Chapter 2: Overview of C

Problem Solving & Program Design in C

Eighth Edition

By Jeri R. Hanly &
Elliot B. Koffman

Addison Wesley
is an imprint of
PEARSON

Outline

2.1 C LANGUAGE ELEMENTS

2.2 VARIABLE DECLARATIONS AND DATA TYPES

2.3 EXECUTABLE STATEMENTS

2.4 GENERAL FORM OF A C PROGRAM

2.5 ARITHMETIC EXPRESSIONS

- *CASE STUDY: Supermarket Coin Processor*

2.6 FORMATTING NUMBERS IN PROGRAM OUTPUT

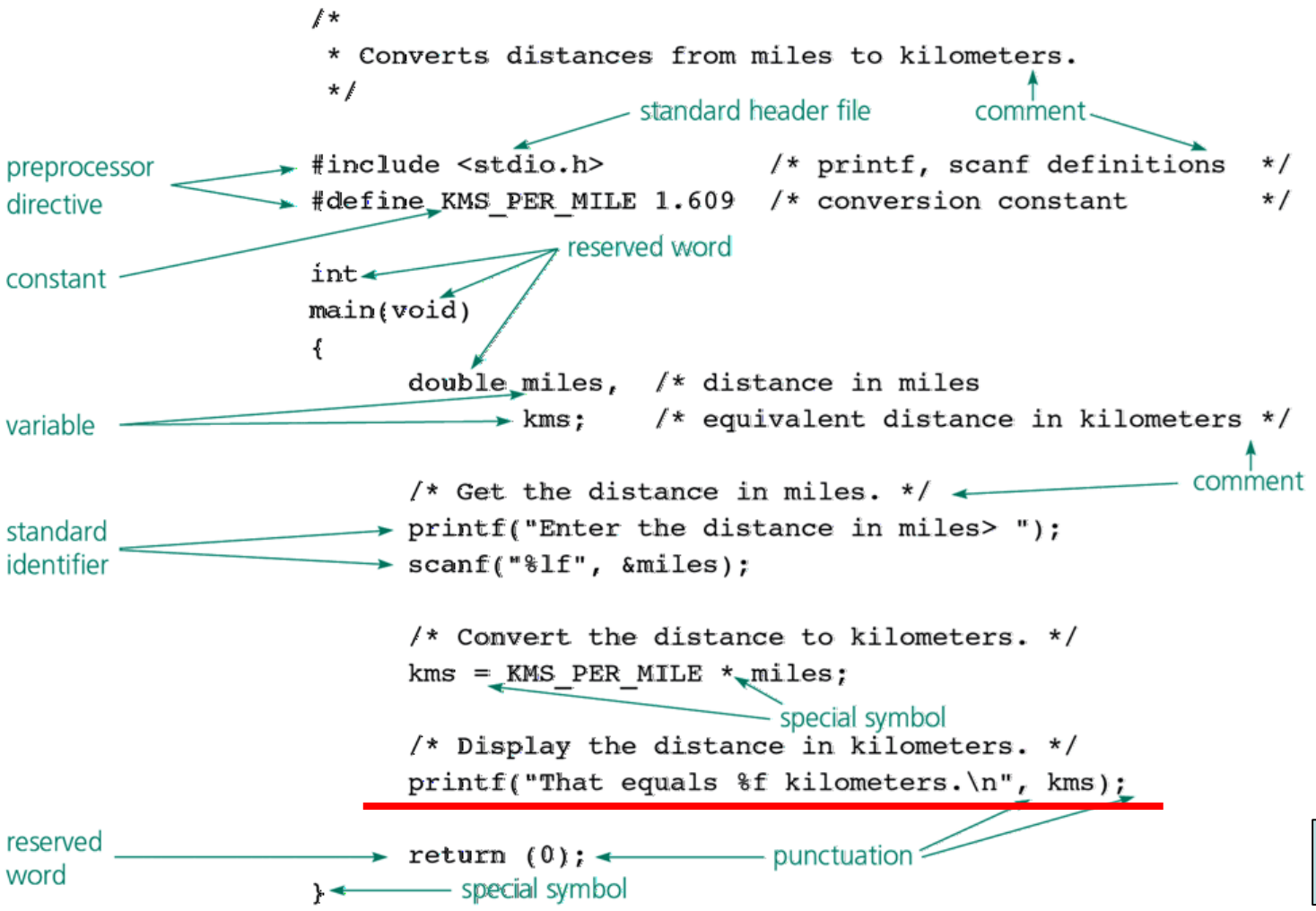
2.7 INTERACTIVE MODE, BATCH MODE, AND DATA FILES

2.8 COMMON PROGRAMMING ERRORS

2.1 C Language Elements

- Preprocessor
  - a system program that modifies the text of a C program before it is compiled
- Preprocessor directives
  - commands that provides instructions to the C preprocessor
    - e.g. #include, #define
- Library
  - a collection of useful functions and symbols that may be accessed by a program
  - each library has a standard header file
    - e.g. stdio.h. math.h

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



Preprocessor Directives(1/2)

- #include
  - gives a program access to a library
- <stdio.h>
  - standard header file
  - include printf 、 scanf

⇒#include <stdio.h>

  - notify the preprocessor that some names used in the program are found in <stdio.h>

Preprocessor Directives (2/2)

- #define
  - using only data values that never change should be given names
- Constant macro
  - a name that is replaced by a particular constant value

⇒#define KMS_PER_MILE 1.609

Comment

- Beginning with /* and ending with */
- Supplementary information
- Ignored by the preprocessor and compiler

Syntax Displays for Preprocessor Directives(1/2)

- #include
  - for defining identifiers from standard libraries
- Syntax：
  - #include<standard header file>
- Examples：
  - #include<stdio.h>
  - #include<math.h>

Syntax Displays for Preprocessor Directives (2/2)

- #define
  - for creating constant macros
- Syntax：
  - #define NAME value
- Examples：
  - #define MIL_PER_KM 0.62137
  - #define PI 3.141593
  - #define MAX_LENGTH 100

Function Main(1/4)

Contains two parts

Part 1: Declarations

- the part of a program that tells the compiler the names of memory cells in a program

Part 2: Executable statements

- program lines that are converted to machine language instructions and executed by the computer

Function Main (2/4)

- Syntax：

```
int
main(void)
 {
    function body
}
```

Function Main (3/4)

- Examples：

```
int
main(void)
{
        printf("Hello world\n");
        return(0);
}
```

Function “Main”(4/4)

- int
  - indicates that the main function returns an integer value (0) to the operating system when it finishes normal execution
- (void)
  - indicate that the main function receives no data from the operating system

Reserved Words

- A word that has special meaning in C

Reserved Word	Meaning
int	main function returns an integer value
void	main function receives no data from the operating system
double	the memory cells store real numbers
return	control from the main function to the operating system

Standard Identifiers

- A word having special meaning but one that a programmer may redefine
  - In Figure 2.1, the standard identifiers printf and scanf are names of operations defines in the standard input/output library.

