

C Programming Language

Operators cont.

Dr. Mostafa A. Elhosseini

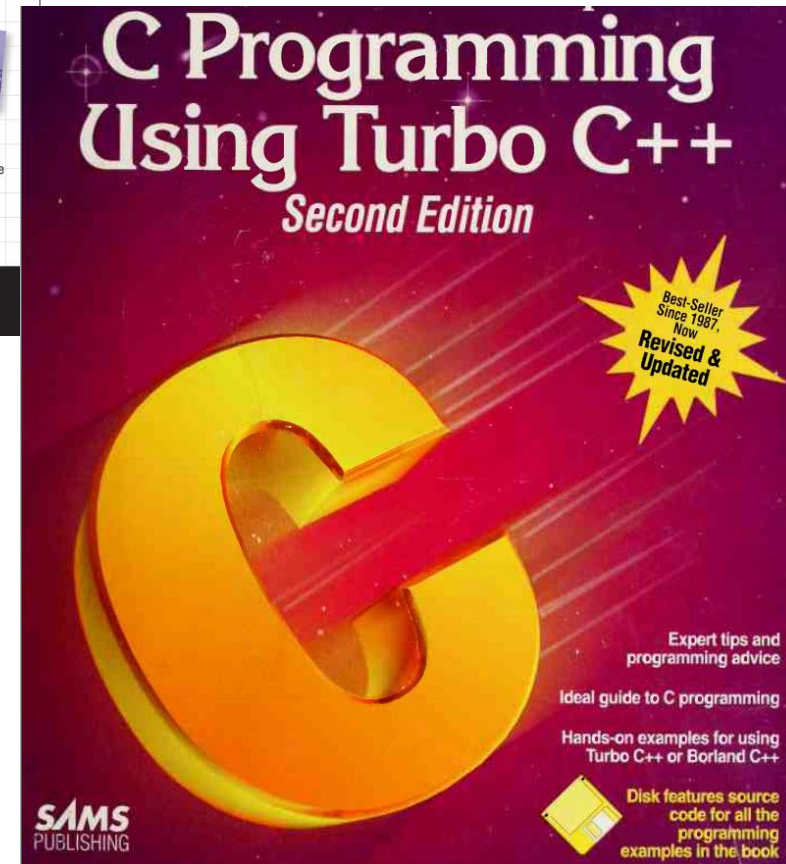
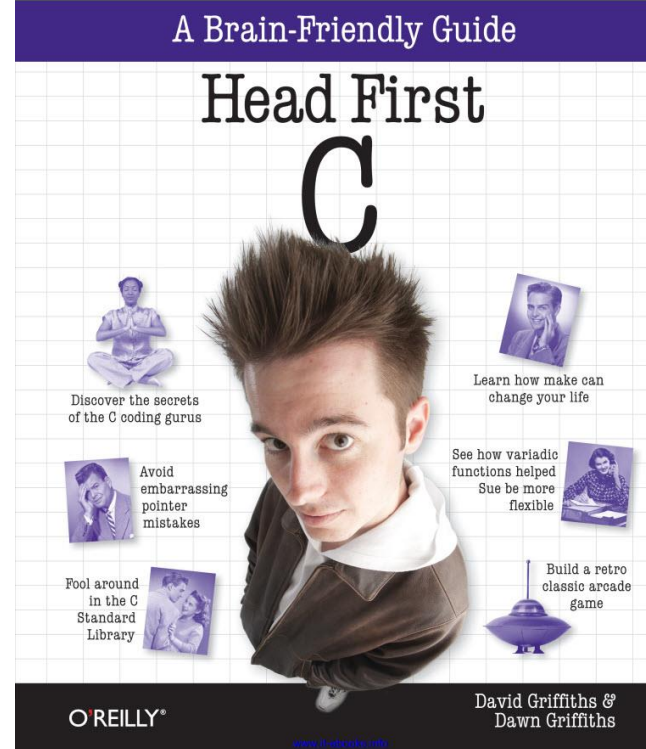
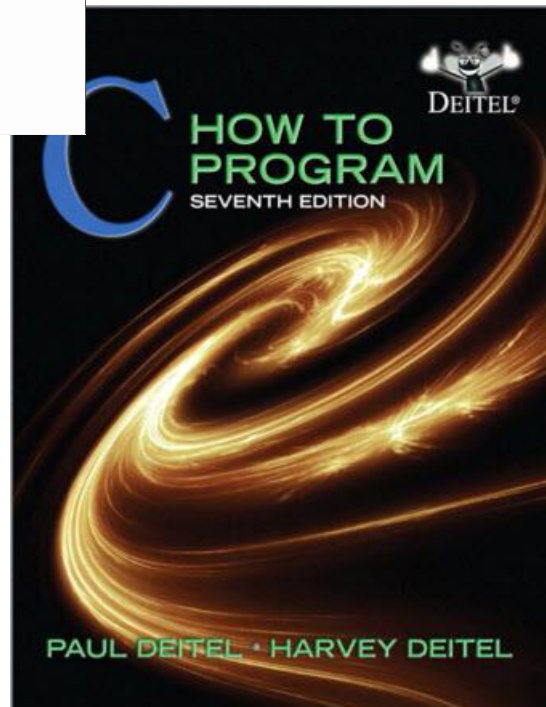
SECOND EDITION

