

OVERVIEW OF C

CHAPTER 2

Problem Solving & Program Design in C

Eighth Edition

Global Edition

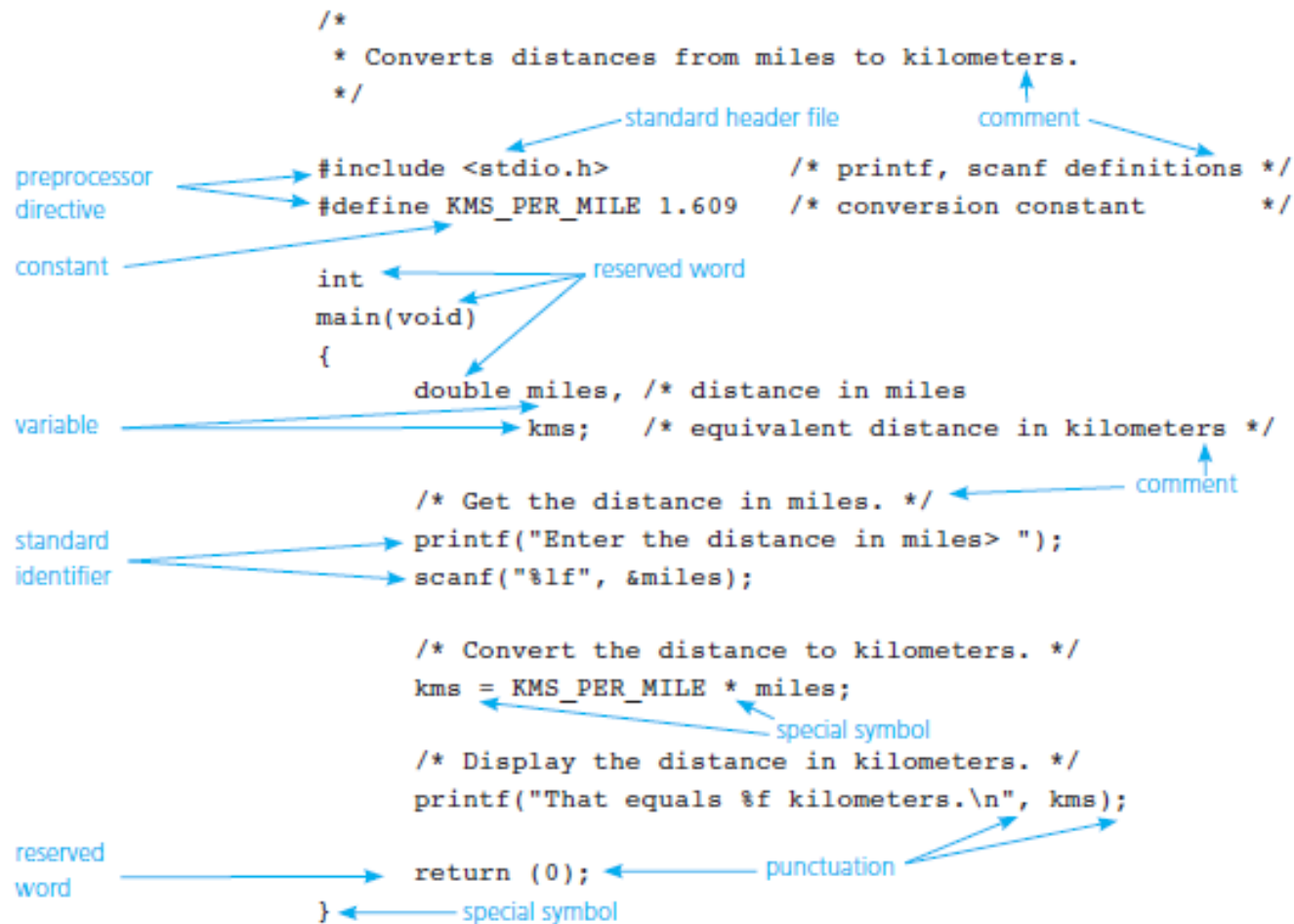
Jeri R. Hanly & Elliot B. Koffman

Figure 2.8

General Form of a C Program

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

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



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.

Language Elements

- comment
 - text beginning with `/*` and ending with `*/` that provides supplementary information but is ignored by the preprocessor and compiler

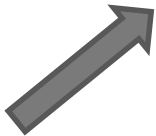
`/* Get the distance in miles */`

`// this is also a comment for 1 line only`

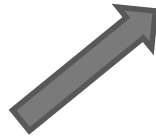
Language Elements

- preprocessor
 - a system program that modifies a C program prior to its compilation
- 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

- 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”



```
#include <stdio.h> /*includes printf, scanf definitions*/  
#include <math.h> /*includes math definitions*/  
#include <string.h> /*includes string (text) manipulation functions*/  
... many more.
```

```
SYNTAX:      #include <standard header file>
```

include directive gives a program access to a library

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;`

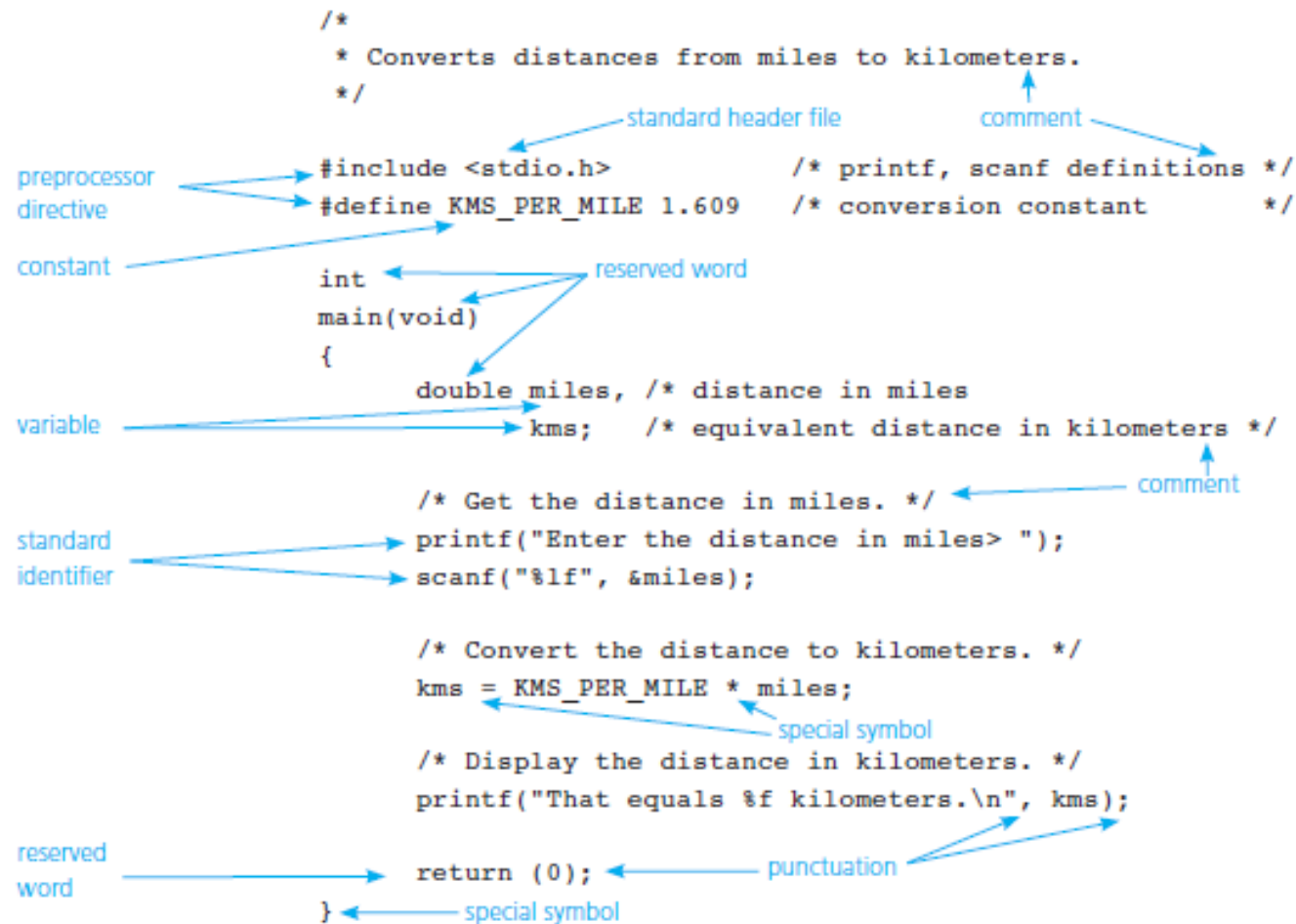
`#define` Directive for Creating Constant Macros

