CEng 230 Introduction to C Programming

Seyyit Alper SERT

Department of Computer Engineering

2017-2018 Fall

Web Pages

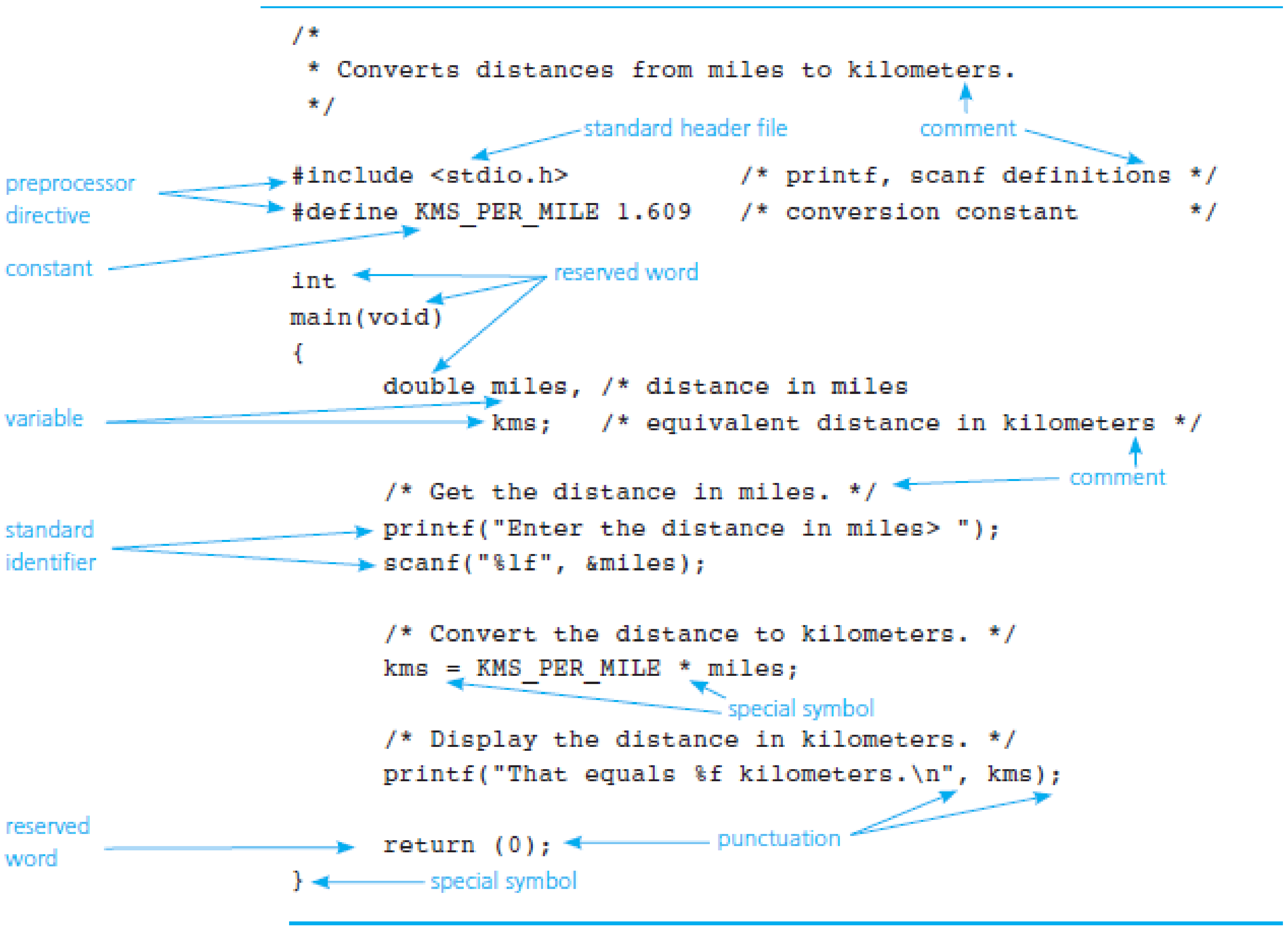
Official Course Page: ceng230.ceng.metu.edu.tr

Learning Management System (LMS): odtuclass.metu.edu.tr

Contact: alper.sert@ceng.metu.edu.tr

Valid variable names
Arithmetic expressions
Logic operators
Unary and binary operators
Assignment operators
Formating the output of numbers
Error types

Elements of a C program



```c
/*
 * Converts distances from miles to kilometers.
 */

#include <stdio.h>              /* printf, scanf definitions */
#define KMS_PER_MILE 1.609      /* conversion constant       */

int
main(void)
{
      double miles, /* distance in miles
             kms;   /* equivalent distance in kilometers */

      /* Get the distance in miles. */
      printf("Enter the distance in miles> ");
      scanf("%lf", &miles);

      /* Convert the distance to kilometers. */
      kms = KMS_PER_MILE * miles;

      /* Display the distance in kilometers. */
      printf("That equals %f kilometers.\n", kms);

      return (0);
}
```

Valid Identifiers

`letter_1, letter_2, inches, cent, CENT_PER_INCH, Hello, variable`

TABLE 2.2 Invalid Identifiers

Invalid Identifier	Reason Invalid
`1Letter`	begins with a letter
`double`	reserved word
`int`	reserved word
`TWO*FOUR`	character * not allowed
`joe's`	character ' not allowed

int1 and Int1 are not the same identifiers/variables

C has the following basic built-in data types.