THE



PROGRAMMING
LANGUAGE

BRIAN W. KERNIGHAN
DENNIS M. RITCHIE



Revise

✂ Arithmetic in C

✂ Operators

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

Agenda

✂ Notes

- Comments
- **compiler** is only concerned with the types of the operands
- Casting
- Logical operators
- Bitwise operators

Note

✧ To cause the compiler to ignore large parts of a C program it is best to enclose the parts by

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      printf("Hello world!\n");
7      #if 0
8          printf("Hi Mostafa");
9          puts("%s", "Play football");
10     #endif
11     return 0;
12 }
```

■ Note

- ✗ The compiler uses the types of the operands to determine how the calculation should be done

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main()
4  {
5      int i = 5, j = 4, k, m;
6      double f = 5, g = 4, h, z;
7      k = i/j;        // 1
8      z = i/j;        // 1.00
9      h = f/g;        // 1.25
10     m = f/g;        // 1
11     printf("\n k = %d", k);
12     printf("\n z = %f", z);
13     printf("\n h = %f", h);
14     printf("\n m = %d", m);
15 }
```

■ Note

- ⚡ If “+” sees an integer on its left and its right, integer addition is performed
- ⚡ With a real on the left and right, real addition is performed instead
 - This is also true for the other operators, “-”, “*”, “/”
- ⚡ The compiler is only concerned with the types of the operands
- ⚡ The cast operator temporarily changes the type of a variable
 - If either operand is a double, the other is automatically promoted

■ Casting

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main()
4  {
5      int i = 5, j = 4, k, m;
6      double f1, f2, f3, f4;
7      f1 = (double) i / j;
8      f2 = i / (double) j;
9      f3 = (double) i / (double) j;
10     f4 = (double) (i / j);
11     printf("\n f1 = %f", f1);
12     printf("\n f2 = %f", f2);
13     printf("\n f3 = %f", f3);
14     printf("\n f4 = %f", f4);
15
16 }
```

"E:\Courses\Taibah\Programming essential using C\Code\te

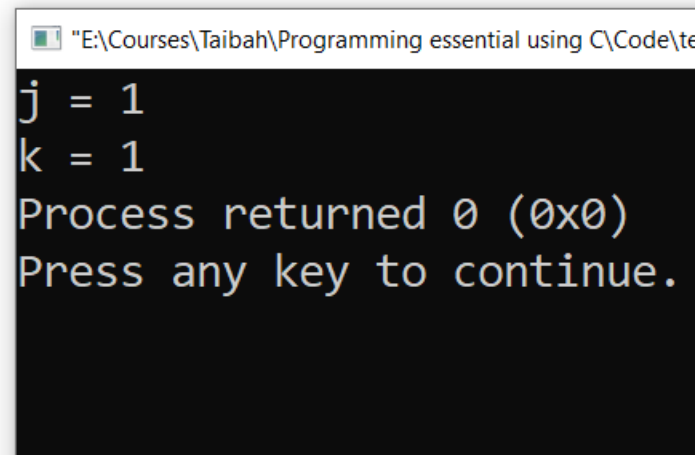
```
f1 = 1.250000
f2 = 1.250000
f3 = 1.250000
f4 = 1.000000
Process returned 0 (0x0)
Press any key to continue.
```

■ Note:

- ✧ There is no Boolean data type in C, integers are used instead
- ✧ The value of 0 (or 0.0) is false
- ✧ Any other value , 1, -1, 0.3 , -20.8, is true
 - Recall back once we study if statement
 - You may need to check the content of a register bit by bit, if a single bit is set, the value is identified as true
- ✧ Testing truth

■ Note:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main()
4  {
5      int i = 10, j, k;
6      j = i>5;
7      k = i<=1000;
8      printf("j = %d",j);
9      printf("\nk = %d",k);
10 }
```