SYNTAX: `#define NAME value`

EXAMPLES: `#define MILES_PER_KM 0.62137`
`#define PI 3.141593`
`#define MAX_LENGTH 100`

INTERPRETATION: The C preprocessor is notified that it is to replace each use of the identifier *NAME* by *value*. C program statements cannot change the value associated with *NAME*.

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



Function main

- Every C program has a main function.

int main (void)

OR

int
main (void)

- These line(s) mark the beginning of the main function where program execution begins.

main Function Definition

SYNTAX: int
 main(void)
 {
 function body
 }

curly braces are used, to show to the compiler where a function starts and ends.

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

```
int main(void)
{
    double miles; /* distance in miles */
    double kms; /* equivalent distance in kilometers */
```

declarations

```
printf("Enter the distance in miles> ");
scanf("%lf", &miles);
```

executable statements

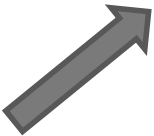
Language Elements


- 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

auto	default	float	register	switch
break	do	for	return	typedef
case	double	goto	short	union
char	else	if	signed	unsigned
const	enum	int	sizeof	void
continue	extern	long	static	volatile
			struct	while

Language Elements

- standard identifier
 - a word having special meaning but one that a programmer may redefine
 - Correspond to standard functions already implemented for a purpose

 `printf("Enter the distance in miles> ");` from **stdio.h**

 `sin(PI);` from **math.h**

In C, standard identifiers (standard functions) can be redefined, but *redefinition is not recommended*.

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

```
double miles; /* distance in miles */  
double kms; /* equivalent distance in kilometers */
```

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 stored in a particular variable.

`int hours;`

`double miles;`

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;`

INTERPRETATION: A memory cell is allocated for each name in the *variable_list*. The type of data (`double`, `int`, `char`) to be stored in each variable is specified at the beginning of the statement. One statement may extend over multiple lines. A single data type can appear in more than one variable declaration, so the following two declaration sections are equally acceptable ways of declaring the variables `rate`, `time`, and `age`.

<code>double rate, time;</code>		<code>double rate;</code>
<code>int age;</code>		<code>int age;</code>
		<code>double time;</code>

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', '*', '!'

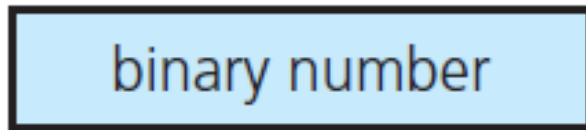
TABLE 2.4 Type double Constants (real numbers)

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

Figure 2.2

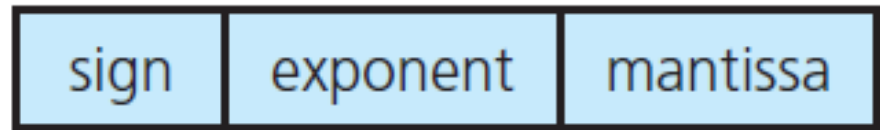
Internal Format of Type int and Type double

type `int` format



32-bits

type `double` format



1 bit

11 bits

52 bits

64-bits

double: The storage area occupied by the number is divided into three sections:
sign is 1 bit (0 for positive numbers, 1 for negative numbers),
mantissa is a binary fraction
exponent is an integer.
mantissa and exponent are chosen so that $real\ number = mantissa \times 2^{exponent}$

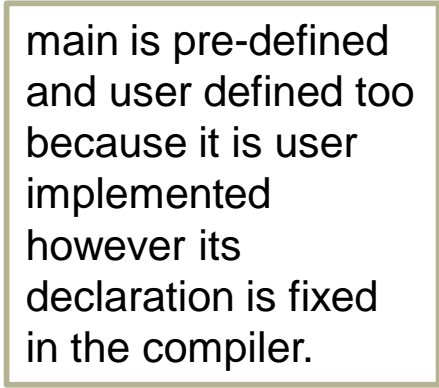
HOA 1, Q1-2

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 as an identifier.
 4. An identifier defined in a C standard library should not be redefined.
 5. C is case sensitive. rate, Rate and RATE are different identifiers.

User-defined identifiers

- Quick Check: Which identifiers are valid?
 - letter_1 ✓
 - 1Letter ✗
 - CENT_PER_INCH ✓
 - variable ✓
 - TWO*FOUR ✗
- What happens when we define below two identifiers?
 - per_capita_meat_consumption_in_1980 ✓
 - per_capita_meat_consumption_in_1995 ✓
 - Both are syntactically correct and there is no rule about length, BUT
 - some ANSI C compilers consider only first 31 characters, therefore, would consider these two identifiers identical



