

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

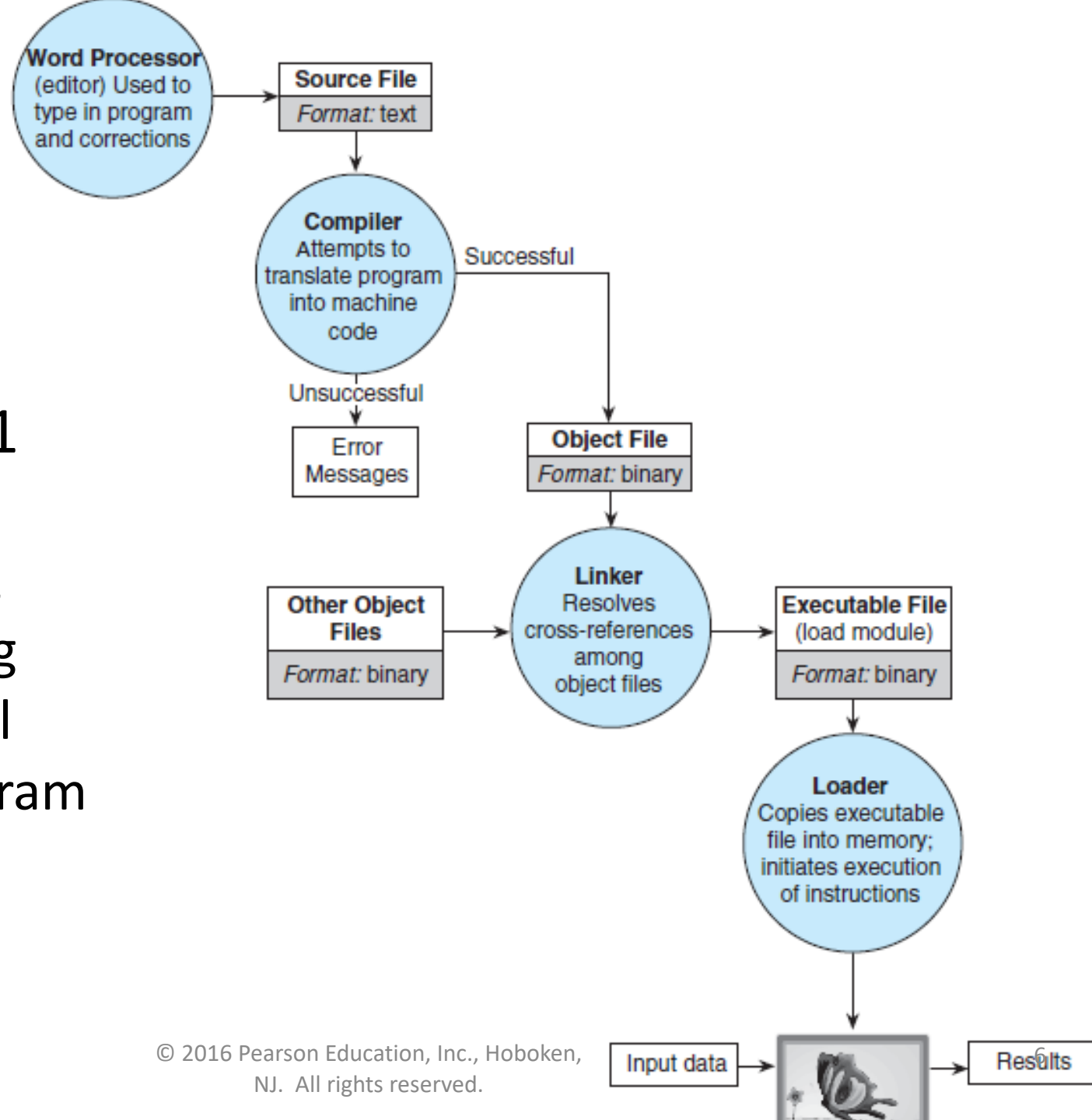
Chapter Objectives

- 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”

The text "stdio.h" is enclosed in a purple oval, highlighting it as an example of a standard header file name.

