# Overview of C
# Chapter 2

*Problem Solving & Program Design in C*

*Eighth Edition*

*Jeri R. Hanly & Elliot B. Koffman*

# Chapter Objectives

- To become familiar with the general form of a C program and the basic elements in a program
- To appreciate the importance of writing comments in a program
- To understand the use of data types and the differences between the data types int, double, and char
- To know how to declare variables

# Chapter Objectives

- To understand how to write assignment statements to change the value of variables
- To learn how C evaluates arithmetic expressions and how to write them in C
- To learn how to read data values into a program and to display results
- To understand how to write format strings for data entry and display

# Chapter Objectives

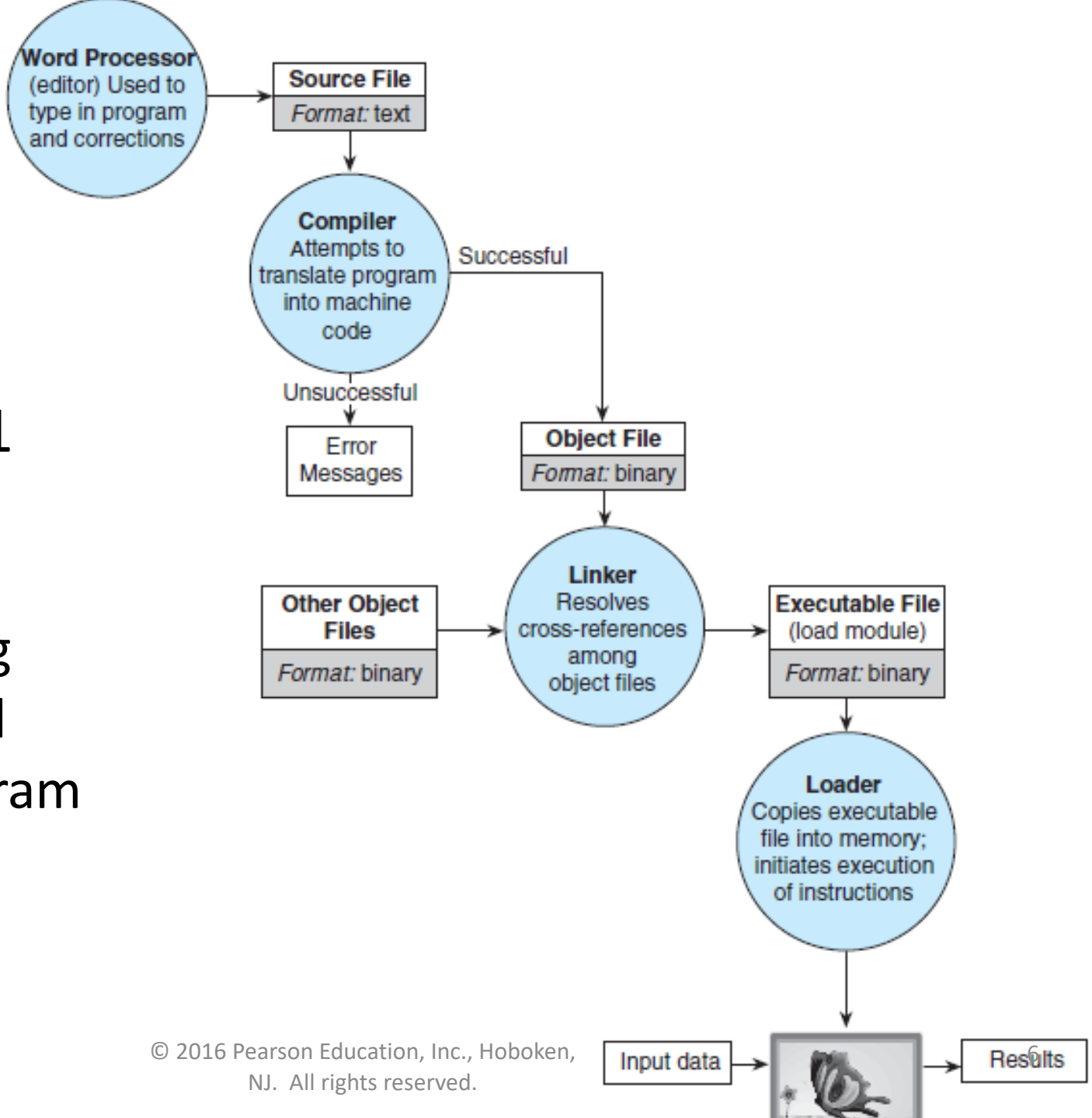
- To learn how to use redirection to enable the use of files for input/output
- To understand the differences between syntax errors, run-time errors, and logic errors, and how to avoid them and to correct them

# C

- A high-level programming language
- Developed in 1972 by Dennis Ritchie at AT&T Bell Labs
- Designed as the language to write the Unix operating system
- Resembles everyday English
- Very popular

# Figure 1.11

## Entering, Translating, and Running a High-Level Language Program



# Language Elements

- preprocessor
  - a system program that modifies a C program prior to its compilation
- library
  - a collection of useful functions and symbols that may be accessed by a program
  - each library has a standard header file whose name ends with the symbols ".h"

stdio.h

# Language Elements

- preprocessor directive
  - a C program line beginning with # that provides an instruction to the preprocessor

```
#include  <stdio.h>

#define  KMS_PER_MILE  1.609
```

# Language Elements

- constant macro
  - a name that is replaced by a particular constant value before the program is sent to the compiler

```
#define  KMS_PER_MILE  1.609
```

*constant*

*constant macro*

```
kms  =  KMS_PER_MILE  *  miles;
```

# Language Elements

- comment
  - text beginning with /* and ending with */ that provides supplementary information but is ignored by the preprocessor and compiler
  - for single-line comments, can use // (introduced in C99)

```
/*  Get the distance in miles  */
// Get the distance in miles
```

# Figure 2.1 C Language Elements in Miles-to-Kilometers Conversion Program

```
/*
 * 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);
}
```

standard header file
comment
preprocessor directive
constant
reserved word
variable
comment
standard identifier
special symbol
reserved word
punctuation
special symbol

# Function main

- Every C program has a main function.

```
int main (void)
```

- These lines mark the beginning of the main function where program execution begins.

