Chapter 2
Introduction to C Programming

C How to Program, 8/e

2.2 A Simple C Program: Printing a Line of Text

- We begin by considering a simple C program.
- Our first example prints a line of text (Fig. 2.1).

```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" );
} // 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.)

- `// Fig. 2.1: fig02_01.c`
  `// A first program in C`
  - begin with `//`, indicating that these two lines are comments.
  - Comments document programs and improve program readability.
  - Comments do not cause the computer to perform any action when the program is run.

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

- Comments are ignored by the C compiler and do not cause any machine-language object code to be generated.
- Comments also help other people read and understand your program.

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

- You can also use `/*...*/` multi-line comments in which everything from `/*` on the first line to `*/` at the end of the line is a comment.
- We prefer `//` comments because they're shorter and they eliminate the common programming errors that occur with `/*...*/` comments, especially when the closing `*/` is omitted.

#include Preprocessor Directive

- **`#include <stdio.h>`**
  - is a directive to the C preprocessor.

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

- Lines beginning with # are processed by the preprocessor before compilation.
- 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`.

Blank Lines and White Space

- You use blank lines, space characters and tab characters (i.e., "tabs") to make programs easier to read.
- Together, these characters are known as white space. White-space characters are normally ignored by the compiler.

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

The `main` Function

- **`int main( void )`**
  - is a part of every C program.
  - The parentheses after `main` indicate that `main` is a program building block called a function.

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

- 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.

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

- For now, simply include the keyword `int` to the left of `main` in each of your programs.
- Functions also can receive information when they're called upon to execute.
- The `void` in parentheses here means that `main` does not receive any information.

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

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

An Output Statement

- `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 the `printf` function (the "f" stands for "formatted"), its argument within the parentheses and the semicolon `(;)`, is called a statement.
- Every statement must end with a semicolon (also known as the statement terminator).
- When the preceding `printf` statement is executed, it prints the message `Welcome to C!` on the screen.
- The characters normally print exactly as they appear between the double quotes in the `printf` statement.

Escape Sequences

- Notice that the characters `\n` were not printed on the screen.
- The backslash (`\`) is called an escape character.
- It indicates that `printf` is supposed to do something out of the ordinary.

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

- 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.
- When a newline appears in the string output by a `printf`, the newline causes the cursor to position to the beginning of the next line on the screen.
- Some common escape sequences are listed in Fig. 2.2.

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

- Because the backslash has special meaning in a string, i.e., the compiler recognizes it as an escape character, we use a double backslash (`\\`) to place a single backslash in a string.
- Printing a double quote also presents a problem because double quotes mark the boundaries of a string—such quotes are not printed.
- By using the escape sequence `\"` in a string to be output by `printf`, we indicate that `printf` should display a double quote.
- The right brace, `}`, indicates that the end of `main` has been reached.

Common Programming Error 2.1

Mistyping the name of the output function `printf` *as* `print` *in a program.*

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

The Linker and Executables

- Standard library functions like `printf` and `scanf` are not part of the C programming language.
- For example, the compiler cannot find a spelling error in `printf` or `scanf`.
- 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.

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

- Now the object program is complete and ready to be executed.
- For this reason, the linked program is called an executable.
- If the function name is misspelled, it's the linker that will spot the error, because it will not be able to match the name in the C program with the name of any known function in the libraries.

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

Using Multiple `printf`s

- The `printf` function can print `Welcome to C!` several different ways.
- For example, the program of Fig. 2.3 produces the same output as the program of Fig. 2.1.
- This works because each `printf` resumes printing where the previous `printf` stopped printing.
- The first `printf` prints `Welcome` followed by a space and the second `printf` begins printing on the same line immediately following the space.

```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" );
} // end function main
```

```
Welcome to C!
```

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

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

- One `printf` can print *several* lines by using additional newline characters as in Fig. 2.4.
- Each time the `\n` (newline) escape sequence is encountered, output continues at the beginning of the next line.