Language Elements

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



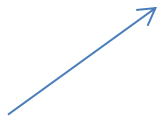
```
#include <stdio.h>  
  
#define KMS_PER_MILE 1.609
```


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



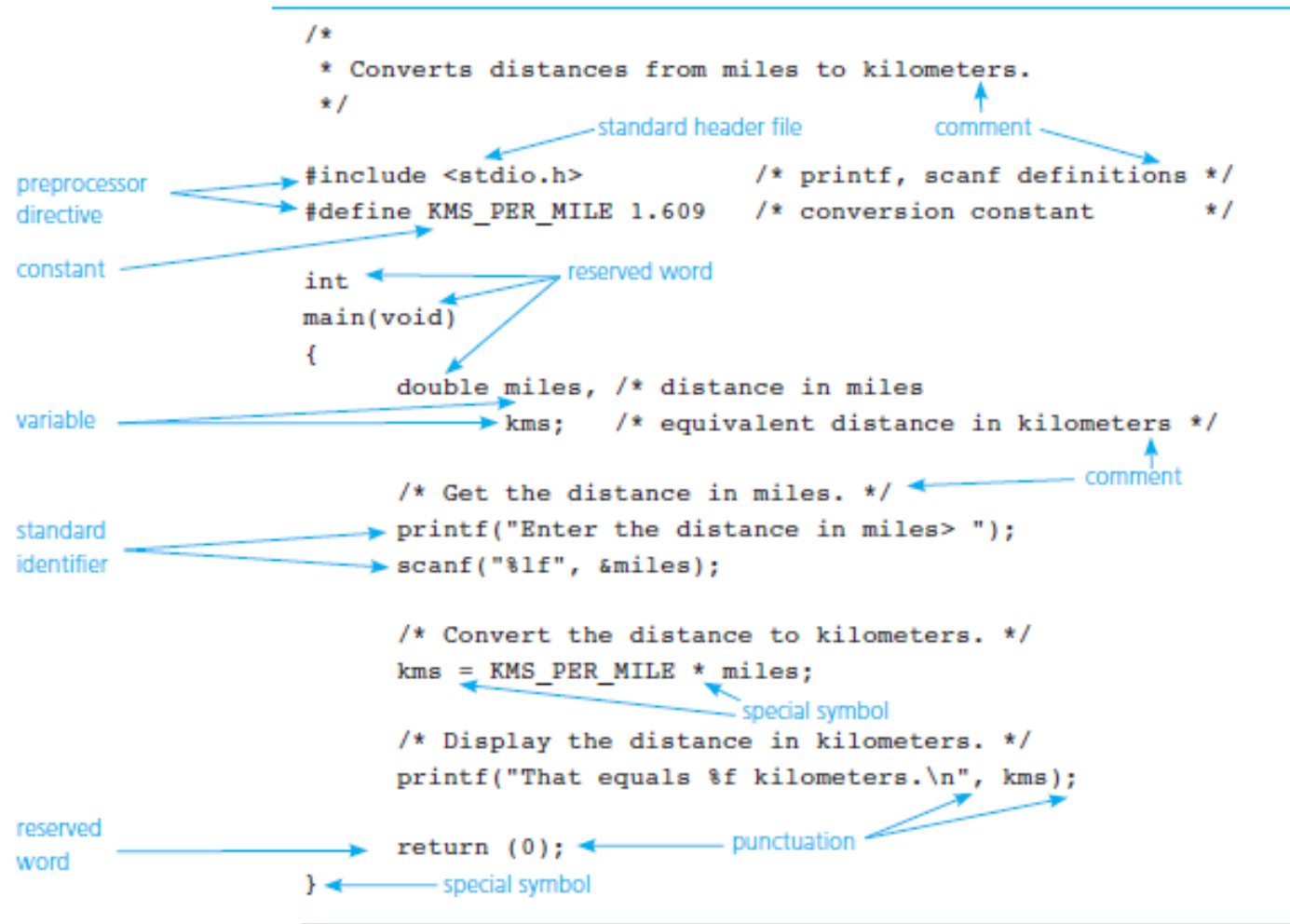
```
kms = KMS_PER_MILE * miles;
```

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



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

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', '*', '!'

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.

