedaWiki to render my non-standard notation, but I still have a lot that needs to be written. Apologies in advance.

In my work, I've ended up coining a few terms, so in the spirit of all-things-Wiki, here are some definitions.

- [Transitional Logic](#) -- Transitional logics are extended, multi-value logics that encompass the concept of values that change over time.
- [Achronous Analysis](#) -- An *achronous analysis* (of an asynchronous circuit) is an analysis that disregards absolute time, and that adopts an [Independent Attribute Model](#) for its values. Our [Transitional Logics](#) are all achronous.

(All of these terms are coined in [Abstract Interpretation of Combinational Asynchronous Circuits](#) (Extended Journal Version))

Highlights [\[edit\]](#)

- [Thesis Progress](#) -- how far I've got, latest draft, etc.
- [My Schedule](#) -- an (incomplete) list of upcoming work-related engagements.
- [Conferences](#) -- links to upcoming conference web sites
- [To Do Lists](#) -- my to-do lists (probably of interest only to myself)
- [Hacking the MediaWiki LaTeX Support](#) -- How to extend MediaWiki's LaTeX support

(unofficial)

Retrieved from "http://findatlantis.com/wiki/index.php/Main_Page"

Views

- [Article](#)
- [Discussion](#)
- [Edit](#)
- [History](#)

Personal tools

- [Create an account or log in](#)

Navigation

- [Main Page](#)
- [Recent changes](#)
- [Random page](#)
- [Help](#)
- [Picture Gallery](#)
- [My Papers](#)
- [My Blog](#)
- [Useful Links](#)
- [How to Contact Me](#)

Search

Toolbox

- [What links here](#)
- [Related changes](#)
- [Upload file](#)
- [Special pages](#)
- [Printable version](#)



- This page was last modified 13:31, 14 December 2005.
- This page has been accessed 1,945 times.
- [About SarahWiki](#)
- [Disclaimers](#)

navigation:

[home](#)

[cisco security](#)

[hybrid security](#)

[yahoo hacking](#)

[c unleashed](#)

[mp3](#)

[links](#)

[ccie](#)

[me](#)

email loopy@loopy.org

[HOME](#) | [INTERNET ACCESS](#) | [BUSINESS](#) | [VOICE](#) | [WIRELESS](#) | [SOFTWARE & TOOLS](#) | [MEMBER CENTER](#) | [CART](#)

Have you KISSED your computer today?

GET A FREE SPYWARE SCAN NOW!

**KISS
Keep Internet
Security Simple**

EarthLink High Speed

Includes FREE AntiVirus,
Firewall, and Spyware
Blocker.

As low as
\$19.95/mo.
for **6 months***

*Offer available in
limited areas and
for a limited time.
[Click here](#) for details.



Get the Samsung® A560 FREE
with EarthLink Dial-Up
for **\$9.95/mo.** [Learn more >](#)

[See All Bundles >](#)

EarthLink
trueVoice

*Save money with
our Internet
phone service*

SIGN UP NOW →

EarthLink Dial-Up

Only **\$9.95/mo.**
for 6 months






[Learn More](#)

**EarthLink To Acquire
New Edge Networks**

**Get help with
common email errors**

**Parental Controls
Keep kids safe online**

Instant Support

-  Billing Issues
-  Local Dial-Up Numbers
-  Email Help
-  Password Help
-  Your Internet Connection

Today on **myEarthLink** [click here](#)

[HOME](#) | [ABOUT US](#) | [CONTACT US](#) | [PARTNER PROGRAMS](#) | [DOWNLOADS](#) | [CAREERS@EARTHLINK](#) | [SITE MAP](#)

© 2006 EarthLink, Inc. All rights reserved.

Members and visitors to the EarthLink Web site agree to abide by our [Policies and Agreements](#).

EarthLink [Privacy Policy](#)



comp.lang.c Frequently Asked Questions

This collection of hypertext pages is Copyright 1995-2005 by Steve Summit. Content from the book "C Programming FAQs: Frequently Asked Questions" (Addison-Wesley, 1995, ISBN 0-201-84519-9) is made available here by permission of the author and the publisher as a service to the community. It is intended to complement the use of the published text and is protected by international copyright laws. The on-line content may be accessed freely for personal use but may not be published or retransmitted without explicit permission.

This page is the top of an HTML version of the Usenet comp.lang.c Frequently Asked Questions (FAQ) list. An FAQ list is a collection of questions commonly asked on Usenet, together with presumably definitive answers, provided in an attempt to keep repeated questions on the newsgroup down to a low background drone so that discussion can move on to more interesting matters. Since they distill knowledge gleaned from many sources and answer questions which are demonstrably Frequent, FAQ lists serve as useful references outside of their originating Usenet newsgroups. This list is, I dare to claim, no exception, and the HTML version you're looking at now, as well as other versions referenced just below are intended to be useful to C programmers everywhere.

Several [other versions](#) of this FAQ list are available, including a [book-length version](#) published by [Addison-Wesley](#). (The book, though longer, also has a few more errors; I've prepared an [errata list](#).) See also question [20.40](#).

These pages are synchronized with the posted Usenet version and the Addison-Wesley book version. Since not all questions appear in all versions, the question numbers are not always contiguous.

[Note to web authors, catalogers, and bookmarkers: the URL `<http://www.c-faq.com/>` is the right way to link to these pages. All other URL's implementing this collection are subject to change.]

You can browse these pages in several ways. The table of contents below is of the list's major sections; these links lead to sub-lists of the questions for those sections. The "all questions" link leads to a list of all the questions; each question is (obviously) linked to its answer. The "section at a time" link arranges that all the questions in each major section are downloaded to your browser on one "page", rather than having each question/answer pair appear on its own page. In either case, the "read sequentially" link leads to the first question; you can then follow the "next" link at the bottom of each question's page to read through all of the questions and answers sequentially.

[Steve Summit](#)

scs@eskimo.com

[section at a time](#)

[1. Declarations and Initializations](#)

[2. Structures, Unions, and Enumerations](#)

[3. Expressions](#)

[4. Pointers](#)

[5. Null Pointers](#)

[6. Arrays and Pointers](#)

[7. Memory Allocation](#)

[8. Characters and Strings](#)

[9. Boolean Expressions and Variables](#)

[10. C Preprocessor](#)

[11. ANSI/ISO Standard C](#)

[12. Stdio](#)

[13. Library Functions](#)

[14. Floating Point](#)

[15. Variable-Length Argument Lists](#)

[16. Strange Problems](#)

[17. Style](#)

[18. Tools and Resources](#)

[19. System Dependencies](#)

[20. Miscellaneous](#)

[Glossary](#)

[Bibliography](#)

[Acknowledgements](#)

[All Questions](#)

[Read Sequentially](#)



[about this FAQ list](#) [about eskimo](#) [search](#) [feedback](#) [copyright](#)



Hosted by

Meta: [\[a\]](#)

The HTML version of this list is now considered the master version; the text version may be updated occasionally, but is now obsolete. Additionally, there are now accompanying [annotations](#) for some of the material. (Eventually, for all of the material.) Annotations are denoted by a bracketed letter a, or occasionally by links in the document text.

[Last modified April 16, 1999 by seebs.] [Revision: yes]

[Copyright 1995, 1996, 1997, 1998, 1999 Peter Seebach. All rights reserved, all wrongs reversed. Unauthorized duplication and distribution prohibited.]

The following FAQ has been a hobby of mine since 1995. If you enjoy it, please consider sending a token donation to help pay for my web page, and help me fund future, similar works. (Yes, I am thinking about doing others.) Email seebs@plethora.net for information.