- int
- float
- double
- char

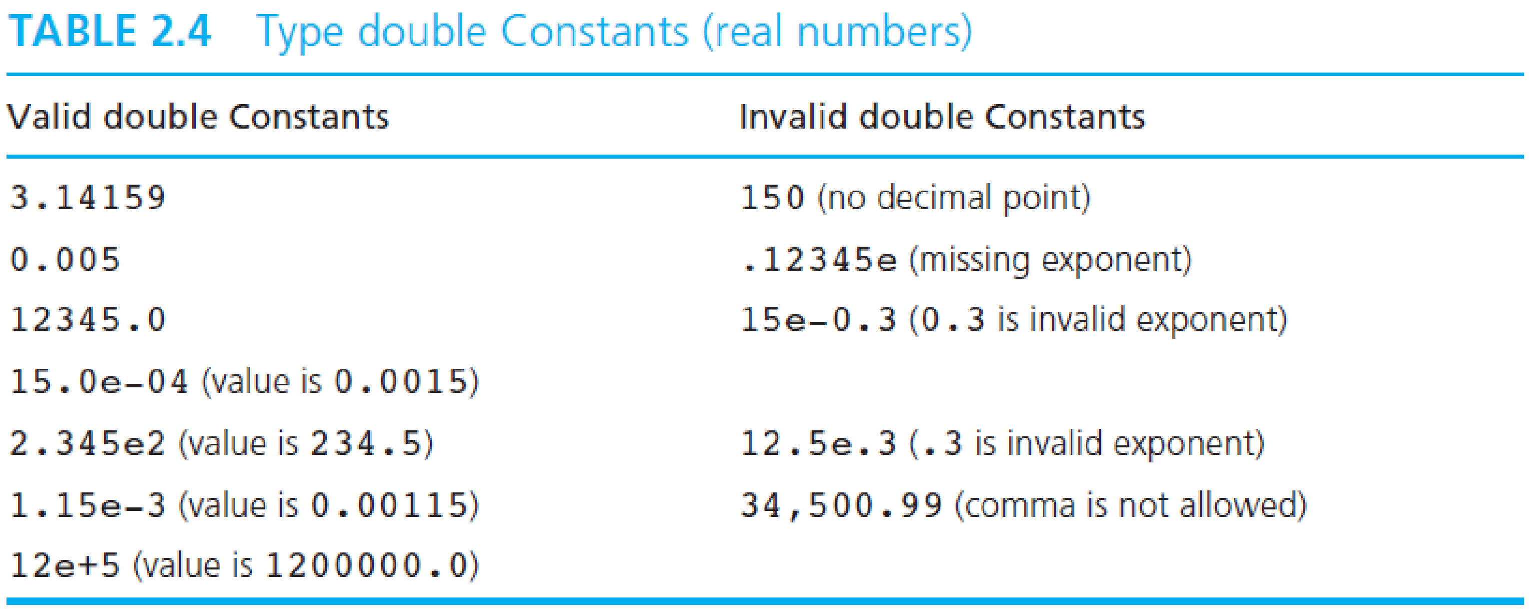


TABLE 2.4 Type double Constants (real numbers)

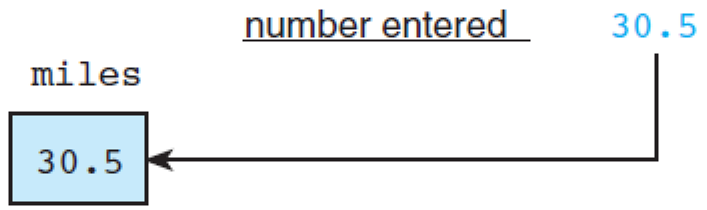
Valid double Constants	Invalid double Constants
`3.14159`	`150` (no decimal point)
`0.005`	`.12345e` (missing exponent)
`12345.0`	`15e-0.3` (`0.3` is invalid exponent)
`15.0e-04` (value is `0.0015`)	
`2.345e2` (value is `234.5`)	`12.5e.3` (`.3` is invalid exponent)
`1.15e-3` (value is `0.00115`)	`34,500.99` (comma is not allowed)
`12e+5` (value is `1200000.0`)	

TABLE 2.8 Placeholders in Format Strings

Placeholder	Variable Type	Function Use
`%c`	`char`	`printf/scanf`
`%d`	`int`	`printf/scanf`
`%f`	`double`	`printf`
`%lf`	`double`	`scanf`

FIGURE 2.6

Effect of `scanf("%lf", &miles);`



```
int first, second;
scanf("%d%d", &first, &second);
```

```
double miles;  /* distance in miles        */
scanf("%lf", &miles);
```

Type conversions (casting)

```
float a = 5.25;
int b = a;
/*Casting from float to int. The value of b here is 5*/
```

```
char c = ‘A’;
int x = c;
/*Casting from char to int.
The value of x here is 65: the ASCII code of ‘A’*/
```

```
int x=7, y=5 ;
float z;
z=x/y;
/* the value of z is 1.00 */
```

```
int x=7, y=5;
float z;
z = (float)x/(float)y;
/ the value of z is 1.4*/
```

Type conversions (casting)

```
printf( "Welcome : %d", (3/2) );
```

Output is : 1 and **fraction** part of the number is lost

```
int sum = 17, count = 5;
double mean;
mean = (double) sum / count;
printf("Value of mean : %f\n", mean );
```

```
Value of mean : 3.400000
```

```
int i = 17;
char c = 'c'; /* ascii value is 99 */
int sum;
sum = i + c;
printf("Value of sum : %d\n", sum );
```

```
Value of sum : 116
```

TABLE 2.7 ASCII Codes for Characters

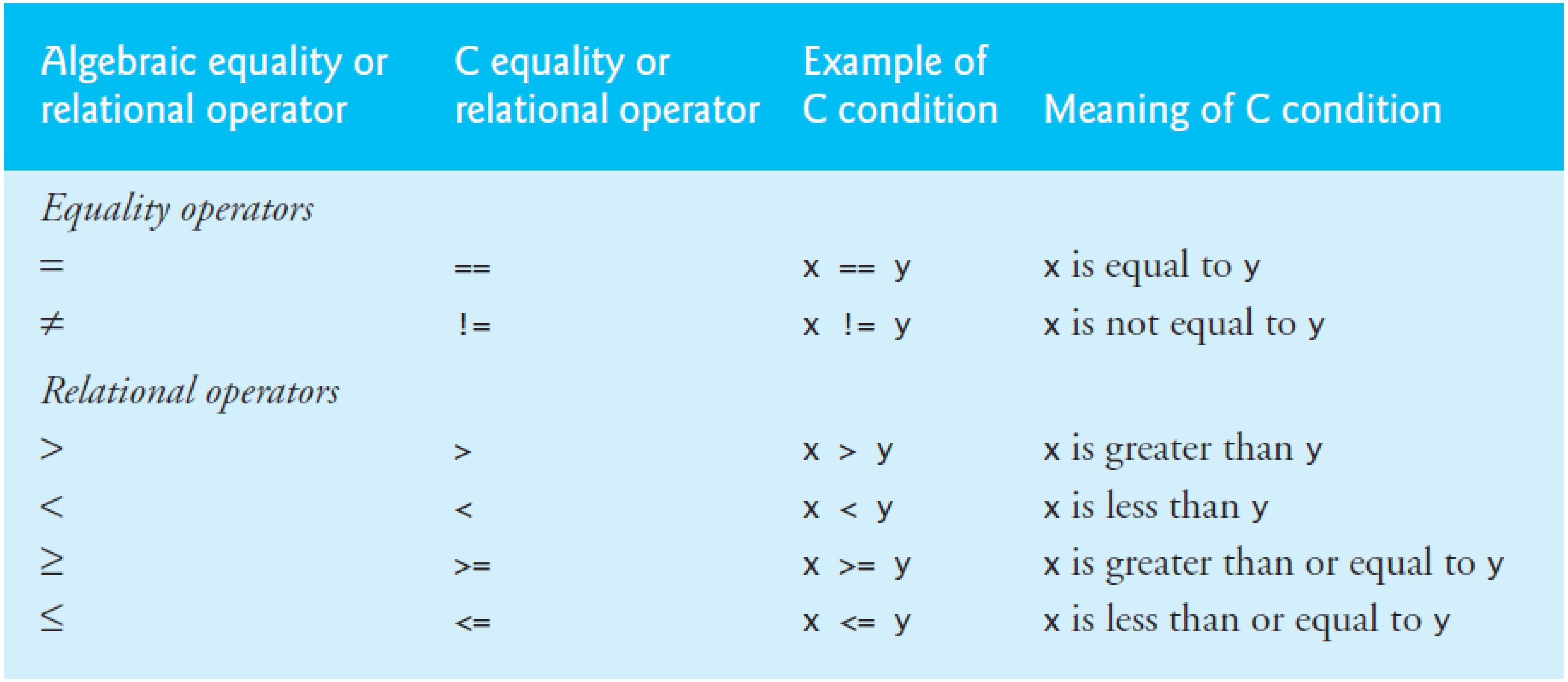
Character	ASCII Code
`' '`	32
`'*'`	42
`'A'`	65
`'B'`	66
`'Z'`	90
`'a'`	97
`'b'`	98
`'z'`	122
`'0'`	48
`'9'`	57