User-Define Identifiers(1/2)

- Syntax rules for identifiers
  - An identifier must consist only of letters, digits, and underscores
  - An identifier cannot begin with a digit
  - A C reserved word cannot be used as an identifier
  - An identifier defined in a C standard library should not be redefined

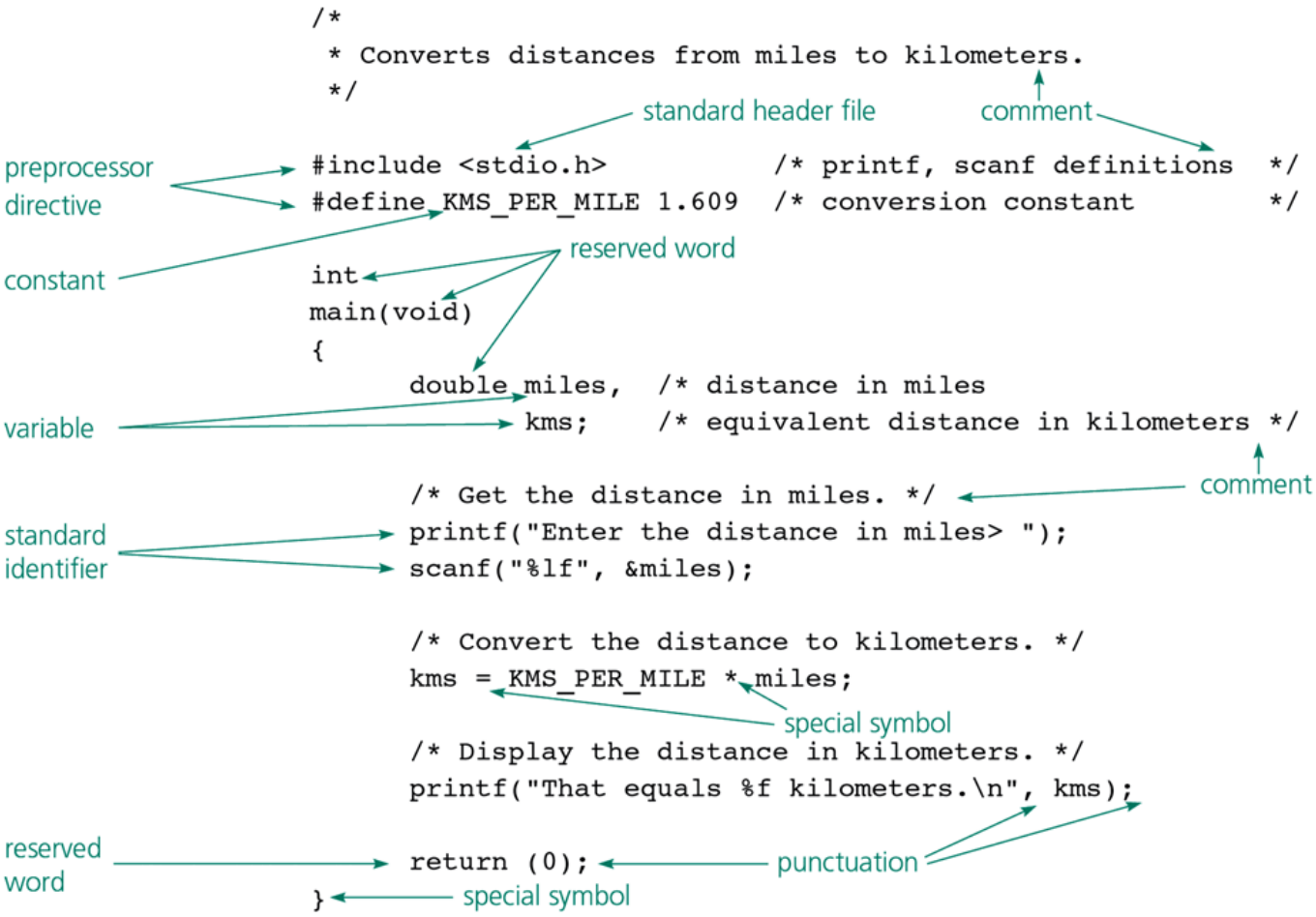
User-Define Identifiers(2/2)

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

Reserved Words vs. Identifiers

Reserved Words	Standard Identifiers	User-Define Identifiers
int	printf	KMS_PER_MILE
void	scanf	main
double		miles
return		kms

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



Uppercase and Lowercase Letters

- Rate, rate, RATE are viewed by the compiler as different identifiers
- Wildly adopted in industry uses all uppercase letters in the names of **constant macros**
- For example

  #define PI 3.14159

  #define KMS_PER_MILE 1.609

2.2 Variable Declaration and Data Types

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

Syntax Display for Declarations

- Syntax :
  - int *variable_list*;
  - double *variable_list;*
  - char *variable_list;*
- Examples :
  - int count, large;
  - double x, y, z;
  - char first_initial;
  - char ans;

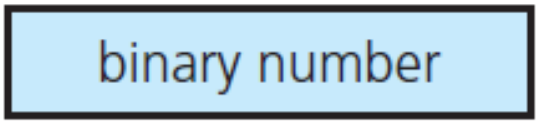
Data Types(1/3)

- Data type
  - a set of values and operations that can be performed on those values
- Data Type int
  - range -32768~32767 (16 bit)
- Data Type double (64 bit)
  - a real number has an integral part and a fractional part that are separated by a decimal point

Data Types(2/3)

- Internal Format of Type int and Type double

type `int` format



type `double` format

1 bit 11 bits 52 bits

$$real\ number = mantissa \times 2^{exponent}$$

Integer Types in C

Type	Range in Typical Microprocessor Implementation
short	-32767 ~ 32767
unsigned short	0 ~ 65535
int	-2147483647 ~ 2147483647
unsigned	0 ~ 4294967295
long	-2147483647 ~ 2147483647
unsigned long	0 ~ 4294967295

Floating-Point Types in C

Type	Approximate Range	Significant Digits
Float(32 bit)	10^{-37} ~10^{38}	6
Double(64 bit)	10^{-307} ~10^{308}	15
long double(80 bit)	10^{-4931} ~10^{4932}	19

Data Types(3/3)

- Data Type char (8 bit)
  - represent an individual character value
  - include a letter, a digit, a special symbol
  - ex. ‘A’ ‘z’ ‘2’ ‘9’ ‘ * ’ ‘ : ’ ‘ ” ’ ‘ ‘

  - char first_initial;
  - first_initial=‘H’; first_initial=72 (ASCII code)

The ASCII Character Set

ASCII stands for American Standard Code for Information Interchange

ASCII originally used seven bits to represent each character, allowing for 128 unique characters

Later extended ASCII evolved so that all eight bits were used

How many characters could be represented?

The ASCII Character Set

A=65=64+1=1000001

Left Digit(s) / *Right Digit*	*ASCII* 0	1	2	3	4	5	6	7	8	9
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT
1	LF	VT	FF	CR	SO	SI	DLE	DC1	DC2	DC3
2	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS
3	RS	US	□	!	“	#	$	%	&	’
4	(	)	*	+	,	–	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	`	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{	|	}	~	DEL		

