مبانی برنامه نویسی
به زبان سی

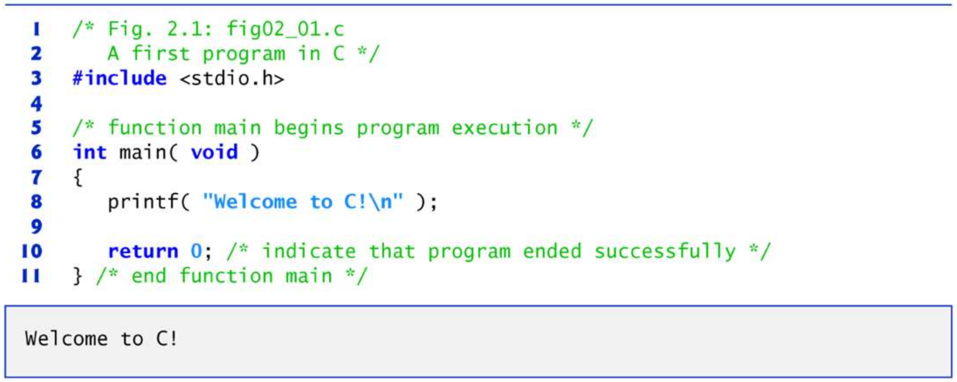
۱۸، ۲۰ ۱۳۹۹

جلسه دوم، سوم

ملکی مجد

طرح کلی این هفته:

- برنامه ساده چاپ
- برنامه جمع دو عدد به درخواست کاربر
- مفهوم حافظه
- محاسبات در زبان سی
- عملگرهای رابطه ای
- تصمیم گیری



```c
/* Fig. 2.1: fig02_01.c
   A first program in C */
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
   printf( "Welcome to C!\n" );

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

```
Welcome to C!
```

Fig. 2.1 | A first program in C.

2.2 A Simple C Program: Printing a Line of Text (Cont.)

- Lines 1 and 2

    -
      ```
      /* Fig. 2.1: fig02_01.c
         A first program in C */
      ```
- begin with `/*` and end with `*/` indicating that these two lines are a comment.
    - You insert comments to document programs and improve program readability.
    - Comments do not cause the computer to perform any action when the program is run.
- `//` single-line comments in which everything from `//` to the end of the line is a comment.

2.2 A Simple C Program: Printing a Line of Text (Cont.)

- Lines beginning with # are processed by the preprocessor before the program is compiled.
  - Line 3 tells the preprocessor to include the contents of the standard input/output header (`<stdio.h>`) in the program.
  - This header contains information used by the compiler when compiling calls to standard input/output library functions such as `printf`.
  - `#include <stdio.h>` is a directive to the C preprocessor.

2.2 A Simple C Program: Printing a Line of Text (Cont.)

- `int main( void ){}`
  - The parentheses after `main` indicate that `main` is a program building block called a function.
  - C programs contain one or more functions, one of which must be `main`.
  - Every program in C begins executing at the function `main`.
  - The keyword `int` to the left of `main` indicates that `main` "returns" an integer (whole number) value.
    - Learn more later!
  - The `void` in parentheses here means that `main` does not receive any information.

  - A left brace, `{`, begins the body of every function (line 7).
  - A corresponding right brace ends each function (line 11).
  - This pair of braces and the portion of the program between the braces is called a block.

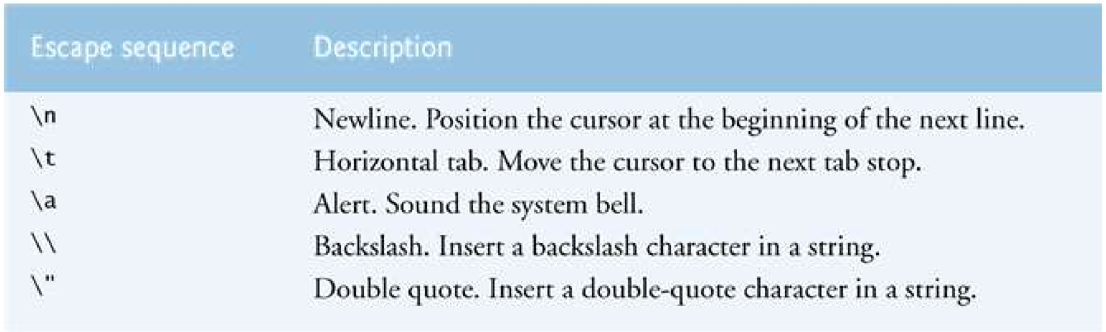
2.2 A Simple C Program: Printing a Line of Text (Cont.)

- `printf( "Welcome to C!\n" );`
  - instructs the computer to perform an action,
    - namely to print on the screen the string of characters marked by the quotation marks.
  - A string is sometimes called a character string, a message or a literal.

2.2 A Simple C Program: Printing a Line of Text (Cont.)

- The entire line, including `printf`, its argument within the parentheses and the semicolon (;), is called a statement.
  - Every statement must end with a

- The characters normally print exactly as they appear between the double quotes in the `printf` statement. **When** encountering a **backslash** in a string, the compiler looks ahead at the next character and combines it with the backslash to form an escape sequence.
  - The escape sequence `\n` means newline.



Escape sequence	Description
`\n`	Newline. Position the cursor at the beginning of the next line.
`\t`	Horizontal tab. Move the cursor to the next tab stop.
`\a`	Alert. Sound the system bell.
`\\`	Backslash. Insert a backslash character in a string.
`\"`	Double quote. Insert a double-quote character in a string.

2.2 A Simple C Program: Printing a Line of Text (Cont.)

- Line 10

