

C Programming Language

Operators

Dr. Mostafa A. Elhosseini

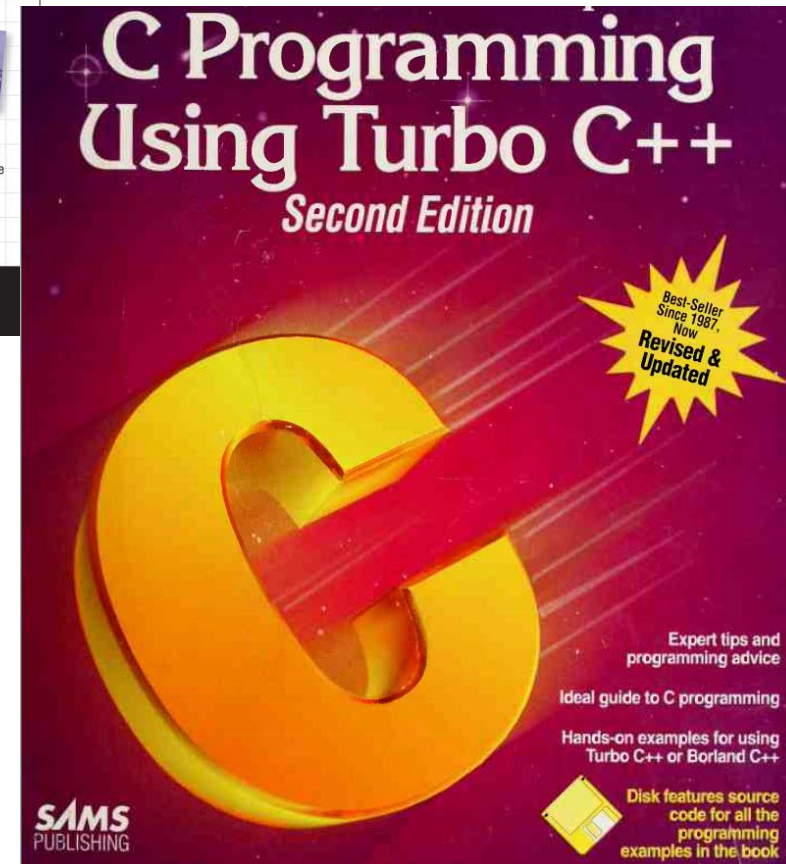
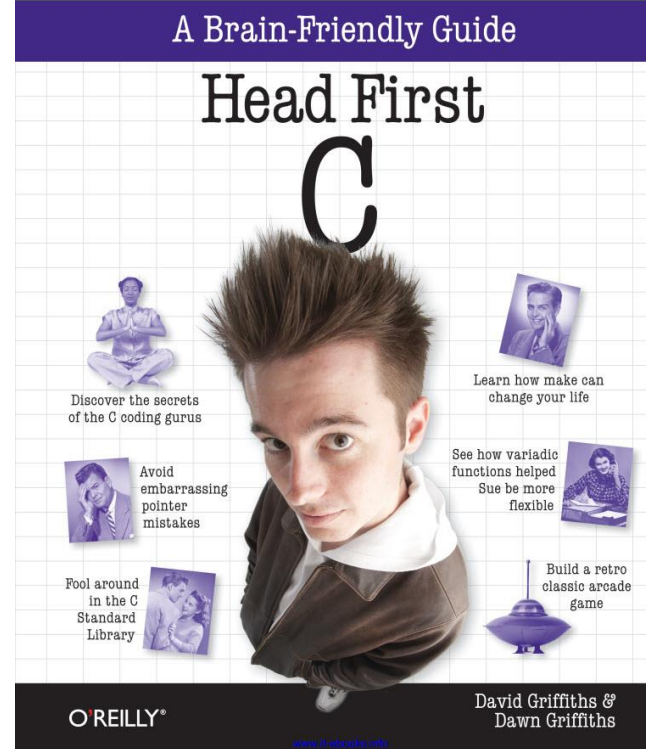
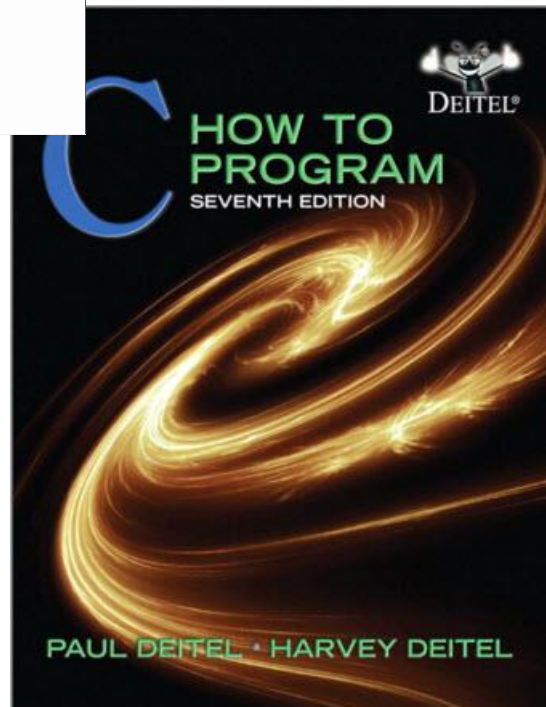
SECOND EDITION

THE



PROGRAMMING
LANGUAGE

BRIAN W. KERNIGHAN
DENNIS M. RITCHIE



Revise

✂ How to read from user

- `scanf( )`
 - & ampersand – address operator
- `getche( )` function
 - Get character with echo
- `getchar( )`
 - Get the character – it did not pass the character typed until the user hits [Return]
- `getch( )`
 - Get character without echo