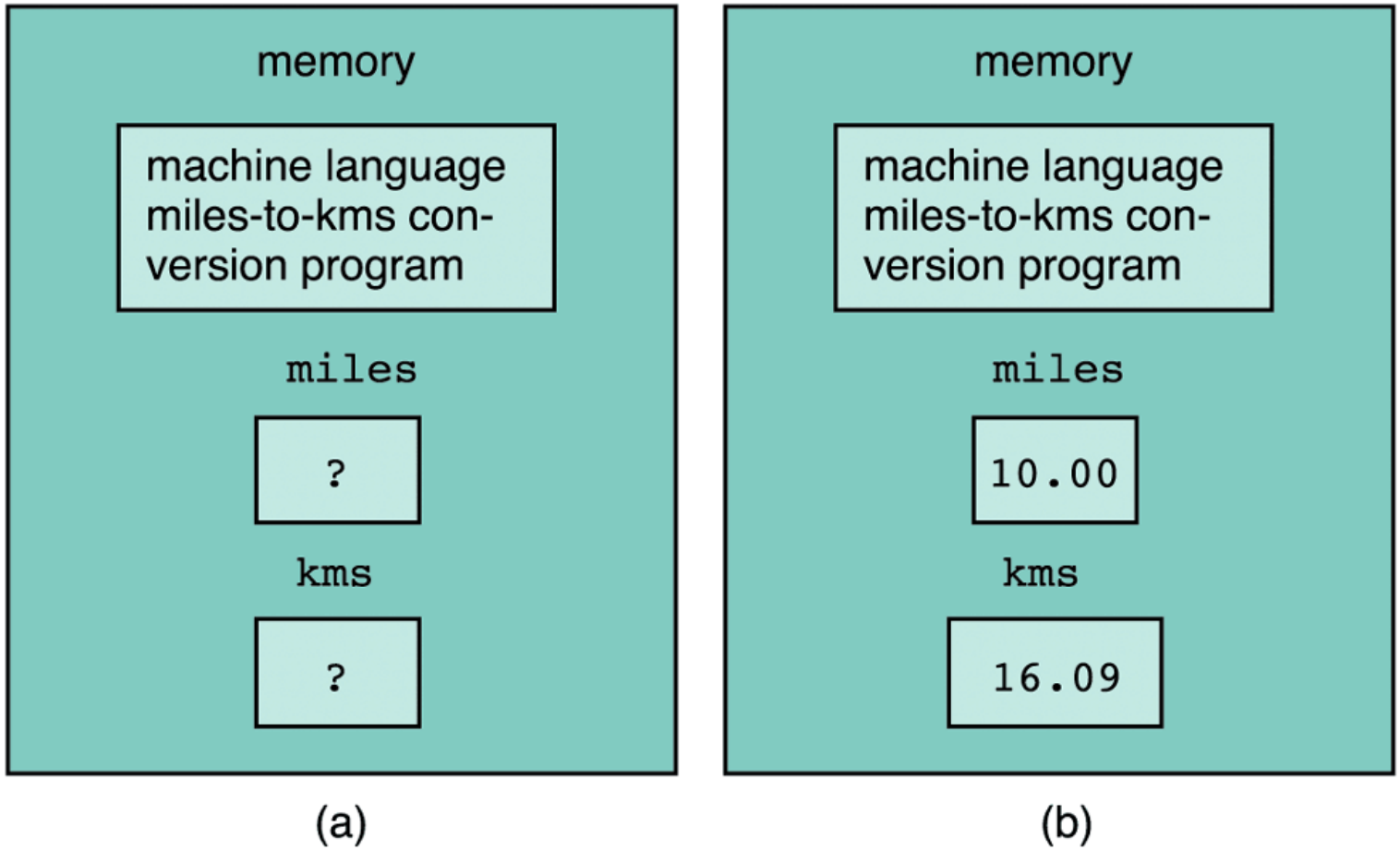
Type double Constants

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)
15.0e-04 (0.0015)	
2.345e2 (234.5)	12.5e.3(.3 is invalid)
1.15e-3 (0.00115)	34,500.99(, is not allowed)
12e+5 (1200000.0)	

2.3 Executable Statements

- Executable Statements
  - statements used to write or code the algorithm and its refinements

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



Assignment Statements

- Assignment Statement
  - an instruction that stores a value or a computational result in a variable
  - Form: *variable* = *expression* ;
  - Example: x = y + z + 2.0 ;

- Kms = KMS_PER_MILE * miles; (Figure 2.4)
- sum = sum + item ; (Figure 2.5)

Figure 2.4 Effect of kms = KMS_PER_MILE * miles;

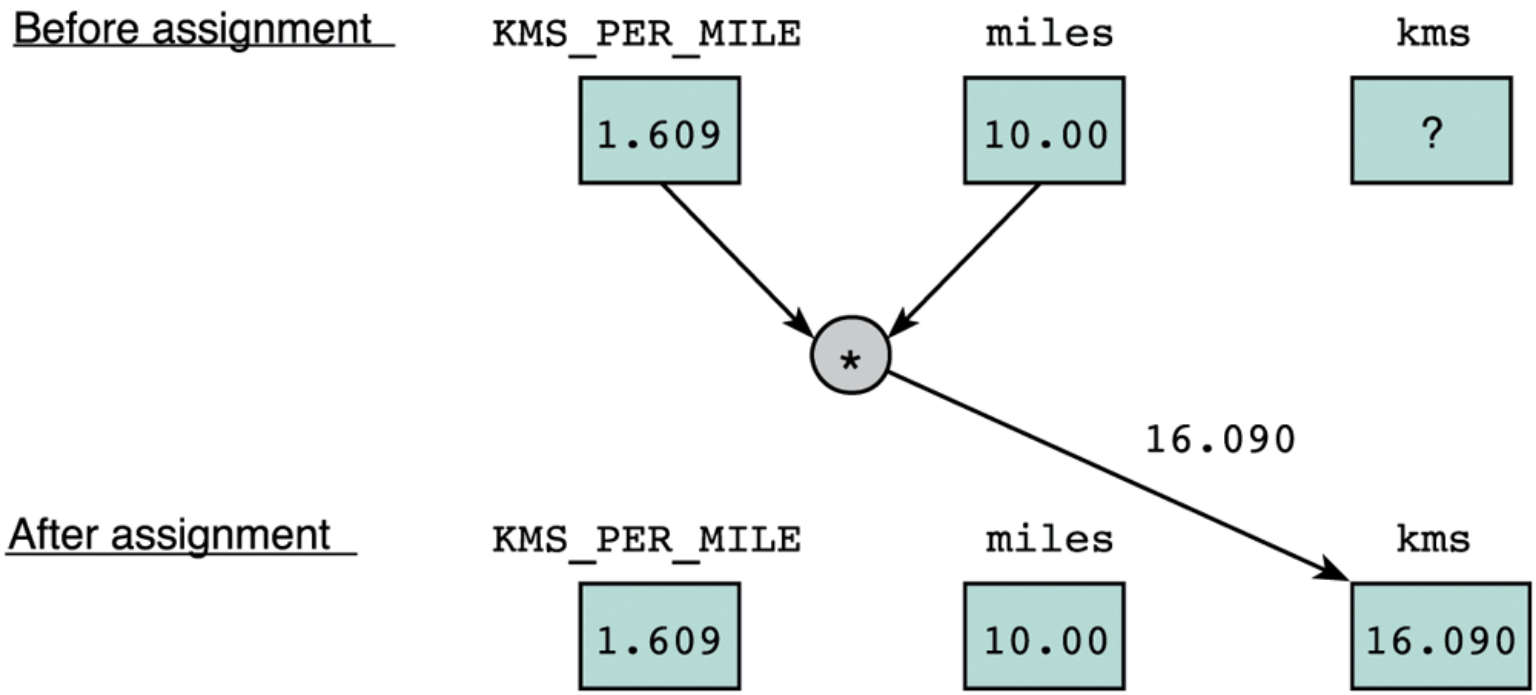
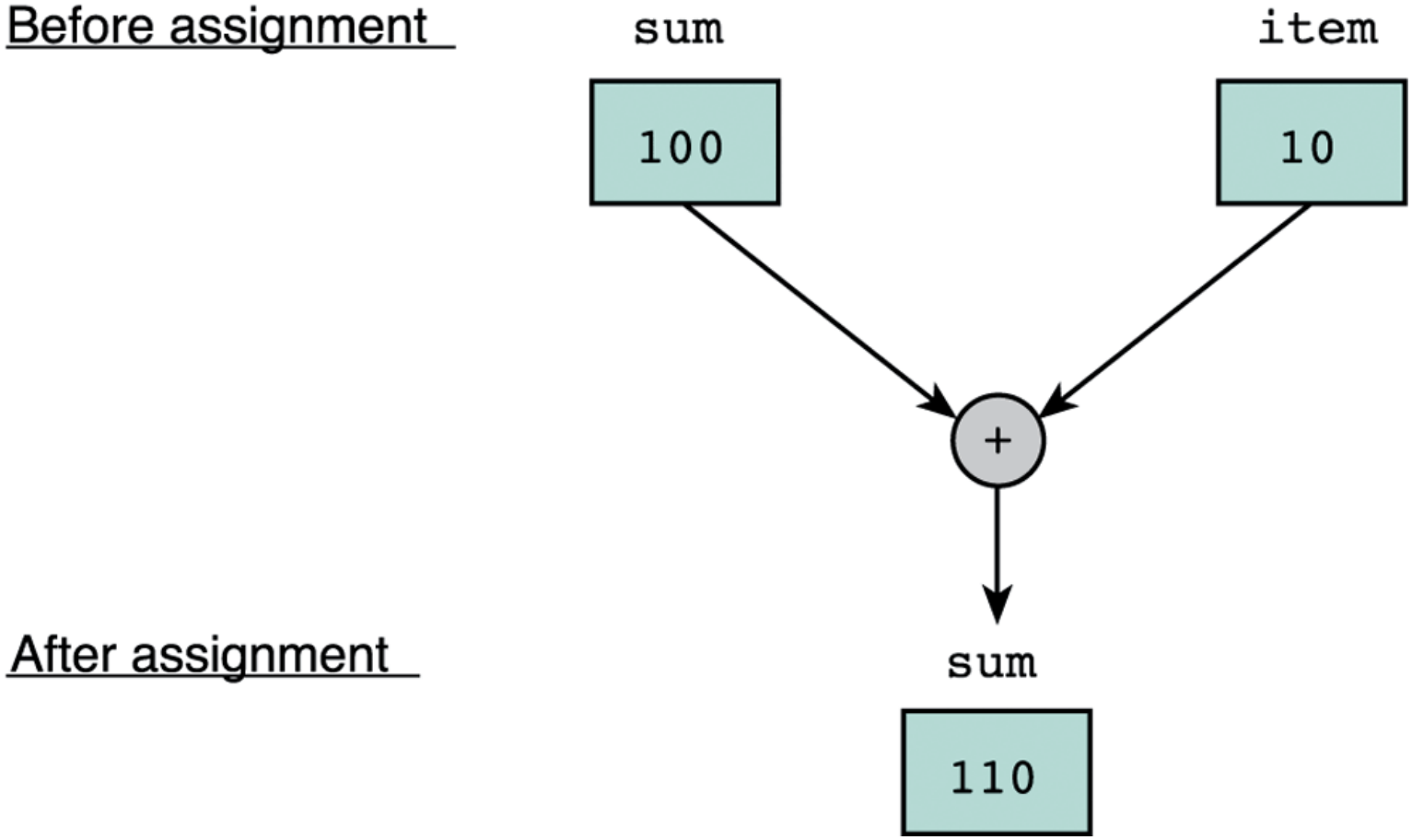