```
    return 0; /* indicate that program ended successfully */
```

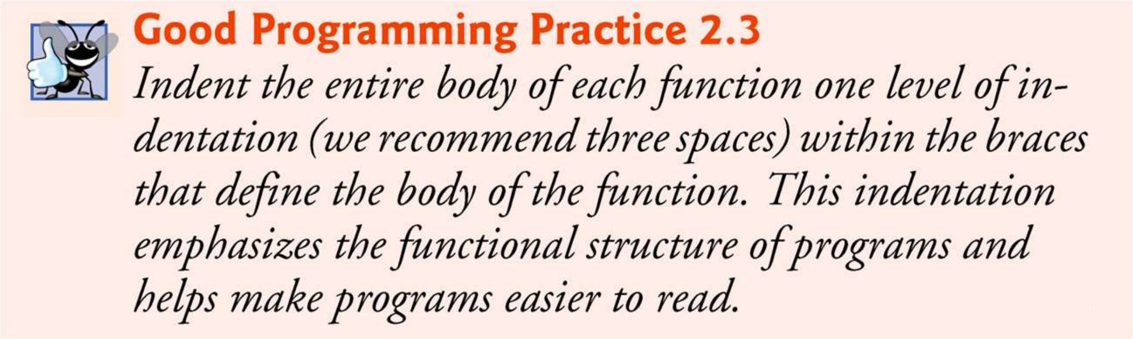
  - is included at the end of every `main` function.
  - The keyword `return` is one of several means we'll use to exit a function.

- The right brace, `}`, (line 12) indicates that the end of `main` has been reached.

2.2 A Simple C Program: Printing a Line of Text (Cont.)

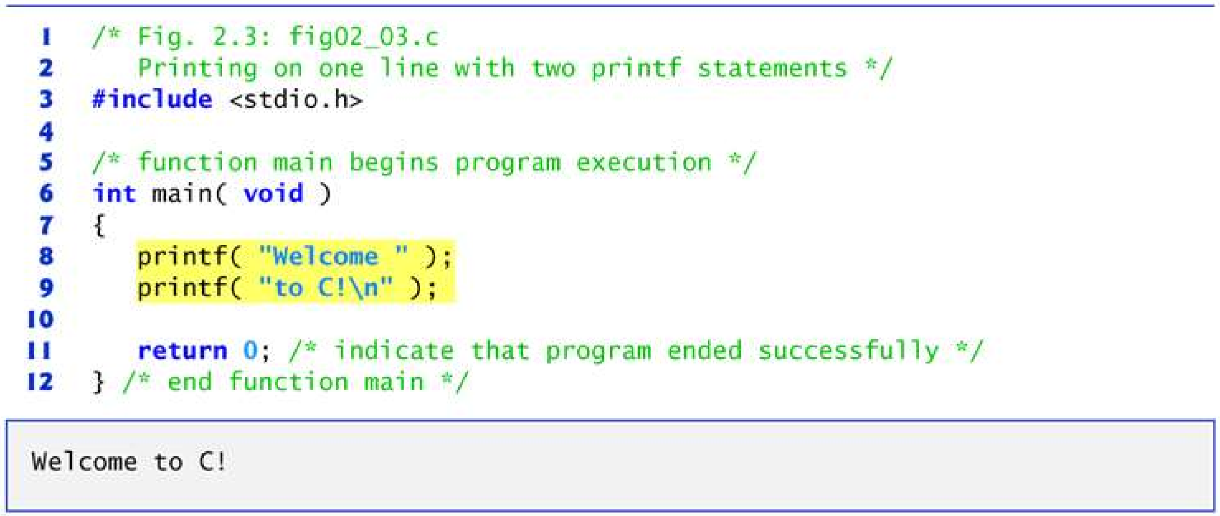
- Standard library functions like `printf` and `scanf` are not part of the C programming language.
  - When the compiler compiles a `printf` statement, it merely provides space in the object program for a “call” to **the library function**.

  - But the compiler does not know where the library functions are—the linker does.
  - When the linker runs, it locates the library functions and inserts the proper calls to these library functions in the object program.



Good Programming Practice 2.3

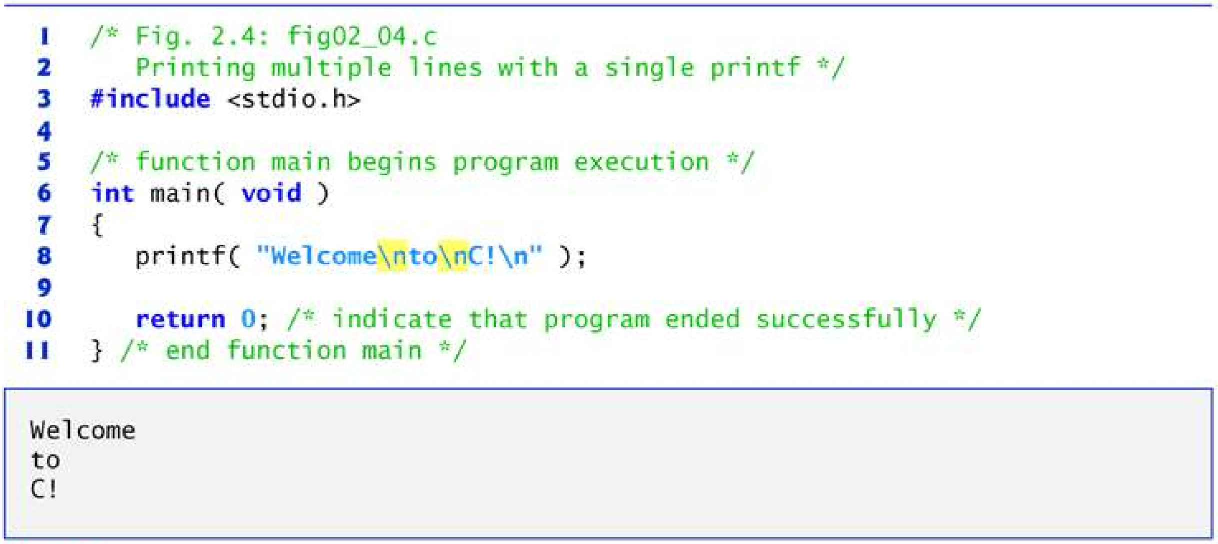
Indent the entire body of each function one level of indentation (we recommend three spaces) within the braces that define the body of the function. This indentation emphasizes the functional structure of programs and helps make programs easier to read.



```c
/* Fig. 2.3: fig02_03.c
   Printing on one line with two printf statements */
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
   printf( "Welcome " );
   printf( "to C!\n" );

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

```
Welcome to C!
```

Fig. 2.3 | Printing on one line with two `printf` statements.



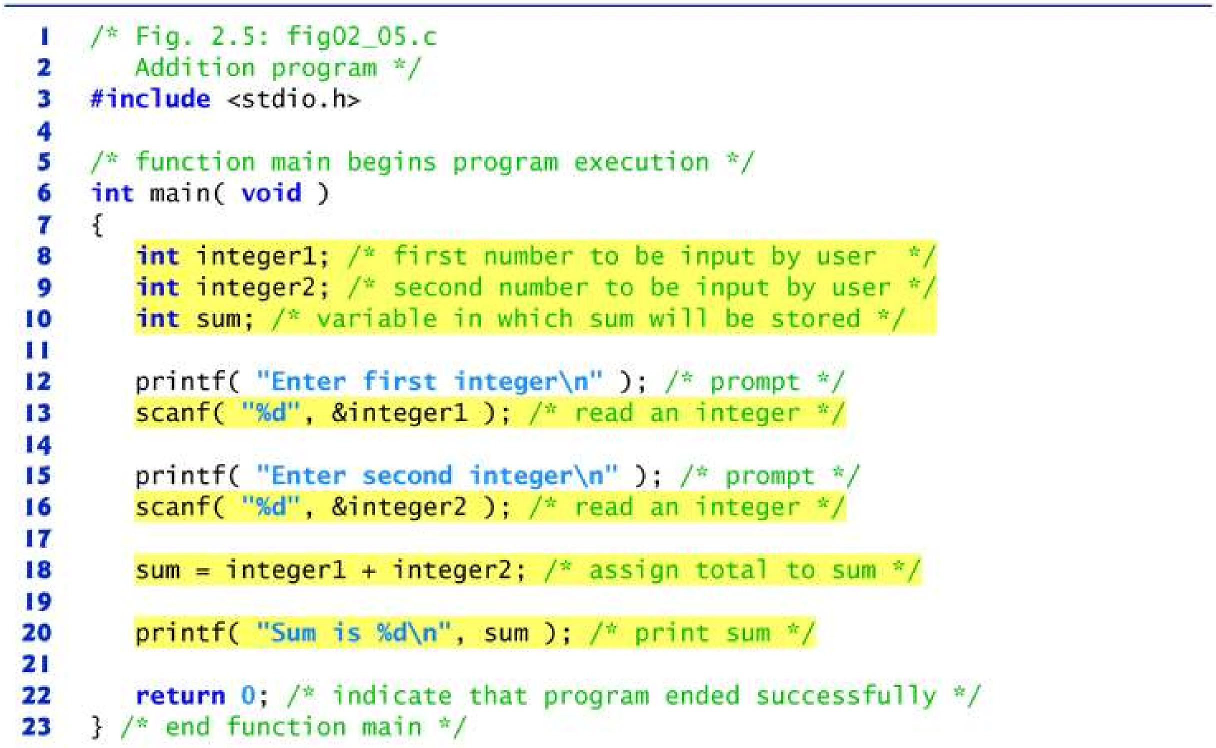
```c
/* Fig. 2.4: fig02_04.c
   Printing multiple lines with a single printf */
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
   printf( "Welcome\nto\nC!\n" );

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

```
Welcome
to
C!
```

Fig. 2.4 | Printing multiple lines with a single `printf`.

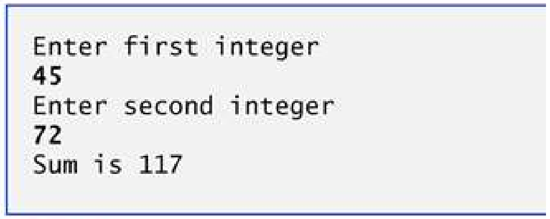
2.3 Another Simple C Program: Adding Two Integers

- Using the Standard Library function `scanf`



```
/* Fig. 2.5: fig02_05.c
   Addition program */
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
   int integer1; /* first number to be input by user  */
   int integer2; /* second number to be input by user */
   int sum; /* variable in which sum will be stored */

   printf( "Enter first integer\n" ); /* prompt */
   scanf( "%d", &integer1 ); /* read an integer */

   printf( "Enter second integer\n" ); /* prompt */
   scanf( "%d", &integer2 ); /* read an integer */

   sum = integer1 + integer2; /* assign total to sum */

   printf( "Sum is %d\n", sum ); /* print sum */

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

Fig. 2.5 | Addition program. (Part 1 of 2.)



```
Enter first integer
45
Enter second integer
72
Sum is 117
```

Fig. 2.5 | Addition program. (Part 2 of 2.)