# Function main

- declarations
  - the part of a program that tells the compiler the names of memory cells in a program
- executable statements
  - program lines that are converted to machine language instructions and executed by the computer

# Language Elements

- reserved word
  - a word that has a special meaning in C
  - identifiers from standard library and names for memory cells
  - appear in lowercase
  - cannot be used for other purposes

```
return  0;
```

# Language Elements

- standard identifier
  - a word having special meaning but one that a programmer may redefine
  - *redefinition is not recommended*

```
printf("Enter the distance in miles> ");
```

# User-defined identifiers

- These name memory cells that hold data and program results and to name operations that we define.
- Naming rules:
  1. An identifier must consist only of letters, digits and underscores.
  2. An identifier cannot begin with a digit.
  3. A C reserved word cannot be used an an identifier.
  4. An identifier defined in a C standard library should not be redefined.

# Figure 2.1 C Language Elements in Miles-to-Kilometers Conversion Program

```
/*
 * 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);
}
```

standard header file
comment
preprocessor directive
constant
reserved word
variable
comment
standard identifier
special symbol
reserved word
punctuation
special symbol

# Variable Declarations

- variable
  - a name associated with a memory cell whose value can change
- variable declarations
  - statements that communicate to the compiler the names of variables in the program and the kind of information stored in each variable

# Variable Declarations

- C requires you to declare every variable used in a program.
- A variable declaration begins with an identifier that tells the C compiler the type of data store in a particular variable.

```
int  hours;
double  miles;
```

# Data Types

- int
  - a whole number
  - 435
- double
  - a real number with an integral part and a fractional part separated by a decimal point
  - 3.14159
- char
  - an individual character value
  - enclosed in single quotes
  - ‘A’, ‘z’, ‘2’, ‘9’, ‘*’, ‘!’