Introduction: [\[a\]](#)

Certain topics never (well, hardly ever) come up on this newsgroup. They are stupid questions, to which the answers are immediately obvious, but they would be more fun to talk about than these arcane details of loop control.

This article, which is posted yearly, attempts to answer these questions definitively, succinctly, and in such a way as to discourage further discussion.

Table of Contents:

- [1. Declarations and Initializations](#)
- [2. Structures, Unions, and Enumerations](#)
- [3. Expressions](#)
- [4. Null Statements](#)
- [5. Arrays and Pointers](#)
- [6. Memory Allocation](#)
- [7. Characters and Strings](#)
- [8. Boolean Expressions and Variables](#)
- [9. C Preprocessor](#)
- [10. ANSI/ISO Standard C](#)
- [11. Stdio](#)
- [12. Library Functions](#)

[13. Floating Point](#)

[14. Variable-Length Argument Lists](#)

[15. Lint](#)

[16. Strange Problems](#)

[17. Style](#)

[18. System Dependencies](#)

[19. Miscellaneous](#)

Herewith, some infrequently-asked questions and their answers:

Section 1: Declarations and Initializations [\[a\]](#)

1.1: How do you decide which integer type to use?

Use `short` when you need to avoid values over 32,767, `int` when you want to store integers, `long` for long numbers (more than 6 digits), and `float` for numbers over 4 billion.

1.2: What should the 64-bit type on new, 64-bit machines be?

`int`.

1.3: If I write the code `int i, j;` can I assume that `(&i + 1) == &j`?

Only sometimes. It's not portable, because in EBCDIC, `i` and `j` are not adjacent. [\[a\]](#)

1.4: What's the best way to declare and define global variables?

In headers; this way, you can get link errors when you include the same header twice. Generally, you will have to define a variable everywhere you want to use it, and then declare it someplace so you know what it is. [\[a\]](#)

1.5: What does `extern` mean in a function declaration?

It refers to a variable which is not actually in your program. For instance,

```
main() {
    extern int bar;
    printf("%d\n", bar);
    return 0;
}
```

will compile without errors because bar is declared as being external. (It won't run, though, because you never assign bar a value.) [\[a\]](#)

1.6: I finally figured out the syntax for declaring pointers to functions, but now how do I initialize one?

With the assignment operator. You were perhaps expecting a screwdriver?

1.7: I've seen different methods used for calling through pointers to functions. What's the story?

In the old days, when Microsoft first invented C, the syntax for calling functions involved more parentheses; this was after their market research indicated that most C programmers would be coming from a Lisp environment. Later, when Kernighan took over the language design (right after AT&T bought Microsoft's language technology), he decided to eliminate the parentheses, but the old form is still allowed.

You do need the parentheses to call a function with more than one argument, for instance,

```
int (*foo)(char *, ...) = printf;
(*foo)("hello, %s\n", "world!");
```

needs the parens, but they would not be needed for

```
foo, "hello, world!\n";
```

(The ``*' just means to execute foo, just like the ``*' on the end of an executable filename in ``ls -F'.) [\[a\]](#)

1.8: What's the `auto` keyword good for?

Declaring vehicles.

1.9: I can't seem to define a linked list successfully. I tried

```
typedef struct {
    char *item;
    NODEPTR next;
} *NODEPTR;
```

but the compiler gave me error messages. Can't a structure in C contain a pointer to itself?

Not exactly; it can contain a pointer to another structure of the same type. Try:

```
typedef struct {
    char *item;
    double *next;
} NODEFAKE;

typedef struct {
    char *item;
    NODEFAKE *next;
} NODEPTR;
```

Make sure that `sizeof(NODEPTR) == sizeof(double)`.

This technique is called a ``backwards reference". [\[a\]](#)

1.10: How do I enter values using hexadecimal?

long ints can be entered using hexadecimal notation; for instance,

```
long int foo = 07;
```

sets foo to hex 7. [\[a\]](#)

1.11: How do I declare an array of N pointers to functions returning pointers to functions returning pointers to characters?

Well, first you need to know how to declare an array of N items of type T - that's

```
T foo[N];
```

Now you need to look at how to declare a pointer to function returning something, say, an object of type S. That's like this:

```
S (*bar)();
```

Now assume that S is ``pointer to function returning pointer to char". We get

```
(char *) (*)() (*bar)().
```

So, the whole thing turns out to be (with appropriate parentheses)

```
(( (char) (*) ) ( (*) ) ) ( ( (*) ( (foo) ) ) ) ( [ (N) ] ) ) ;
```

If your compiler complains, break this down into subexpressions.

To call it, just use

```
foo[i]();
```

This works because, in C, declaration reflects use, but it's one of those weird distorted mirrors. [\[a\]](#)

Section 2: Structures, Unions, and Enumerations [\[a\]](#)

2.1: What is the difference between an `enum` and a series of preprocessor `#defines`?

The `enum` doesn't require the preprocessor.

2.2: I heard that structures could be assigned to variables and passed to and from functions, but *K&R I* says not.

K&R I was wrong; they hadn't actually learned C very well before writing the book. Later, Ritchie got a job at Bell Labs, and worked closely with the authors of C, allowing the 2nd edition of the book to be much more accurate. (Kernighan already worked at Bell Labs, as a video game developer.) [\[a\]](#)

2.3: How does struct passing and returning work?

The structures are put into the low part of the VGA card's VRAM. They are then removed before the next video update. This is why struct passing was not supported for a long time; VGA cards were prohibitively expensive.

If you try to pass very large structures on the stack, you may see odd screen graphics. [\[a\]](#)

2.4: Why can't you compare structs?

Compare them to what? A summer's day?

2.5: How can I read/write structs from/to data files?

Loop with `putchar`. Be careful; if your machine uses signed chars by default, all of the sign bits in your structure elements will be reversed. [\[a\]](#)

2.6: How can I determine the byte offset of a field within a structure?

It's generally 4 times the number of members of the structure. It may be more or less on some machines. [\[a\]](#)

2.7: How can I access structure fields by name at run time?

foo."name" should work. You may need to overload the . operator, which, in turn, may overload your C compiler.

2.8: Why does sizeof report a larger size than I expect for a structure type, as if there was padding at the end?

Because there's padding at the end. **Duh.**

2.9: My compiler is leaving holes in structures, which is wasting space and preventing ``binary" I/O to external data files. Can I turn off the padding, or otherwise control the alignment of structs?

Sure. What you do to eliminate the padding in structures is use unions; for instance,

```
struct foo {
    char c;
    long l;
    char d;
    char e;
    char f;
};
```

may cause struct foo to be padded to 12 bytes, rather than the correct size of 8. Try

```
union foo {
    double _d;
    char c, d, e, f;
    long l;
};
```

which will be 8 bytes. (The double is for alignment.) [\[a\]](#)

2.10: Can I initialize unions?

Depends. They may go on strike when provoked. Luckily, if your program involves air traffic control, the

ISO standard guarantees that Ronald Reagan will fire any unions that go on strike, and replace them with structs, which should be close enough. [\[a\]](#)

2.11: How can I pass constant values to routines which accept struct arguments?

Try `foo((struct foo) 3)`. [\[a\]](#)

Section 3: Expressions [\[a\]](#)

3.1: Why doesn't the code `a[i] = i++;` work?

You didn't declare either `i` or `a`.

3.2: Under my compiler, the code

```
int i = 7;
printf("%d\n", i++ * i++);
```

prints 49. Regardless of the order of evaluation, shouldn't it print 56?

No. The only logical answer would be 81 - two postfix ++'s are automatically converted to prefix.