2.3 Another Simple C Program: Adding Two Integers (Cont.)

- Lines 8–10

    -
      ```
      int integer1; /* first number to be input by user  */
      int integer2; /* second number to be input by user */
      int sum; /* variable in which sum will be stored */
      ```
- are definitions.

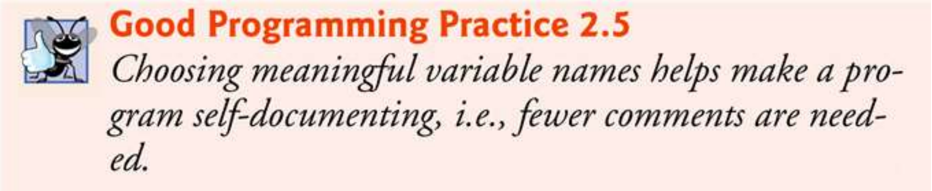
- The names `integer1`, `integer2` and `sum` are the names of variables.

- A variable is a **location in memory** where a value can be stored for use by a program.
    - These definitions specify that the variables `integer1`, `integer2` and `sum` are of type `int`, which means that these variables will hold integer values, i.e., whole numbers such as 7, –11, 0, 31914 and the like.

2.3 Another Simple C Program: Adding Two Integers (Cont.)

- **variables** must be defined **with a name** and a **data type**
  - after the left brace that begins the body of `main`
  - And before they can be used in a program.
- The preceding definitions could have been combined into a single definition statement as follows:
  - `int integer1, integer2, sum;`

- A variable name in C is any valid identifier.
  - An identifier is a series of characters consisting of letters, digits and underscores (_) that does not begin with a digit.
  - C is case sensitive—uppercase and lowercase letters are different in C, so `a1` and `A1` are different identifiers



Good Programming Practice 2.5

Choosing meaningful variable names helps make a program self-documenting, i.e., fewer comments are needed.

2.3 Another Simple C Program: Adding Two Integers (Cont.)

- A syntax error is caused when the compiler cannot recognize a statement.
- The compiler normally issues an error message to help you locate and fix the incorrect statement.
- Syntax errors are violations of the language.
- Syntax errors are also called compile errors, or compile-time errors.

2.3 Another Simple C Program: Adding Two Integers (Cont.)

- `scanf( "%d", &integer1 ); /* read an integer */`
  - uses `scanf` to obtain a value from the user.
  - The `scanf` function reads from the standard input, which is usually the keyboard.
- This `scanf` has two arguments, `"%d"` and `&integer1`.
  - The first argument, the format control string, indicates the type of data that should be input by the user.
  - The `%d` conversion specifier indicates that the data should be an integer (the letter `d` stands for “decimal integer”).

  - The `%` in this context is treated by `scanf` (and `printf` as we’ll see) as a special character that begins a conversion specifier.
  - The second argument of `scanf` begins with an ampersand (&)—called the address operator in C—followed by the variable name.

2.3 Another Simple C Program: Adding Two Integers (Cont.)

- Ampersand
  - The ampersand, when combined with the variable name, tells `scanf` the location (or address) in memory at which the variable `integer1` is stored.
  - The computer then stores the value for `integer1` at that location.
  - The use of ampersand (&) is often confusing to novice programmers or to people who have programmed in other languages that do not require this notation.
  - For now, just remember to precede each variable in every call to `scanf` with an ampersand.

2.3 Another Simple C Program: Adding Two Integers (Cont.)

- The assignment statement in line 18
    - `sum = integer1 + integer2; /* assign total to sum */`
  - calculates the sum of variables `integer1` and `integer2` and assigns the result to variable `sum` using the assignment operator =.

- The = operator and the + operator are called binary operators because each has two operands.
  - The + operator’s two operands are `integer1` and `integer2`.
  - The = operator’s two operands are `sum` and the value of the expression `integer1 + integer2`.

2.3 Another Simple C Program: Adding Two Integers (Cont.)

- Line 20
    - `printf( "Sum is %d\n", sum ); /* print sum */`
- calls function `printf` to print the literal `Sum is` followed by the numerical value of variable `sum` on the screen.

    - This `printf` has two arguments, `"Sum is %d\n"` and `sum`.
    - The first argument is the format control string.
        - It contains some literal characters to be displayed, and it contains the conversion specifier `%d` indicating that an integer will be printed.
    - The second argument specifies the value to be printed.

2.3 Another Simple C Program: Adding Two Integers (Cont.)

- We could have combined the previous two statements into the statement

    -
      ```
      printf( "Sum is %d\n", integer1 + integer2 );
      ```

فعالیت کلاسی

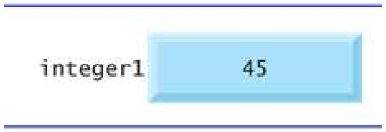
- برنامه ای بنویسید که از ورودی کنسول یک عدد صحیح x بخواند و حاصل عبارت x*x-x را در خروجی کنسول چاپ کند.

2.4 Memory Concepts

- Variable names such as `integer1`, `integer2` and `sum` actually **correspond to locations** in the computer's memory.

- Every variable has a name, a type and a value.

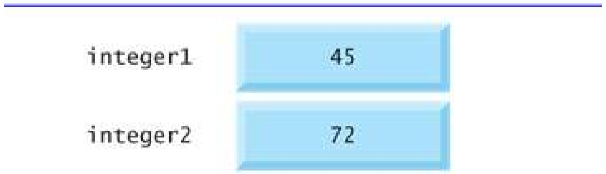
- when the statement `scanf( "%d", &integer1 );`
  - is executed, the value typed by the user is placed into a memory location to which the name `integer1` has been assigned.
  - Suppose the user enters the number `45` as the value for `integer1`. The computer will place `45` into location `integer1`.



2.4 Memory Concepts (Cont.)

- Whenever a value is placed in a memory location,
  - the value replaces the previous value in that location; thus,
  - placing a new value into a memory location is said to be destructive.

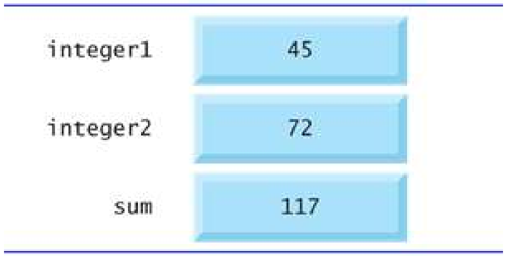
- `scanf( "%d", &integer2 ); /* read an integer */`
  - executes, suppose the user enters the value `72`.
  - This value is placed into location `integer2`, and memory appears as in Fig. 2.7.
  - These locations are not necessarily adjacent in memory.



Memory locations after both variables are input.

2.4 Memory Concepts (Cont.)

- `sum = integer1 + integer2; /* assign total to sum */`
  - that performs the addition also replaces whatever value was stored in `sum`.