# Figure 2.2
## Internal Format of Type int and Type double

type `int` format

| binary number |
|---|

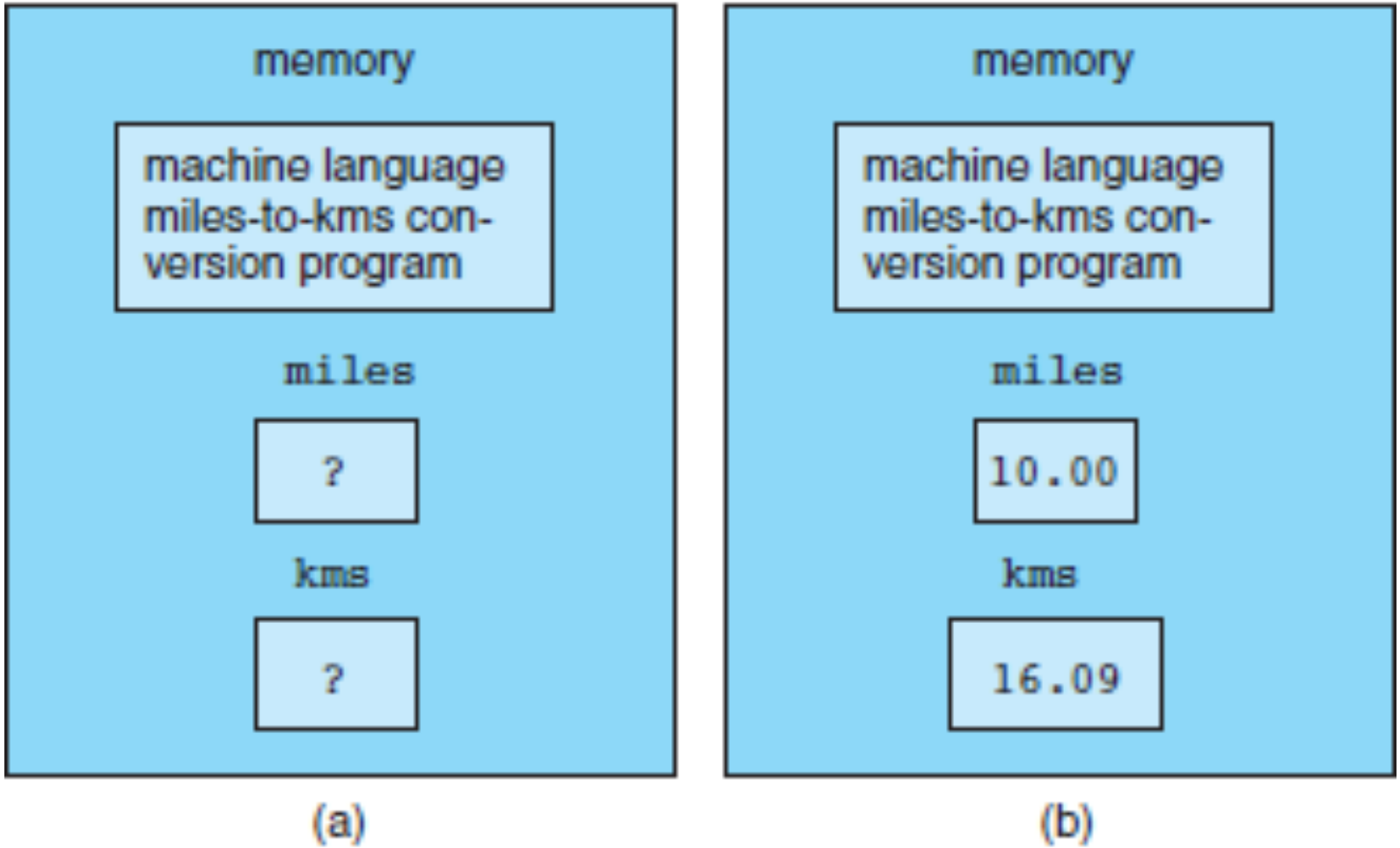
type `double` format

| sign | exponent | mantissa |
|---|---|---|

# Executable Statements

- Follow the declarations in a function.
- Used to write or code the algorithm and its refinements.
- Are translated into machine language by the compiler.
- The computer executes the machine language version.

# Figure 2.3 Memory(a) Before and (b) After Execution of a Program

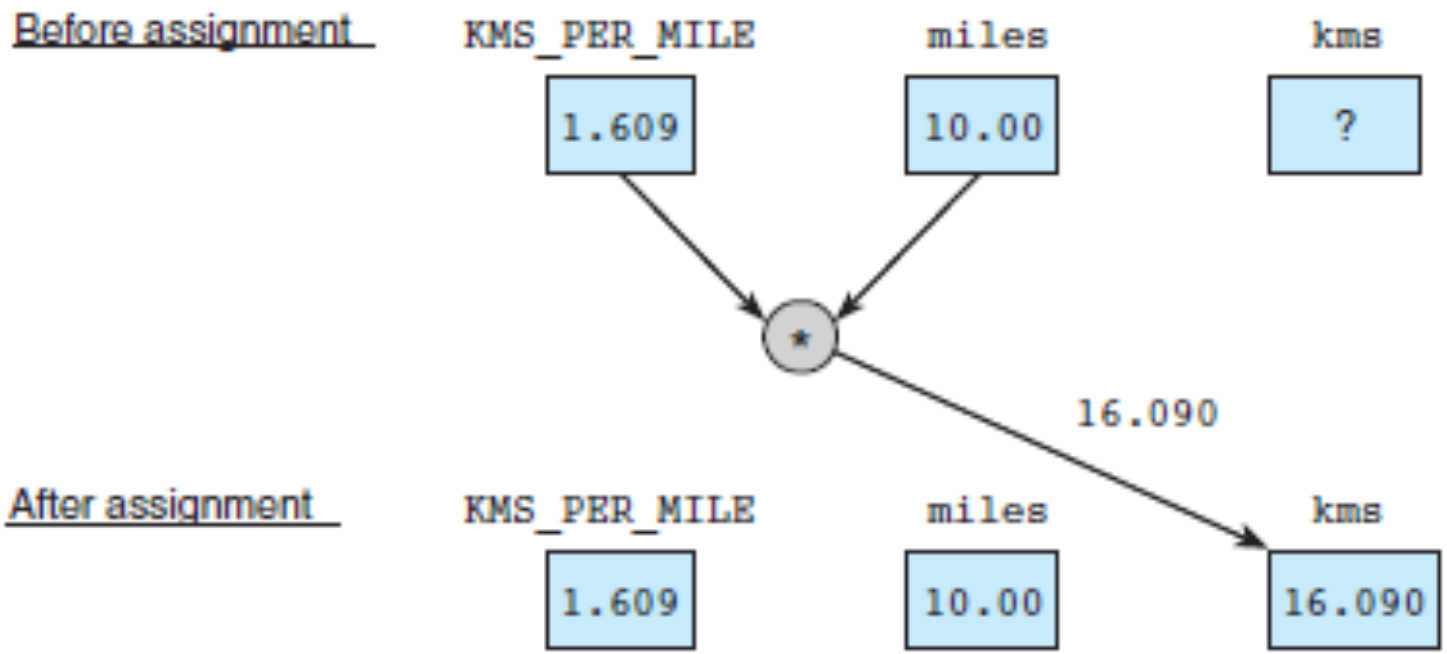


# Executable Statements

- assignment statement
  - an instruction that stores a value of a computational result in a variable

```
kms = KMS_PER_MILE * miles;
```