3.3: I've experimented with the code

```
int i = 2;
i = i++;
```

on several compilers. Some gave `i` the value 2, some gave 3, but one gave 4. I know the behavior is undefined, but how could it give 4?

Because `i` is 2, the loop is executed twice.

3.4: People keep saying the behavior is undefined, but I just tried it on an ANSI-conforming compiler, and got the results I expected.

They were probably wrong. Flame them mercilessly. Be sure before you do that your compiler is *really* ANSI conforming, though. If it turns out you were wrong, they get a legal claim on your first-born.

3.5: Can I use explicit parentheses to force the order of evaluation I want? Even if I don't, doesn't precedence dictate it?

No. To force order of evaluation, you must threaten it. Take the comma operator hostage. Using it, you can force the other operators to do what you want. [\[a\]](#)

3.6: But what about the `&&`, `||`, and comma operators? I see code like `if((c = getchar()) == EOF || c == '\n') ...`

As noted, once you've captured the comma operator, the others become docile.

3.7: If I'm not using the value of the expression, should I use `i++` or `++i` to increment a variable?

`++i`. Only losers and idiots use `i++`. This is different if your native language would idiomatically use `++i` increment, but in English and related languages, you must use `++i`. Note that a modern program must use both, dependent on the current locale.

3.8: Why is `i = ++i` undefined?

Because it is unclear whether it is shorthand for `i = 42;` or `i = (char *) "forty two";`.

Given the ambiguity, the standards committee decided to leave it undefined.

Section 4: Null Statements [\[a\]](#)

4.1: What is this infamous null statement, anyway?

A null statement is an expression statement consisting solely of the terminating semicolon. The optional expression is dropped. It can be distinguished from any other statement by byte count or study of side-effects.

4.2: How do I `get` a null statement in my programs?

In ANSI C, there are six types of statements; labeled statements, compound statements, expression-statements, selection statements, iteration statements, and jump statements. All of them, except the jump and expression statements, are defined in terms of optional preceding text, and other statements. The jump statements are never null statements. An expression statement is considered to be "a null statement" if the optional expression part of it has been left out. A null statement can appear on its own, or (most frequently) as the statement body of an iteration statement. These two null statements are equivalent, though neither of them is equivalent to any non-null statement. [\[*\]](#)

You may accidentally get a null statement by deleting the body of a non-null statement.

[*] Actually, they are functionally equivalent to a large set of non-null statements, namely, those with no side-effects. However, the FDA has yet to approve any such, as their lack of side effects is conjectured, and not clinically proven. This applies only to the ANSI standard, and not the ISO standard, as the FDA has no jurisdiction outside the U.S. [\[a\]](#)

4.3: Is there more than one null statement?

Sort of. You can use ```;"`, ```0;"`, or ```1;"` - they will all act like a null statement. Only the first is a ```true` null statement (all bits zero). They are basically equivalent. Note that `(void *) 0;` is a null statement of type pointer to void, for instance. [\[a\]](#)

4.4: But I thought `{ }` was a null statement!

No. `{ statement-list[opt] }` is a compound statement. An empty block is not the same as a null statement, however, although it can be used in many of the same places. It's really a null block. (You can convert it with a cast, but it's not directly compatible. For instance, you can't use a null block as one of the controlling statements of a for loop.)

4.5: I use the statement `#define NULLSTMT(F) (F) ;` to allow me to cast a null statement to an appropriate type.

This trick, though popular in some circles, does not buy much. The resulting code is invalid, and will not compile. This (in the author's opinion) outweighs any arguable type consistency. It may be more common in industrial code. If it becomes common practice, C++ will probably legalize it. [\[a\]](#)

4.6: I use the statement `#define NULLSTMT(F) (F) 0;` to allow me to cast a null statement to an appropriate type.

This trick will likely work, but think: what does it really buy you? Mostly, it will indicate to even the most casual observer that you are shakey on the concept of null statements, making it harder for them to check your code.

4.7: But wouldn't it be better to use `;` (rather than `0;`) in case the value of 0 changes, perhaps on a machine with nonzero no-op instructions?

No. The `'0'` of `'0 ;'` is not evaluated as an instruction, rather, it is just ignored. The only reason to use `'0 ;'` instead of `' ;'` is to help keep 1-heavy code properly balanced (in C, which uses binary representations for numbers, it is possible for code to become unbalanced; an unbalanced binary tree is a common source of poor performance).

4.8: Is a null statement a null pointer?

No. A null pointer is a pointer where all of the address bits are zero (no matter what the segment bits are), and can be obtained by typing `(char *) (int) 0`. A null statement is not a pointer to anything. They are not interchangeable, although you can combine them to get an effectively-null statement, such as `NULL ;`. This does not buy you anything. [\[a\]](#)

4.9: I'm still confused. I just can't understand all this null statement stuff.

Follow these two simple rules:

1. When you don't want to do anything in source code, don't write it.
2. If you need a null statement to round out an expression, use an unadorned `;` to provide it.
3. Send large donations, checks, and money orders to the author of the FAQ, or the moderator of the group, whichever you prefer. Then, cross the top question off the FAQ, answer the question at the bottom, and mail it to three people. Within two weeks, you will receive 729 answers to various questions! Do not break the chain; Emily Postnews broke the chain, and now no one listens to her. [\[a\]](#)

Section 5: Arrays and Pointers

5.1: I had the definition `char a[6]` in one source file, and in another I declared `extern char a[]`. Why did it work?

The declaration `extern char a[]` simply matches the actual definition. The type ```array-of-type-T"` is the same as ```array-of-type-T."` Go ahead and use `extern char a[]`. (For greater portability, use it in both files, not only in one of them.) [\[a\]](#)

5.2: But I heard that `char a[]` was different from `char a[6]`.

This is true. However, the declaration `a[]` is compatible with the definition `a[6]`.

5.3: So what is meant by the "equivalence of pointers and arrays" in C?

Very little.

5.4: Then why are array and pointer declarations interchangeable as function formal parameters?

Classism. We consider arrays "second class objects". They don't vote, and they get treated as pointers. Additionally, they're merely objects, not citizens. Marx wrote about this a lot. [\[a\]](#)

5.5: Why doesn't `sizeof` properly report the size of an array which is a parameter to a

function?

Part of the ANSI conspiracy to restrict people to passing pointers; this was undertaken after the first discovery that passing large arrays recursively could cause crashes. Since then, with the passing of MS-DOS, it has become a non-issue; since all serious machines have virtual memory, you can pass as much data as you want on the stack without detectable problems. [\[a\]](#)

5.6: Someone explained to me that arrays were really just constant pointers.

Cool. Someone I know says he saw Elvis in a local bar. [\[a\]](#)

5.7: Practically speaking, what is the difference between arrays and pointers?

About the difference between alcohol and marijuana; they have different characteristics, and that's not a problem if you don't mix them too carelessly.

5.8: I came across some ``joke" code containing the ``expression" 5["abcdef"]. How can this be legal C?

It was added to allow people to avoid the character constant 'f' which may not be available on some systems. (Actually, it's a side-effect of the equivalence of arrays and pointers.) [\[a\]](#)

5.9: How would I initialize an entire array from standard input?

You have to use a loop. For instance, the following code reads the numbers zero through 99 into the array a.

```
for (i = 0; i < 100; ++i)
    a[i] = (scanf, ("%d", i));
```

Make sure to include `<stdio.h>`, or this may not work. [\[a\]](#)

Section 6: Memory Allocation

6.1: Why doesn't this fragment work?

```
char *answer
printf("Type something:\n");
gets(answer);
printf("You typed \"%s\"\n", answer);
```


The semicolon after ``answer" is missing. [\[a\]](#)

6.2: I have a function that is supposed to return a string, but when it returns to its caller, the returned string is garbage.