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

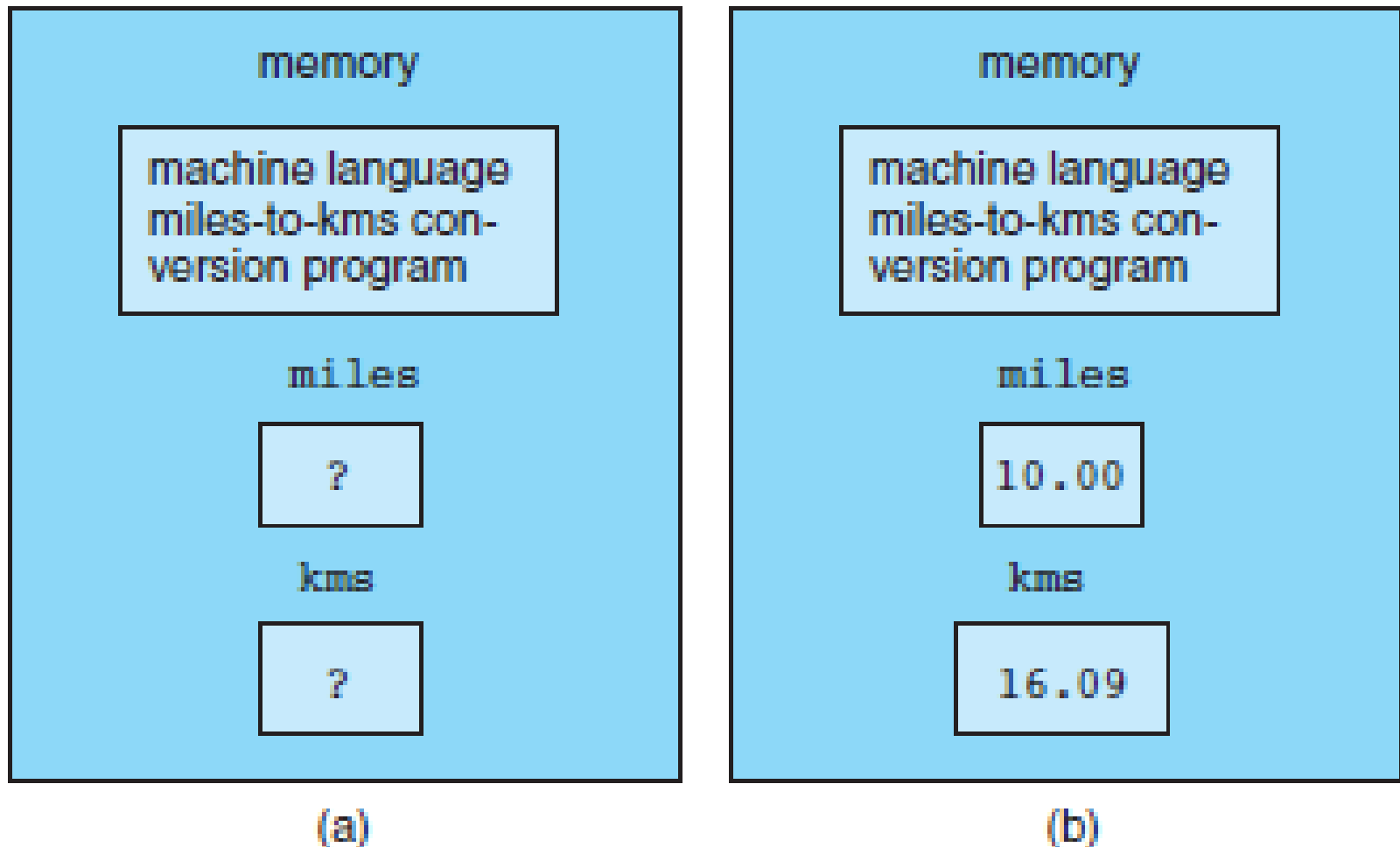
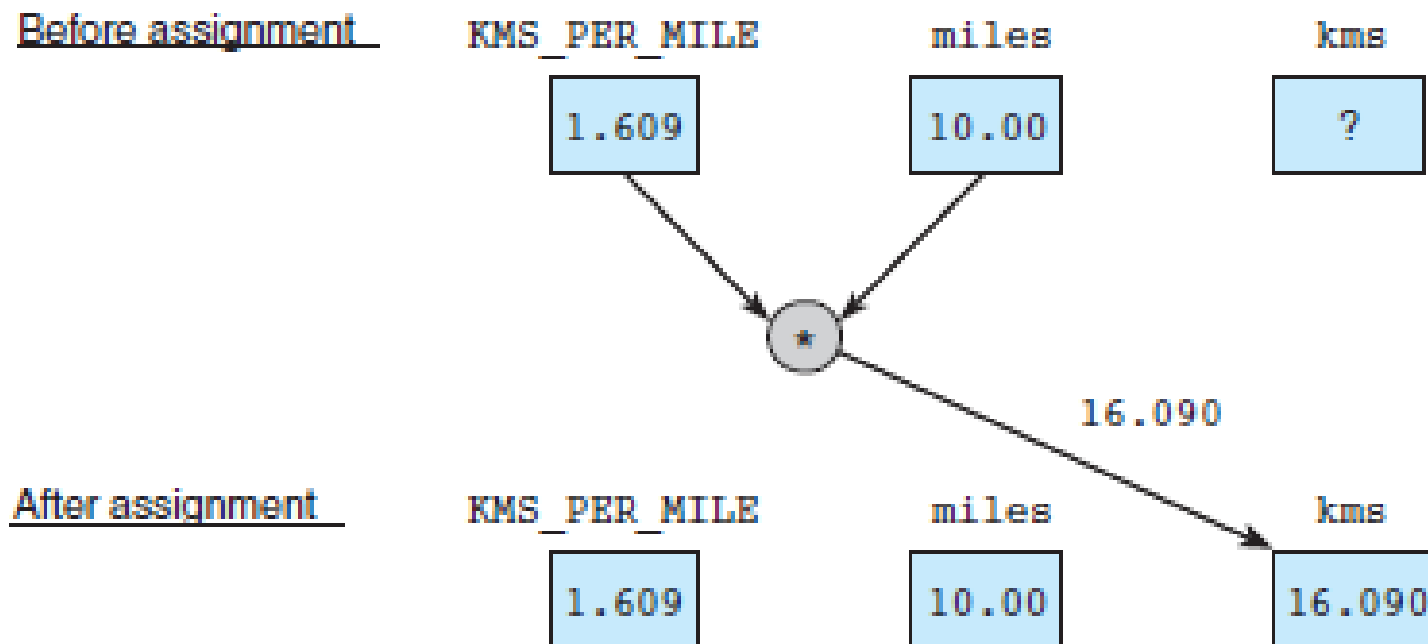


Figure 2.4

Effect of $\text{kms} = \text{KMS_PER_MILE} * \text{miles};$



Executable Statements

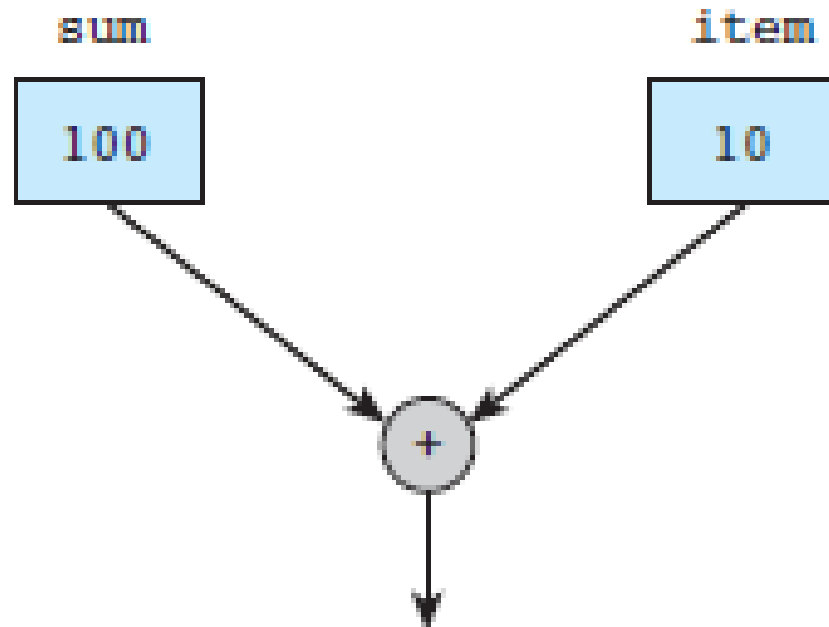
- Assignment is not the same as an algebraic equation.
- The expression to the right of the assignment operator is first evaluated.
- Then the variable on the left side of the assignment operator is assigned the value of that expression.