# Figure 2.4
# Effect of kms = KMS_PER_MILE * miles;

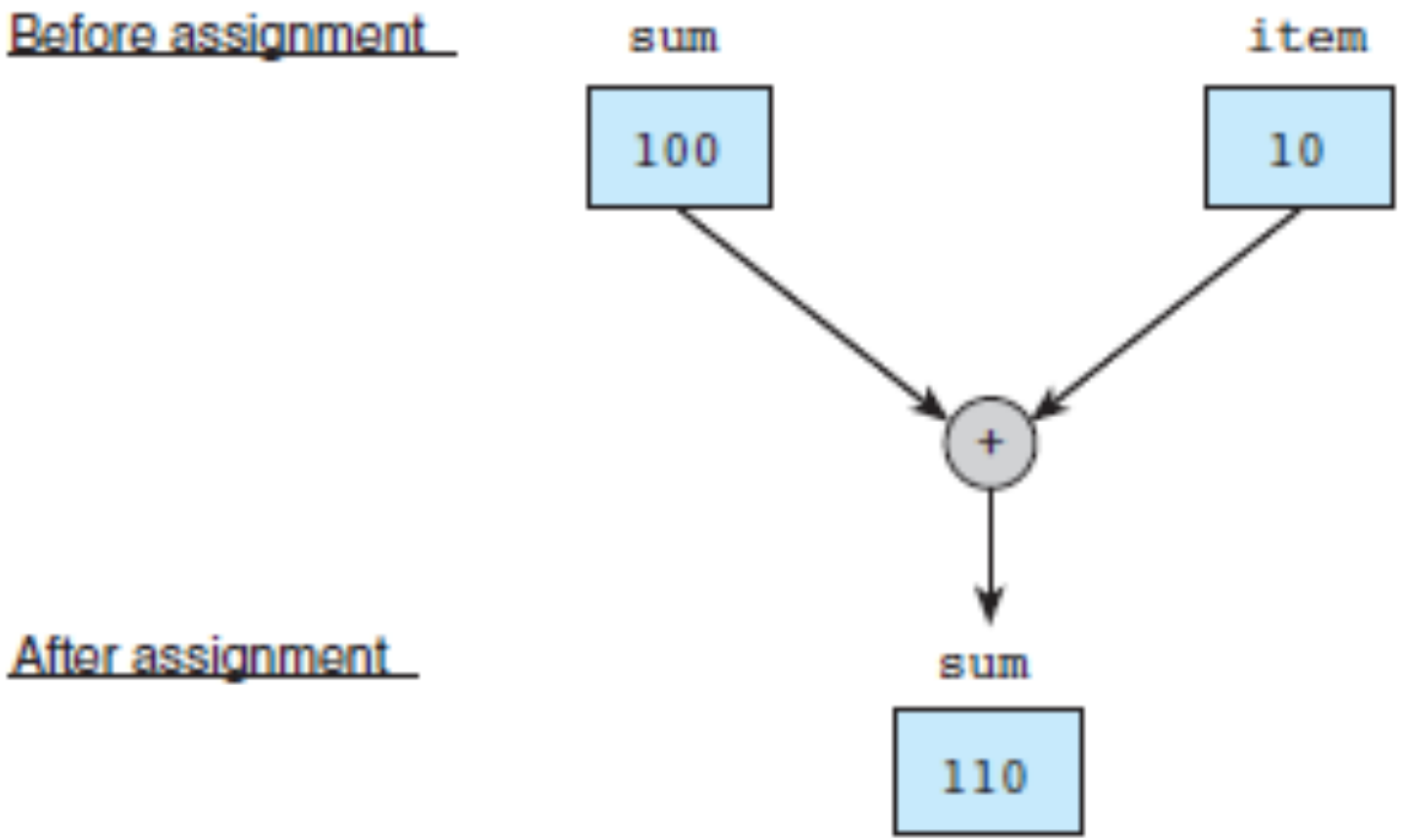


# Executable Statements

- Assignment is not the same as an algebraic equation.
- The expression to the right of the assignment operator is first evaluated.
- Then the variable on the left side of the assignment operator is assigned the value of that expression.

```
sum = sum + item;
```

# Figure 2.5
# Effect of sum = sum + item;



# Input /Output Operations and Functions

- input operation
  - an instruction that copies data from an input device into memory
- output operation
  - an instruction that displays information stored in memory
- input/output function
  - a C function that performs an input or output operation
- function call
  - calling or activating  function

# The printf Function

- Displays a line of program output.
- Useful for seeing the results of a program execution.

```
printf("That equals %f kilometers. \n", kms);
```

# The printf Function

- function argument
  - enclosed in parentheses following the function name
  - provides information needed by the function

```
printf("That equals %f kilometers. \n", kms);
```

function name

# The printf Function

- format string
  - in a call to printf, a string of characters enclosed in quotes, which specifies the form of the output line

```
printf("That equals %f kilometers. \n", kms);
```

# The printf Function

- print list
  - in a call to printf, the variables or expressions whose values are displayed
- placeholder
  - a symbol beginning with % in a format string that indicates where to display the output value

```
printf("That equals %f kilometers. \n", kms);
```

# Placeholders in format string

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

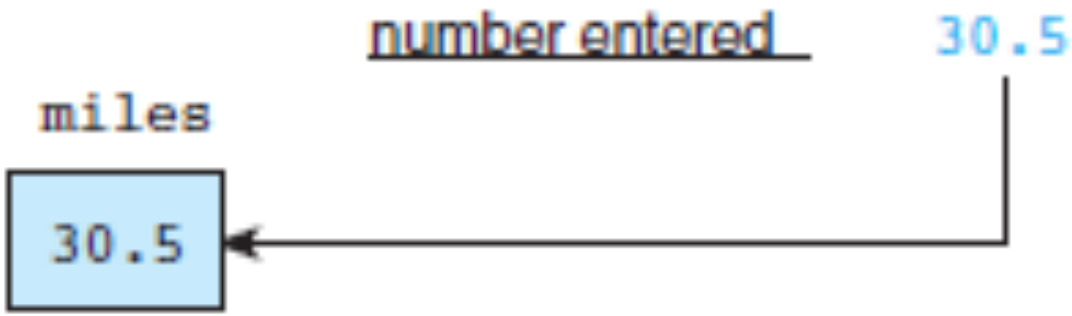
# The scanf Function

- Copies data from the standard input device (usually the keyboard) into a variable.