✂ How to define constant

- Rvalue: Right value

Agenda

✧ Arithmetic in C

✧ Operators

- Arithmetic operators
- Relational operators
- Precedence
- Increment / Decrement
- Comment
- Secure programming

Operator

- ✧ **Operators** are words or symbols that cause a program **to do something** to variables
- ✧ For instance, the **arithmetic operators** (+) and (-) cause a program to add or subtract two numbers
- ✧ There are many different kinds of operators — mention the most common
 - Arithmetic
 - Relational
 - Increment/decrement
 - Assignment

Arithmetic operators

C operation	Arithmetic operator	Algebraic expression	C expression
Addition	+	$f + 7$	<code>f + 7</code>
Subtraction	-	$p - c$	<code>p - c</code>
Multiplication	*	bm	<code>b * m</code>
Division	/	x / y or $\frac{x}{y}$ or $x \div y$	<code>x / y</code>
Remainder	%	$r \bmod s$	<code>r % s</code>

An attempt to divide by zero is normally undefined on computer systems and generally results in a fatal error, i.e., an error that causes the program to terminate immediately without having successfully performed its job.

Precedence of arithmetic operators

Operator(s)	Operation(s)	Order of evaluation (precedence)
()	Parentheses	Evaluated first. If the parentheses are nested, the expression in the <i>innermost</i> pair is evaluated first. If there are several pairs of parentheses “on the same level” (i.e., not nested), they’re evaluated left to right.
*	Multiplication	Evaluated second. If there are several, they’re evaluated left to right.
/	Division	
%	Remainder	
+	Addition	Evaluated third. If there are several, they’re evaluated left to right.
-	Subtraction	
=	Assignment	Evaluated last.

Arithmetic operators

- ✧ It is also possible to **include expressions** involving arithmetic operators (and other kinds of operators as well) **directly into printf()** and other kinds of statements
- ✧ An **entire expression** can be used almost **anyplace a variable can be used**

Arithmetic Assignment Operator

✧ If you **compare a C program** with a program with the same purpose written in **another language**, you may well find that the **C source file is shorter** — **C has several operators that can compress** often-used programming statements

- | | | |
|-------------------------|---|---------------|
| ■ Total = total +number | ➔ | total+=number |
| ■ Total = total -number | ➔ | total-=number |
| ■ Total = total *number | ➔ | total*=number |
| ■ Total = total /number | ➔ | total/=number |
| ■ Total = total %number | ➔ | total%=number |

Relational Operator

- ✧ it will be used while dealing with **loops and decisions**
 - is a certain variable greater than 100? — if so stop looping
 - is the character the user just typed equal to a space? If so, increment the count of the number of words
 - determine whether a person's grade on an exam is greater than or equal to 60
 - In C, **true** is represented by the **integer 1**, and **false** values are represented by the integer **0**
 - Any nonzero integer is recognized by compiler as true value

Relational Operator

Algebraic equality or relational operator	C equality or relational operator	Example of C condition	Meaning of C condition
<i>Equality operators</i>			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y
<i>Relational operators</i>			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y

- A **syntax error** occurs if the two symbols in any of the operators ==, !=, >= and <= are separated by spaces
- Confusing the **equality operator ==** with the **assignment operator**.
- To avoid this confusion, the **equality operator** should be read “**double equals**” and the assignment operator should be read “**gets**” or “**is assigned the value of**”

Notes:

- Although it's allowed, there should be no more than one statement per line in a program
- A lengthy statement may be spread over several lines. If a statement must be split across lines, choose breaking points that make sense (such as after a comma in a comma-separated list).
- If a statement is split across two or more lines, indent all subsequent lines.
- It's not correct to split identifiers

Precedence of arithmetic operators

Operators	Associativity
()	left to right
* / %	left to right
+ -	left to right
< <= > >=	left to right
== !=	left to right
=	right to left

The Increment // decrement operator

- ✧ The **increment** operator **++** adds 1 to its operand,
- ✧ while the **decrement** operator **--** **subtracts 1**
- ✧ **++** and **--** may be used either as **prefix operators** (before the variable, as in **++n** or **postfix** (after the variable **n++**) — in both cases, the effect is to increment n
- ✧ But the expression **++n** increments n before its value is used
- ✧ While **n++** increments n after its value has been used.

Comments

- It is helpful to be able to **put comments** into the source code file that can **read by humans** but are **invisible to the compiler**
- A **comment begins** with the **two-character** symbol slash-asterisk (**/***) and ends with an asterisk-slash (***/**)
- Comments should be added anyplace there's **possibility of confusion**

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  main()
5  {
6      float years,days;           /* initialize variables*/
7      printf("Please type your age in years: "); /* print prompt*/
8      scanf("%f",&years);         /* get age in years from user*/
9      days = years*365 ;          /* calculate age in days*/
10     printf("You are %f days old",days); /* print result*/
11
12
13 }
14
```

Secure C Programming

- One such guideline is to avoid using printf with a single string argument
- If you need to **display a string that terminates with a newline**, use the **puts** function
- `printf( "Welcome to C!\n" );` $\rightarrow$ `puts( "Welcome to C!" );`
- We **did not include \n** in the preceding string because puts adds it automatically
- If you need to display a string without a terminating newline character, use **printf with two arguments**—a **"%s"** format control string and the string to display.
- `printf( "Welcome " );` $\rightarrow$ `printf( "%s", "Welcome " );`