What is the result of printf("%d", ‘d’– ‘a’);

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

Operator	Sample expression	Explanation
`++`	`++a`	Increment `a` by 1, then use the new value of `a` in the expression in which `a` resides.
`++`	`a++`	Use the current value of `a` in the expression in which `a` resides, then increment `a` by 1.
`--`	`--b`	Decrement `b` by 1, then use the new value of `b` in the expression in which `b` resides.
`--`	`b--`	Use the current value of `b` in the expression in which `b` resides, then decrement `b` by 1.

Fig. 3.12 | Increment and decrement operators



Algebraic equality or relational operator	C equality or relational operator	Example of C condition	Meaning of C condition
Equality operators			
=	`==`	`x == y`	x is equal to y
≠	`!=`	`x != y`	x is not equal to y
Relational operators			
>	`>`	`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

Fig. 2.12 | Equality and relational operators.

! exclamation mark

= is assignment and == is an equality operator

```c
/* Fig. 2.13: fig02_13.c
   Using if statements, relational
   operators, and equality operators */
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
   int num1; /* first number to be read from user  */
   int num2; /* second number to be read from user */

   printf( "Enter two integers, and I will tell you\n" );
   printf( "the relationships they satisfy: " );

   scanf( "%d%d", &num1, &num2 ); /* read two integers */

   if ( num1 == num2 ) {
      printf( "%d is equal to %d\n", num1, num2 );
   } /* end if */

   if ( num1 != num2 ) {
      printf( "%d is not equal to %d\n", num1, num2 );
   } /* end if */

   if ( num1 < num2 ) {
      printf( "%d is less than %d\n", num1, num2 );
   } /* end if */


   if ( num1 > num2 ) {
      printf( "%d is greater than %d\n", num1, num2 );
   } /* end if */

   if ( num1 <= num2 ) {
      printf( "%d is less than or equal to %d\n", num1, num2 );
   } /* end if */

   if ( num1 >= num2 ) {
      printf( "%d is greater than or equal to %d\n", num1, num2 );
   } /* end if */

   return 0; /* indicate that program ended successfully */
} /* end function main */
```

```
Enter two integers, and I will tell you
the relationships they satisfy: 3 7
3 is not equal to 7
3 is less than 7
3 is less than or equal to 7
```

Assignment operators

=

+=

-=

*=

/=

%=

a+=10; is the same with
a=a + 10;

Logic operators

&&	**AND**
||	**OR**

```
if (  (a > = 25) || (b==4)  )

if (  (a > = 25) && (b!=4)  )
```

Operator(s)	Operation(s)	Order of evaluation (precedence)
( )	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost 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 Division Remainder	Evaluated second. If there are several, they're evaluated left to right.
+ -	Addition Subtraction	Evaluated last. If there are several, they're evaluated left to right.

Rules for Evaluating Expressions

a. *Parentheses rule:* All expressions in parentheses must be evaluated separately. Nested parenthesized expressions must be evaluated from the inside out, with the innermost expression evaluated first.

b. *Operator precedence rule:* Operators in the same expression are evaluated in the following order:

unary `+`, `-`	first
`*`, `/`, `%`	next
binary `+`, `-`	last

c. *Associativity rule:* Unary operators in the same subexpression and at the same precedence level (such as + and –) are evaluated right to left (*right associativity*). Binary operators in the same subexpression and at the same precedence level (such as + and –) are evaluated left to right (*left associativity*).

TABLE 2.9 Arithmetic Operators

Arithmetic Operator	Meaning	Examples

`+`	addition	`5 + 2 is 7`
		`5.0 + 2.0 is 7.0`

`-`	subtraction	`5 - 2 is 3`
		`5.0 - 2.0 is 3.0`

`*`	multiplication	`5 * 2 is 10`
		`5.0 * 2.0 is 10.0`

`/`	division	`5.0 / 2.0 is 2.5` $\boxed{\texttt{5 / 2 is 2}}$
`%`	remainder	`5 % 2 is 1`

TABLE 2.10 Results of Integer Division

3 / 15 = 0	18 / 3 = 6
15 / 3 = 5	16 / −3 = -5
16 / 3 = 5	0 / 4 = 0
17 / 3 = 5	4 / 0 is undefined

TABLE 2.11 Results of % Operation

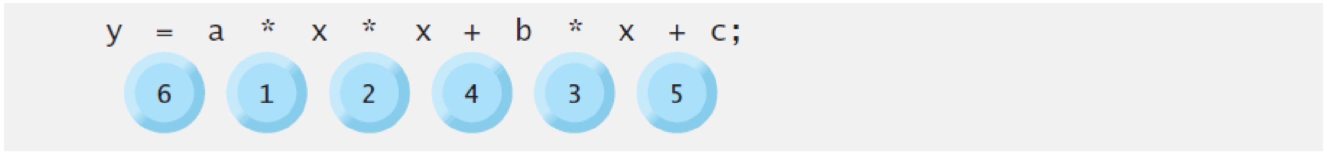
3 % 5 = 3	5 % 3 = 2
4 % 5 = 4	5 % 4 = 1
5 % 5 = 0	15 % 5 = 0
6 % 5 = 1	15 % 6 = 3
7 % 5 = 2	15 % −7 = 1
8 % 5 = 3	15 % 0 is undefined