Figure 2.5 Effect of sum = sum + item;



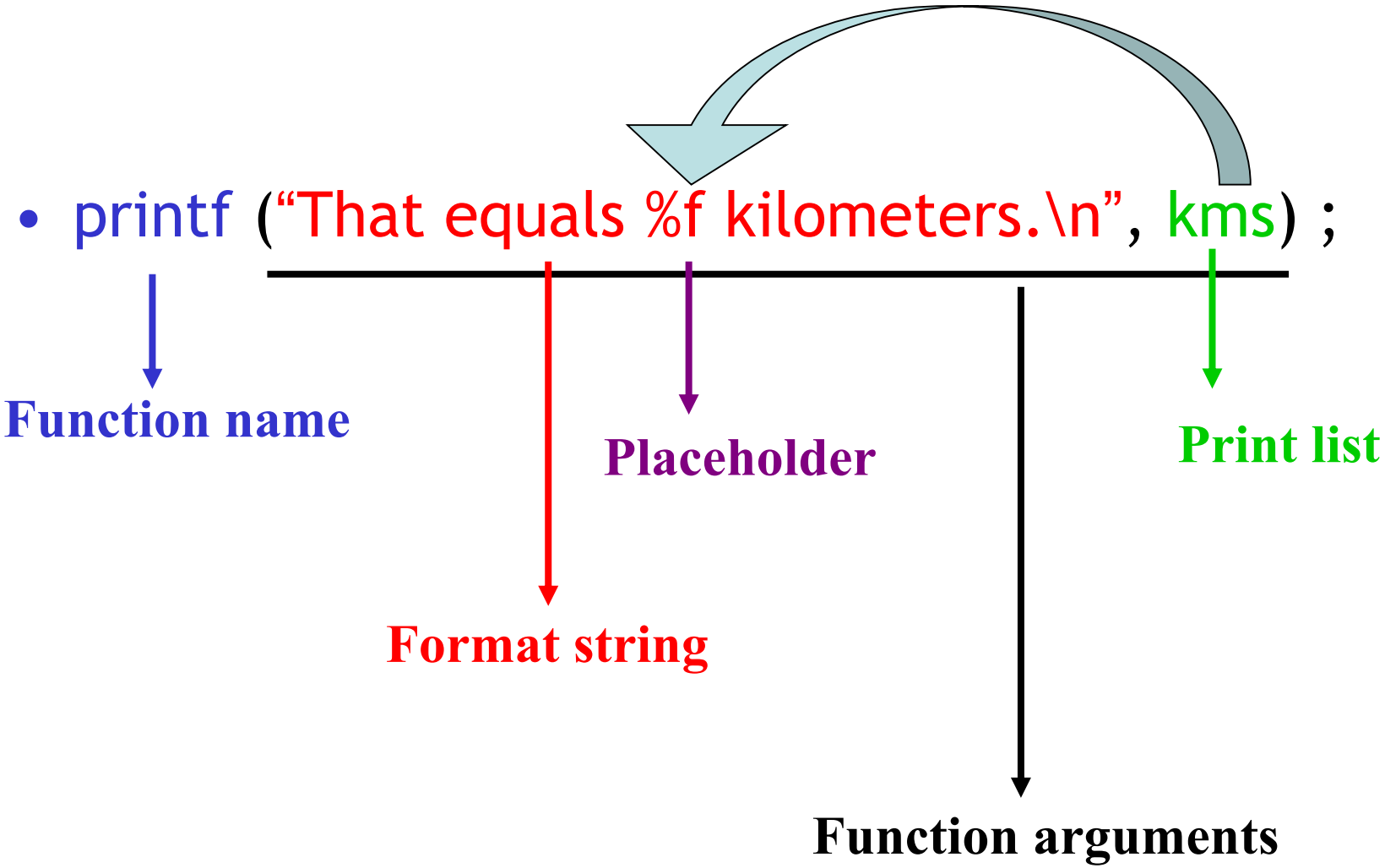
Input/Output Operations and Functions

- Input operation
  - an instruction that copies data from an input device into memory e.g. scanf
- Output operation
  - an instruction that displays information stored in memory e.g. printf
- Input/output function
  - A C function that performs an input or output operation e. g scanf, printf
- Function call
  - Calling or activating a function