`sum = sum + item;`

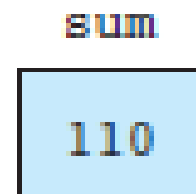
Figure 2.5

Effect of $\text{sum} = \text{sum} + \text{item};$

Before assignment



After assignment



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

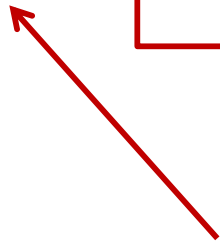
- Displays a line of program output.
- Useful for seeing the results of a program execution.

```
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

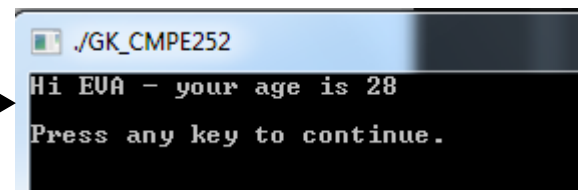
Placeholder	Variable Type	Function Use
% c	char	printf/scanf
% d	int	printf/scanf
% f	double	printf
% lf	double	scanf

Quick Check

- What if we have multiple placeholders?
- `#_to_be_printed_variables = #_placeholders`
- C matches them in left-to-right order

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int age = 22;
6      char c_1 = 'E';
7      char c_2 = 'V';
8      char c_3 = 'A';
9
10     printf("Hi %c%c%c, your age is %d\n", c_1, c_2, c_3, age);
11     return 0;
12 }
```

How do you get this output with 3 characters and 1 integer?

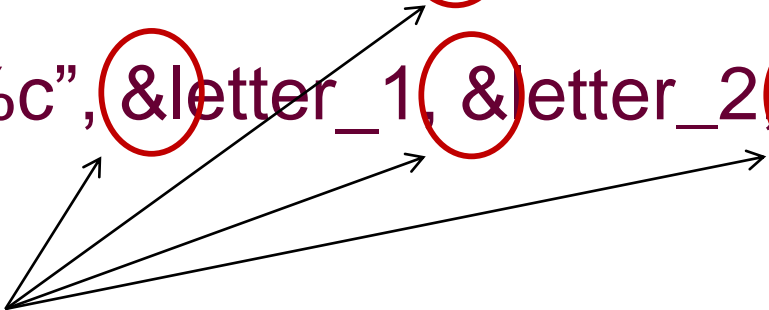


The terminal window shows the output of the C program. The title bar reads `./GK_CMPE252`. The output is `Hi EVA - your age is 28` followed by a prompt `Press any key to continue.` on the next line.

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);
```



Ampersand & : address of operator in C

Figure 2.6

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

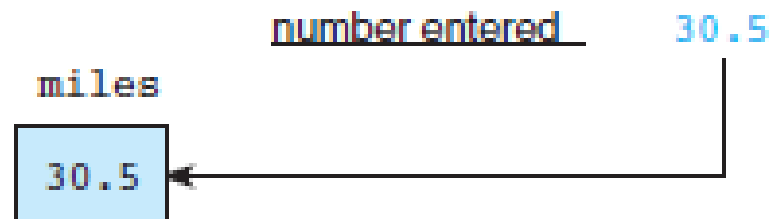
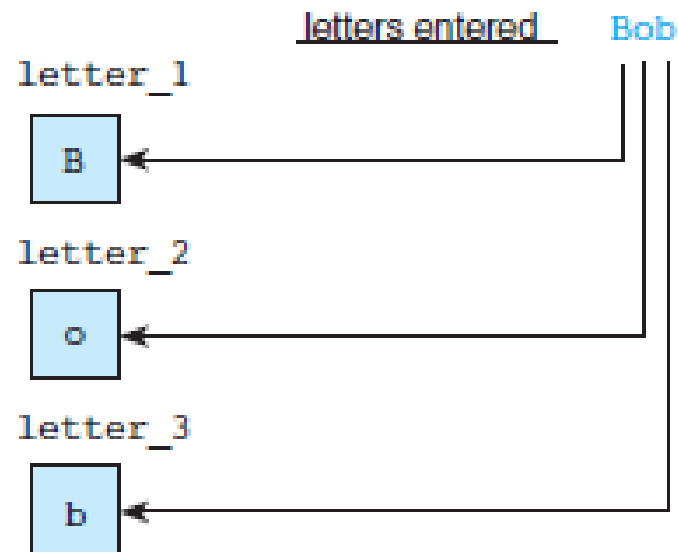


Figure 2.7

Scanning Data Line Bob



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

Syntax Display for scanf Function Call

SYNTAX: `scanf (format string, input list);`

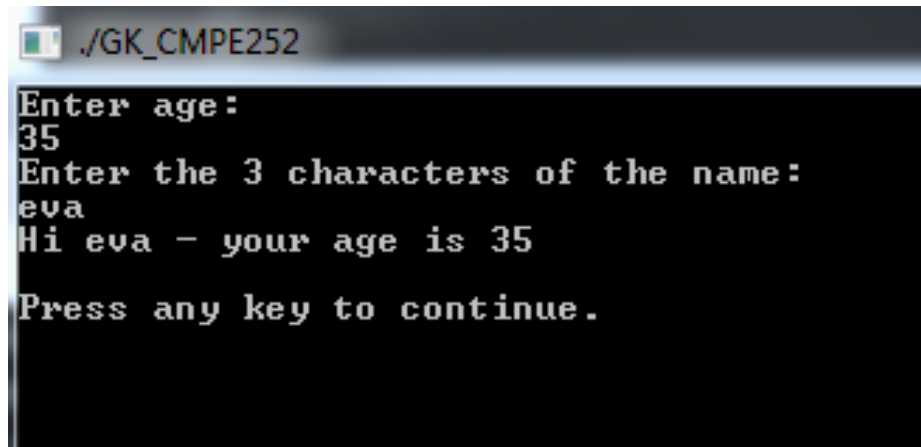
EXAMPLE: `scanf ("%c%d", &first_initial, &age);`

INTERPRETATION: The `scanf` function copies into memory data typed at the keyboard by the program user during program execution. The *format string* is a quoted string of placeholders, one placeholder for each variable in the *input list*. Each `int`, `double`, or `char` variable in the *input list* is preceded by an ampersand (&). Commas are used to separate variable names. The order of the placeholders must correspond to the order of the variables in the *input list*.

You must enter data in the same order as the variables in the *input list*. You should insert one or more blank characters or carriage returns between numeric items. If you plan to insert blanks or carriage returns between character data, you must include a blank in the format string before the `%c` placeholder.

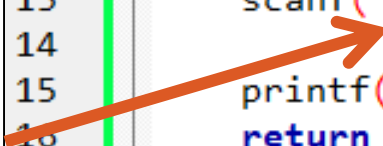
Quick Check

Write a C code which has an integer and 3 char variables, and produces the given output



```
./GK_CMPE252
Enter age:
35
Enter the 3 characters of the name:
eva
Hi eva - your age is 35
Press any key to continue.
```

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int age;
6      char c_1;
7      char c_2;
8      char c_3;
9
10     printf("Enter age:\n");
11     scanf("%d", &age);
12     printf("Enter the 3 characters of the name: \n");
13     scanf(" %c%c%c", &c_1, &c_2, &c_3);
14
15     printf("Hi %c%c%c, your age is %d\n", c_1, c_2, c_3, age);
16     return 0;
17 }
18
```



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);`

HOA2 – Q3,4,5

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

TABLE 2.10 Results of Integer Division

3 / 15 = 0	18 / 3 = 6
15 / 3 = 5	16 / -3 varies
16 / 3 = 5	0 / 4 = 0
17 / 3 = 5	4 / 0 is undefined

Expressions with Multiple Operators

- unary operator
 - an operator with one operand
 - unary plus (+), unary negation (-)
 - ex. $x = -y$; ($x = +y$ is useless, generally unary(+) don't used)
- 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

$x * y * z + a / b - c * d$

can be written in a more readable form using parentheses:

$(x * y * z) + (a / b) - (c * d)$

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 level are evaluated left to right.