The screenshot shows a terminal window with a black background and yellow text. The window title is "E:\Courses\Taibah\Programming essential using C\Code\te". The output of the program is displayed as follows:

```
j = 1
k = 1
Process returned 0 (0x0)
Press any key to continue.
```

Logical operators

- && AND
- || OR
- ! NOT

- ✧ As soon as C decided that a condition evaluates to true or false, no further evaluation is done
- ✧ Remember to use parenthesis with conditions otherwise your program may not mean what you think
 - ✧ Return back once conditional statement is given

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main()
4  {
5      int i = 10;
6      if (!i == 5)    // change to (!(i == 5))
7          printf("i is not equal to five\n");
8      else
9          printf("i is equal to five\n");
10 }
11
```

"E:\Courses\Taibah\Programming essential using C\Code\test01\bin\Debug\test01.exe"

i is equal to five

Bitwise operators

- ✧ Computers **represent all data** internally as **sequences of bits**.
 - Each **bit** can assume the **value 0** or the **value 1**.
 - On most systems, a sequence of **8 bits** forms a **byte**—the **typical storage unit** for a variable of **type char**.
 - **Other data types** are stored in **larger numbers of bytes**.
- ✧ The **bitwise operators** are used to **manipulate the bits of integral operands**, both **signed** and **unsigned**.
- ✧ **Unsigned integers** are **normally used** with the **bitwise operators**.

Bitwise operators

✧ C has the following bit operators which may only be applied to integer types:

- `&` bitwise and
- `|` bitwise or
- `^` bitwise xor
- `~` one's complement
- `>>` right shift
- `<<` left shift

✧ Kernighan and Ritchie needed to manipulate hardware registers in their PDP-11, they needed the proper tools

- Bit manipulation operators
- Kernighan and Ritchie do more bitwise manipulation than logical `&&`
 - Reserve `&` to bitwise instead of writing `&&`

Bitwise operators

OR |

x	y	z
0	0	0
0	1	1
1	0	1
1	1	1

And &

x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

XOR ^

x	y	z
0	0	0
0	1	1
1	0	1
1	1	0

one's complement

x	z
0	1
1	0

- The **bitwise AND** operator **&** is used to **mask off** some set of bits

AND $\Rightarrow$ the AND operation also clears bits of a binary number.

$\Downarrow$ the task of clearing a bit in a binary number is called MASKING

AND	<u>XXXX</u>	XXXX	unknown number
	0000	1111	mask
	0000	XXXX	Result
	<u>~~~~</u>		

$\Downarrow$ masked - cleared

- The **bitwise OR** operator **|** is used to **set “turn bits on”**

OR $\Rightarrow$ sets any bit of a binary number

OR $\Rightarrow$

X	X	X	X
0	0	0	0

X	X	X	X
1	1	1	1

 unknown

X	X	X	X
		1	1

Setting

the right most four bits set,
while the left most four bits remain
changed

- The **bitwise exclusive OR** operator $\wedge$ is used to **control flipping of bits**

Exclusive — OR (XOR)

✚ XOR instruction is useful if some bits of a register or memory location must be inverted. this instruction allows part of a number to be inverted or complemented