You probably returned a pointer to a local array. That doesn't work. Try using a temporary file, instead. For instance:

```
char *getstr(void) {
    FILE *fp = tmpfile();
    fputs(gets(NULL), fp);
    return (char *) fp;
}
```

6.3: Why does some code carefully cast the values returned by malloc to the pointer type being allocated?

In interrupt-riddled code, it may be necessary to cast values to force the CPU to resolve pointer types. [\[a\]](#)

6.4: You can't use dynamically-allocated memory after you free it, can you?

Yes. However, what happens when you do is not clearly defined.

6.5: How does free() know how many bytes to free?

Interrupt 41h. On macs, amigas, and other ``big-endian" processors, that would be interrupt 14h; be wary of portability problems. [\[a\]](#)

6.6: So can I query the malloc package to find out how big an allocated block is?

Not exactly; because the objects are dynamically allocated, their size can change at run time, so this will not be reliable. If you restrict your allocation to allocating sizeof(void *) bytes at a time, you will find that you can use sizeof() to get the size of a block, in the obvious way.

6.7: I'm allocating structures which contain pointers to other dynamically-allocated objects. When I free a structure, do I have to free each subsidiary pointer first?

No. You just have to keep track of them somewhere else also.

6.8: Was Proust's masterwork, *A Remembrance of Things Past*, the basis for the C library's allocation scheme, based largely on contextual analysis?

The standard does not specify an allocation scheme; the famous author the allocation scheme is based on is implementation specified. Proust is a common choice, however.

6.9: I have a program which mallocs but then frees a lot of memory, but memory usage (as reported by ps) doesn't seem to go back down.

You're probably not freeing the memory completely. Try replacing `'free (foo) ;'` with

```
free(foo);
free(foo);
free(foo);
```

in case the first `free ()` frees the memory only partially. (Unix wizards may recognize the parallel with syncing three times before rebooting.)

Alternatively, `free(foo) + 4;` may free the remaining four bytes. (Before using this, make sure `realloc(foo, 0)` returned 4).

Section 7: Characters and Strings

7.1: How can I get the numeric (character set) value corresponding to a character, or vice versa?

The obvious way is to write a function to do the conversion. (Error checking has been omitted for brevity.)

```
int ctoi(char c) {
    static unsigned char *ary;
    /* initialize the array */
    if (!ary) {
        int i;

        ary = malloc( UCHAR_MAX + 2 );
        for (i = 0; i < UCHAR_MAX + 1; ++i) {
            ary[i] = i;
        }
        ary[UCHAR_MAX + 1] = '\\0';
    }
    if (c) {
        unsigned char *t;

        /* we have to skip the leading NUL */
```

```

        t = strchr(ary + 1, c);
        if (!t)
            return 0;
        return t - ary;
    } else {
        /* special case for NUL character */
        return 0;
    }
}

```

There are various clever tricks you can use to get around writing the function, but most are too complicated for beginners. [\[a\]](#)

Section 8: Boolean Expressions and Variables

8.1: What is the right type to use for boolean values in C? Why isn't it a standard type? Should `#defines` or `enums` be used for the true and false values?

`int (*)(int, char **)` makes a good boolean type. You can use `main` for true, and `exit` for false. On some compilers, you may need to cast `exit()` to an appropriate type.

8.2: Isn't `#defining TRUE` to be 1 dangerous, since any nonzero value is considered "true" in C? What if a built-in boolean or relational operator "returns" something other than 1?

Very good! For instance, one program I saw used

```
#define TRUE(x) ((x) & 0x100)
```

for compatability with a specific release of a FORTRAN compiler, which used 0 for `.FALSE.` and 256 for `.TRUE.` - this allowed them to change their code with every new release of the FORTRAN compiler, and kept them alert to changes. This has no relationship to the boolean or logical operators in C, which always return 0 or 1.

8.3: What is truth?

It is not a saffron-robed monk, pissing in the snow. [\[a\]](#)

Section 9: C Preprocessor

9.1: How can I use a preprocessor `#if` expression to tell if a machine is big-endian or

little-endian?

`#ifdef __BIG_ENDIAN` should work on all known machines; Borland defines it. [[a](#)]

9.2: I've got this tricky processing I want to do at compile time and I can't figure out a way to get cpp to do it.

Poor baby.

9.3: How can I list all of the pre-#defined identifiers?

`#define __ALL_CPP_IDS` - put this in a source file, and run it through your C preprocessor.

9.4: How can I write a cpp macro which takes a variable number of arguments?

Try something like this:

```
#define add(x) (x)
#define add(x, y) (x + y)
#pragma induction add
```

9.5: Shouldn't the following code:

```
#define ROSE 1
#define CHRYSANTHEMUM 2
#define RHODODENDRON 3
#define WATER_LILY 4
printf("%d\n", CHRYSATHNEMUM);
```

print ``2"?

You misspelled CHRYSANTHEMUM. Use abbreviations for long flower names in C code.

Section 10: ANSI C

10.1: What is the ``ANSI C Standard?"

A whiny bunch of lusers who haven't written as many books as Herbert Schildt. [[a](#)]

10.2: How can I get a copy of the Standard?

Use a copier.

10.3: Does anyone have a tool for converting old-style C programs to ANSI C, or vice versa, or for automatically generating prototypes?

A router helps, but your best bet is still the band saw. Quick, efficient, and powerful.

10.4: I'm trying to use the ANSI ``stringizing" preprocessing operator # to insert the value of a symbolic constant into a message, but it keeps stringizing the macro's name rather than its value.

This is because " 3 " is not a legal integral constant in C - it's a string constant.

10.5: I don't understand why I can't use const values in initializers and array dimensions, as in

```
const int n = 7;  
int a[n];
```

Because you're not using C++.

10.6: What's the difference between ``char const *p" and ``char * const p"?

One `` " character. There are some trivial differences having to do with the distinction between a pointer to a constant, and a constant pointer, but since you can cast either to a (char *) it hardly matters.

10.7: Can I declare main as void, to shut off these annoying ``main returns no value" messages? (I'm calling exit(), so main doesn't return.)

Certainly. You can also declare it as double. It may not compile, or it may crash, but who cares? No lousy bunch of whining lusers is going to tell *you* what to do.

10.8: Why does the ANSI Standard not guarantee more than six monospace characters of external identifier significance?

Because none of the members of the committee had names over six letters, or in which letters other than the first were capitalized. [\[a\]](#)

10.9: What is the difference between memcpy and memmove?

memmove moves memory, and memcpy copies it. memmove may not be supported on machines without internal robot arms. Do not use memmove while the machine is powered up - you can destroy your memory.

10.10: Why won't the Frobozz Magic C Compiler, which claims to be ANSI compliant, accept this code? I know that the code is ANSI, because gcc accepts it.

The Frobozz Magic Company lies through its teeth. Consider: does Flood Control Dam #3 actually control floods? Didn't think so. The wands are excellent for making useless via casts of Float, though. [\[a\]](#)

10.11: Why can't I perform arithmetic on a void * pointer?

You're too big and clumsy. When you try to push the numbers together, you lose your balance. Perhaps you should get some angels from the rave over on pin 3.

10.12: What are #pragmas and what are they good for?

They are useful ways to eliminate compiler features which are not helpful to your goals; contrast #utility, which introduces useful compiler features, and #absolutist, which introduces those compiler features believed to be right. #relativist is supported by some compilers.