The printf Functon(1/3)

- Function argument
  - enclosed in parentheses
  - provide information needed by the function
- Format string
  - a string of characters enclosed in quotes(“ ”)
  - specify the form of the output line
- Print list
  - the variables or expressions whose values are displayed

The printf Functon(2/3)



The printf Function(3/3)

- Placeholder
  - a symbol beginning with %
  - indicate where to display the output value
- Newline escape sequence
  - the character sequence \n
  - used in a format string to terminate an output line

Placeholders in Format Strings

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

Syntax Display for Function Call

- Syntax:
  - printf(*format string, print list*);
  - printf(*format string*);
- Examples:
  - printf("I am %d years old, and my gpa is %f\n", age, gpa);
  - printf("Enter the object mass in grams> ");

Displaying Prompts

- prompt (prompting message)
  - a message displayed to indicate what data to enter and in what form
- Example
  - printf("Enter the distance in miles> ");
  - scanf("%lf", &miles);

The scanf Functon

- scanf("%lf", &miles); (Figure 2.5)
- scanf("%c%c%c", &letter_1, &letter_2, &letter_3); (Figure 2.6)
- &
  - The C address-of operator
  - Tell the scanf function where to find each variable into which it is to store a new value

Beware the difference between SCANF and PRINTF in input arguments

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

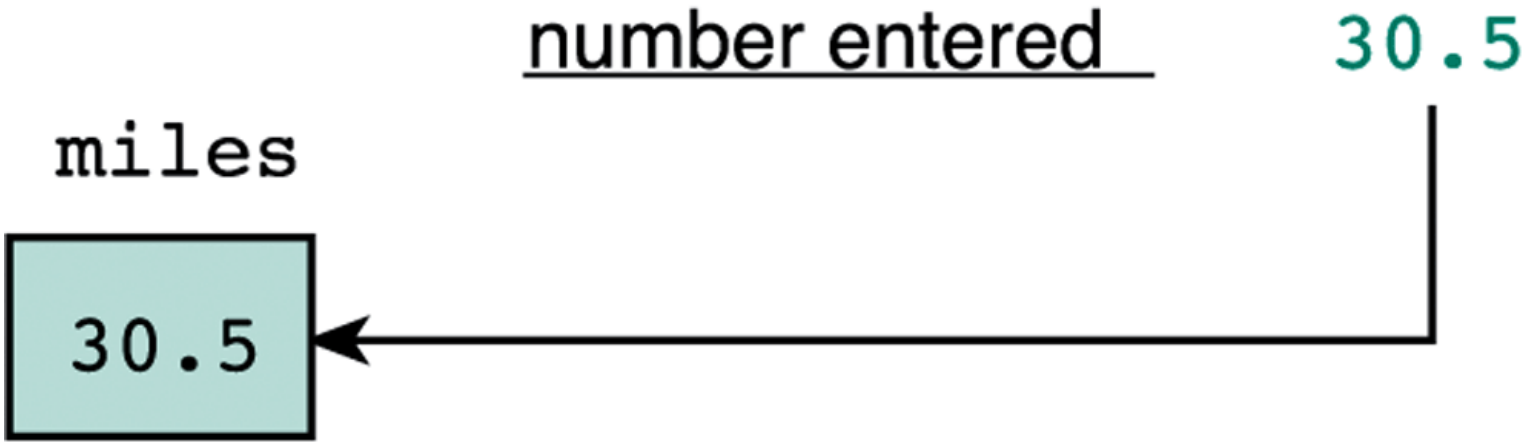
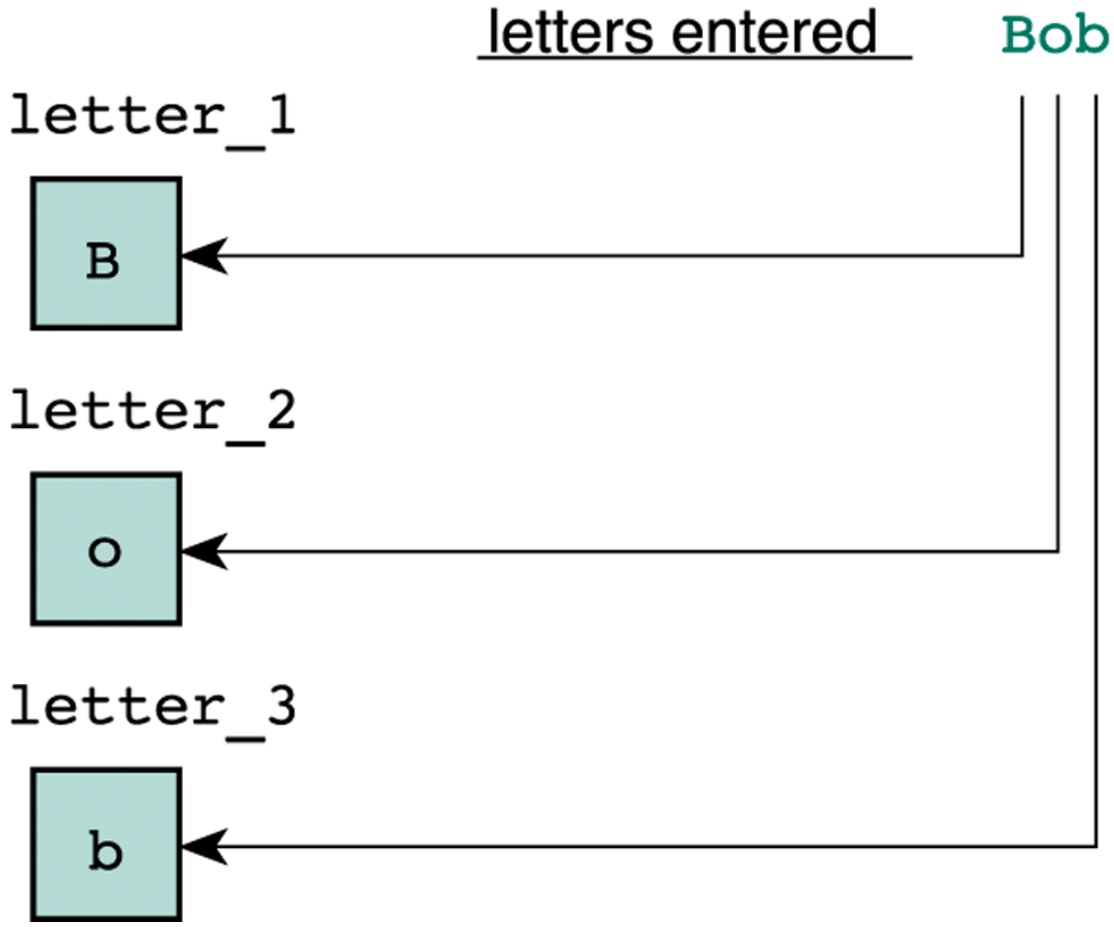


Figure 2.7 Scanning Data Line Bob

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



Syntax Display for scanf Function Call

- Syntax:
  - scanf(fortmating string, input list);
- Example:
  - scanf("%c%d", &first_initial, &age);

- Note: To skip spaces before scanning a character, put a ***blank*** in the format string before the %c placeholder.