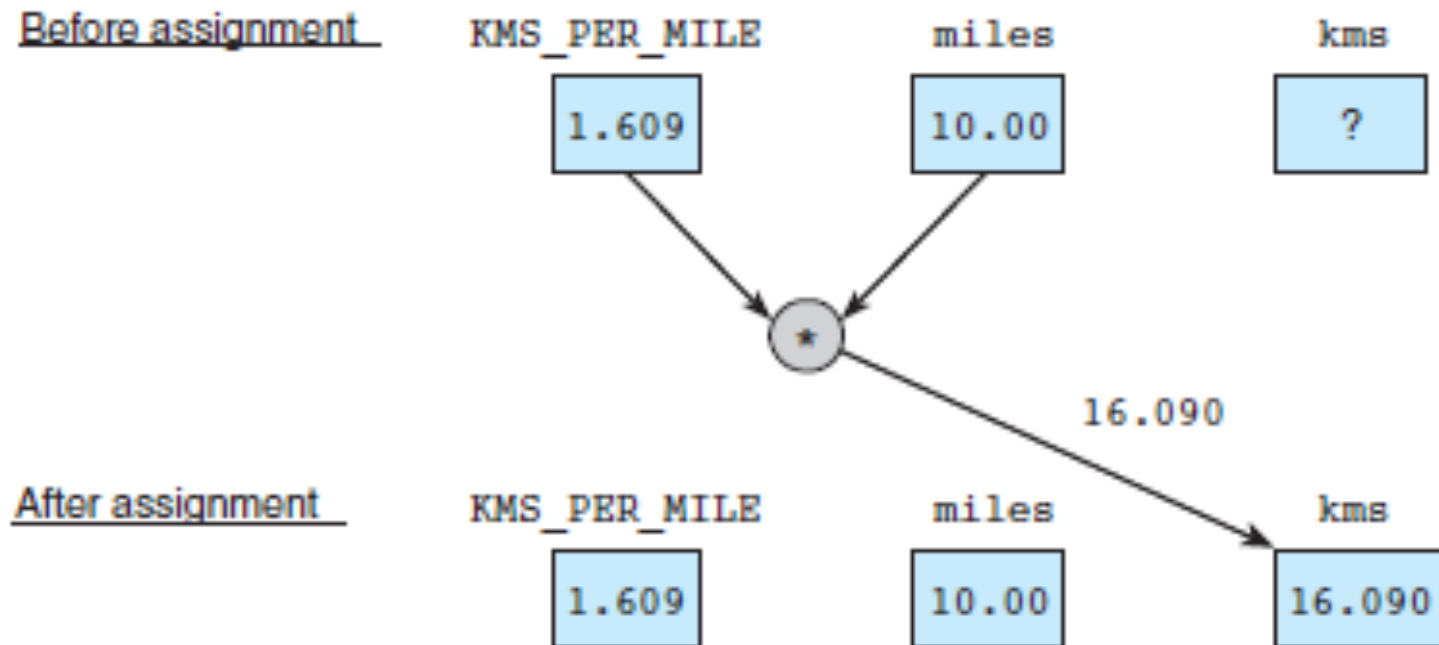
Executable Statements

- 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 $\text{kms} = \text{KMS_PER_MILE} * \text{miles}$;