TABLE 2.13 Mathematical Formulas as C Expressions

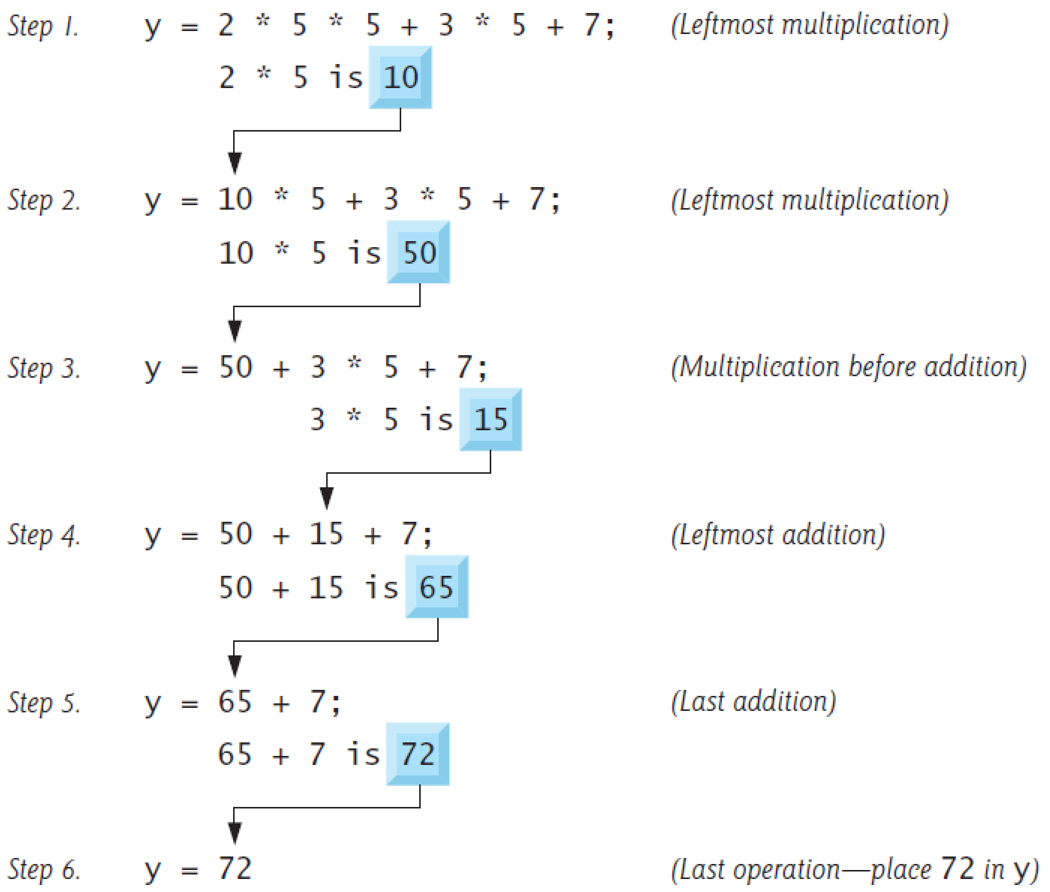
Mathematical Formula	C Expression
1. $b^2 - 4ac$	`b * b - 4 * a * c`
2. $a + b - c$	`a + b - c`
3. $\frac{a + b}{c + d}$	`(a + b) / (c + d)`
4. $\frac{1}{1 + x^2}$	`1 / (1 + x * x)`
5. $a \times -(b + c)$	`a * -(b + c)`

Evaluation of a Second-Degree Polynomial

To develop a better understanding of the rules of operator precedence, let's see how C evaluates a second-degree polynomial.