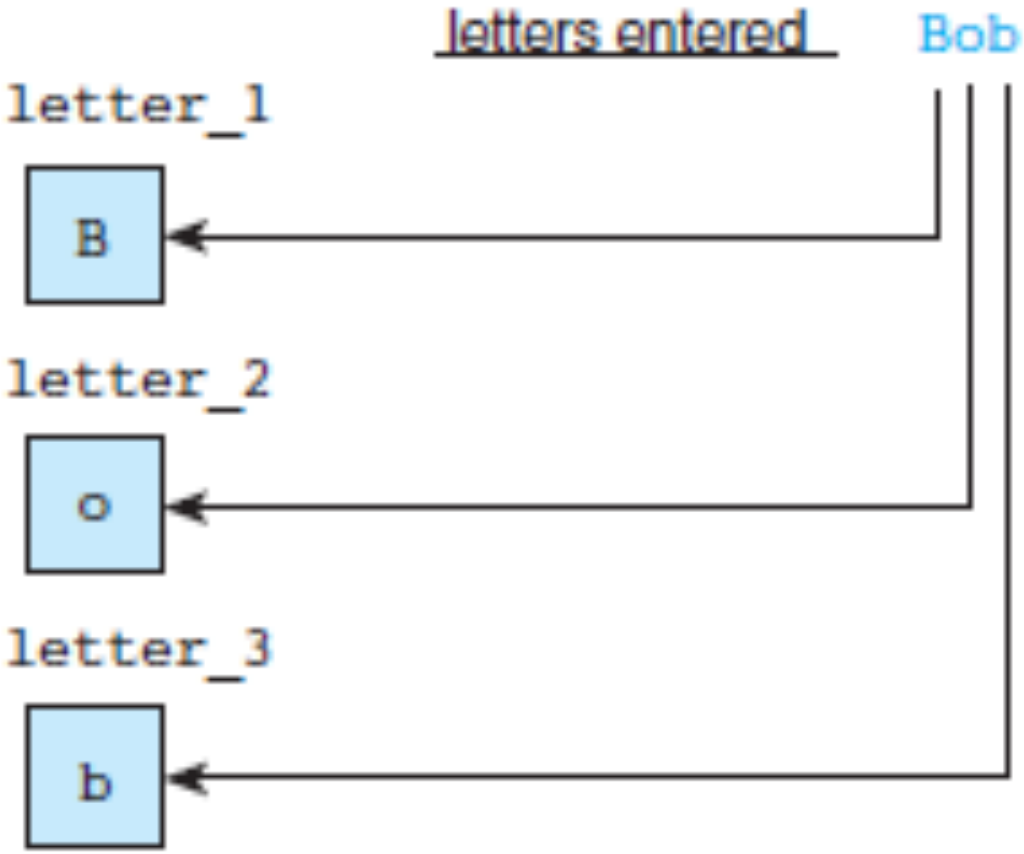
```
scanf("%lf", miles);
```

```
scanf("%c%c%c", &letter_1, &letter_2, &letter_3);
```

# Figure 2.6
# Effect of scanf("%lf", &miles);



# Figure 2.7
# Scanning Data Line Bob



# The return Statement

- Last line in the main function.
- Transfers control from your program to the operating system.
- The value 0 is optional; indicates that your program executed without an error.

```
return (0);
```

# Figure 2.8
# General Form of a C Program

*preprocessor directives*
*main function heading*
{
    *declarations*
    *executable statements*
}

# Program Style

- Use spaces consistently and carefully.
  - One is required between consecutive words in a program.
  - Improves readability.
- Use comments to document your program.
  - Also enhances readability.

# Arithmetic Operators

| Arithmetic Operator | Meaning | Example |
|---|---|---|
| + | addition | 5 + 2 is 7<br>5.0 + 2.0 is 7.0 |
| – | subtraction | 5 – 2 is 3<br>5.0 – 2.0 is 3.0 |
| * | multiplication | 5 * 2 is 10<br>5.0 * 2.0 is 10.0 |
| / | division | 5.0 / 2.0 is 2.5<br>5 / 2 is 2 |
| % | remainder | 5 % 2 is 1 |

# Data Type of an Expression

- mixed-type expression
  - an expression with operands of different types
- mixed-type assignment
  - the expression being evaluated and the variable to which it is assigned have different data types
- type cast
  - converting an expression to a different type by writing the desired type in parentheses in front of the expression

# Expressions with Multiple Operators

- unary operator
  - an operator with one operand
  - unary plus (+), unary negation (-)
  - ex. x = -y;
- binary operator
  - an operator with two operands
  - ex. x = y + z;