The return Statement

- Syntax:
  - return *expression*;
- Example:
  - return(0);

2.4 General Form of A C Program

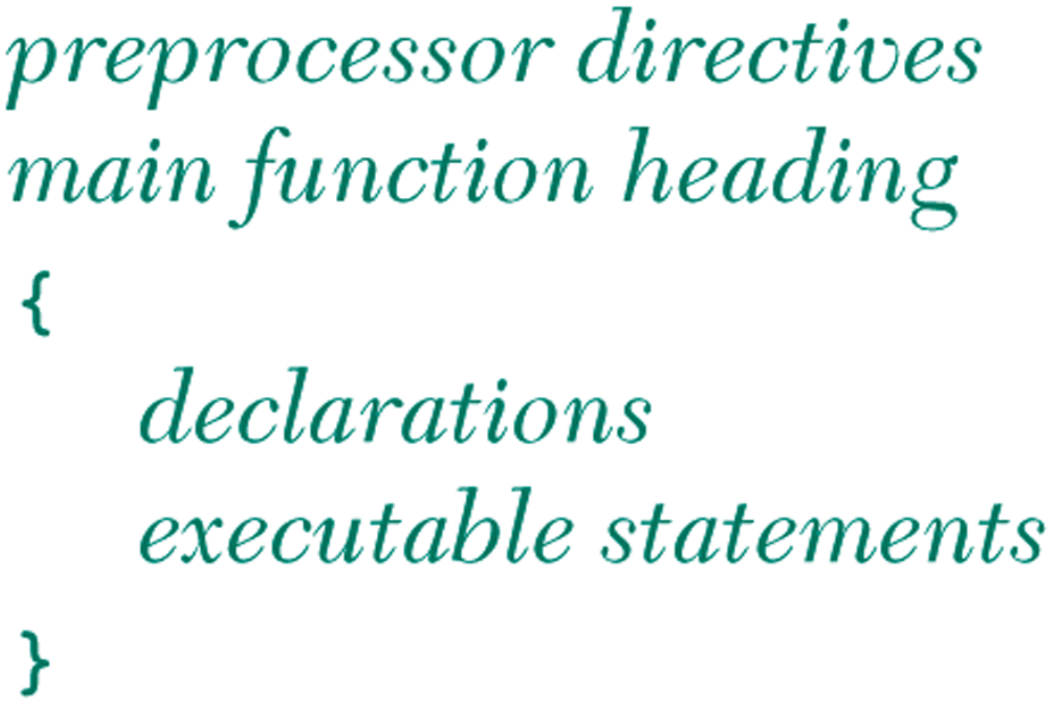


Figure 2.8 General Form of a C Program

Program comment

SYNTAX: /* *comment text* */

EXAMPLES:
```
/* This is a one-line or partial-line comment */
/*
 * This is a multiple-line comment in which the stars
 * not immediately preceded or followed by slashes
 * have no special syntactic significance, but simply
 * help the comment to stand out as a block. This
 * style is often used to document the purpose of a
 * program.
 */
```

Program Style

- Spaces in Programs
  - careful use of blank spaces
- Comments in Programs
  - header section of a program consists of a series of comments specifying
    - the programmer's name
    - the data of the current version
    - a brief description of what the program does

[Page 91, Examples] ←Very Important

Example of program style

```
/*
 * Programmer: William Bell    Date completed: May 9, 2003
 * Instructor: Janet Smith     Class: CIS61
 *
 * Calculates and displays the area and circumference of a
 * circle
 */
```

2.5 Arithmetic Expressions

Arithmetic Operator	Meaning	Examples
+	addition	5 + 2 is 7
-	subtraction	5 - 2 is 3
*	multiplication	5 * 2 is 10
/	division	5 / 2 is 2
%	remainder	5 % 2 is 1

Operators / and %

- If the / operator is used with a negative and a positive integer, the result may vary from one C implementation to another
- The / operation is undefined when the divisor is 0 (4/0 is undefined)
- The % operation is undefined when the divisor is 0 and varies from one implementation to another if the divisor is negative

Results of / and % operations

- 3 / 15 = 0 16 / 3 = 5
- 16 / -3 varies
- 4 / 0 undefined
- 3 % 5 = 3 5 % 4 = 1
- 15 % -7 varies
- 15 % 0 undefined
- 299 equals (299/100)*100+(299%100)

Data Type of an Expression

- Mixed-type expression
  - an expression with operands of different types
  - Ex: int m = 3, n = 2;

```
double p, x, y;
p = 2.0;
x = m / p;
y = m / n;
```

what is the value of y ?

Type Conversion through Casts

- type cast
  - converting an expression to a different type by writing the desired type in parentheses in front of the expression (Table 2.12 Page 97)

Application	Example
Avoiding integer division	int num_students, total_score; double average; average = (double)total_score / (double)num_students;
Rounding a number	double x; int rounded_x; rounded_x = (int)(x + 0.5);

Expressions with Multiple Operators

- Unary operator
  - an operator with one operand
  - Ex: x = -y,
- Binary operator
  - an operator with two operands
  - Ex: x = y + z

Rules for Evaluating Expression

- Parentheses rule
  - all expressions in parentheses must be evaluated separately
  - nested parenthesized expressions must be evaluated from the inside out
- Operator precedence rule
  - unary +, - first
  - * , / , % next
  - binary + , - last
- Associativity rule
  - right associativity (unary operator are evaluated right to left)
  - left associativity (binary operator are evaluated left to right)

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

- Example 2.5:

  The formula for th
  area of a circle
  - a = πr^2
  - area = PI * radius * radius
  - (Figure 2.9)