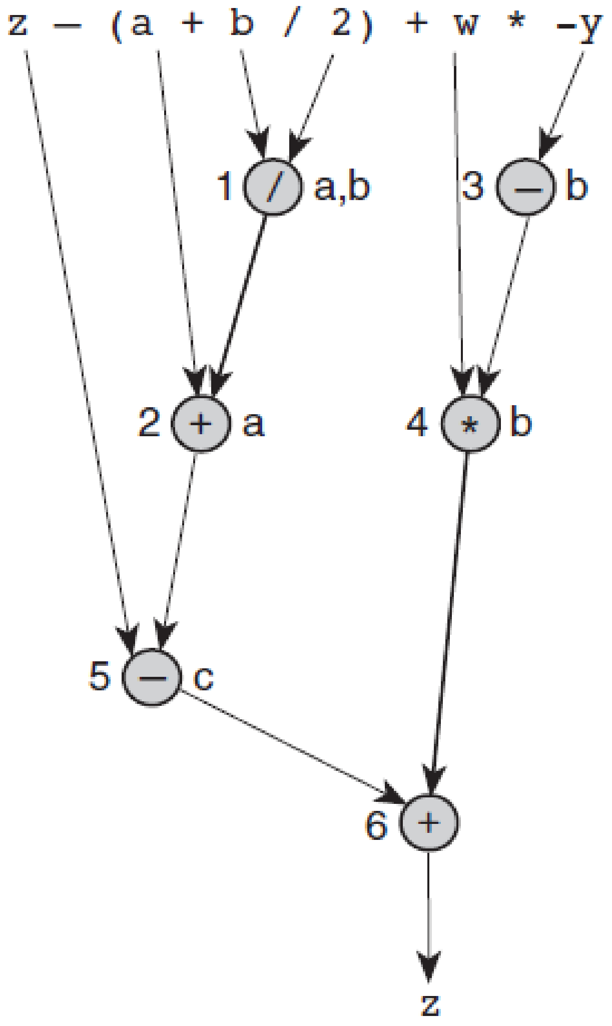


```
y = a * x * x + b * x + c;
```

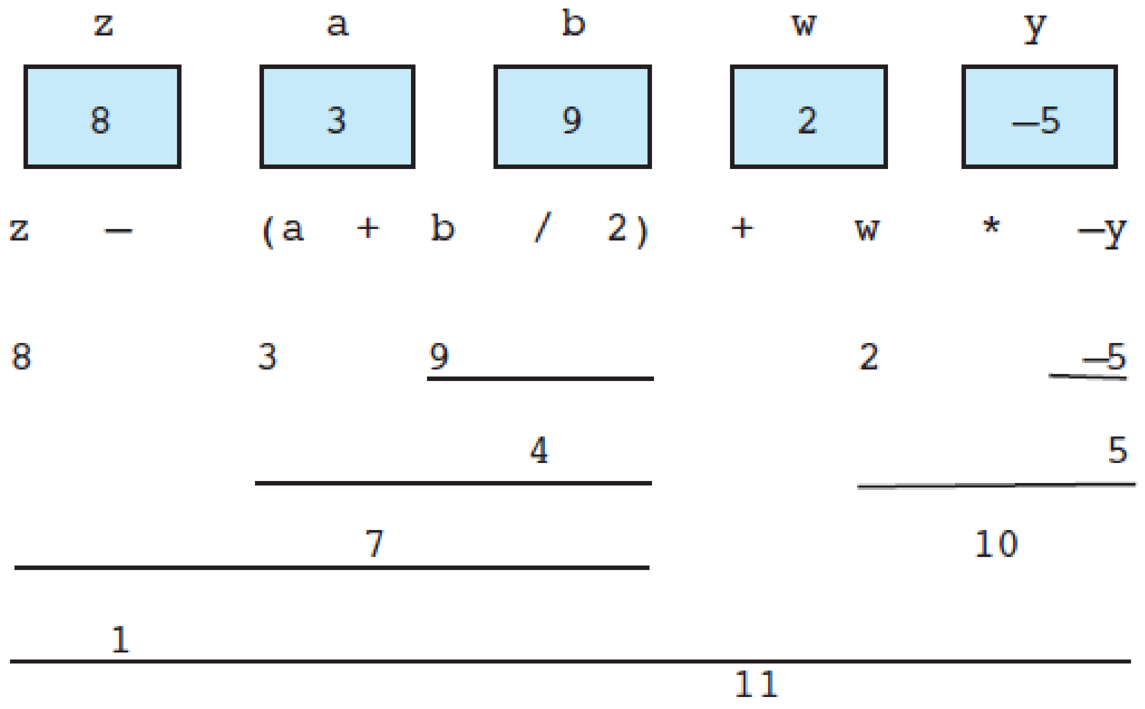


```
Step 1.   y = 2 * 5 * 5 + 3 * 5 + 7;      (Leftmost multiplication)
              2 * 5 is 10

Step 2.   y = 10 * 5 + 3 * 5 + 7;         (Leftmost multiplication)
              10 * 5 is 50

Step 3.   y = 50 + 3 * 5 + 7;             (Multiplication before addition)
                   3 * 5 is 15

Step 4.   y = 50 + 15 + 7;                (Leftmost addition)
              50 + 15 is 65

Step 5.   y = 65 + 7;                     (Last addition)
              65 + 7 is 72

Step 6.   y = 72                          (Last operation—place 72 in y)
```

Fig. 2.11 | Order in which a second-degree polynomial is evaluated.

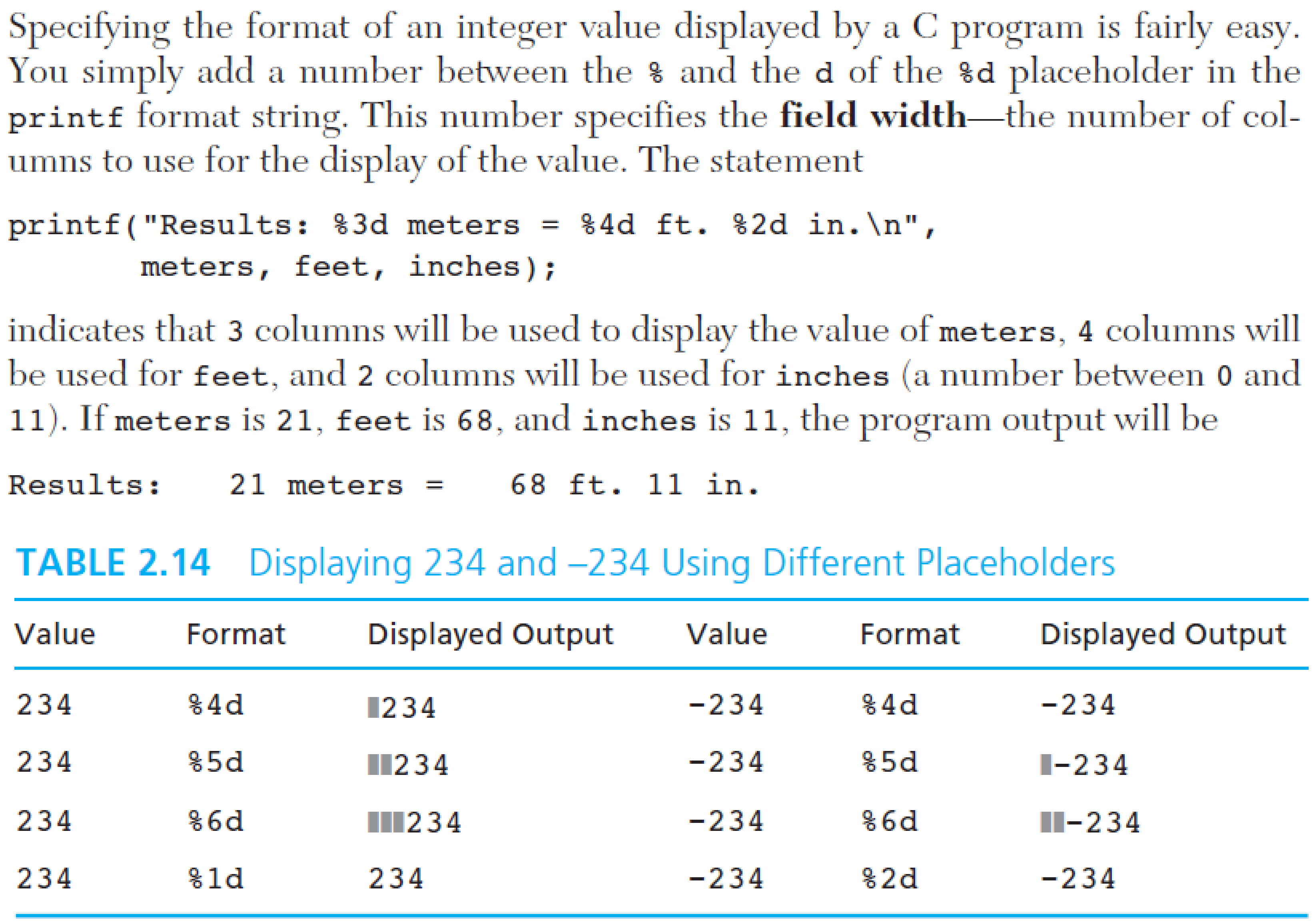


z – (a + b / 2) + w * -y
1 / a,b
3 – b
2 + a
4 * b
5 – c
6 +
z



z
a
b
w
y
8
3
9
2
–5
z – (a + b / 2) + w * –y
8
3
9
2
–5
4
5
7
10
1
11

Formating the output of integer values



Specifying the format of an integer value displayed by a C program is fairly easy. You simply add a number between the `%` and the `d` of the `%d` placeholder in the `printf` format string. This number specifies the **field width**—the number of columns to use for the display of the value. The statement

```
printf("Results: %3d meters = %4d ft. %2d in.\n",
       meters, feet, inches);
```

indicates that `3` columns will be used to display the value of `meters`, `4` columns will be used for `feet`, and `2` columns will be used for `inches` (a number between `0` and `11`). If `meters` is `21`, `feet` is `68`, and `inches` is `11`, the program output will be

```
Results:   21 meters =    68 ft. 11 in.
```