Figure 2.9

Evaluation Tree for $\text{area} = \text{PI} * \text{radius} * \text{radius};$

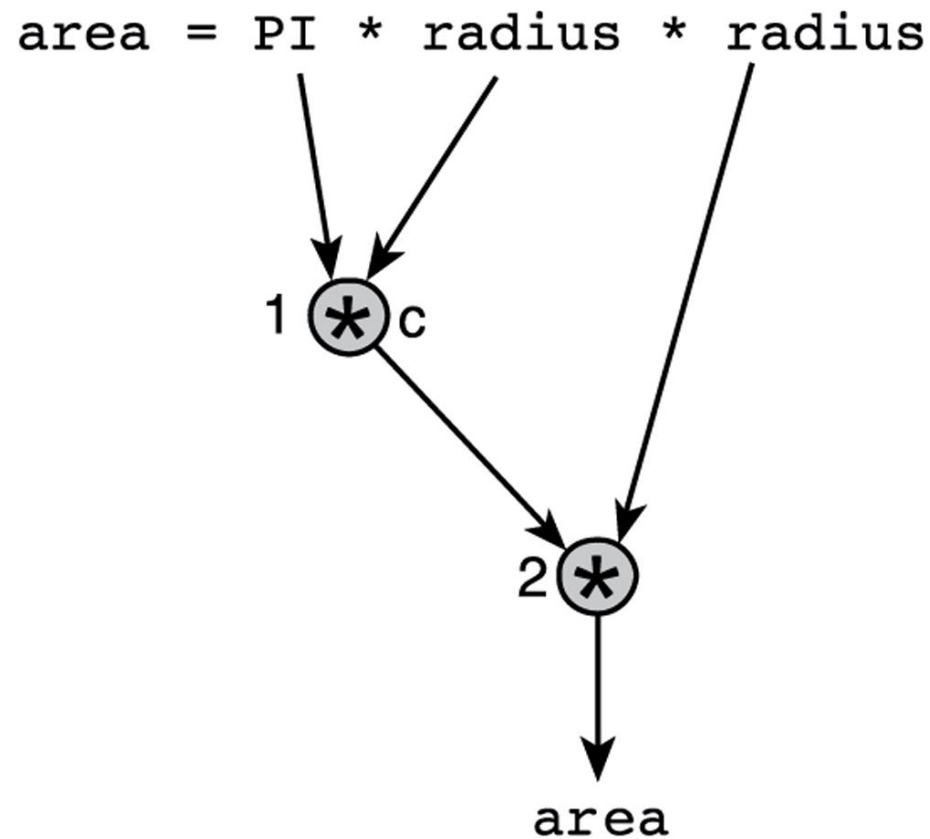


Figure 2.10

Step-by-Step Expression Evaluation

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

Writing Mathematical Formulas in C

You may encounter two problems in writing a mathematical formula in C. First, multiplication often can be implied in a formula by writing the two items to be multiplied next to each other, for example, $a = bc$. In C, however, you must always use the `*` operator to indicate multiplication, as in

```
a = b * c;
```

The other difficulty arises in formulas with division. We normally write the numerator and the denominator on separate lines:

$$m = \frac{y - b}{x - a}$$

In C, however, the numerator and denominator are placed on the same line. Consequently, parentheses are often needed to separate the numerator from the denominator and to indicate clearly the order of evaluation of the operators in the expression. The above formula would be written in C as

```
m = (y - b) / (x - a);
```

TABLE 2.13 Mathematical Formulas as C Expressions

Mathematical Formula	C Expression
1. $b^2 - 4ac$	<code>b * b - 4 * a * c</code>
2. $a + b - c$	<code>a + b - c</code>
3. $\frac{a + b}{c + d}$	<code>(a + b) / (c + d)</code>
4. $\frac{1}{1 + x^2}$	<code>1 / (1 + x * x)</code>
5. $a \times -(b + c)$	<code>a * -(b + c)</code>

Table 2.13 shows several mathematical formulas rewritten in C.

The points illustrated in these examples can be summarized as follows:

- Always specify multiplication explicitly by using the operator `*` where needed (formulas 1 and 4).
- Use parentheses when required to control the order of operator evaluation (formulas 3 and 4).
- Two arithmetic operators can be written in succession if the second is a unary operator (formula 5).

Figure 2.11

Evaluation Tree and Evaluation for

$$v = (p2 - p1) / (t2 - t1);$$

Draw the evaluation tree for the calculation of v

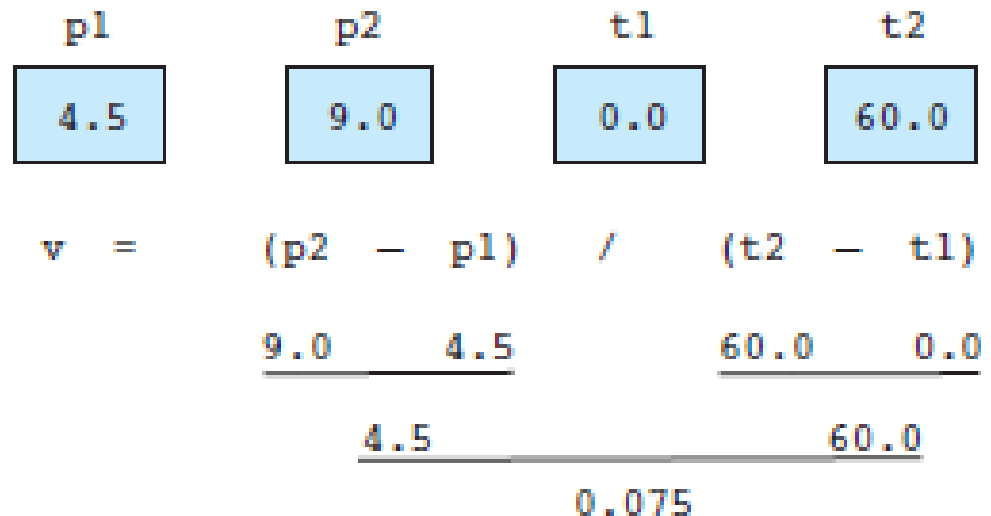
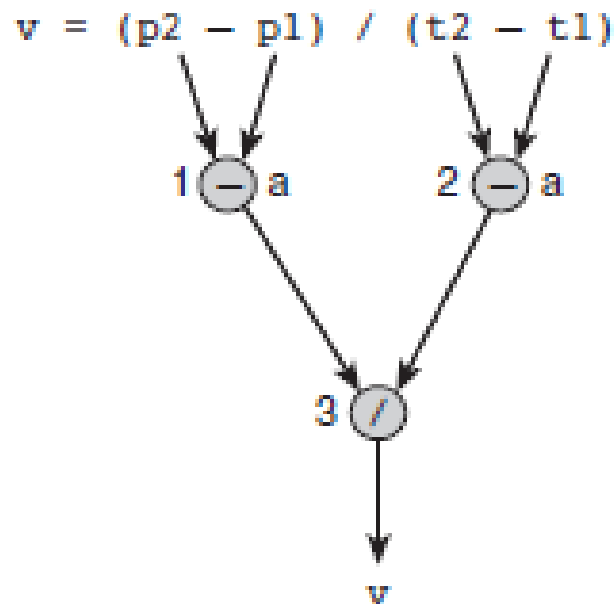
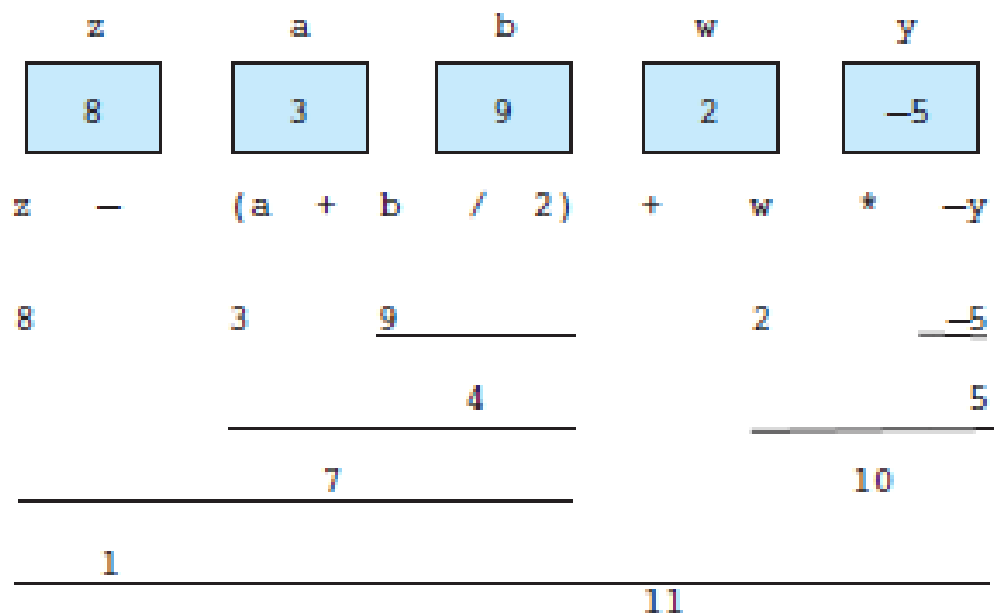
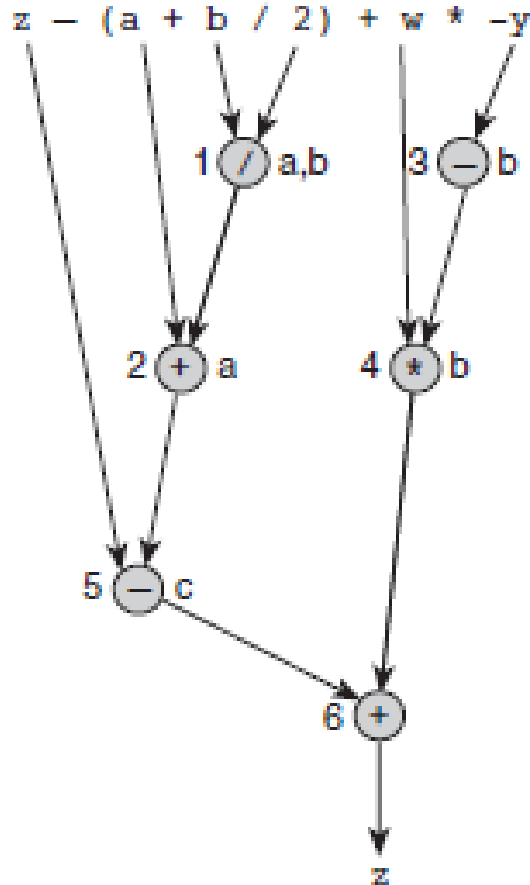