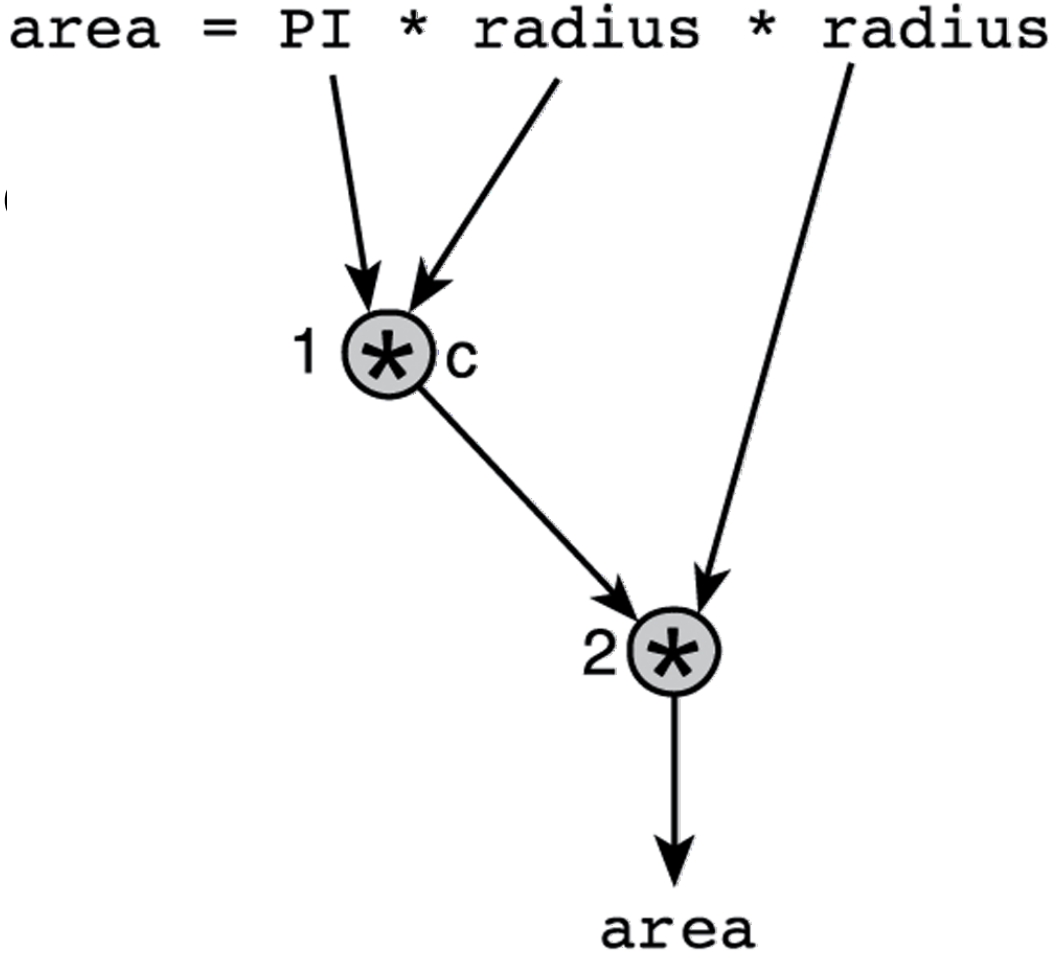


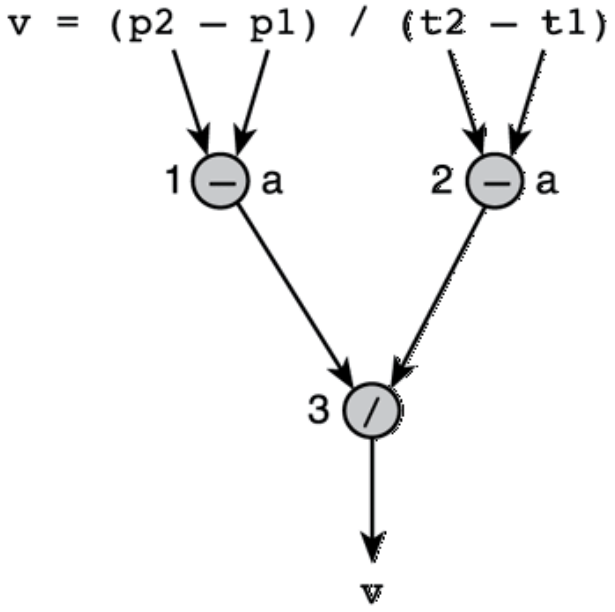
Figure 2.10 Step-by-Step Expression Evaluation

```
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);

- Example 2.6 :The formula for the average velocity

$$v = \frac{p2-p1}{t2-t1}$$



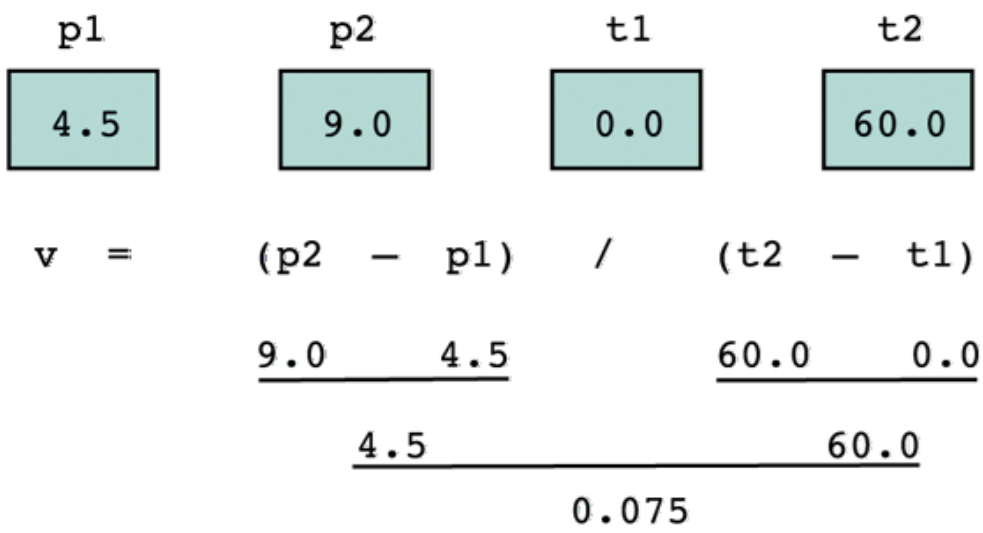
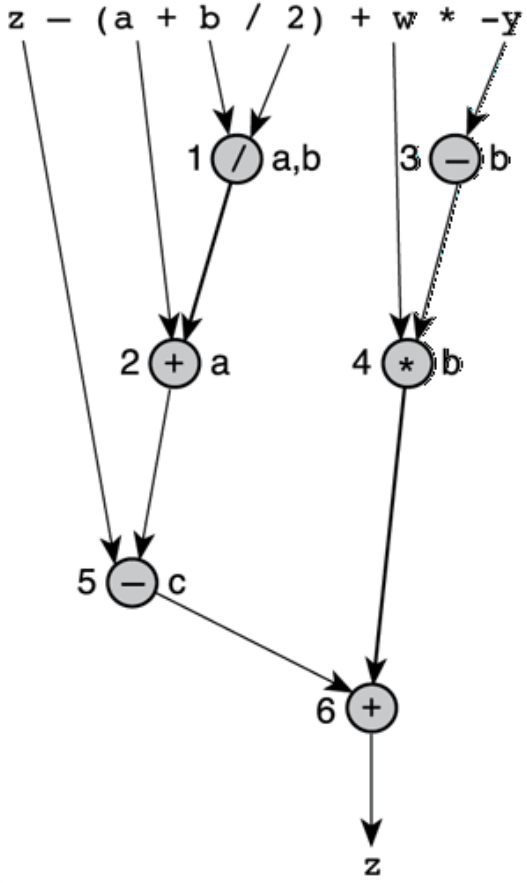
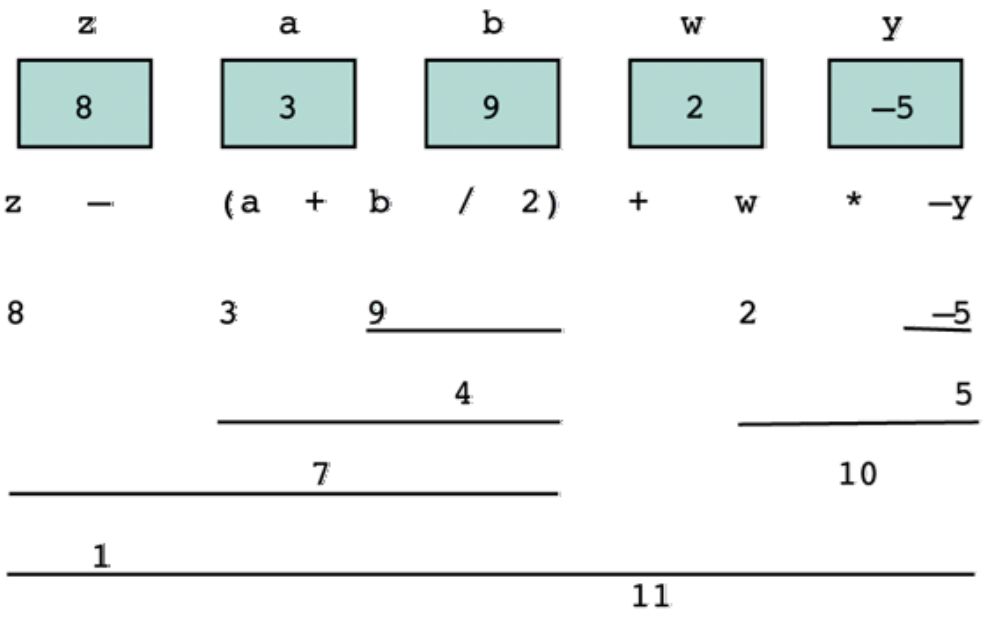


Figure 2.12 Evaluation Tree and Evaluation for z - (a + b / 2) + w * - y

- Example 2.7





Mathematical Formula as C Expression

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

CASE STUDY: Supermarket Coin Processor (1/4)

Step 1: Problem

- You are drafting software for the machines placed at the front of supermarkets to convert change to personalized credit slips. In this draft, the user will manually enter the number of each kind of coin in the collection, but in the final version, these counts will be provided by code that interacts with the counting devices in the machine.

CASE STUDY:
Supermarket Coin Processor (2/4)

Step 2: Analysis

- problem inputs
  - char first, middle, last
  - int dollars
  - int quarters
  - int dimes
  - int nickels
  - int pennies
- problem outputs
  - int total_dollars
  - int change
- additional program variables
  - int total_cents

CASE STUDY: Supermarket Coin Processor (3/4)

Step 3: Design

- initial algorithm
  1. get and display the customers initials
  2. get the count of each kind of coins
  3. compute the total value in cents
  4. find the value in dollars and change
  5. display the value in dollars and change
- refinement

  3.1 find the equivalent value of each kind of coin in pennies and add these values

  4.1 total_dollars is the integer quotient of total_cents and 100

  4.2 change is the integer remainder of total_cents and 100

CASE STUDY:
Supermarket Coin Processor (4/4)

Step 4: Implementation (Figure 2.13)

Step 5: Testing

Figure 2.13 Finding the Value of Coins