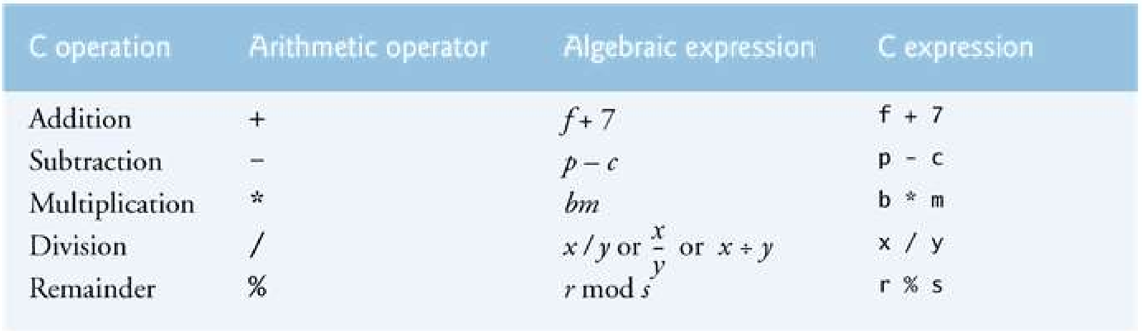


Memory locations after a calculation.

- `integer1 are integer2` were used, but not destroyed, as the computer performed the calculation.
- Thus, when a value is read from a memory location, the process is said to be nondestructive.

2.5 Arithmetic in C

- The C arithmetic operators are summarized in following:



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`

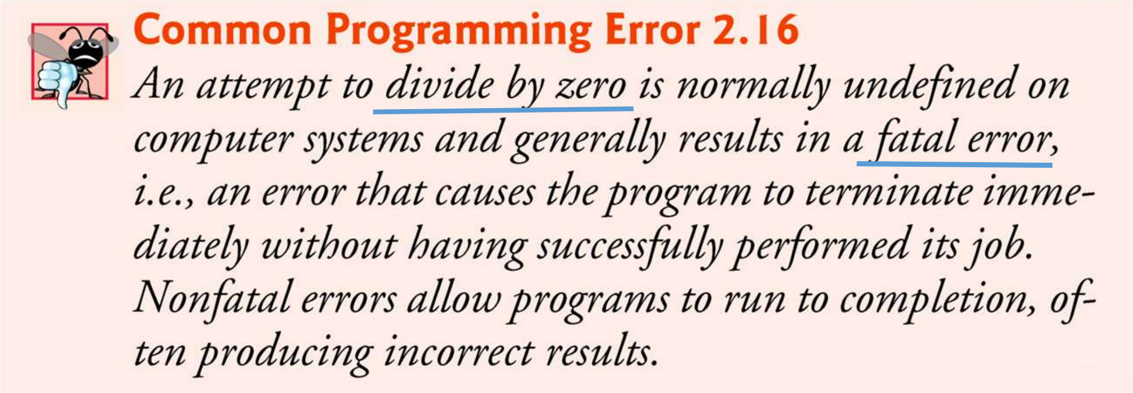
- The asterisk (`*`) indicates multiplication and the percent sign (`%`) denotes the remainder operator.
  - C requires that multiplication be explicitly denoted by using the * operator as in `a * b`.

2.5 Arithmetic in C (Cont.)

- The arithmetic operators are all **binary** operators.

- Integer division yields an integer result.
  - For example, the expression `7 / 4` evaluates to `1` and the expression `17 / 5` evaluates to `3`.

- C provides the remainder operator, `%`, which yields the remainder after integer division.
  - The remainder operator is an integer operator that can be used only with integer operands.
  - The expression `x % y` yields the remainder after `x` is divided by `y`.
  - Thus, `7 % 4` yields `3` and `17 % 5` yields `2`.



Common Programming Error 2.16

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. Nonfatal errors allow programs to run to completion, often producing incorrect results.

2.5 Arithmetic in C (Cont.)

- Parentheses are used in C expressions in the same manner as in algebraic expressions.
  - For example, to multiply `a` times the quantity `b + c` we write `a * ( b + c )`.

- As in algebra, it is acceptable to place **unnecessary** parentheses in an expression to make the expression clearer.
  - These are called redundant parentheses.

2.5 Arithmetic in C (Cont.)

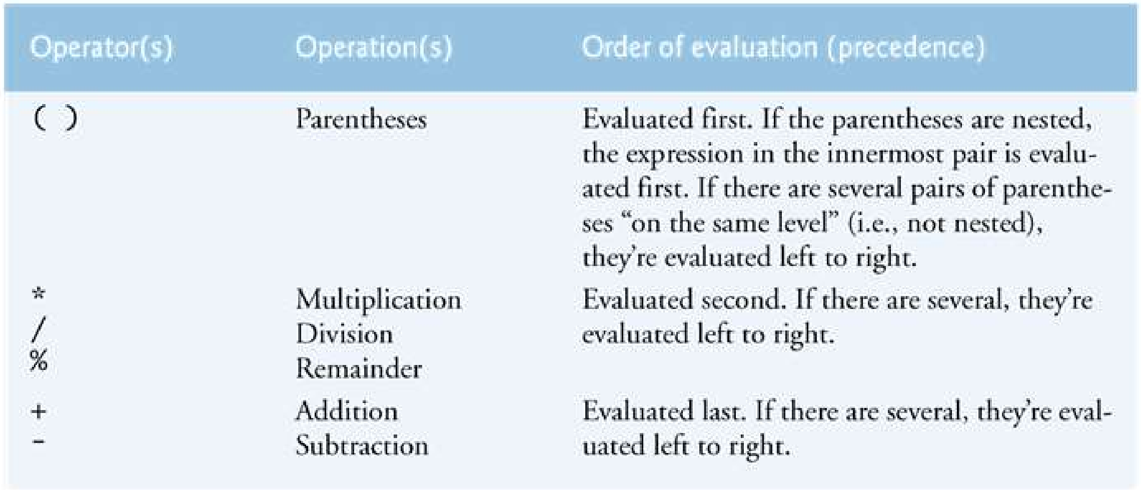
- C applies the operators in arithmetic expressions in a precise sequence determined by the following rules of operator precedence, which are generally the same as those in algebra:
  - Operators in expressions contained within pairs of parentheses are evaluated first.
  - Thus, *parentheses may be used to force the order of evaluation to occur in any sequence you desire.*
  - *Parentheses are said to be at the "highest level of precedence." In cases of nested, or embedded, parentheses, such as*
    - `( ( a + b ) + c )`
  - the operators in the innermost pair of parentheses are applied first.