TABLE 2.14 Displaying 234 and –234 Using Different Placeholders

Value	Format	Displayed Output	Value	Format	Displayed Output
234	%4d	■234	-234	%4d	-234
234	%5d	■■234	-234	%5d	■-234
234	%6d	■■■234	-234	%6d	■■-234
234	%1d	234	-234	%2d	-234

Formating the output of double values

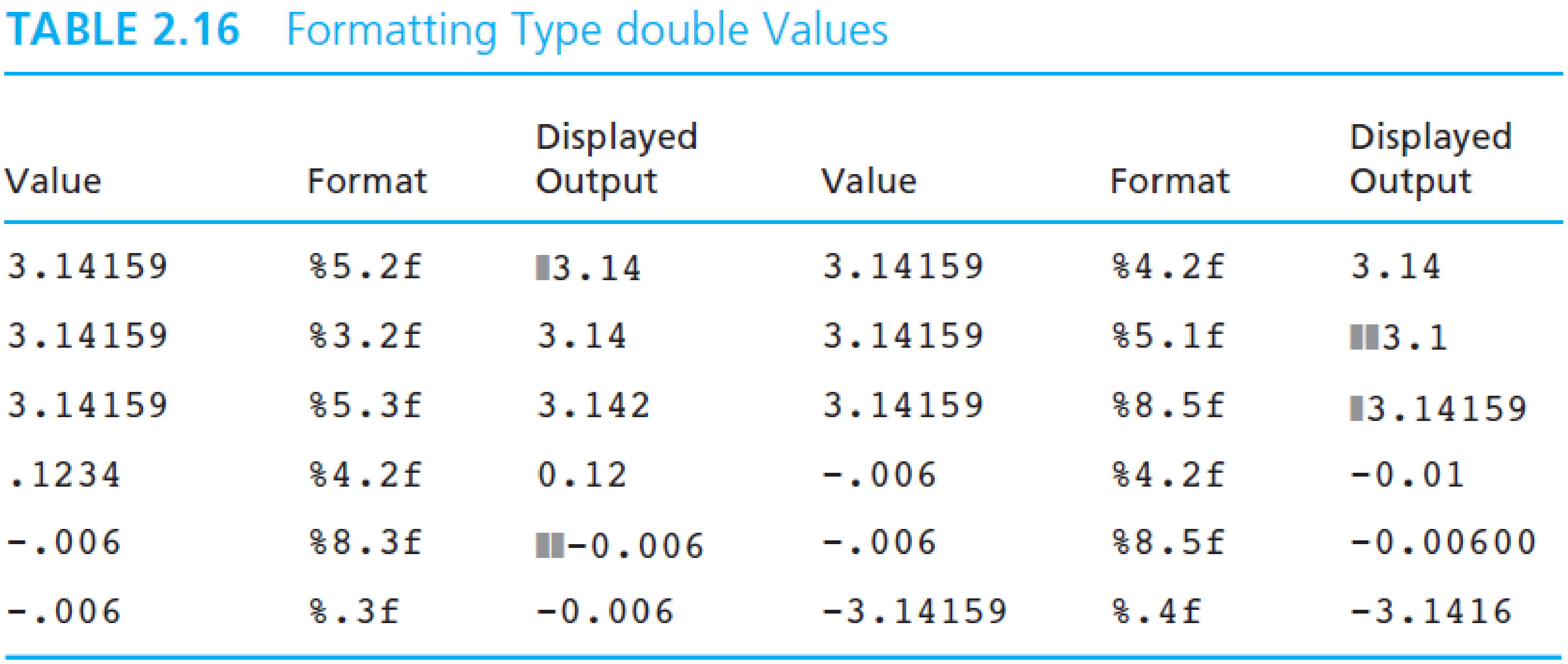


TABLE 2.16 Formatting Type double Values

Value	Format	Displayed Output	Value	Format	Displayed Output
`3.14159`	`%5.2f`	`▮3.14`	`3.14159`	`%4.2f`	`3.14`
`3.14159`	`%3.2f`	`3.14`	`3.14159`	`%5.1f`	`▮▮3.1`
`3.14159`	`%5.3f`	`3.142`	`3.14159`	`%8.5f`	`▮3.14159`
`.1234`	`%4.2f`	`0.12`	`-.006`	`%4.2f`	`-0.01`
`-.006`	`%8.3f`	`▮▮-0.006`	`-.006`	`%8.5f`	`-0.00600`
`-.006`	`%.3f`	`-0.006`	`-3.14159`	`%.4f`	`-3.1416`