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;`

The `printf` Function

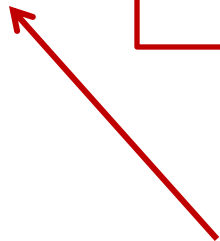
- Displays a line of program output.
- Useful for seeing the results of a program execution.

```
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);`



Let's write a C program

That stores an int, double, and char variable, and prints them all out.

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.

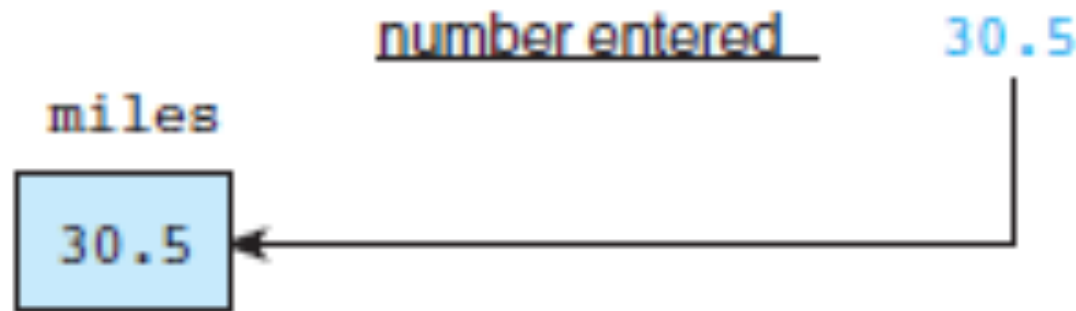
```
scanf("%lf", miles);
```

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

- Must pass address of variable to store using the addressof operator (&)

Figure 2.6

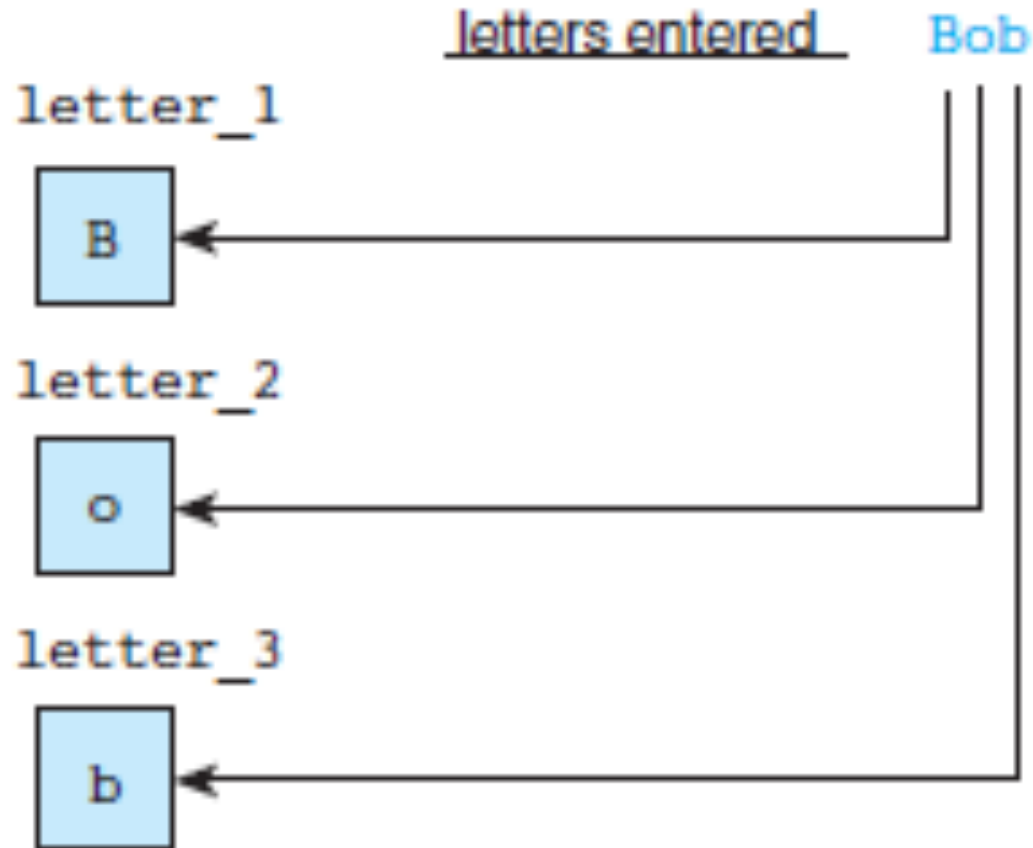
Effect of `scanf("%lf", &miles);`



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

Figure 2.7

Scanning Data Line **Bob**



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

Let's edit our C program

So that it gets the data from the user.

The `return` Statement

- Last line in the main function.
- Transfers control from your program to the operating system.
- The value 0 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  
}
```


Arithmetic Operators

Arithmetic Operator	Meaning	Example
+	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 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

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 (setting sign)
 - `*`, `/`, `%` next
 - binary `+`, `-` last
- Note prefix and postfix increment/decrement!
 - `++a` and `--a` are executed before value is used
 - `a++` and `a--` are executed after value is used