2.5 Arithmetic in C (Cont.)

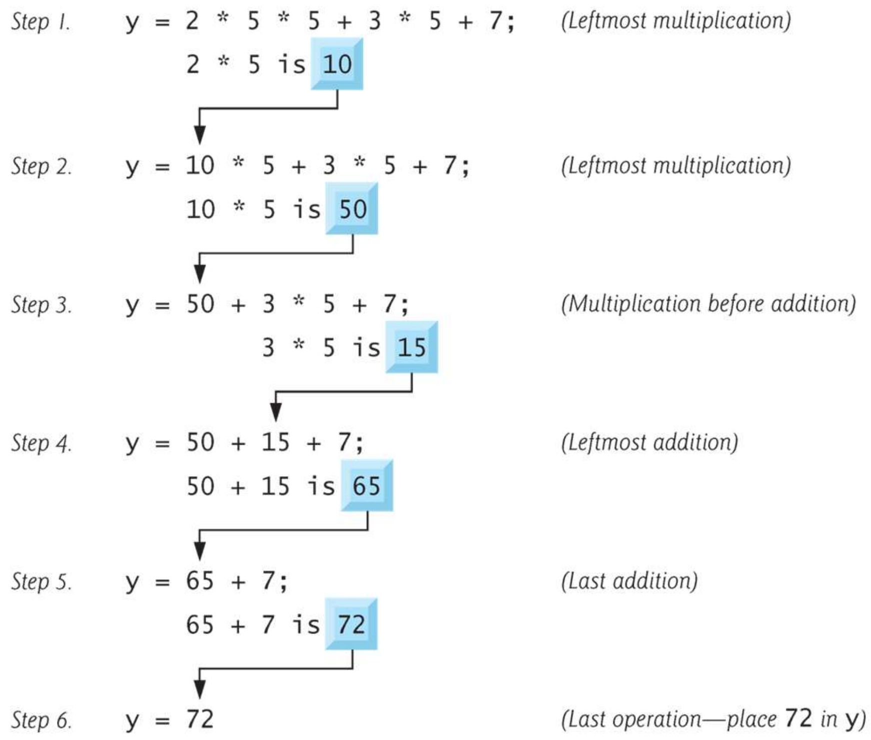
- Multiplication, division and remainder operations are applied first.
  - If an expression contains several multiplication, division and remainder operations, evaluation proceeds **from left to right**. Multiplication, division and remainder are said to be on the **same level** of precedence.
- Addition and subtraction operations are evaluated next.
  - If an expression contains several addition and subtraction operations, evaluation proceeds **from left to right**. Addition and subtraction also have the same level of precedence, which is lower than the precedence of the multiplication, division and remainder operations.

2.5 Arithmetic in C (Cont.)

- The rules of operator **precedence** specify the order C uses to evaluate expressions.
  - When we say evaluation proceeds from left to right, we're referring to the associativity of the operators.



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.



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.

2.6 Decision Making: Equality and Relational Operators

- Executable C statements either
  - ❖perform actions (such as calculations or input or output of data) or
  - ❖make decisions (we'll soon see several examples of these).

- We might make a decision in a program, for example, to determine if a person's grade on an exam is greater than or equal to 60 and if it is to print the message "Congratulations! You passed."

- In what follows
  - We introduces a simple version of C's `if` statement that allows a program to make a decision based on the truth or falsity of a statement of fact called a condition.

2.6 Arithmetic in C (Cont.)

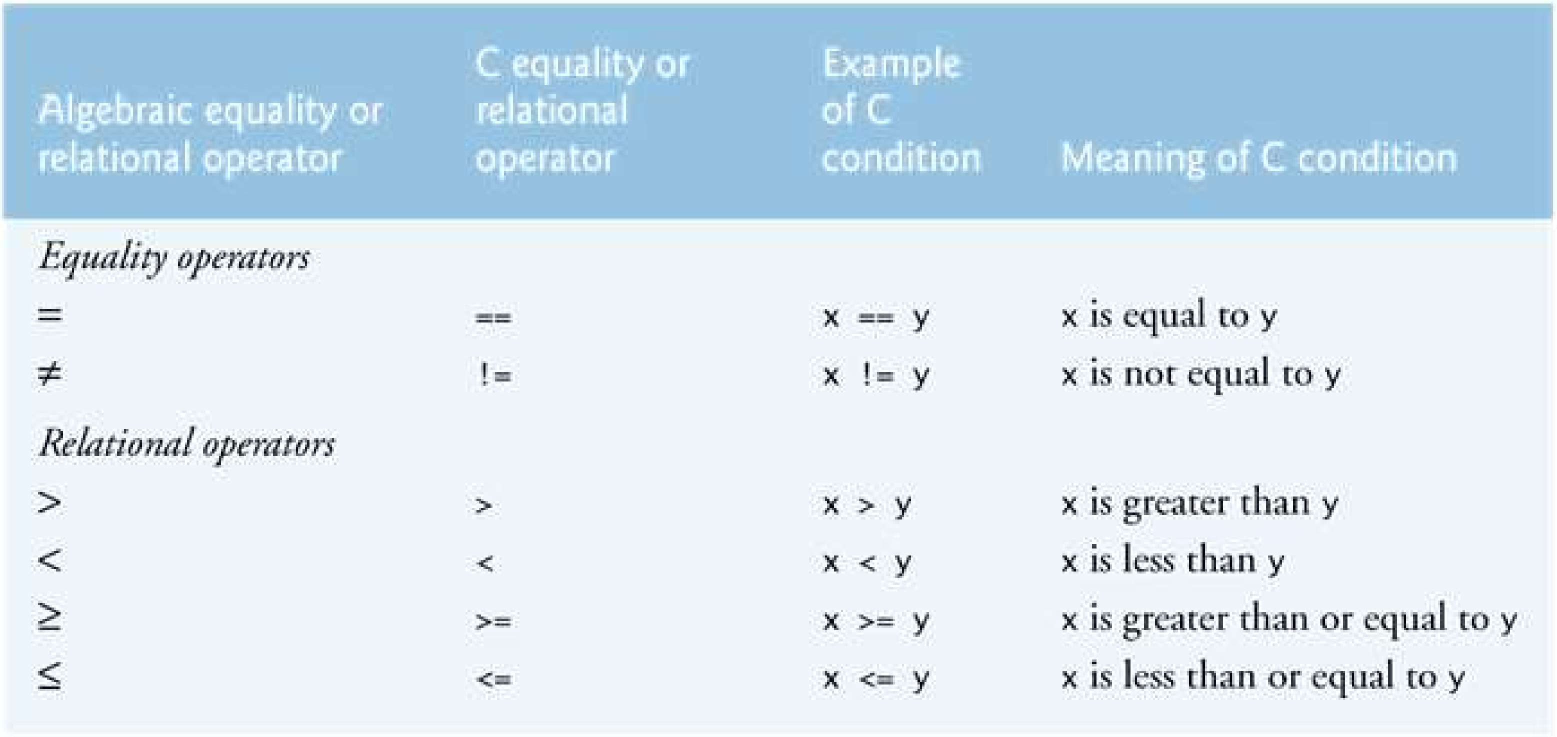
```
if ( -condition- ){
        -the body of if-
}
-next statement-
```

- If the condition is met (i.e., the condition is true) the statement in the body of the `if` statement is executed.
- If the condition is not met (i.e., the condition is false) the body statement is not executed.
- Whether the body statement is executed or not, after the `if` statement completes, execution proceeds with the next statement after the `if` statement.
- Conditions in `if` statements are formed by
  - using the equality operators and relational operators.