Figure 2.12

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

Draw the evaluation tree for the calculation



Data Type of an Expression

Expressions having the same type of operands return the type of the operands:

TABLE 2.9 Arithmetic Operators

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

What if types of the operands are different, e.g `<int>/<double> = ?`:

What is the result of

`(5 / 2.0) = ?`

`5.1 + 2 = ?`

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

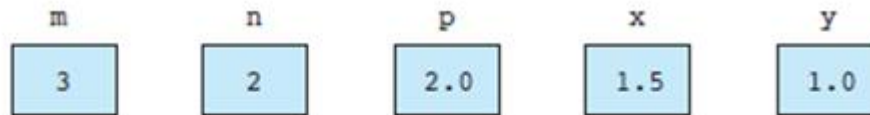
Mixed Type Expression

- In general, an expression of the form
(A <*arithmetic_operator*> B)

if *both* A and B are of type int ;
then expression is of type int

otherwise, if one of them is double,
expression is of type double

Mixed Type Assignment



In a mixed-type assignment such as

```
y = m / n;
```

$m / n \rightarrow \text{int}$

y is what it was declared as !!! – important!

```
x = 9 * 0.5;
```

```
n = 9 * 0.5; —————> double expression assigned to int variable ?
```

Only the integer part is stored, therefore,

$x = 4.5$ (since it is double)

$n = 4$ (since it was declared as int)

Type Conversion through casts

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

C allows the programmer to convert the type of an expression by placing the desired type in parentheses before the expression, an operation called a **type cast**. In the previous section, we saw that the fractional part of a real value is lost when it is assigned to an `int`. Use a type cast to show that this happens.

```
n = (int)(9 * 0.5);
```

Two common uses of type casts are shown in Table 2.12—avoiding integer division when computing an average and rounding a type `double` value by adding 0.5 and converting the result to `int`.

`x=(int) (9 * 0.5) ; -> (we know that x was declared as double)`

So what is the result ?

`x = 4.0 ; (expression result casted to integer, then assigned to double)`

Casting Example

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5      int num_students = 7;
6      int total_score = 556;
7      double avg1, avg2;
8      avg1 = (double)total_score/(double)num_students;
9      avg2 = (double)(total_score/num_students);
10     printf("avg1: %f, avg2: %f", avg1, avg2);
11 }
```

- avg1: 79.428571
- avg2: 79.000000

Characters as Integers

- Characters are actually represented as an integer with 1 byte in the memory.
- The ASCII character set is used to represent characters.
- **→ American Standard Code for Information Interchange (ASCII)**
- The ASCII code associates an integer value for each symbol in the character set, such as letters, digits, punctuation marks, special characters, and control characters.

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

The ASCII table has 128 characters, with values from 0 through 127

Characters as Integers

Since characters are represented by integer codes, C permits conversion of type `char` to type `int` and vice versa. For example, you could use the following to find out the code your implementation uses for a question mark:

```
qmark_code = (int) '?';  
printf("Code for ? = %d\n", qmark_code);
```

You can perform arithmetic operations on characters. For example, the expression `'A' + 1` adds 1 to the code for `'A'` and its value is the next character after `'A'` which is `'B'` in ASCII.

Chars as Integers

```
char c1 = 'A';  
char c2 = c1 + 1;  
printf("%c", c2);
```

Output?

B

Check ASCII table!

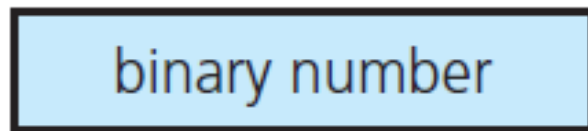
Numerical Inaccuracies

- **Representational error** (sometimes called *round-off error*):
 - When processing data of type **double**, an error may occur in representing real numbers
 - certain fractions cannot be represented exactly in the decimal number system (e.g., the fraction $1/3$ is $0.333333 \dots$),
 - So some fractions cannot be represented exactly as binary numbers in the mantissa of the type double format.
 - Error will depend on the number of bits used in the mantissa: the more bits, the smaller the error.

Figure 2.2

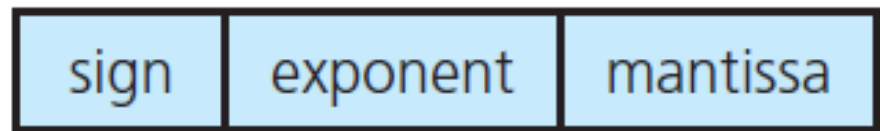
Internal Format of Type int and Type double

type `int` format



32-bits

type `double` format



1 bit

11 bits

52 bits

64-bits

double: The storage area occupied by the number is divided into three sections:
The **sign** (0 for positive numbers, 1 for negative numbers),
The **mantissa** is a binary fraction.
The **exponent** is an integer.
The mantissa and exponent are chosen so that $\text{real number} = \text{mantissa} \times 2^{\text{exponent}}$

Numerical Inaccuracies

- Cancellation error
 - Errors may occur when manipulating very large and very small real numbers.
 - When you add a large number and a small number, the larger number may “cancel out” the smaller number,

e.g. $10000000.0 + 0.0000001234$ is equal to 10000000.0 on some computers

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
 - Arithmetic overflow can occur when processing very large integer values as well.

SUPERMARKET COIN VALUE PROGRAM

Case Study

Problem

- Manually enter the number of each kind of coin
- Calculate total cents
- Output the total in dollars and leftover change as a personal credit slip
 - $total\ dollars = \frac{total\ cents}{100}$
 - $change = total\ cents \% 100$

1 dollar = 100 cents
 1 quarter = 25 cents
 1 dime = 10 cents
 1 nickel = 5 cents

