

THE
C
PROGRAMMING
LANGUAGE

Second Edition

Brian W. Kernighan • Dennis M. Ritchie

AT&T Bell Laboratories
Murray Hill, New Jersey



PRENTICE HALL, Englewood Cliffs, New Jersey 07632

**EXPERIAN EXHIBIT 1014
IPR2023-01406**

Library of Congress Cataloging-in-Publication Data

Kernighan, Brian W.

The C programming language.

Includes index.

1. C (Computer program language) I. Ritchie,
Dennis M. II. Title.
QA76.73.C15K47 1988 005.13'3 88-5934
ISBN 0-13-110370-9
ISBN 0-13-110362-8 (pbk.)

Copyright © 1988, 1978 by Bell Telephone Laboratories, Incorporated.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher. Printed in the United States of America. Published simultaneously in Canada.

UNIX is a registered trademark of AT&T.

This book was typeset (`pic|tbl|eqn|troff -ms`) in Times Roman and Courier by the authors, using an Autologic APS-5 phototypesetter and a DEC VAX 8550 running the 9th Edition of the UNIX® operating system.

Prentice Hall Software Series
Brian Kernighan, Advisor

Printed in the United States of America

10 9 8 7

ISBN 0-13-110362-8 {PBK}
ISBN 0-13-110370-9

Prentice-Hall International (UK) Limited, *London*
Prentice-Hall of Australia Pty. Limited, *Sydney*
Prentice-Hall Canada Inc., *Toronto*
Prentice-Hall Hispanoamericana, S.A., *Mexico*
Prentice-Hall of India Private Limited, *New Delhi*
Prentice-Hall of Japan, Inc., *Tokyo*
Simon & Schuster Asia Pte. Ltd., *Singapore*
Editora Prentice-Hall do Brasil, Ltda., *Rio de Janeiro*

```
#define VTAB '\xb'      /* ASCII vertical tab */
#define BELL '\x7'      /* ASCII bell character */
```

The complete set of escape sequences is

<code>\a</code>	alert (bell) character	<code>\\</code>	backslash
<code>\b</code>	backspace	<code>\?</code>	question mark
<code>\f</code>	formfeed	<code>\'</code>	single quote
<code>\n</code>	newline	<code>\"</code>	double quote
<code>\r</code>	carriage return	<code>\ooo</code>	octal number
<code>\t</code>	horizontal tab	<code>\xhh</code>	hexadecimal number
<code>\v</code>	vertical tab		

The character constant `'\0'` represents the character with value zero, the null character. `'\0'` is often written instead of `0` to emphasize the character nature of some expression, but the numeric value is just `0`.

A *constant expression* is an expression that involves only constants. Such expressions may be evaluated during compilation rather than run-time, and accordingly may be used in any place that a constant can occur, as in

```
#define MAXLINE 1000
char line[MAXLINE+1];
```

or

```
#define LEAP 1 /* in leap years */
int days[31+28+LEAP+31+30+31+30+31+31+30+31+30+31];
```

A *string constant*, or *string literal*, is a sequence of zero or more characters surrounded by double quotes, as in

```
"I am a string"
```

or

```
" " /* the empty string */
```

The quotes are not part of the string, but serve only to delimit it. The same escape sequences used in character constants apply in strings; `\"` represents the double-quote character. String constants can be concatenated at compile time:

```
"hello," " world"
```

is equivalent to

```
"hello, world"
```

This is useful for splitting long strings across several source lines.

Technically, a string constant is an array of characters. The internal representation of a string has a null character `'\0'` at the end, so the physical storage required is one more than the number of characters written between the quotes. This representation means that there is no limit to how long a string can be, but programs must scan a string completely to determine its length. The standard library function `strlen(s)` returns the length of its character

In a context where no value is wanted, just the incrementing effect, as in

```
if (c == '\n')
    nl++;
```

prefix and postfix are the same. But there are situations where one or the other is specifically called for. For instance, consider the function `squeeze(s,c)`, which removes all occurrences of the character `c` from the string `s`.

```
/* squeeze: delete all c from s */
void squeeze(char s[], int c)
{
    int i, j;

    for (i = j = 0; s[i] != '\0'; i++)
        if (s[i] != c)
            s[j++] = s[i];
    s[j] = '\0';
}
```

Each time a non-`c` occurs, it is copied into the current `j` position, and only then is `j` incremented to be ready for the next character. This is exactly equivalent to

```
if (s[i] != c) {
    s[j] = s[i];
    j++;
}
```

Another example of a similar construction comes from the `getline` function that we wrote in Chapter 1, where we can replace

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

by the more compact

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

As a third example, consider the standard function `strcat(s,t)`, which concatenates the string `t` to the end of the string `s`. `strcat` assumes that there is enough space in `s` to hold the combination. As we have written it, `strcat` returns no value; the standard library version returns a pointer to the resulting string.

```

/* 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 */
        ;
}

```

As each character is copied from *t* to *s*, the postfix ++ is applied to both *i* and *j* to make sure that they are in position for the next pass through the loop.

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

Exercise 2-5. 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.) □

2.9 Bitwise Operators

C provides six operators for bit manipulation; these may only be applied to integral operands, that is, `char`, `short`, `int`, and `long`, whether signed or unsigned.

<code>&</code>	bitwise AND
<code> </code>	bitwise inclusive OR
<code>^</code>	bitwise exclusive OR
<code><<</code>	left shift
<code>>></code>	right shift
<code>~</code>	one's complement (unary)

The bitwise AND operator `&` is often used to mask off some set of bits; for example,

```
n = n & 0177;
```

sets to zero all but the low-order 7 bits of *n*.

The bitwise OR operator `|` is used to turn bits on:

```
x = x | SET_ON;
```

sets to one in *x* the bits that are set to one in `SET_ON`.

The bitwise exclusive OR operator `^` sets a one in each bit position where its operands have different bits, and zero where they are the same.