Errors

Compile time → Syntax errors

- When the code violated the grammer rules of C.
- Compiler detecs these errors

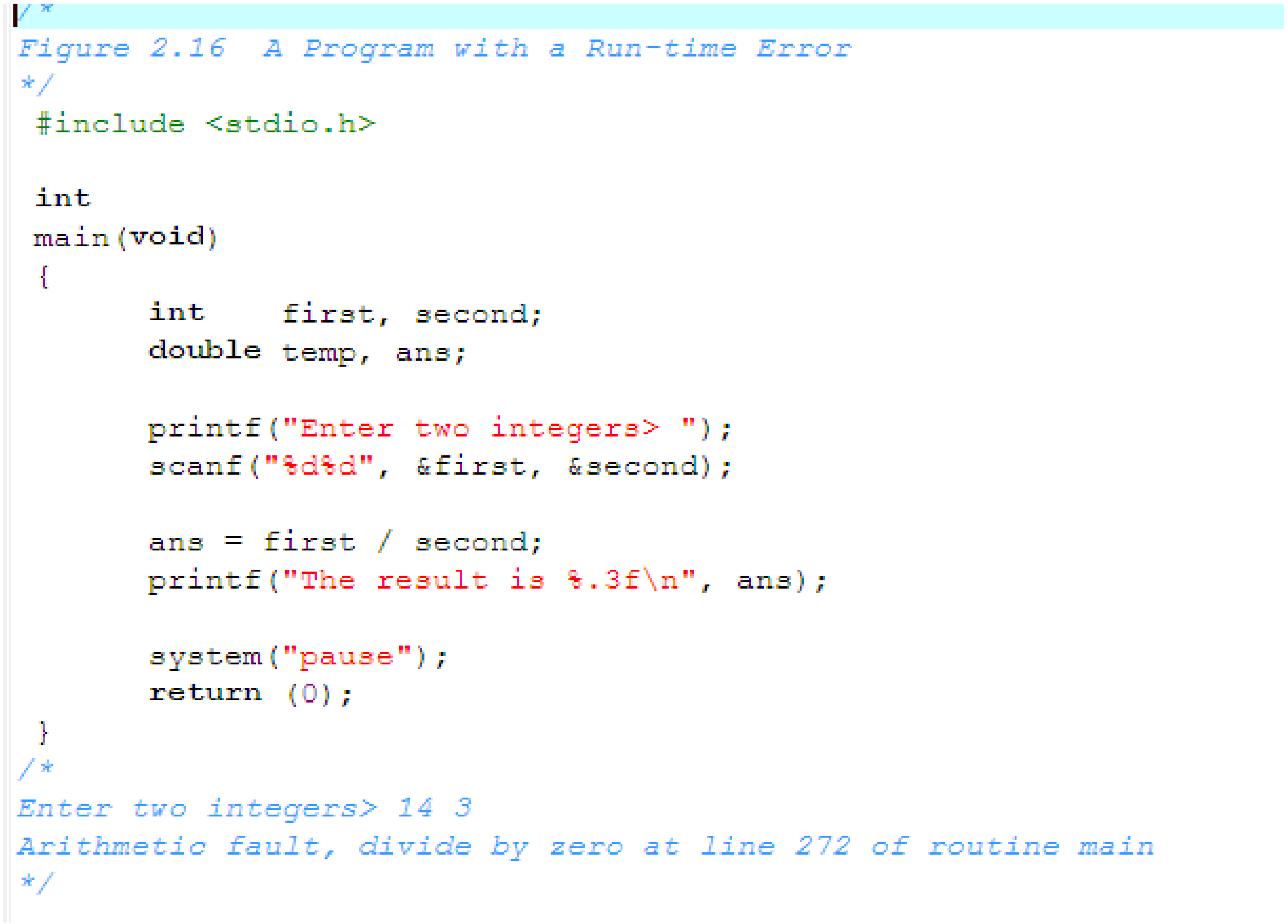
Run-time errors

- happen when the program directs the computer to an illegal operation.
- Such as Division by zero

Logic errors

- A faulty algorithm
- It gives no error message.

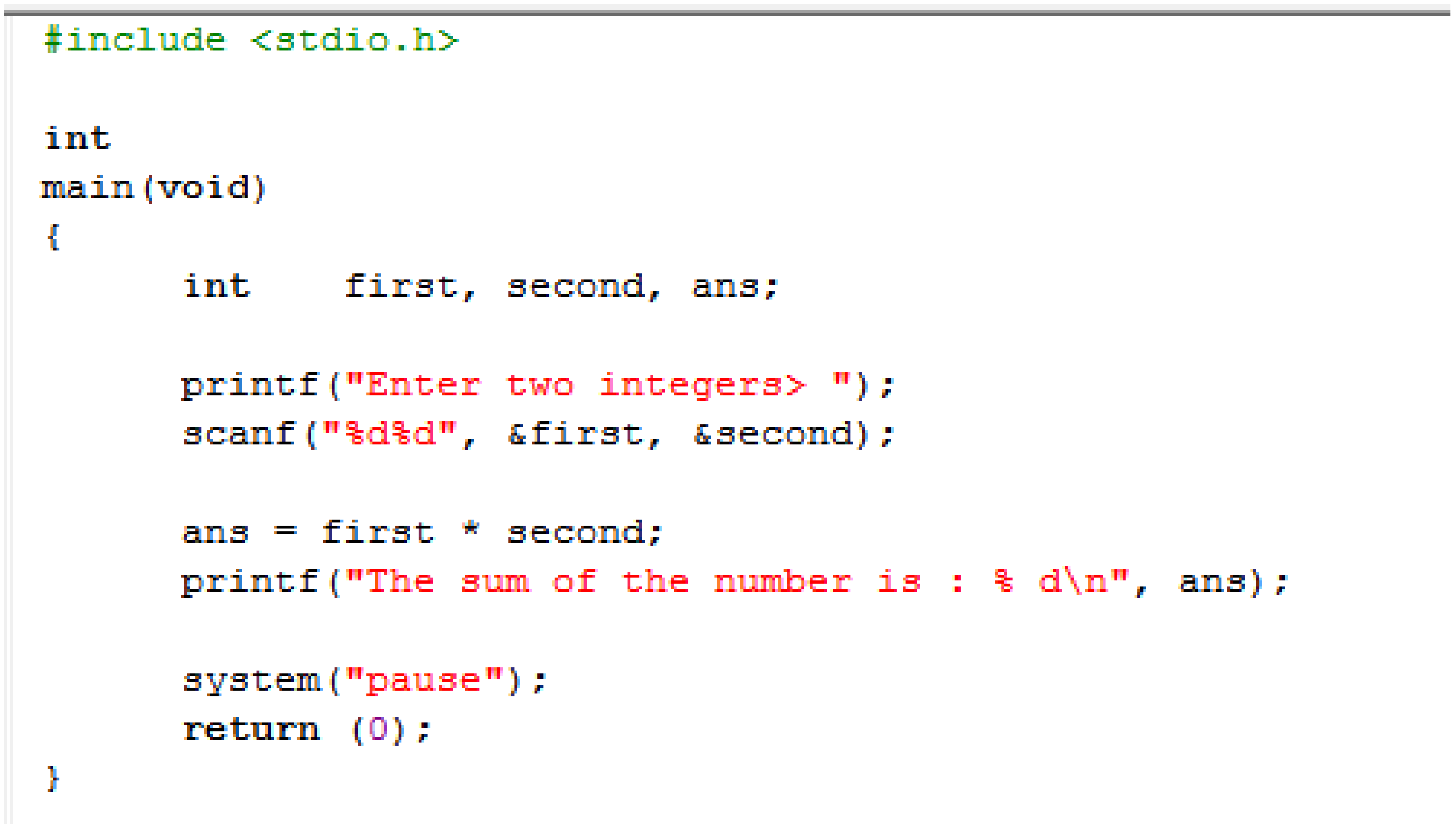
Run-time error (division by zero)



```
/*
Figure 2.16  A Program with a Run-time Error
*/
 #include <stdio.h>

 int
 main(void)
 {
       int    first, second;
       double temp, ans;

       printf("Enter two integers> ");
       scanf("%d%d", &first, &second);

       ans = first / second;
       printf("The result is %.3f\n", ans);

       system("pause");
       return (0);
 }
/*
Enter two integers> 14 3
Arithmetic fault, divide by zero at line 272 of routine main
*/
```