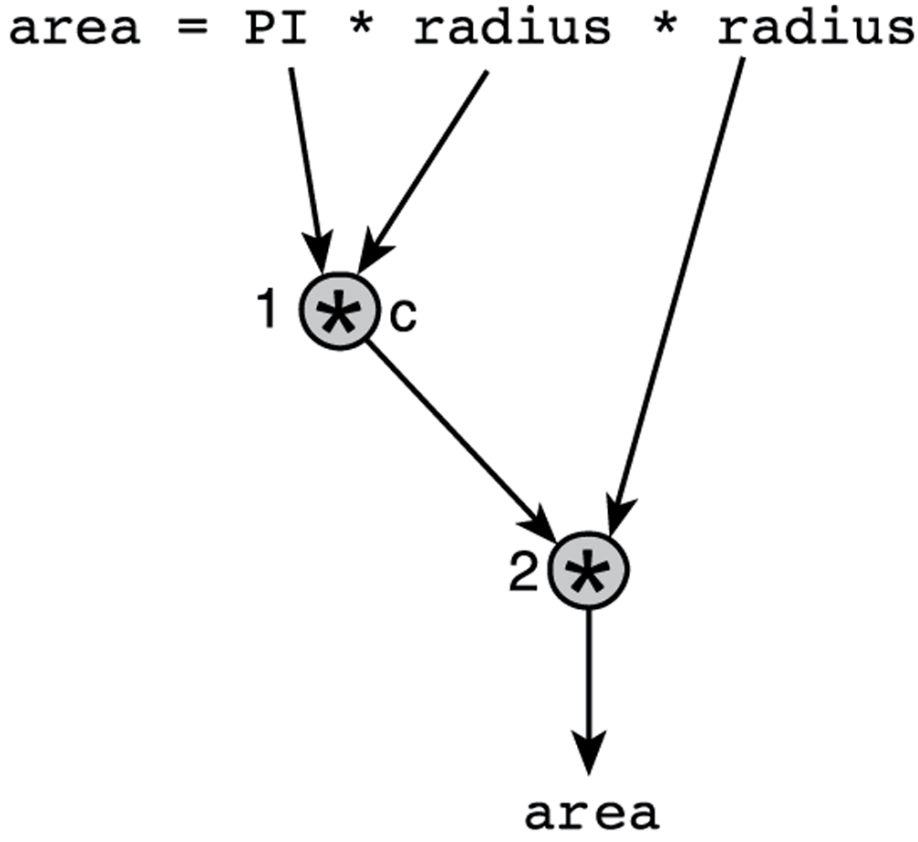
# Rules for Evaluating Expressions

- Parentheses rule
  - all expression must be evaluated separately
  - nested parentheses evaluated from the inside out
  - innermost expression evaluated first
- Operator precedence rule
  - unary +, - first
  - *, /, % next
  - binary +, - last

# Rules for Evaluating Expressions

- Right Associativity
  - Unary operators in the same subexpression and at the same precedence level are evaluate right to left.
- Left Associativity
  - Binary operators in the same subexpression and at the same precedence lever are evaluated left to right.

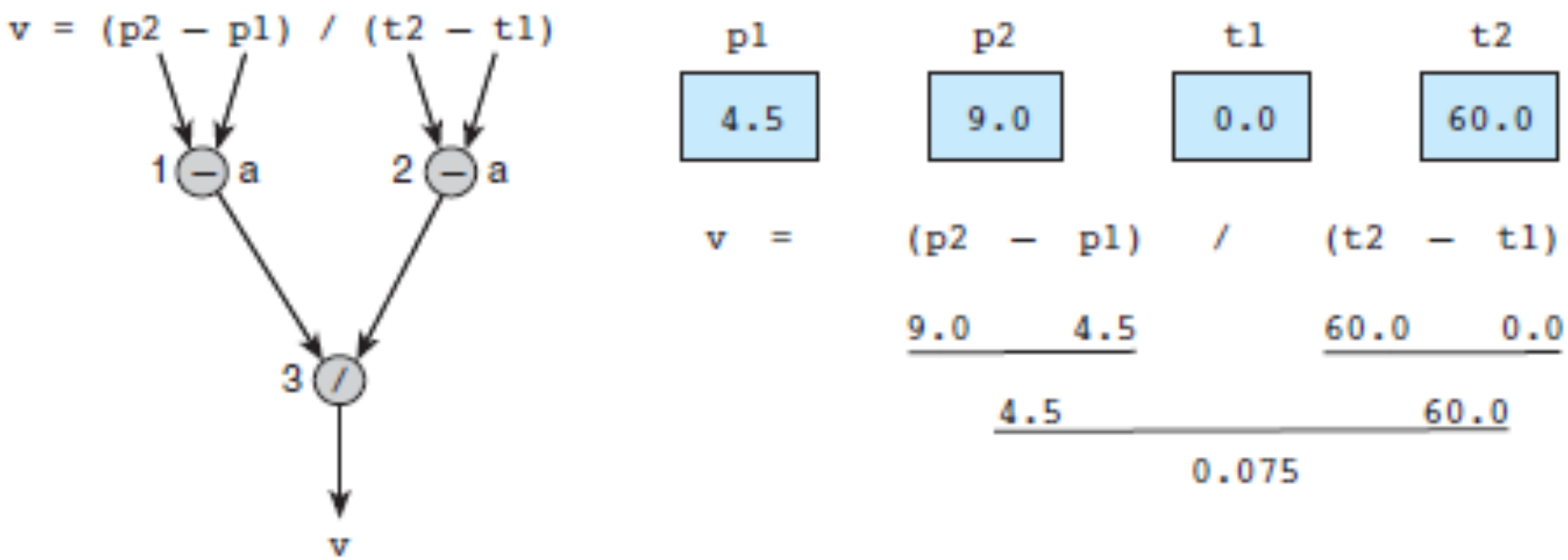
# Figure 2.9
# Evaluation Tree for
# area = PI * radius * radius;