10.13: What does ``#pragma once" mean? I found it in some header files.

It means that your program will only run once; it's used to create ``crippled demos". [\[a\]](#)

10.14: People seem to make a point of distinguishing between implementation-defined, unspecified, and undefined behavior. What's the difference?

There isn't really one; people just enjoy flaming over nits. (To be technical, one has a hyphen, one has a space, and one is a single word.)

10.15: Is C an acronym?

Yes, it stands for ``C". It's another of those funky recursive acronyms.

Section 11: Stdio

11.1: What's wrong with this code:

```
char c;
while((c = getchar()) != EOF)...
```

You forgot to include space for the terminating NUL character, so the compiler can't find the end of `c` without overwriting other memory. In all probability, after the user types ```n<return>`", your code will look like

```
char cn
while((c = getchar()) != EOF)...
```

which won't compile.

Also, the ellipsis is not legal outside of function prototypes.

Try

```
char c[2]; /* include space for terminating NUL */
while ((c = getchar()) != EOF)
    ;
```

Note the use of the null statement to absorb the NUL. (See [Section 4](#).)

11.2: How can I print a ```%` character in a `printf` format string? I tried ```\%` but it didn't work.

Break the `'%'` sign out. i.e., `fprintf("foo " "% " "%d\n", foo);`

Alternatively, try

`sprintf("o" "/" "o")` to get a `"% "`.

The astute reader will notice that the latter example uses `sprintf`, and the former `fprintf` - this is because `sprintf()` works by characters, or strings, while `fprintf` (```fast printf`) works on files. [[a](#)]

11.3: Why doesn't the code `scanf("%d", i);` work?

You need to do this a bit differently; you should always check for the return from `scanf`, so try something like

```
i = 1;
if ((scanf, "%d", i) == 1)
```

to make sure you're reading correctly. (The assignment to `i` is so that, if `scanf` fails, you still have a legal value in `i`.)

11.4: Once I've used `freopen`, how can I get the original `stdout` (or `stdin`) back?

Call `main()` - the environment will be restored.

11.5: Why won't the code

```
while(!feof(infp)) {
    fgets(buf, MAXLINE, infp);
    fputs(buf, outfp);
}
```

work?

Because the end of file character is not detected on files named ```infp`". (Introverted-iNtuitive-Feeling-Perceptive, that is.) Also, it may be that the file was opened in text mode, where an end of file is read as a capital 'Z' on most machines, and `feof()` only looks for 'control Z'.

11.6: Why does everyone say not to use `gets()`?

Because they're trying to spoil your fun. `gets()` can make an otherwise droll and predictable program a lot more exciting.

11.7: Why does `errno` contain `ENOTTY` after a call to `printf`?

Because `stdout` is not a mammal. [\[a\]](#)

11.8: My program's prompts and intermediate output don't always show up on the screen, especially when I pipe the output through another program.

Have you turned your monitor on? If not, try hitting the ```PrtSc`" key, which will re-enable the electron guns.

11.9: How can I read one character at a time, without waiting for the `RETURN` key?

Ask the user to press enter after hitting a single character. [\[a\]](#)

11.10: People keep telling me that `getch()` is not standard, but my C compiler has it. Are they wrong?

They've been programming more than ten years. You haven't. Draw your own conclusions. That's right! They hadn't noticed it. No doubt their compilers have it too, and its behavior is identical everywhere else in the world, also. That would explain everything.

11.11: What does it matter that `getch()` isn't standard; it works, doesn't it?

Well, that would depend on the definition you're using for ``works".

11.12: I tried to port some code from a PC to a unix machine, and now it crashes immediately on startup. It isn't using `getch()` - it's reading directly from the keyboard. How can this be wrong?

The chances are you forgot to run the Unix linker; currently your code is linked to your PC hardware, and won't run anywhere else until it's linked to the new hardware. It may also need to be linked to someone with a brain.

11.13: How can I redirect `stdin` or `stdout` to a file from within a program?

```
execlv("main()" "> file", argv);
```

11.14: How can I recover the file name given an open file descriptor?

You will have to search the filesystem for files of the same size as the file you're reading, and compare information in them to find the file you're working on.

11.15: How do I open Flood Control Dam #3?

PUSH THE YELLOW BUTTON. TURN THE BOLT WITH THE WRENCH. (You must have the wrench, first.) [\[a\]](#)

Section 12: Library Subroutines

12.1: How can I convert numbers to strings (the opposite of `atoi`)? Is there an `itoa` function?

There's frequently an `itoa` function. Better yet, write your own; it'll be good practice. On some

implementations, `(char *) x`; will convert `x` to a string.

12.2: How can I get the current date or time of day in a C program?

```
fprintf(stderr, "please enter the current time and date..."); fflush(stderr); gets(stdin);
```

12.3: I need a random number generator.

Count errors in Herbert Schildt's C books. No one has detected any consistent pattern.

12.4: How can I get random integers in a certain range?

`random(n)` returns random numbers between `n` and `INT_MAX`.

12.5: Each time I run my program, I get the same sequence of numbers back from `rand()`.

This is so your results will be reproducible.

12.6: I need a random true/false value, so I'm taking `rand() % 2`, but it's just alternating 0, 1, 0, 1, 0...

That seems pretty random to me.

12.7: I need some code to do regular expression matching.

So do I. Let me know if you find some.

12.8: I read through the standard library, but there's no function to multiply two floating point numbers! Help!

Many C compilers offer an extension `__mult` to do just this. If your compiler doesn't, just hang tight; ANSI is likely to add it in the next revision.

For now, you can try

```
float mult(float m, n)
{
    float i = 0, j = 0;
    for (i = 0; i < n; ++i)
        j += m;
    return j;
}
```

}

which is fine as long as n is an integer.

12.9: How do I get past the snake?

Release the bird. You will have to drop the rod to get the bird in the cage. [\[a\]](#)

Section 13: Floating Point

13.1: My floating-point calculations are acting strangely and giving me different answers on different machines.

One of the machines is probably a Pentium. Scrap it and get a real machine. [\[a\]](#)

13.2: I'm trying to do some simple trig, and I am `#including <math.h>`, but I keep getting `undefined: _sin` compilation errors.

You forgot to define the `sin()` function. Most math texts should cover it in some detail. The easiest way to fix this should be:

```
double sin(double x) {
    return sqrt(1 - cos(x) * cos(x));
}
```

Warning: You *must not* declare this function as `extern`, or you will still have link problems.

13.3: Why doesn't C have an exponentiation operator?

It does. It looks like the multiplication operator, but you use it more. For instance, the C way of expressing " x squared" is `x*x`. " x cubed" would be `x*x*x`. Easy, isn't it?

13.4: How do I round numbers?

Multiply by 10. *Numerical Recipies in C* has a section on this, but there's reputedly a bug in their algorithm.

13.5: How do I test for IEEE NaN and other special values?

Using an electron microscope; the patterns are obvious once you know them.

13.6: I'm having trouble with a Turbo C program which crashes and says something like ``floating point formats not linked."

Turbo C is notoriously buggy. Get a compiler with floating point support.

13.7: What is so ``unsafe" about floating point?

Have you tried EXAMINE STICK? The stick has a sharp point, which punctures the raft, which no longer floats. Don't bring the stick into the raft with you.

13.8: Which is larger, ``2" or ``2.0"?

Numerical Recipes in C has a function for comparing two values to see which is greater. It may have a slight bug, where it would report incorrect results if the numbers differ by less than `FLOAT_MAX / INT_MAX`.