2.6 Arithmetic in C (Cont.)

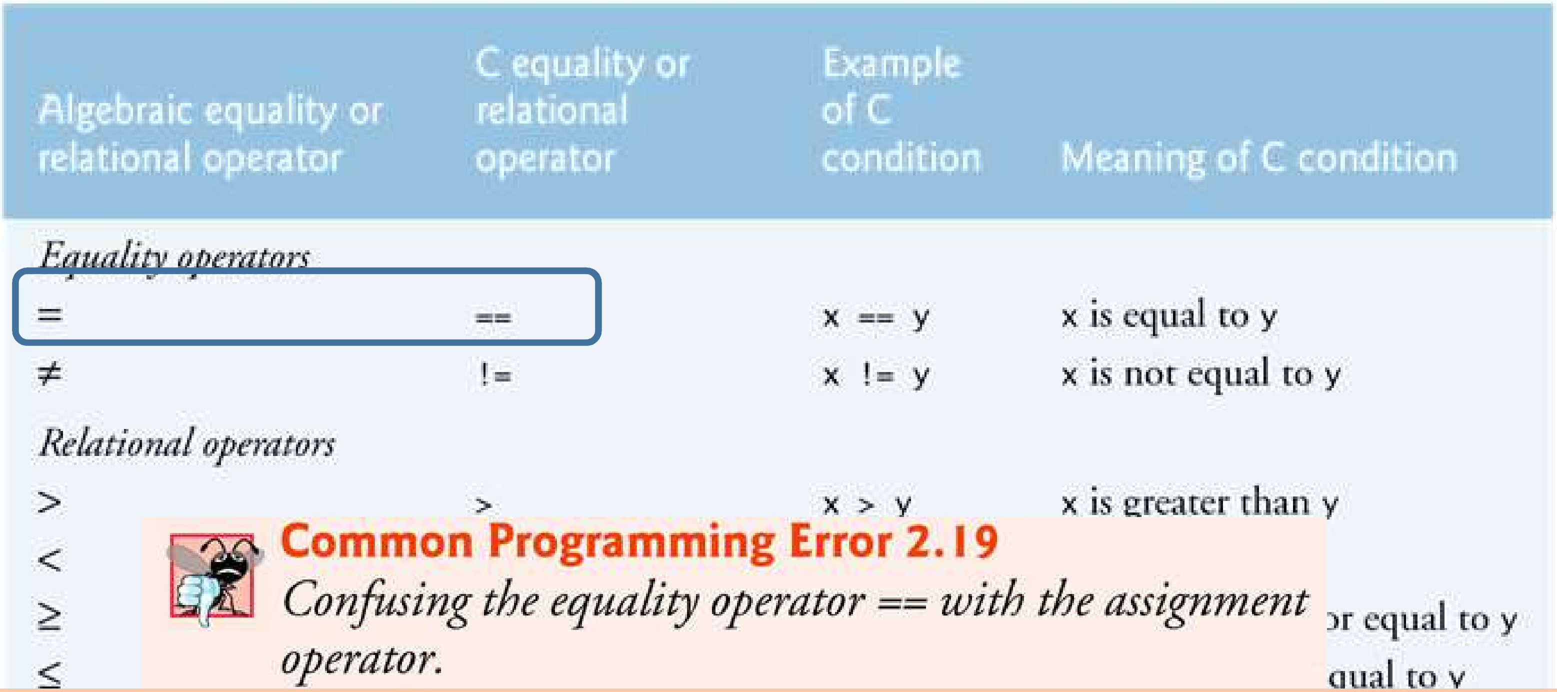
- The relational operators all have the **same level of precedence** and they associate **left to right**.
- The equality operators have a lower level of precedence than the relational operators and they also associate left to right.
- In C, a condition may actually be any expression that generates a zero (false) or nonzero (true) value.

Equality and relational 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

Equality and relational operators

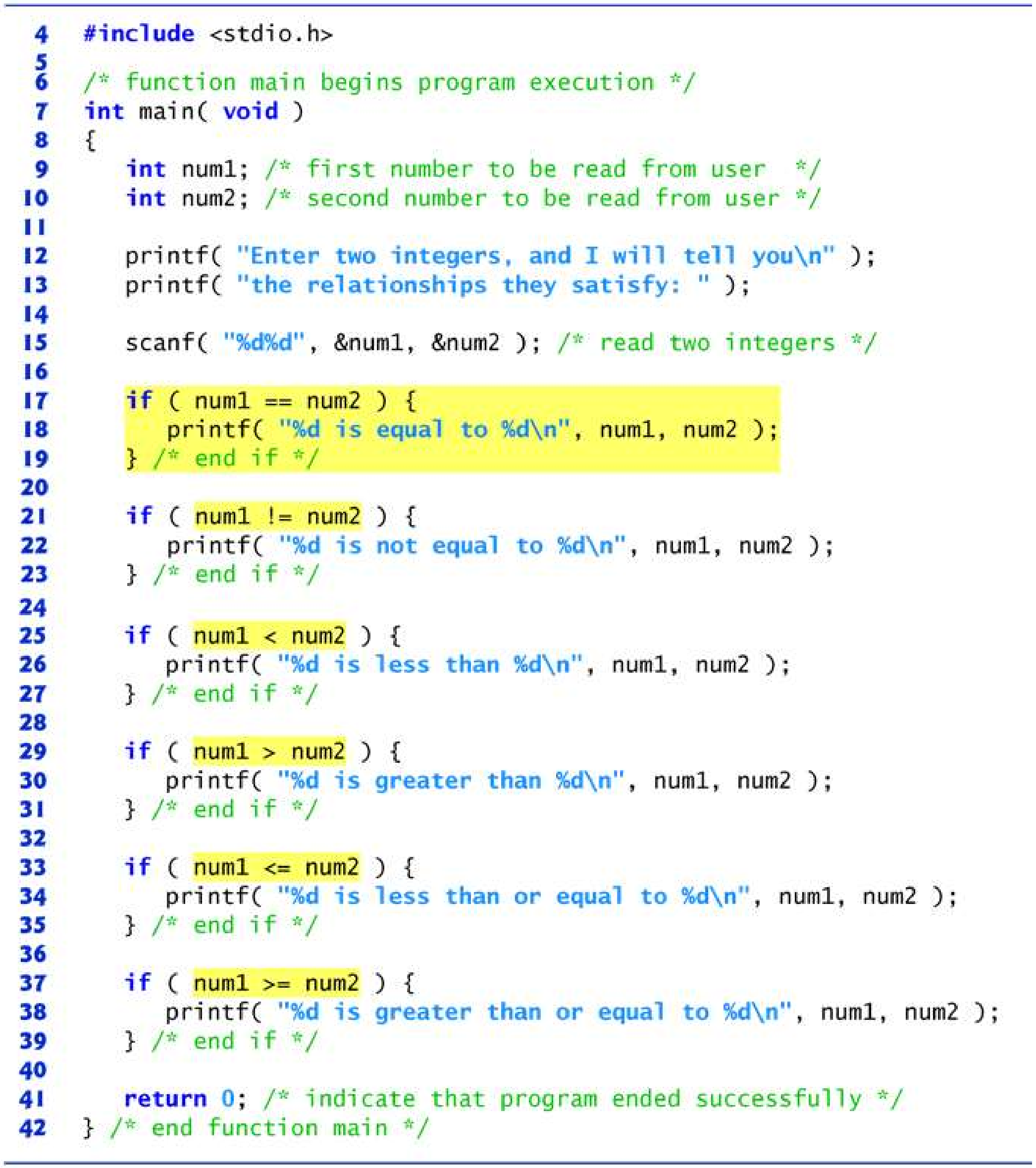


Algebraic equality or relational operator	C equality or relational operator	Example of C condition	Meaning of C condition
Equality operators			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y
Relational operators			
>	>	x > y	x is greater than y
<			
≥			or equal to y
≤			qual to y