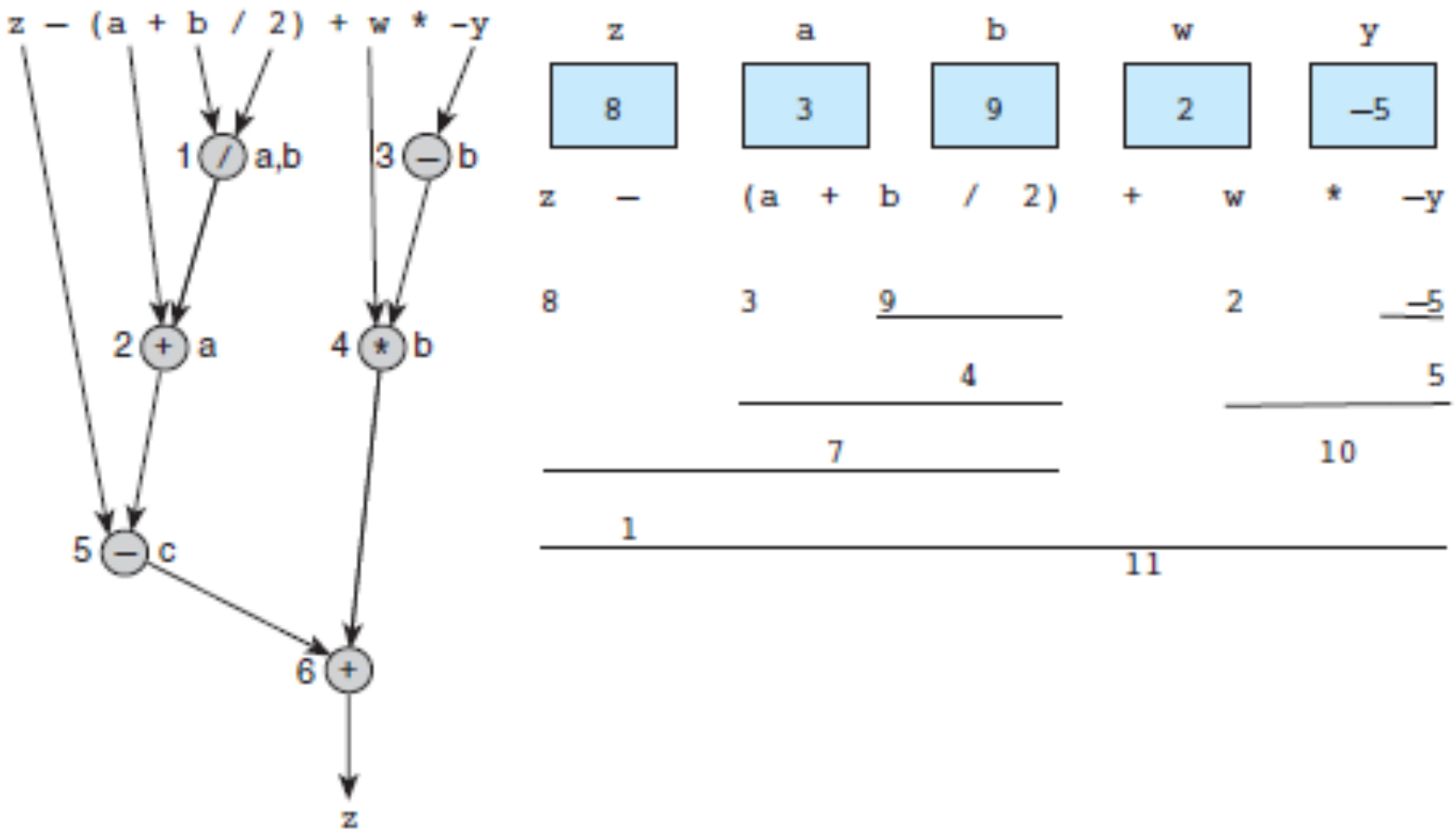
# Figure 2.10
# Step-by-Step Expression Evaluation

```
area    =      PI    *   radius   *   radius
            3.14159        2.0          2.0
                  6.28318
                                 12.56636
```

# Figure 2.11
# Evaluation Tree and Evaluation for v = (p2 - p1) / (t2 - t1);



# Figure 2.12
# Evaluation Tree and Evaluation for z - (a + b / 2) + w * -y



# Numerical Inaccuracies

- representational error
  - an error due to coding a real number as a finite number of binary digits
- cancellation error
  - an error resulting from applying an arithmetic operation to operands of vastly different magnitudes
  - effect of smaller operand is lost

# Numerical Inaccuracies

- arithmetic underflow
  - an error in which a very small computational result is represented as zero
- arithmetic overflow
  - an error that is an attempt to represent a computational result that is too large

# Supermarket Coin Value Program

Case Study

# Figure 2.13

```
/*
 * Determines the value of a collection of coins.
 */
#include <stdio.h>
int
main(void)
{
 char first, middle, last; /* input - 3 initials              */
 int pennies, nickels;     /* input - count of each coin type */
 int dimes, quarters;      /* input - count of each coin type */
 int dollars;              /* input - count of each coin type */
 int change;               /* output - change amount          */
 int total_dollars;        /* output - dollar amount          */
 int total_cents;          /* total cents                     */

     /* Get and display the customer's initials. */
     printf("Type in your 3 initials and press return> ");
     scanf("%c%c%c", &first, &middle, &last);
     printf("\n%c%c%c, please enter your coin information.\n",
            first, middle, last);

     /* Get the count of each kind of coin. */
     printf("Number of $ coins > ");
     scanf("%d", &dollars);
     printf("Number of quarters> ");
     scanf("%d", &quarters);
     printf("Number of dimes   > ");
     scanf("%d", &dimes);
     printf("Number of nickels > ");
     scanf("%d", &nickels);
     printf("Number of pennies > ");
     scanf("%d", &pennies);

     /* Compute the total value in cents. */
     total_cents = 100 * dollars +25 * quarters + 10 * dimes +
                   5 * nickels + pennies;

     /* Find the value in dollars and change. */
     total_dollars = total_cents / 100;
     change = total_cents % 100;

     /* Display the credit slip with value in dollars and change. */
```

*(Continued)*

# Figure 2.13

```
    printf("\n\n%c%c%c Coin Credit\nDollars: %d\nChange: %d cents\n",
            first, middle, last, total_dollars, change);

    return (0);
}

Type in your 3 initials and press return> JRH
JRH, please enter your coin information.
Number of $ coins > 2
Number of quarters> 14
Number of dimes   > 12
Number of nickels > 25
Number of pennies > 131

JRH Coin Credit
Dollars: 9
Change:  26 cents
```

# Formatting Numbers in Program Output

- field width
  - the number of columns used to display a value

- When formatting doubles, you must indicate the total field width needed and the number of decimal places desired.