13.9: When I try to compile the following code, I get the error ``invalid use of floating point", what does this mean?

```
x=663608941*y%pow(2,32);
```

Remember that `*` is the indirection operator, as well as the multiplication operator; try putting spaces before and after the ```*`" so the compiler knows what you mean. Do the same with the `%` operator.

13.10: How can I copy a float into a string?

```
strcpy(string_var, float_var);
```

13.11: What are float variables, anyway?

The term ``float variable" is actually redundant; they are simply variables whose value can ``float" during execution. For instance:

```
float f, g = 3;
f = g; /* f ``floats' to g */
```

Easy!

Section 14: Variable-Length Argument Lists

14.1: How can I write a function that takes a variable number of arguments?

By declaring it with a variable number of arguments in the prototype. Use only the arguments declared at any given time.

14.2: How can I write a function that takes a format string and a variable number of arguments, like printf, and passes them to printf to do most of the work?

Redefine printf; the call to ``printf" inside yours will be resolved to the library version, because the C language doesn't allow recursion.

14.3: How can I discover how many arguments a function was actually called with?

_args is an external integer constant. It evaluates to three times the number of arguments the current function was called with. You can then look at _argdata[args] to get the address of the last arg, _argdata[args - 1] to get the size of the last arg, and _argdata[args - 2] to get the type of the last arg (as an int).

N.B. You **MUST** not refer to _args or _argdata between the ()'s of a function call; their value will be indeterminate. Use temporary storage.

14.4: Why doesn't

```
printf("hello, ", "world!", '\n');
```

work? I thought printf() took a variable number of arguments.

It will probably work some of the time; the number of arguments used by printf() may vary, as it is a variadic function.

Section 15: Lint**15.1: I just typed in this program, and it's acting strangely. Can you see anything wrong with it?**

Yes. There's too much lint in it. You should get a shop vac.

15.2: How can I shut off the ``warning: possible pointer alignment problem" message lint gives me for each call to malloc?

Don't run lint. Alternatively, provide a prototype of ``extern double * malloc()" to make the return from malloc() be more strongly aligned.

15.3: Where can I get an ANSI-compatible lint?

You may wish to check your spouse's navel occasionally, especially if your spouse works for a standards committee.

15.4: What does LINT stand for, anyway?

Lexeme Interpreter aNd Tester.

Section 16: Strange Problems

16.1: Something really strange happened when I ran this code!

No, it didn't. [\[a\]](#)

Section 17: Style

17.1: Here's a neat trick:

```
if(!strcmp(s1, s2))
```

Is this good style?

Not really; it's too similar to

```
if (!strncmp(s1, s2))
```

which invokes undefined behavior, so it might be confusing.

17.2: Here's an even neater trick:

```
volatile int True_Tester = 3;
#define TRUE (!True_Tester == !True_Tester)
#define FALSE ((!TRUE) != (!TRUE))

#define STR DISSIMILAR(x, y) (strcmp((x), (y)) != FALSE)
```

Isn't this cool?

Very impressive. The volatile int type assures that even seemingly redundant calculations involving

True_Tester will be performed, making sure that if the compiler's ANSI-compliant values of 0 for false and 1 for true vary during runtime, your program will detect it - and producing meaningful error messages if this change occurs during a boolean computation! Similarly, the STR_DISSIMILAR macro allows you to make quite clear what the real effects of strcmp() are.

However, you must be careful; if this code is included twice, it may produce errors, due to the multiple definitions of the ``True_Tester" variable. You may wish to declare it ``extern" (See [Question 1.5.](#))

17.3: What's the best style for code layout in C?

There are many systems of indentation advocated, but all of them have the same basic flaw; they will mislead the reader when the actual code logic does not follow the indentation. It is better to avoid indentation entirely, so the reader will not be misled.

17.4: Is goto a good thing or a bad thing?

Yes.

17.5: No, really, should I use goto statements in my code?

Any loop control construct can be written with gotos; similarly, any goto can be emulated by some loop control constructs and additional logic.

However, gotos are unclean. For instance, compare the following two code segments:

do {	foo();
foo();	if (bar())
if (bar())	goto SKIP;
break;	baz();
baz();	quux();
quux();	
} while (1 == 0);	SKIP:
buz();	buz();

Note how the loop control makes it quite clear that the statements inside it will be looped on as long as a condition is met, where the goto statement gives the impression that, if bar() returned a nonzero value, the statements baz() and quux() will be skipped.

17.6: What's this ``white space" I keep hearing about?

White space is a racist, segregational term. Implicitly, ``dark" or ``colored" space (i.e., the '_' character) is not good enough to separate tokens. More interestingly, the white space characters keep the other tokens

apart. They say it's for parsing, but there's ample evidence the goal of white space is to keep the other characters from ``taking over" the program. This is disguised by the description of C as ``white space insensitive" - a simple ploy for sympathy.

Section 18: System Dependencies

18.1: How can I read a single character from the keyboard without waiting for a newline?

Try 'stty eol ^M' to wait for a carriage return.

18.2: How can I find out if there are characters available for reading (and if so, how many)? Alternatively, how can I do a read that will not block if there are no characters available?

The buffer is normally at ``&main - 0100". Lower if you have more than 256 characters of typeahead.

18.3: How can I clear the screen? How can I print things in inverse video?

You can clear the screen by sending several formfeed characters. Additionally, some operating systems (like NetBSD) support a feature called ``whiteouts". [[a](#)]

18.4: How do I read the mouse?

Flip it over, put on your reading glasses.

18.5: How can my program discover the complete pathname to the executable file from which it was invoked?

By asking the user.

18.6: How can a process change an environment variable in its caller?

Only by force. Example code for Unix:

```
memmove(getppid() + getenv(NULL), getpid() + getenv(NULL),
        sizeof(envIRON));
```

18.7: How can I check whether a file exists? I want to query the user before overwriting existing files.

Time an attempt to truncate it to zero length; if it takes more than 20-30 ms, the file existed. The exact

values will depend on the system and the load; before testing, create several large files and time attempts to truncate them, for calibration.

18.8: How can I find out the size of a file, prior to reading it in?

There are two good ways:

1. Vernier calipers work well.
2. `mmap()` the file, then use `sizeof()`.

18.9: I tried to use the second strategy above. I used `mmap()` to map `stdin`, then tried to use `sizeof`. But, when my user is about to write something very long, `mmap()` fails! How can I prevent this?

`mmap()` only 1k at a time, then, when you've read the first kilobyte of your input, use

```
memmove(mapped_addr, mapped_addr + 1024, 1024);
```

to move in the next kilobyte of data.

18.10: How can I implement a delay, or time a user's response, with sub-second resolution?

Time writes of large files to disks; then you can wait for a certain amount of time by writing a certain amount of data, and time a response by how much you could write before the response arrived.

You may need to delete spare or unnecessary files to do this; for best results, use a loop like the following to eliminate temporary files:

```
d = opendir(s);
while (r = readdir(d)) {
    /* remove files matching tmpnam's return, which is
     * the temporary file name. */
    if (strcmp(d->d_name, tmpnam())) {
        remove(d->d_name);
    }
}
closedir(d);
```

18.11: How can I read in an object file and jump to routines in it?

`fopen` and `goto`.

18.12: How can I invoke an operating system command from within a program?

Ask the user to open a new shell. The best way to do this is

```
system("echo Please open a new shell now.");  
sprintf(cmdstring, "echo Enter the command '%s' in it.", cmd);  
system(cmdstring);
```

This will not work if you haven't declared cmdstring properly.

18.13: How can I ensure objects of my class are always created via ``new" rather than as locals or global/static objects?