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;`

`area = PI * radius * radius`

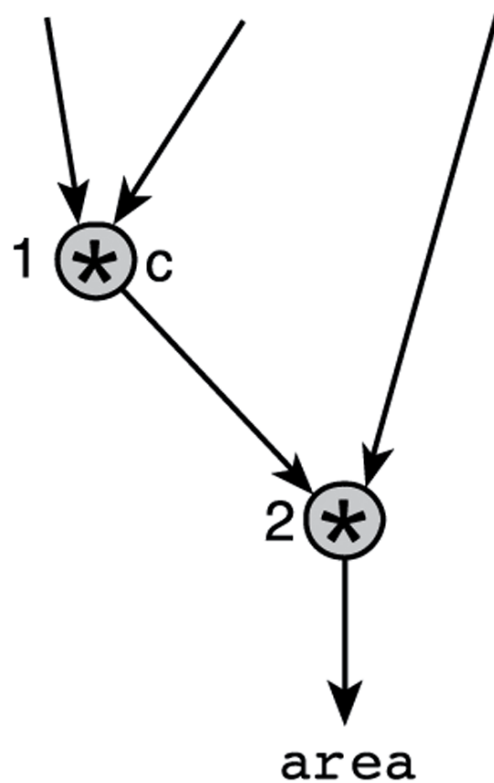


Figure 2.10

Step-by-Step Expression Evaluation

$$\begin{array}{rccccccc} \text{area} & = & & \text{PI} & * & \text{radius} & * & \text{radius} \\ & & & 3.14159 & & 2.0 & & 2.0 \\ & & & \hline & & & 6.28318 & & & & \\ & & & & & & & \hline & & & & & & & 12.56636 \end{array}$$

Figure 2.11

Evaluation Tree and Evaluation for $v = (p2 - p1) / (t2 - t1)$;