```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" );
} // 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

- Our next program (fig. 2.5) uses the Standard Library function `scanf` to obtain two integers typed by a user at the keyboard, computes the sum of these values and prints the result using `printf`.
- [In the input/output dialog of Fig. 2.5, we emphasize the numbers entered by the user in **bold.]**

```c
// 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 entered by user
   int integer2; // second number to be entered by user

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

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

   int sum; // variable in which sum will be stored
   sum = integer1 + integer2; // assign total to sum

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

Variables and Variable Definitions

- ```
  int integer1; // first number to be entered by user
  int integer2; // second number to be entered by user
  int sum; // variable in which sum will be stored
  ```
  are definitions.
- The names `integer1`, `integer2` and `sum` are the names of variables—locations in memory where values 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 they'll 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.)

- All variables must be defined with a name and a data type 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;`

  but that would have made it difficult to describe the variables with corresponding comments

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

Identifiers and Case Sensitivity

- 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.

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

Prompting Messages

- `printf( "Enter first integer\n" ); // prompt`
  - displays the literal "`Enter first integer`" and positions the cursor to the beginning of the next line.
  - This message is called a prompt because it tells the user to take a specific action.

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

The `scanf` Function and Formatted Inputs

- The next statement
  - `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.

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

- This `scanf` has two arguments, `"%d"` and `&integer1`.
- The first, 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.)

- The `&`, 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 that the user enters 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.)

- When the computer executes the preceding `scanf`, it waits for the user to enter a value for variable `integer1`.
- The user responds by typing an integer, then pressing the *Enter* key to send the number to the computer.
- The computer then assigns this number, or value, to the variable `integer1`.
- Any subsequent references to `integer1` in this program will use this same value.
- Functions `printf` and `scanf` facilitate interaction between the user and the computer.
- Because this interaction resembles a dialogue, it's often called interactive computing.

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

- `printf( "Enter second integer\n" ); // prompt`
  - displays the message `Enter second integer` on the screen, then positions the cursor to the beginning of the next line.
  - This `printf` also prompts the user to take action.
- `scanf( "%d", &integer2 ); // read an integer`
  - obtains a value for variable `integer2` from the user.

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

Assignment Statement

- The assignment statement
  - `sum = integer1 + integer2; // assign total to sum`

  calculates the total of variables `integer1` and `integer2` and assigns the result to variable `sum` using the assignment operator =.
- The statement is read as, "`sum` *gets* the value of `integer1 + integer2`." Most calculations are performed in assignments.
- 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`.

Common Programming Error 2.3

A calculation in an assignment statement must be on the right side *of the = operator. It's a compilation error to place a calculation on the* left side *of an assignment operator.*

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

Printing with a Format Control String

- `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.
  - Notice that the conversion specifier for an integer is the same in both `printf` and `scanf`.

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

Calculations in `printf` Statements

- We could have combined the previous two statements into the statement
  - `printf( "Sum is %d\n", integer1 + integer2 );`
- The right brace, }, at line 21 indicates that the end of function `main` has been reached.

Common Programming Error 2.4

Forgetting to precede a variable in a `scanf` *statement with an ampersand (&) when that variable should, in fact, be preceded by an ampersand results in an execution-time error. On many systems, this causes a "segmentation fault" or "access violation." Such an error occurs when a user's program attempts to access a part of the computer's memory to which it does not have access privileges. The precise cause of this error will be explained in Chapter 7.*

Common Programming Error 2.5

Preceding a variable included in a `printf` *statement with an ampersand when, in fact, that variable should not be preceded by an ampersand.*

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.
- In the addition program of Fig. 2.5, when the statement
  - `scanf( "%d", &integer1 ); // read an integer`
- is executed, the value entered 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` as shown in Fig. 2.6.

Fig. 2.6 | Memory location showing the name and value of a variable.

2.4 Memory Concepts (Cont.)

- Whenever a value is placed in a memory location, the value replaces the previous value in that location; thus, this process is said to be destructive.
- When the statement
  - `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.

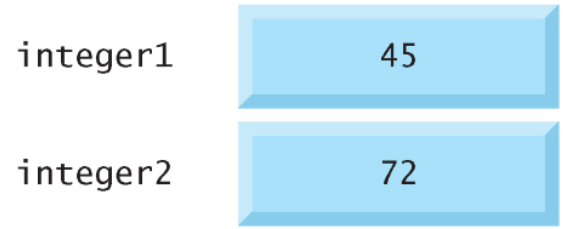

Fig. 2.7 | Memory locations after both variables are input.

2.4 Memory Concepts (Cont.)

- Once the program has obtained values for `integer1` and `integer2`, it adds these values and places the total into variable `sum`.
- `sum = integer1 + integer2; // assign total to sum`
  - replaces whatever value was stored in `sum`.

2.4 Memory Concepts (Cont.)

- This occurs when the calculated total of `integer1` and `integer2` is placed into location `sum` (destroying the value already in `sum`).
- After `sum` is calculated, memory appears as in Fig. 2.8.
- The values of `integer1` and `integer2` appear exactly as they did before they were used in the calculation.

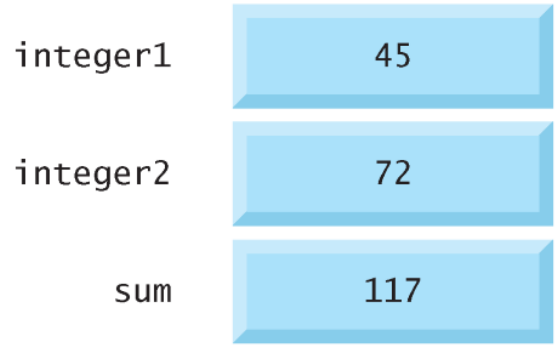

Fig. 2.8 | Memory locations after a calculation.

2.4 Memory Concepts (Cont.)

- They 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.

2.6 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 whether a person's grade on an exam is greater than or equal to 60 and whether the program should print the message "Congratulations! You passed."
- This section 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 Decision Making: Equality and Relational Operators

- If the condition is true (i.e., the condition is met) the statement in the body of the `if` statement is executed.
- If the condition is false (i.e., the condition isn't met) 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 summarized in Fig. 2.12.

2.6 Decision Making: Equality and Relational Operators

- 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*.

2.6 Decision Making: Equality and Relational Operators

- Figure 2.13 uses six `if` statements to compare two numbers entered by the user.
- If the condition in any of these `if` statements is true, the `printf` statement associated with that `if` executes.