# Interactive Mode, Batch Mode, and Data Files

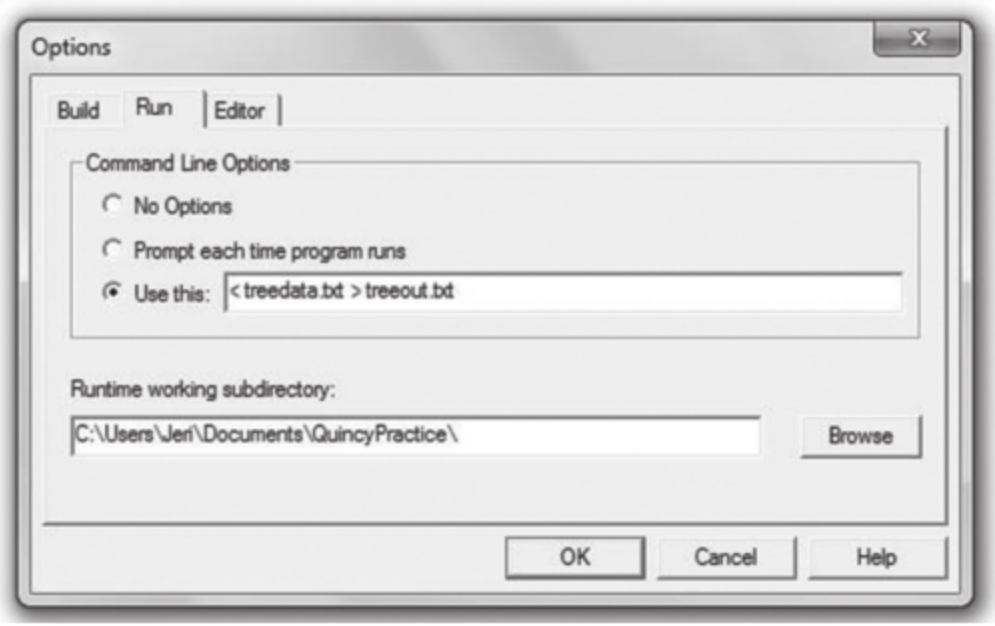
- interactive mode
  - a mode of program execution in which the user responds to prompts by entering (typing in) data
- batch mode
  - a mode of program execution in which the program scans its data from a previously prepared data file

# Figure 2.14 Batch Version of Miles-to-Kilometers Conversion Program

```
/* 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 and echo the distance in miles. */
     scanf("%lf", &miles);
     printf("The distance in miles is %.2f.\n", miles);

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

     /* Display the distance in kilometers. */
     printf("That equals %.2f kilometers.\n", kms);

     return (0);
}
```

```
The distance in miles is 112.00.
That equals 180.21 kilometers.
```

# Figure 2.15
# An IDE (Quincy 2005) Allows Developer to Set Command-Line Options



# Common Programming Errors

- debugging
  - removing errors from a program
- syntax error
  - a violation of the C grammar rules
  - detected during program translation (compilation)
- run-time error
  - an attempt to perform an invalid operation
  - detected during program execution
- logic errors
  - an error caused by following an incorrect algorithm

# Figure 2.16

## Compiler Listing of a Program with Syntax Errors

```
221 /* Converts distances from miles to kilometers. */
222
223 #include <stdio.h>          /* printf, scanf definitions  */
266 #define KMS_PER_MILE 1.609  /* conversion constant        */
267
268 int
269 main(void)
270 {
271       double kms
272
273       /* Get the distance in miles. */
274       printf("Enter the distance in miles> ");
***** Semicolon added at the end of the previous source line
275       scanf("%lf", &miles);
***** Identifier "miles" is not declared within this scope
***** Invalid operand of address-of operator
276
277       /* Convert the distance to kilometers. */
278       kms = KMS_PER_MILE * miles;
***** Identifier "miles" is not declared within this scope
279
280       /* Display the distance in kilometers. * /
281       printf("That equals %f kilometers.\n", kms);
282
283       return (0);
284 }
***** Unexpected end-of-file encountered in a comment
***** "}" inserted before end-of-file
```

# Figure 2.17

A Program with a Run-Time Error

```
111 #include <stdio.h>
262
263 int
264 main(void)
265 {
266         int    first, second;
267         double temp, ans;
268
269         printf("Enter two integers> ");
270         scanf("%d%d", &first, &second);
271         temp = second / first;
272         ans = first / temp;
273         printf("The result is %.3f\n", ans);
274
275         return (0);
276 }

Enter two integers> 14 3
Arithmetic fault, divide by zero at line 272 of routine main
```

# Figure 2.18

Revised Start of main Function for Supermarket Coin Value Program

```
int
main(void)
{
     char first, middle, last; /* input - 3 initials            */
     int pennies, nickels;   /* input - count of each coin type */
     int dimes, quarters;    /* input - count of each coin type */
     int dollars;            /* input - count of each coin type */
     int change;                /* output - change amount       */
     int total_dollars;         /* output - dollar amount       */
     int total_cents;           /* total cents                  */
     int year;                  /* input — year                 */

     /* Get the current year.                                   */
     printf("Enter the current year and press return> ");
     scanf("%d", &year);

     /* Get and display the customer's initials.                */
     printf("Type in 3 initials and press return> ");
     scanf("%c%c%c", &first, &middle, &last);
     printf("\n%c%c%c, please enter your coin information for %d.\n",
            first, middle, last, year);
...
```

# Figure 2.19

A Program That Produces Incorrect Results Due to & Omission

```
#include <stdio.h>

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

    printf("Enter two integers> ");
    scanf("%d%d", first, second); /* ERROR!! should be &first, &second */
    sum = first + second;
    printf("%d + %d = %d\n", first, second, sum);

    return (0);
}
```

```
Enter two integers> 14   3
5971289 + 5971297 = 11942586
```

# Wrap Up

- Every C program has preprocessor directives and a main function.
- The main function contains variable declarations and executable statements.
- C's data types enable the compiler to determine how to store a value in memory and what operations can be performed on that value.