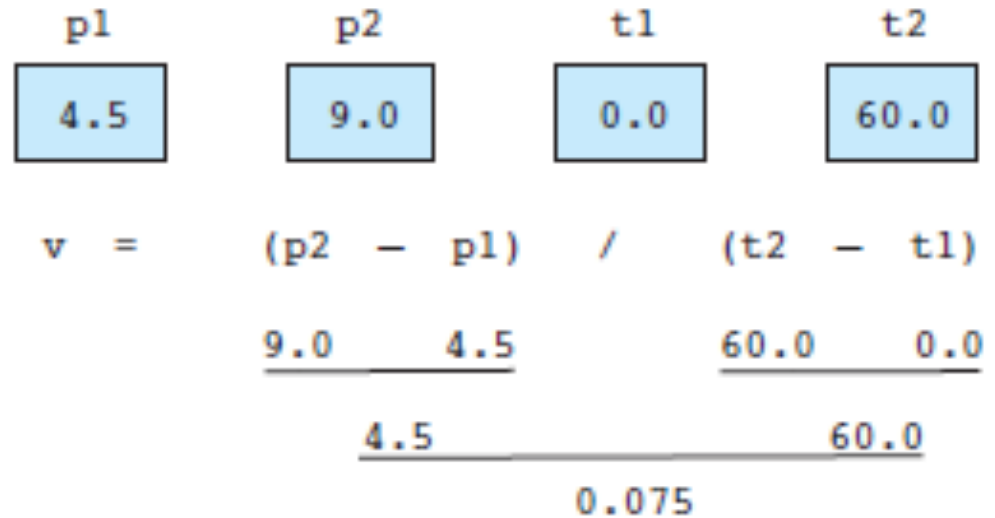
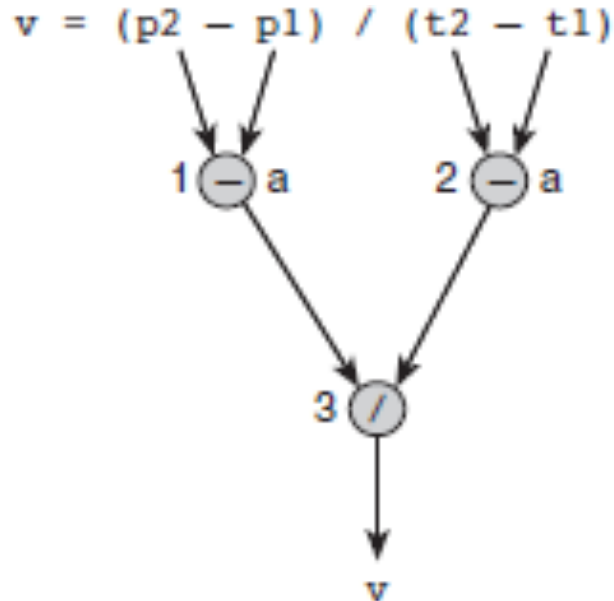
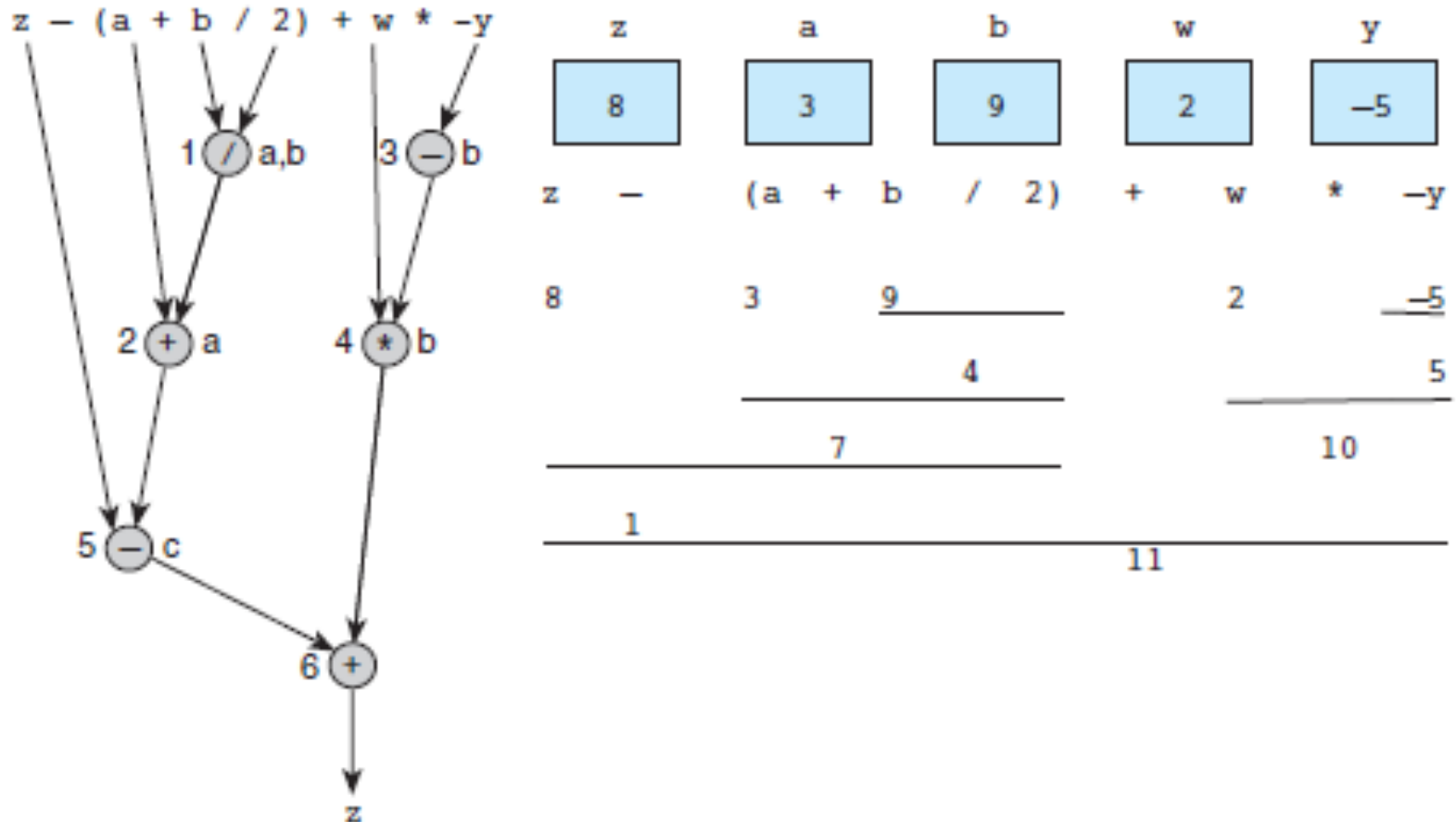


Figure 2.12

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



Let's edit our C program

To have an add function that takes in two integer arguments and returns their sum.

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

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.

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 2.17

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 2.19

A Program That Produces Incorrect Results Due to & Omission

```
1. #include <stdio.h>
2.
3. int
4. main(void)
5. {
6.     int    first, second, sum;
7.
8.     printf("Enter two integers> ");
9.     scanf("%d%d", first, second); /* ERROR!! should be &first, &second */
10.    sum = first + second;
11.    printf("%d + %d = %d\n", first, second, sum);
12.
13.    return (0);
14. }
```

```
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.