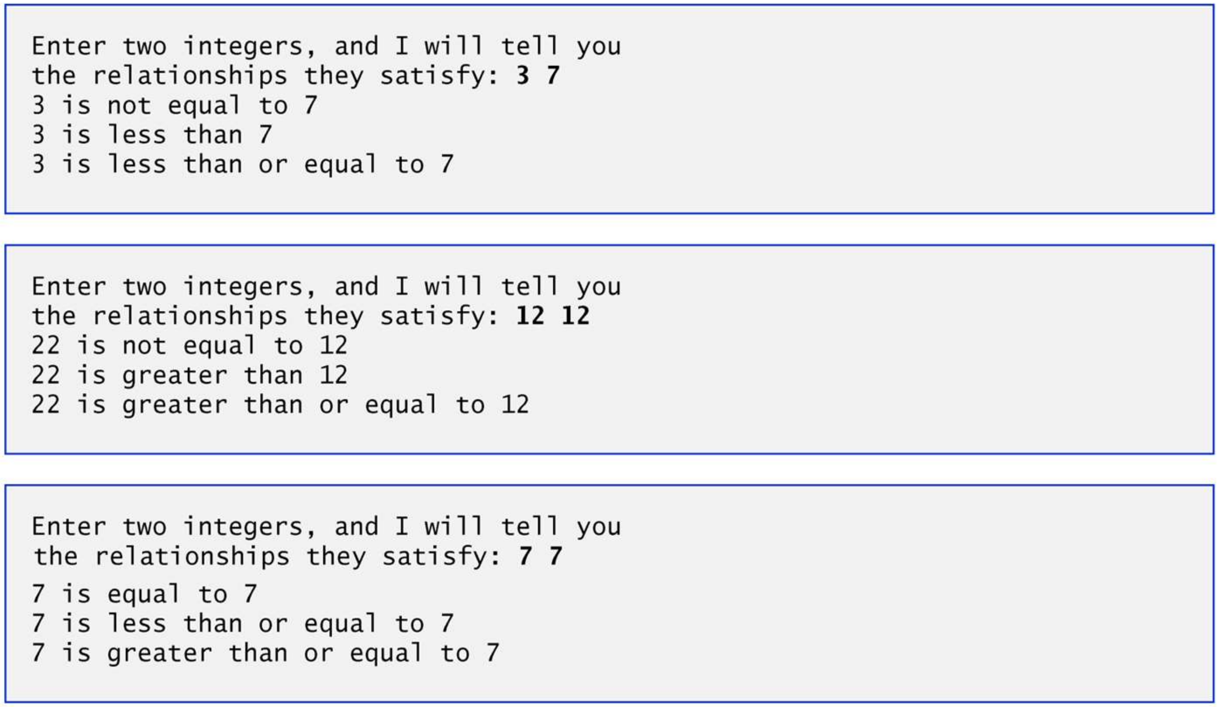
Common Programming Error 2.19

Confusing the equality operator == with the assignment operator.

confusing these operators may not necessarily cause an easy-to-recognize compilation error, but may cause extremely subtle logic errors.



```c
#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 */
```

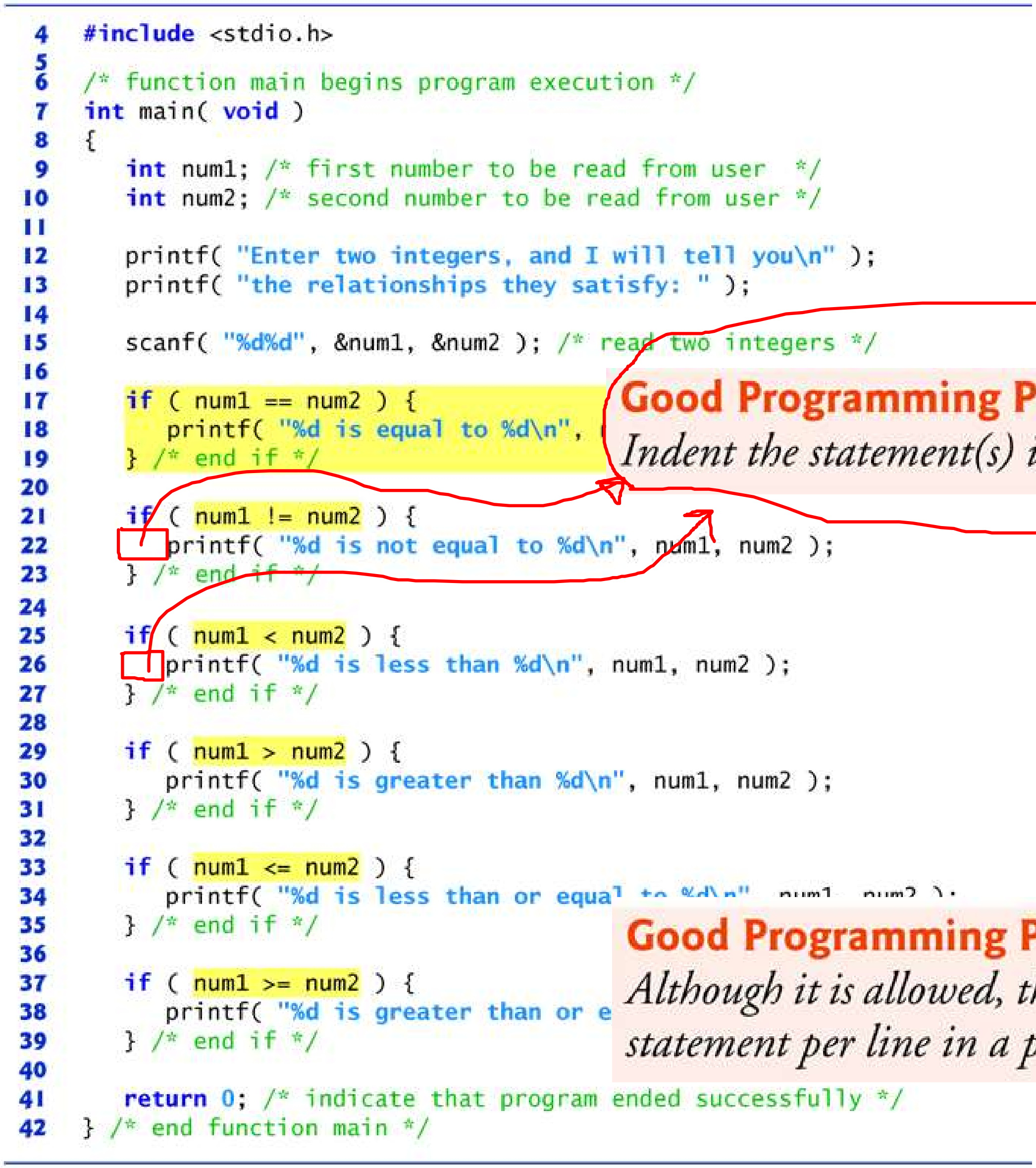


```
Enter two integers, and I will tell you
the relationships they satisfy: 3 7
3 is not equal to 7
3 is less than 7
3 is less than or equal to 7
```

```
Enter two integers, and I will tell you
the relationships they satisfy: 12 12
22 is not equal to 12
22 is greater than 12
22 is greater than or equal to 12
```

```
Enter two integers, and I will tell you
the relationships they satisfy: 7 7
7 is equal to 7
7 is less than or equal to 7
7 is greater than or equal to 7
```