Read the C++ FAQ.

Section 19: Miscellaneous

19.1: What can I safely assume about the initial values of variables which are not explicitly initialized? If global variables start out as ``zero," is that good enough for null pointers and floating-point zeroes?

They're always zero. [[a](#)]

19.2: How can I write data files which can be read on other machines with different word size, byte order, or floating point formats?

The traditional solution, pioneered by Microsoft, is to sell enough copies of your proprietary, slow, and limited software that everyone else supports your formats.

19.3: How can I insert or delete a line (or record) in the middle of a file?

Using fcntl(), lock the line or record in the file exclusively. Now, using another thread, read the file, at each byte, trying to write that byte back. Whenever you succeed, write that byte into another file. Then copy the new file over the old file, releasing the lock first.

19.4: How can I return several values from a function?

Code like this ought to work.

```
long int foo() {
    return 2L + 3; /* returns both values */
}
```

19.5: If I have a char * variable pointing to the name of a function as a string, how can I call that function?

Try the following:

```
eval(s);
```

Now all you need to do is write eval().

19.6: I seem to be missing the system header file <math.h>. Can someone send me a copy?

A lot of people claim that it is useless to send people headers from other machines. Not so! It can be informative, and can show you a lot about how blatantly stupid your request was, although it can't show you anything you wouldn't have known in an instant had you thought before posting.

Of course, we'd be happy to send you the header files...

----cut here----

```
/* math.h rev 7.0b (3/7/95) */
/* RCS log: #log% - can anyone tell me why this doesn't work?
 * - joe, 2/12/93
 */

/*
 * Copyright 1995 Berserkley Software Systems && Analytic Overdrive
 */

/* Parts of this header, including in particular the second and
 * third clauses of the first sentence of the fourth comment, were
 * based on copyright agreements from other sources, including
 * Xerox corporation.
 */

/*
 * math.h - math related macros and headers
 */
```

```

#ifndef _MATH_H
#define _MATH_H

/*
 * global data and definitions
 */

#ifdef __LITERAL_BIBLICAL_FUNDEMENTALISM
#define PI 3.0 /* 1 Kings 7:23 */
#endif

/*
 * common (portable) structures and functions
 */

/*
 * machine specific data
 */
#include <machine/math.h>

#endif /* _MATH_H // prevent multiple inclusion by using C++ comments*/
----cut here----

```

(Morons.) [[a](#)]

19.7: How can I call FORTRAN (C++, BASIC, Pascal, Ada, LISP, perl) functions from C? (And vice versa?)

You can do things like this:

```

DO CALL FORTRAN;          fortran();
__LINE__ BASIC;           basic();
sub pascal;               pascal();
((((lisp))))              lithp(); [*]
&perl_c;                  perl();

```

(You can't call Ada from C; it's unsafe.)

[*] C is pass by value, of course.

19.8: Does anyone know of a program for converting Pascal or FORTRAN (or LISP, Ada,

awk, ``Old" C, ...) to C?

Nope. However, the psychic friends network may have a lead. And they're not just a psychic, they're also a friend.

19.9: Is C++ a superset of C? Can I use a C++ compiler to compile C code?

C++ is a superset of something, we're not sure what. You can use a C++ compiler to compile C code, but the results may surprise you.

19.10: Where can I get copies of all these public-domain programs?

From <ftp://ftp.microsoft.com/> . Some of the code may look copyrighted; don't worry! The small companies that wrote it in the first place are not available for comment.

19.11: When will the next International Obfuscated C Code Contest (IOCCC) be held? How can I get a copy of the current and previous winning entries?

Next week. You missed the deadline. Tough, sucker.

19.12: Why don't C comments nest? How am I supposed to comment out code containing comments? Are comments legal inside quoted strings?

We believe it has something to do with captivity; C comments in the wild mate and nest normally. The San Diego Zoo believes it has managed to convince some C comments to nest, but it's hard to tell how much of that is really in the preprocessor, and how much of it is just bovine fecal matter.

19.13: How can I get the ASCII value corresponding to a character, or vice versa?

`chr$(foo)`; You would have known this if you had an integer basic in ROM.

19.14: How can I implement sets and/or arrays of bits?

With linked lists of bitfields. You may also wish to simply use a large set of constants and some clever use of the switch statement, i.e.:

```
enum { zero, one, two, three };

int bitwise_or(int n, int m) {
    switch (n) {
        case three:
```

```

        return three;
        break;
    case two:
        switch (m) {
            case one: case three: return three; break;
            default: return two; break;
        }
        break;
    case one:
        switch (m) {
            case two: case three: return three; break;
            default: return one; break;
        }
        break;
    default: case zero:
        switch (m) {
            case one: return one; break;
            case two: return two; break;
            case three: return three; break;
            case zero: default: return zero; break;
        }
        break;
}
}

```

Obviously, you'll need to increase this slightly to deal with more than two bits. This is much more readable than the alleged ``C" solution:

```
int bitwise_or(int n,int m){return n|m;}
```

Note how the lack of whitespace around operators obscures the functionality of the code. A clear argument for explicit statement of program logic over arcane operators, if I ever saw one.

The enum at the top isn't declared ``const int", because the resulting ``const poisoning" would require casts during all of the switch statements.

19.15: What is the most efficient way to count the number of bits which are set in a value?

Start a counter at zero and add one to it for each bit set. Some operating systems may provide a call to do this. For values over INT_MAX/2, start the counter at CHAR_BIT * sizeof(int) and subtract one for each bit not set.

19.16: How can I make this code more efficient?

Remove the comments; the no-op instructions generated by comments can slow your code down significantly. Similarly, shorten variable names. Most compilers, to implement pass by value, actually pass the names of variables in the stack; shorter variable names will reduce stack usage, and consequently execution time. If your compiler has good loop optimization, replace

```
foo( );
```

with

```
do {
    foo( );
} while (1 != 1);
```

which will likely receive more optimization.

19.17: Are pointers really faster than arrays? How much do function calls slow things down? Is ++i faster than i = i + 1?

Yes. About 10 ms per call. Only on machines which feature preincrement addressing.

19.18: This program crashes before it even runs! (When single-stepping with a debugger, it dies before the first statement in main.)

You probably declared main as `void main(void)`. It's also possible that the first statement in main is `abort()`; - by the as if rule, the compiler can abort at any time before then, too. Some compilers have bugs, and will produce buggy code for any module which includes the letters ```a"`, ```b"`, ```o"`, ```r"`, and ```t"` in that order before the first function declaration.

19.19: What do ``Segmentation violation" and ``Bus error" mean?

C programs are very territorial, and divide their code into segments. Violating these segments can trigger riots; similarly, pointers and integral constants are at the front of the bus, whereas arrays, strings, and other second-class data types are required to be at the rear of the bus. When they start forgetting their places, you can get a bus error. This is what the whole ```integral"` type thing is about - integrated bussing.

19.20: My program is crashing, apparently somewhere down inside malloc, but I can't see anything wrong with it.

Your vendor's library is buggy; complain loudly. Don't send them any example code; they just ask for that so they can steal your trade secrets.

19.21: Does anyone have a C compiler test suite I can use?

Yes. Unfortunately, it's probably broken. It's hard to tell.

19.22: Where can I get a YACC grammar for C?

You can't; YACC is written in C.

19.23: I need code to parse and evaluate expressions.

Ask any first year CS student. You may also wish to use your C compiler.

19.24: I need a sort of an ``approximate" strcmp routine, for comparing two strings for close, but not necessarily exact, equality.

Just try comparing pointers near the original pointers.