```

Type in your 3 initials and press return> ggk
ggk, 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

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

Figure 2.13

```

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

```

Figure 2.13

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


Formatting Numbers in Program Output

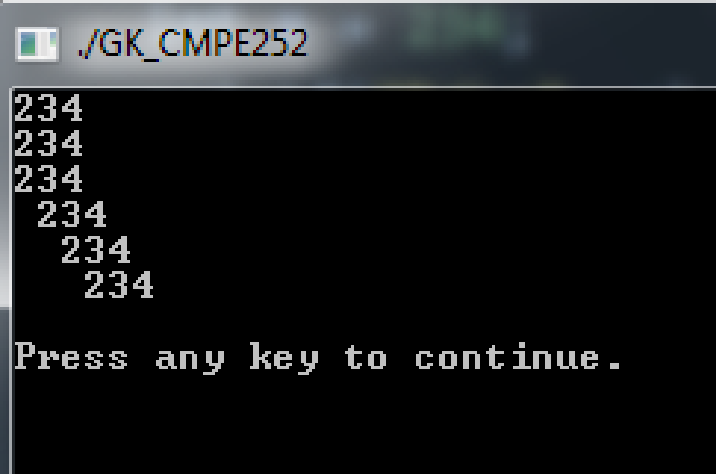
- field width
 - the number of columns used to display a value
- When formatting doubles, you must indicate the total field width needed and the number of decimal places desired.

Formatting Numbers in Program Output

```
int main(void)
{
    int s = 234;

    printf("%d\n", s);
    printf("%2d\n", s);
    printf("%3d\n", s);
    printf("%4d\n", s);
    printf("%5d\n", s);
    printf("%6d\n", s);

    return 0;
}
```

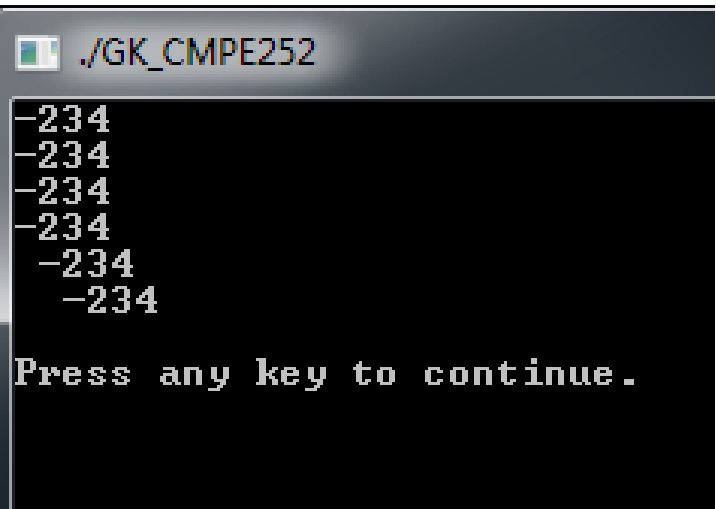


```
./GK_CMPE252
234
234
234
 234
 234
 234
Press any key to continue.
```

```
int main(void)
{
    int s = -234;

    printf("%d\n", s);
    printf("%2d\n", s);
    printf("%3d\n", s);
    printf("%4d\n", s);
    printf("%5d\n", s);
    printf("%6d\n", s);

    return 0;
}
```



```
./GK_CMPE252
-234
-234
-234
 -234
 -234
 -234
Press any key to continue.
```

Formatting Numbers in Program Output

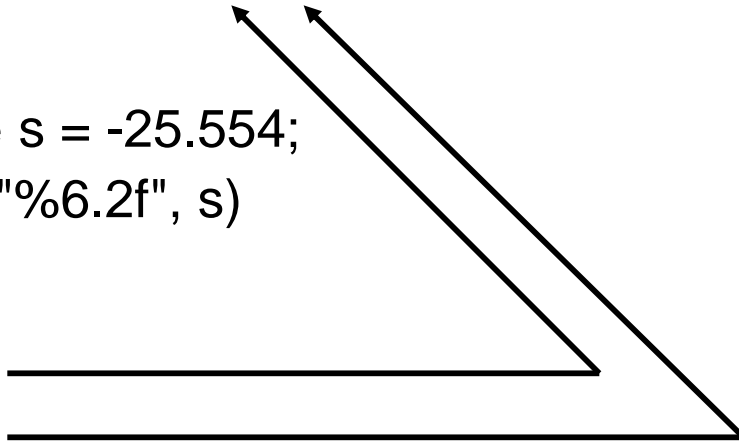
- What does `%n.mf` means?

- E.g.

```
double s = -25.554;  
printf ("%6.2f", s)
```

n = 6

m = 2



What to print on screen?