Fig. 2.13 | Using `if` statements, relational operators, and equality operators. (Part 3 of 3.)



```
#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", 
   } /* 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 e
   } /* end if */

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

Good Programming Practice 2.12

Indent the statement(s) in the body of an `if` *statement.*

Good Programming Practice 2.14

Although it is allowed, there should be no more than one statement per line in a program.

2.6 Arithmetic in C (Cont.)

- The program uses `scanf` (line 15) to input two numbers.
- Each conversion specifier has a corresponding argument in which a value will be stored.
- The first `%d` converts a value to be stored in variable `num1`, and the second `%d` converts a value to be stored in variable `num2`.
- Indenting the body of each `if` statement and placing blank lines above and below each `if` statement enhances program readability.

2.6 Arithmetic in C (Cont.)

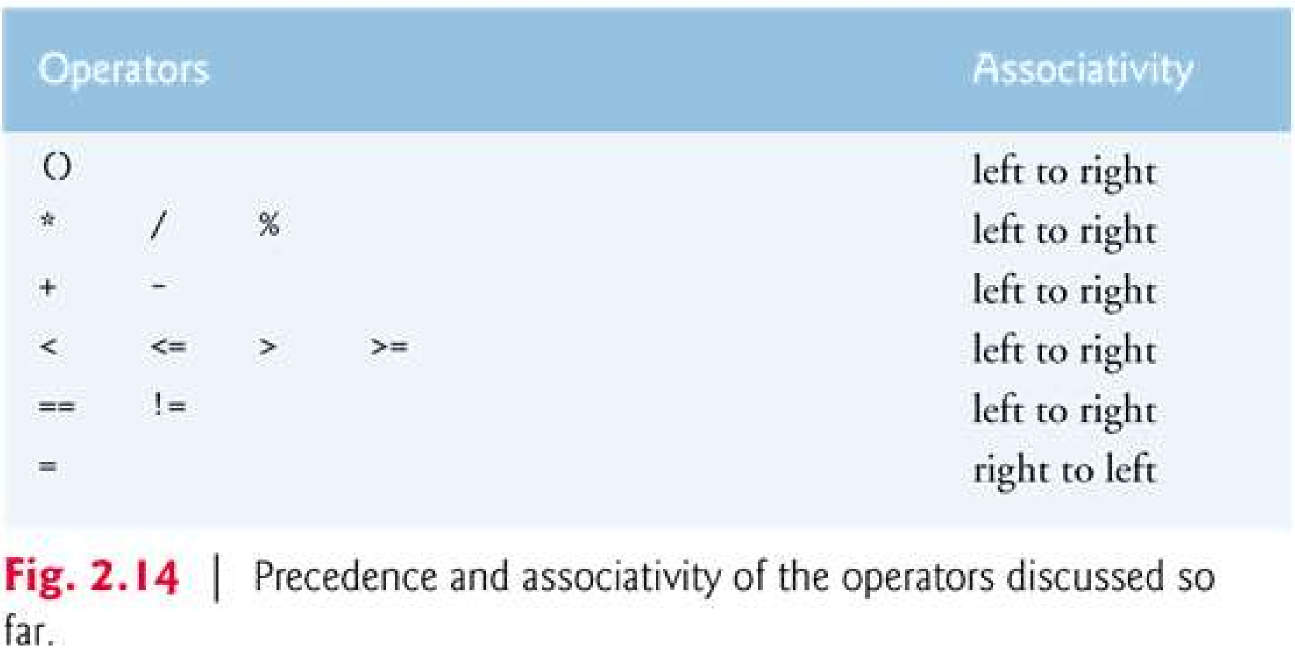
- A left brace, `{`, begins the body of each `if` statement (e.g., line 17).
- A corresponding right brace, `}`, ends each `if` statement’s body (e.g., line 19).
- Any number of statements can be placed in the body of an `if` statement.
- The comment (lines 1–3) in Fig. 2.13 is split over three lines.
- In C programs, white space characters such as tabs, newlines and spaces are normally ignored.
- So, statements and comments may be split over several lines.
- It is not correct, however, to split identifiers.

2.6 Arithmetic in C (Cont.)

- Figure 2.14 lists the precedence of the Operators are shown top to bottom in decreasing order of precedence introduced in this chapter.
- Operators are shown top to bottom in decreasing order of precedence.
- The equals sign is also an operator.
- All these operators, with the exception of the assignment operator =, associate from left to right.
- The assignment operator (=) associates from right to left.

So far **Operators**

are shown top to bottom in decreasing order of precedence

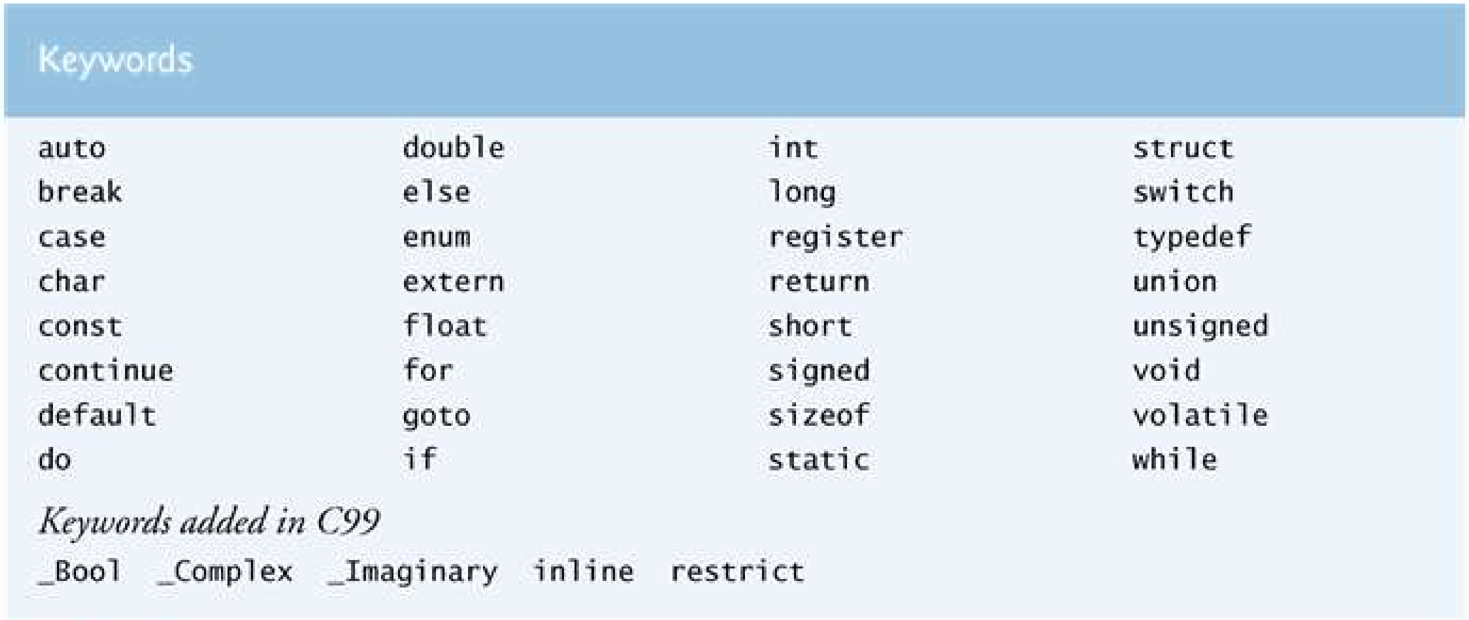


Operators	Associativity
()	left to right
* / %	left to right
+ -	left to right
< <= > >=	left to right
== !=	left to right
=	right to left

Fig. 2.14 | Precedence and associativity of the operators discussed so far.

keywords or reserved words of the language

not to use these as identifiers such as variable names



Keywords			
auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while
Keywords added in C99			
_Bool _Complex _Imaginary inline restrict			

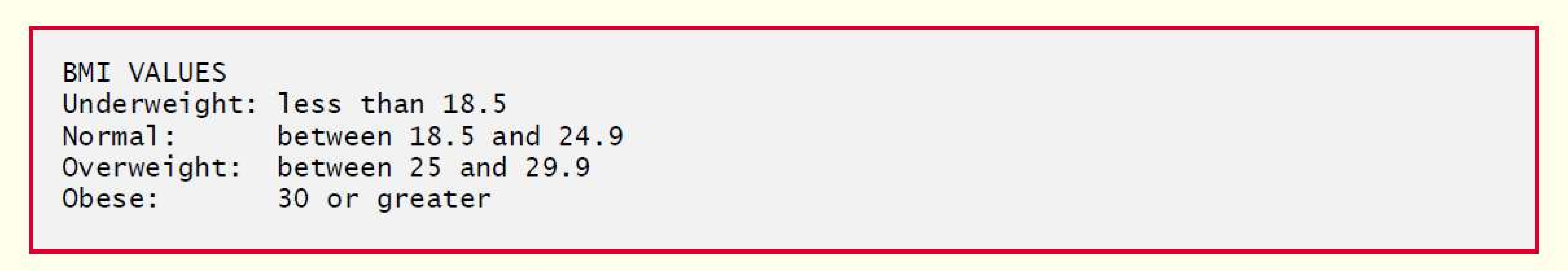
Fig. 2.15 | C's keywords.

نوشتن برنامه محاسبه BMI و تشخیص محدوده آن

برنامه ای بنویسید که از کاربر به ترتیب وزن به کیلوگرم و قد به متر را بگیرد.
طبق فرمول زیر BMI او را محاسبه کند و نمایش بدهد.

$$BMI = \frac{weightInKilograms}{heightInMeters \times heightInMeters}$$

طبق جدول زیر به کاربر اعلام کند که در کدام دسته قرار می گیرد.



```
BMI VALUES
Underweight: less than 18.5
Normal:      between 18.5 and 24.9
Overweight:  between 25 and 29.9
Obese:       30 or greater
```