19.25: Will 2000 be a leap year?

That's a hard question. I'd suggest using an encyclopedia, or possibly a dictionary - look up ``yes".

19.26: How do you pronounce ``char"?

Like the first word of ``char *". The accent is generally on the first syllable.

19.27: Is this FAQ for real?

sigh I knew someone would ask that. (Editorial note: I recieved several corrections to minor factual errors when I first posted this.)

If you actually want to know something about C, get a good book (*K&R* is recommended), and check out the real FAQ, which is posted monthly in comp.lang.c, and available by anonymous ftp from rtfm.mit.edu.

I have a small web page of C stuff:

<http://www.plethora.net/~seebs/c>

and there is an excellent site at Lysator:

<http://www.lysator.liu.se/c>

For extra credit, see if you can figure out what all of the examples really do; most of them will compile, and all of them can be gotten to compile with sufficient #defines. (I think.) [[a](#)]

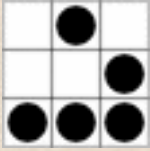
Credits:

The original comp.lang.c FAQ is maintained by Steve Summit, and many of the questions were stolen from it. Some of the idiotic misconceptions here are original, but many are from other sources. People have really said or advocated several of these; think about it. The Zork series may well be trademarked, but it was certainly an excellent game. Some of the mistakes may look similar to things warned against in *C Traps and Pitfalls*. And, of course, if Dennis Ritchie hadn't written C, these jokes would be much harder to understand.

Several people have contributed answers or questions, but I have lost the names in the mists of time.

Comments about this page can be sent to

seebs@plethora.net



The Jargon File, version 4.4.7

This page indexes all the WWW resources associated with the Jargon File and its print version, *The New Hacker's Dictionary*. It's as official as anything associated with the Jargon File gets.

On 23 October 2003, the Jargon File achieved the dubious honor of being cited in the SCO-vs.-IBM lawsuit. See the [FUD](#) entry for details.

- [Browse the Jargon File.](#)
- [What's new in the Jargon File.](#)
- [Other HTML-accessible versions of the Jargon File](#)
- [Search for Jargon terms](#)
- [Download the Jargon File in different forms.](#)
- [How to add or change entries in the Jargon File](#)
- [So, you want to quote the Jargon File?](#)
- [So, you want to mirror or re-package the Jargon File?](#)
- [View the Jargon File's change log](#)
- [Read this if you think The New Hacker's Dictionary is bogus](#)
- [Related resources](#)
- [The Book on the File: The New Hacker's Dictionary](#)
- [Order the book version from MIT Press](#)

This site has won awards; see the [trophy page](#).

MicroFirm

Become a
SNIPPETS sponsor

[Click here to learn more](#)

DIGITAL
MARS

["Free C/C++ Compilers"](#)

 Burton Systems Software

["High-Performance Version
Control"](#)

snippets.org

Free educational resources on the Internet

Friday, 03-Feb-2006, 09:45:58 GMT

Last modified: 18-Jan-2004, 00:22:45 GMT

327493 visitors since 15-Jul-2002

Mission: SNIPPETS.ORG is dedicated to providing free informational and educational resources on the World Wide Web. Currently, the two principle topics are programming and do-it-yourself (DIY) audio. The fields covered by SNIPPETS.ORG are generally technology and arts. As soon as enough free material is collected, new sections will be added related to photography and digital imaging.

The one common factor in everything you'll find on this site is that it's all free. Programming source code is free. Tools and utilities are free. And, of course, information is always free. While SNIPPETS.ORG provides many links to commercial sites, it is a not-for-profit operation - nothing here is for sale!

If you wish to contribute content, information, or entire web sites to SNIPPETS.ORG, please [contact me](#).

Support: This brings up the issue of support... Running a dedicated server such as SNIPPETS.ORG isn't prohibitively expensive, but the monetary costs are not trivial, either. Many web sites fit the same business model as commercial broadcasting. In contrast SNIPPETS.ORG uses the business model of public broadcasting. Companies which sponsor SNIPPETS.ORG will have their logo and link published discretely, but intrusive ads (e.g. pop-ups, animated GIF's, huge logos, ads everywhere) are prohibited. No other form of advertising is solicited or accepted. Only enough sponsorship to cover recurring operating expenses is actively solicited. For information about becoming a SNIPPETS.ORG sponsor, please [contact me](#).

Other maintenance and operational expenses are paid for by donations from users like yourself. If you find the information on this site valuable, I'd like to encourage you to make a donation to help keep this site operational and up-to-date. For your convenience, we use PayPal, which allows you to make secure donations via credit card or electronic check. Note that we keep no statistics on donations, so you needn't worry about your email address winding up on any third party mailing lists or being used in any sort of SNIPPETS.ORG fund drive. We hope you find the information useful. If so, your continued support will be appreciated!

Bob Stout

SNIPPETS.ORG sites:

SNIPPETS

The SNIPPETS archives project consists of application-specific discussion groups and associated repositories of free software and links. The areas currently supported include C, C++, Java, D, Python, Perl, Embedded, DOS, and Applications, although some of these are still a work in progress.

LDSG

The DIY Loudspeaker Designers Selection Guide (the LDSG), a primary net resource for amateur and professional loudspeaker designers. The LDSG only lists drivers and kits which have been reliably recommended. Additionally, it contains instructional information, design tips, and extensive links.

ALSR

John Whittaker's research into the dipole application of acoustic line sources and planar loudspeaker systems, and their interaction in real-world listening environments. Also, the design and construction of true ribbon line source drivers.

FILTERS

John Pomann's analog active filter designs, crossover applications, kits, and DIY resources.

Horns

John Sheerin's site devoted horn loudspeaker systems - their theory, several projects, and numerous links.

©1998-2003 by Bob Stout, all rights reserved

GRAPHICS BY

Image Magicians

powered by



Tested with the following web browsers: [Netscape 4.79](#), [Microsoft Internet Explorer 6.0](#), [Mozilla 1.5b](#).

Wotsit's Format

The programmer's resource

[Home Page](#)

[Full List](#)

[Graphics Files](#)

[3D Graphics Files](#)

[Movies/Animations](#)

[Archive Files](#)

[Binaries](#)

[Spreadsheet/
Database](#)

[Financials/Stocks](#)

[Font Files](#)

[Game Files](#)

[Text Files/
Documents](#)

[Internet Related](#)

[Sound and Music](#)

[Windows Files](#)

Wot it is...

Welcome to Wotsit's Format, the complete programmer's resource on the net. This site contains file format information on hundreds of different file types and all sorts of other useful programming information; algorithms, source code, specifications, etc.

The search engine on this site is the simplest way to get to any information that you are looking for. If you have any useful programming documents or information, please [Submit it](#) so that other programmers can make use of it. Suggestions for improving this site are also welcome.

Wot's useful...

Some documents on this site require the following viewer:
[Adobe Acrobat](#)

Thanks to...

Thanks to everyone who helps the site by and correcting my mistakes!

FAQ –

[Frequently asked questions.](#)

About Me –

[Information about the author of this site.](#)

Submit –

[Submit information to Wotsit's Format](#)

Feedback –

[Send feedback about Wotsit's Format.](#)

[GIS Formats](#)

[Comms Formats](#)

[Printer Formats](#)

[Hardware Formats](#)

[Miscellaneous](#)

Hosted By



Visitors Now: 46

Total Visitors*: 2835

Downloads*: 2244

in the last 21.5 hours

This site is © Paul Oliver 1996-2002. All rights reserved. Please read the [Disclaimer](#). If you have any comments or suggestions then please mail me [Here](#)