If the value of variable “second” is given as zero

Logic error



```
#include <stdio.h>

int
main(void)
{
        int    first, second, ans;

        printf("Enter two integers> ");
        scanf("%d%d", &first, &second);

        ans = first * second;
        printf("The sum of the number is : % d\n", ans);

        system("pause");
        return (0);
}
```

Data Type of an Expression

The data type of an expression depends on the type(s) of its operands.

if both operands are int → int

otherwise → double

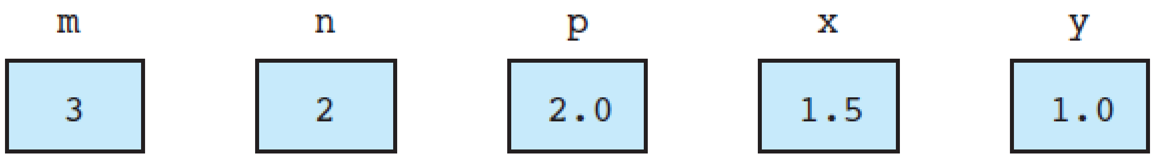
An expression that has operands of both type int and double is a **mixed-type expression .**

The data type of such a mixed-type expression will be **double** .

When an assignment statement is executed, the expression is first evaluated; then the result is assigned to the variable listed to the left of the assignment operator (=).

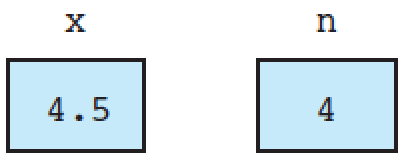
Either a type double or a type int expression may be assigned to a type double

```
int  m, n;

double  p , x , y;

 m = 3;
 n = 2;
 p = 2.0;
 x = m / p;     3/2.0
 y = m / n;     3/2
```



```
x = 9 * 0.5;
n = 9 * 0.5;
```

evaluates to the real number `4.5`. If `x` is of type `double`, the number `4.5` is stored in `x`, as expected. If `n` is of type `int`, only the integral part of the expression value is stored in `n`, as shown.



Sample questions

Evaluate the following expressions with `7` and `22` as operands. (be careful that the values are integers)

```
22 / 7     7 / 22     22 % 7     7 % 22
```

Evaluate the following, assuming that letters have consecutive character codes.

a. `(int)'D' - (int)'A'`
b. `(char)((int)'C' + 2)`
c. `(int)'6' - (int)'7'`

4) What will be the output of the following C program?

```
#include <stdio.h>
#define X 5+3
int main() {
int a = X / 2;
printf("%d", a);
return 0; }
```

a) 2 **b)** 4 [✓ **c)** 6] **d)** 8 **e)** 6.5

5) What will be the output of the following C program?

```
#include <stdio.h>
int main() {
  double x, y;
  x = 7;
  x = x / 2;
  y = x + x / 2;
  printf("%.2f %.2f", x, y);
  return 0; }
```

a) 3.00 3.00 **b)** 3.0 4.50 **c)** 3.50 3.50
[✓ **d)** 3.50 5.25] **e)** 4.50 5.25

6) What will be the output of the following C program?

```
  #include <stdio.h>
int
             main
      (void) {
    int a; double b; printf("%d %.2f", a=5+3/2,
b=5+3/2);
    return 0; }
```

[✓ **a)** 6 6.00] **b)** 6 6.50
c) 7 7.00 **d)** 7 7.50
e) This program will not compile successfully because of bad indentation.

```
# include <stdio.h>
int main (void){
printf ("%c,%d,%c,%d",'a','a', 97, 97);
return 0;}
```

a) a,97,a,98 **b)** 97,a,97,a **c)** 97,97,a,a
d) a,97,a,97 **e)** a,97,97,a

9) What will be the output of the following code segment?

```
double pi= 22/7;
printf("%3.2f", pi);
```

a)3.142857 **b)**3.142 **c)**3.14 **d)**03.14 **e)**3.00

10) What would be the output after execution of the following code?

```
int x=5,y=3;
y+=5-y+x++;
x=y%x;
printf("%d", x);
```

a) 3 **b)** 5 **c)** 6 **d)** 4 **e)** 2

12) What would be the output after execution of the following code?

```
int x=2;
double y=22/5*(double)x;
printf("%.2f", y);
```

a) 2.20 **b)** 2.00 [✓ **c)** 8.00] **d)** 8.80 **e)** 2.80

16) What would be the output after execution of the following code?

```
int b,a=3,c=5;
b=12+a--/++c-(--a);
printf("%d",b);
```

[✓ **a)** 10] **b)** 12 **c)** 9 **d)** 11 **e)** 8

6) What could be the output of the following code segment?

printf("%07.4f", 22/7.0);

a) 03.143
b) 003.143
c) .314285
[✓ **d)** 03.1429]
e) .003143

15) What will be the output of the following code segment?

```
int a=4,b=3;
a= 4*3-2+b--/2*3%2*4-2;
printf("%d",a--);
```

a) 8 **b)** 7 [✓ **c)** 12] **d)** 10 **e)** 11

Hint: b--/2*3%2*4 : execute this part from left to right

2) What will be the output after the input of 13 ?

```
int s;
scanf("%d", &s);
printf ("%d", s%2+--s);
```

This one is first executed

a) 12 **b)** 6 **c)** 7 **d)** 8 [✓ **e)** None of them]

4) What will be the output after the input of 7?

```
int x;
printf ("Enter a number");
scanf("%d",&x);
printf("%d %d %d", x - 1, x, x--);
```

a) 5 6 7 **b)** 4 6 6 **c)** 6 7 6 [✓ **d)** 6 7 7] **e)** 4 5 6

4) If *a* is 5, *b* is 4 , *c* is 10 what is the output?

```
a=b=c+6%2;
printf("%d %d %d", a,b,c);
```

a) `5 10 10` **b)** `13 13 10` **c)** `10 10 10` **d)** `5 4 10` **e)** `13 10 10`

5) What is the output of the below code segment ?

```
int i=32;
char c;
c=i;
printf("%d", c);
```

a) 23 **b)** 'c' **c)** 69 **d)** 'E' **e)** 32

8) What is the C equivalent of the following expression?

$$x = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

a) x=(-b-sqrt(b*b-4*a*c))/(2*a)
b) x=(-b-sqrt(b*2-4*a*c)/2*a
c) x=(-b-sqrt(b*2-4ac)/2a
d) x=((-b)-sqrt(b*2-4ac)/2a
e) x=-b-sqrt(b*b-4*a*c)/2*a

Homework (Assignment-2)
Write a program that:

1. Reads two integers and two float numbers from the user
2. Writes examples with preincrement (++a), postincrement operators (a--).
3. Writes examples with += and -= operators
4. Casts the integers to float values and writes results
5. Writes the result of the expression

   3+(17*2-9)/2*3-29/2

Upload the .c file to LMS