```
1  // Fig. 2.13: fig02_13.c
2  // Using if statements, relational
3  // operators, and equality operators.
4  #include <stdio.h>
5
6  // function main begins program execution
7  int main( void )
8  {
9     printf( "Enter two integers, and I will tell you\n" );
10    printf( "the relationships they satisfy: " );
11
12    int num1; // first number to be read from user
13    int num2; // second number to be read from user
14
15    scanf( "%d %d", &num1, &num2 ); // read two integers
16
17    if ( num1 == num2 ) {
18       printf( "%d is equal to %d\n", num1, num2 );
19    } // end if
20
```

Fig. 2.13 | Using `if` statements, relational operators, and equality operators. (Part 1 of 3.)

```
21      if ( num1 != num2 ) {
22         printf( "%d is not equal to %d\n", num1, num2 );
23      } // end if
24
25      if ( num1 < num2 ) {
26         printf( "%d is less than %d\n", num1, num2 );
27      } // end if
28
29      if ( num1 > num2 ) {
30         printf( "%d is greater than %d\n", num1, num2 );
31      } // end if
32
33      if ( num1 <= num2 ) {
34         printf( "%d is less than or equal to %d\n", num1, num2 );
35      } // end if
36
37      if ( num1 >= num2 ) {
38         printf( "%d is greater than or equal to %d\n", num1, num2 );
39      } // end if
40   } // end function main
```

Fig. 2.13 | Using `if` statements, relational operators, and equality operators. (Part 2 of 3.)

```
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: 22 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.)

2.6 Decision Making: Equality and Relational Operators

- The program uses `scanf` 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 the variable `num1`, and the second `%d` converts a value to be stored in variable `num2`.

Common Programming Error 2.9

Placing commas (when none are needed) between conversion specifiers in the format control string of a `scanf` *statement.*

2.6 Decision Making: Equality and Relational Operators

Comparing Numbers

- The if statement

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

  compares the values of variables `num1` and `num2` to test for equality.
- If the conditions are `true` in one or more of the `if` statements, the corresponding body statement displays an appropriate line of text.
- Indenting the body of each `if` statement and placing blank lines above and below each `if` statement enhances program readability.

Common Programming Error 2.10

Placing a semicolon immediately to the right of the right parenthesis after the condition in an `if` *statement.*

2.6 Decision Making: Equality and Relational Operators

- A left brace, `{`, begins the body of each `if` statement
- A corresponding right brace, `}`, ends each `if` statement's body
- Any number of statements can be placed in the body of an `if` statement.