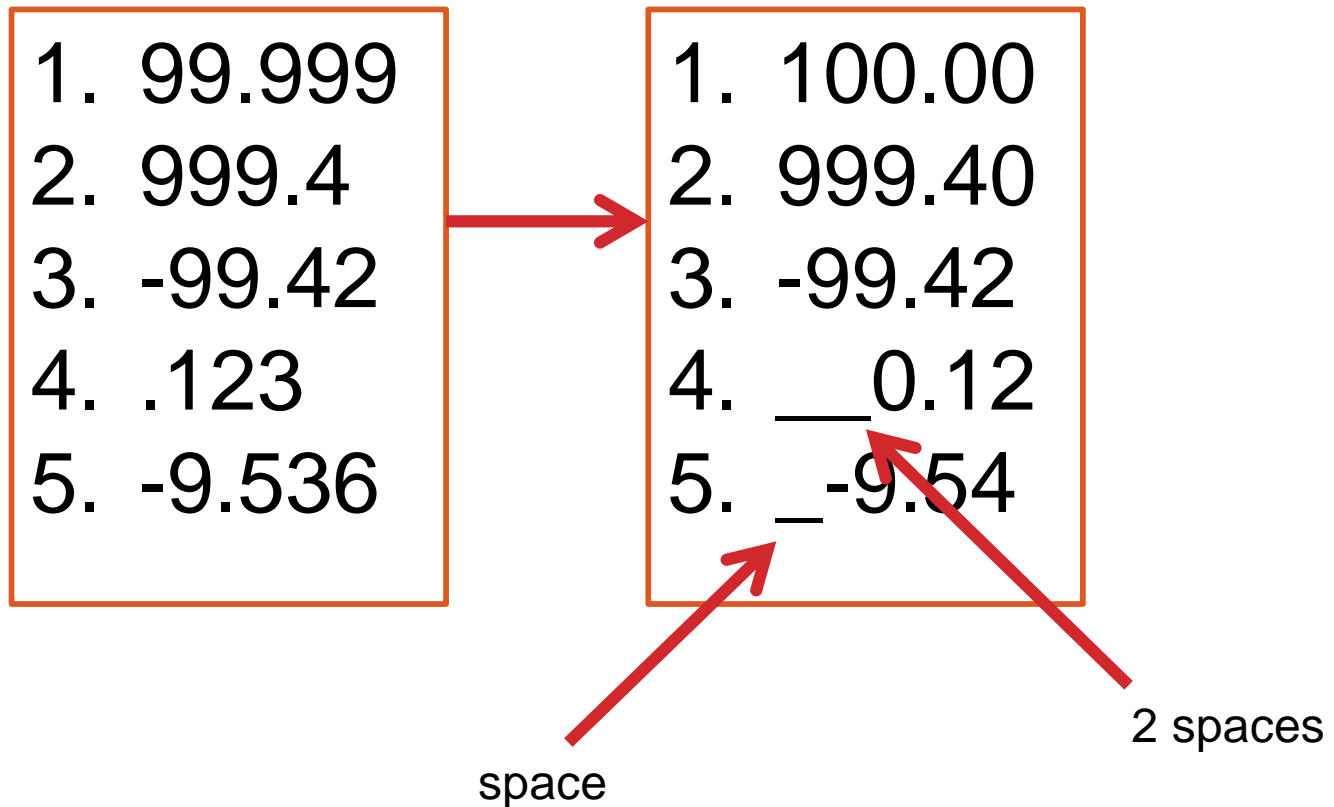
-25.55

Then, what is n and m?

n = total field width

m = number of decimal places

Quick Check (using %6.2f)



Remove Leading Blanks

- HOW?
- Simply omit total width and use as `%.mf` or `%d`
- E.g.

```
double s = -25.554;  
printf("%.2f", s);
```

```
-25.55
```

```
double s = -25.554;  
printf("%8.2f", s);
```

```
__-25.55
```

2 spaces



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 – our case until now
- 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

```
int main(void)
{
    double miles, kms;

    scanf("%lf", &miles);
    printf("The distance in miles is %.2f.\n", miles);
    kms = KMS_PER_MILE * miles;
    printf("That equals %f kilometers.\n", kms);

    return (0);
}
```

```
/* Figure 1.13 Miles-to-Kilometers Conversion Program */
/*
 * Converts distance in miles to kilometers.
 */

#include <stdio.h>           /* printf, scanf definitions */
#define KMS_PER_MILE 1.609   /* conversion constant */

int main(void)
{
    double miles, /* input - distance in miles. */
           kms;   /* output - 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);
}
```

```
34
The distance in miles is 34.00.
That equals 54.71 kms.

Press any key to continue.
```

How about using **input redirection**?
Take input from a file instead of
keyboard?

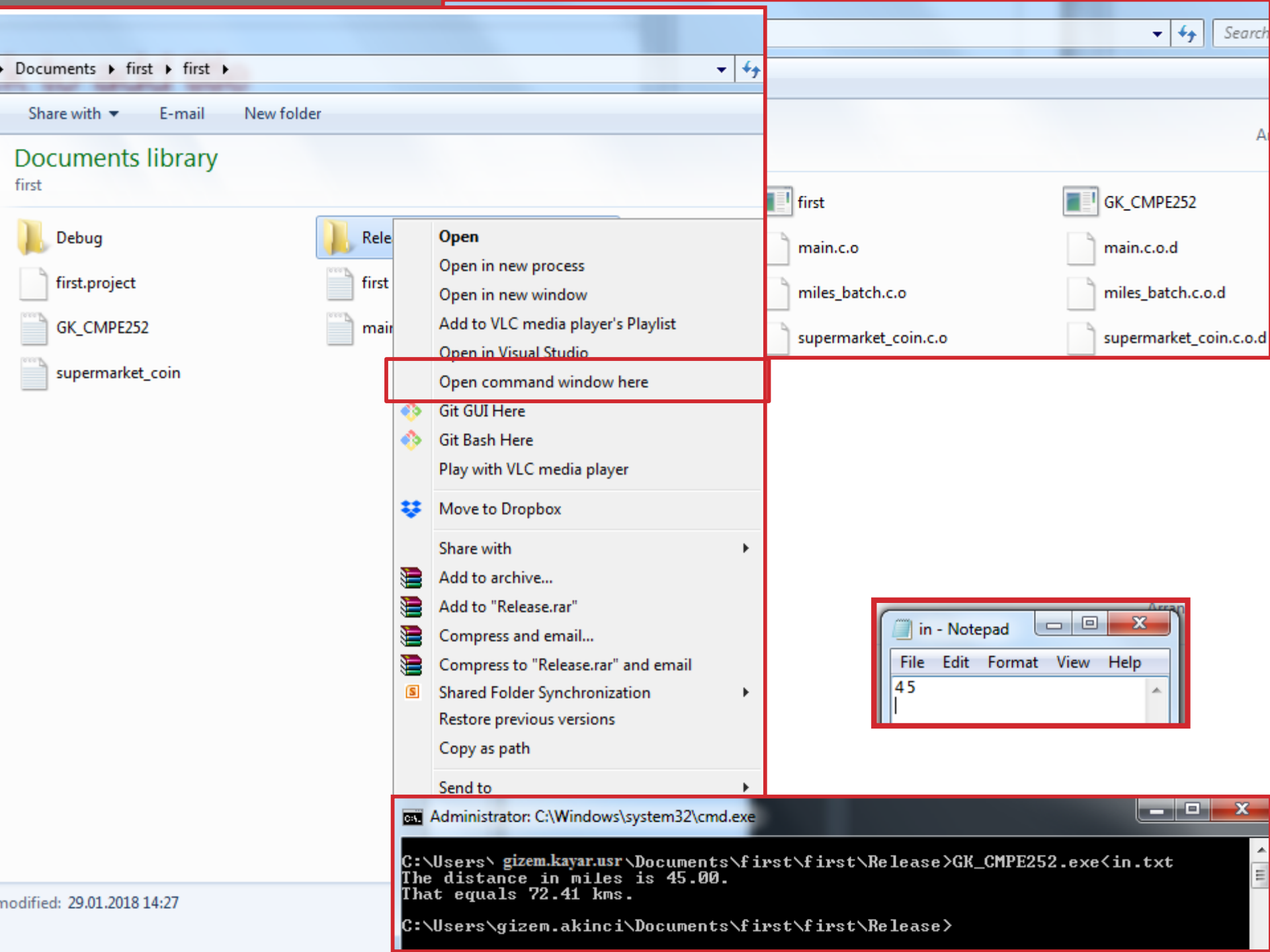
Input/Output Redirection

```
progrname.exe<in.txt
```

```
progrname.exe>out.txt
```

You can also setup the options of your IDE for this task!!!

More will come about file I/O in upcoming weeks.



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
Quickcheck

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
```

No semicolon

miles not declared

Comment closing is wrong
No space!!

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

May or may not cause to program to abort in run time

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.

References

1. Problem Solving & Program Design in C, Jeri R. Hanly & Elliot B. Koffman, Pearson 8. Edition, Global Edition
2. https://en.wikipedia.org/wiki/Programming_languages_used_in_most_popular_websites
3. https://en.wikibooks.org/wiki/C_Programming/Procedures_and_functions
4. <https://cdn.guru99.com/images/uploads/2012/07/GarbageCollection4.jpg>
5. https://en.wikipedia.org/wiki/GNU_Compiler_Collection
6. <http://www.mingw.org>