```
1.  /*
2.   * Determines the value of a collection of coins.
3.   */
4.  #include <stdio.h>
5.  int
6.  main(void)
7.  {
8.       char first, middle, last;/ * input - 3 initials           */
9.       int pennies, nickels;  /* input - count of each coin type */
10.      int dimes, quarters;   /* input - count of each coin type */
11.      int dollars;           /* input - count of each coin type */
12.      int change;               /* output - change amount       */
13.      int total_dollars;        /* output - dollar amount       */
14.      int total_cents;          /* total cents                  */
15.
16.      /* Get and display the customer's initials. */
17.      printf("Type in 3 initials and press return> ");
18.      scanf("%c%c%c", &first, &middle, &last);
19.      printf("\n%c%c%c, please enter your coin information.\n",
20.             first, middle, last);
21.
22.      /* Get the count of each kind of coin. */
23.      printf("Number of $ coins > ");
24.      scanf("%d", &dollars);
25.      printf("Number of quarters> ");
```

(continued)

Figure 2.13 Finding the Value of Coins (cont'd)

```
26.      scanf("%d", &quarters);
27.      printf("Number of dimes   > ");
28.      scanf("%d", &dimes);
29.      printf("Number of nickels > ");
30.      scanf("%d", &nickels);
31.      printf("Number of pennies > ");
32.      scanf("%d", &pennies);
33.
34.      /* Compute the total value in cents. */
35.      total_cents = 100 * dollars +25 * quarters + 10 * dimes +
36.                    5 * nickels + pennies;
37.
38.      /* Find the value in dollars and change. */
39.      dollars = total_cents / 100;
40.      change = total_cents % 100;
41.
42.      /* Display the credit slip with value in dollars and change. */
43.      printf("\n\n%c%c%c Coin Credit\nDollars: %d\nChange:  %d cents\n",
44.             first, middle, last, dollars, change);
45
46.      return (0);
47. }

Type in 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
```

2.6 Formatting Numbers in Program Output

- Field width (the number of columns used to display a value)

Printf("Results: %3d meters = %4d ft.\n",meters, feet);

Results: ▒2 1 meters = ▒▒6 8 ft.

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

Formatting Values of Type double

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

Program-Controlled Input and Output Files

- declare a file pointer variable
  - File *inp , /* pointer to input file */
    outp ; / pointer to output file */
- the calls to function fopen
  - inp = fopen("b:distance.dat", "r") ;
  - outp = fopen("b:distance.out", "w") ;
- use of the functions
  - fscanf(inp, "%lf", &miles);
  - fprintf(outp, "The distance in miles is %.2f. \n", miles);
- end of use
  - fclose(inp);
  - fclose(outp);

Miles-to-Kilometers Conversion Program with Named Files

```
1.  /* Converts distances from miles to kilometers.     */
2.
3.  #include <stdio.h>     /* printf, scanf, fprint, fscanf, fopen, fclose
4.                           definitions                */
5.  #define KMS_PER_MILE 1.609 /* conversion constant   */
6.
7.  int
8.  main(void)
9.  {
10.       double miles, /* distance in miles                                    */
11.              kms;   /* equivalent distance in kilometers                   */
12.       FILE   *inp,  /* pointer to input file                               */
13.              *outp; /* pointer to output file                              */
14.
15.       /* Open the input and output files.     */
16.       inp = fopen("b:distance.dat", "r");
17.       outp = fopen("b:distance.out", "w");
18.
19.       /* Get and echo the distance in miles. */
20.       fscanf(inp, "%lf", &miles);
21.       fprintf(outp, "The distance in miles is %.2f.\n", miles);
22.
23.       /* Convert the distance to kilometers. */
24.       kms = KMS_PER_MILE * miles;
25.
26.       /* Display the distance in kilometers. */
27.       fprintf(outp, "That equals %.2f kilometers.\n", kms);
28.
29.       /* Close files. */
30.       fclose(inp);
31.       fclose(outp);
32.
33.       return (0);
34. }
```

Contents of input file `distance.dat`

```
112.0
```

Contents of output file `distance.out`

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

2.8 Common Programming Errors

- Syntax Errors (Figure 2.15)
  - missing semicolon at the end of the variable declaration
  - undeclared variable miles
  - last comment is not closed because of blank in */ close-comment sequence
- Run-Time Errors (Figure 2.16)
  - an attempt to perform an invalid operation, detected during program execution

Figure 2.15 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 2.16 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
```

Common Programming Errors

- Undetected Errors
- Logic Errors
  - an error caused by following an incorrect algorithm

Figure 2.17 Revised Start of main Function for Coin Evaluation

```
1.  int
2.  main(void)
3.  {
4.        char first, middle, last; /* input - 3 initials            */
5.        int pennies, nickels;  /* input - count of each coin type */
6.        int dimes, quarters;   /* input - count of each coin type */
7.        int change;              /* output - change amount        */
8.        int dollars;             /* output - dollar amount        */
9.        int total_cents;         /* total cents                   */
10.       int year;                /* current year                  */
11.
12.       /* Get the current year.                                   */
13.       printf("Enter the current year and press return> ");
14.       scanf("%d", &year);
15.
16.       /* Get the program user's initials.                        */
17.       printf("Type in 3 initials and press return> ");
18.       scanf("%c%c%c", &first, &middle, &last);
19.       printf("Hello %c%c%c, let's check your coins' value in %d.\n",
20.              first, middle, last, year);
21.       ...
```

Figure 2.18 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
```

Question?

- A good question deserves a good grade…