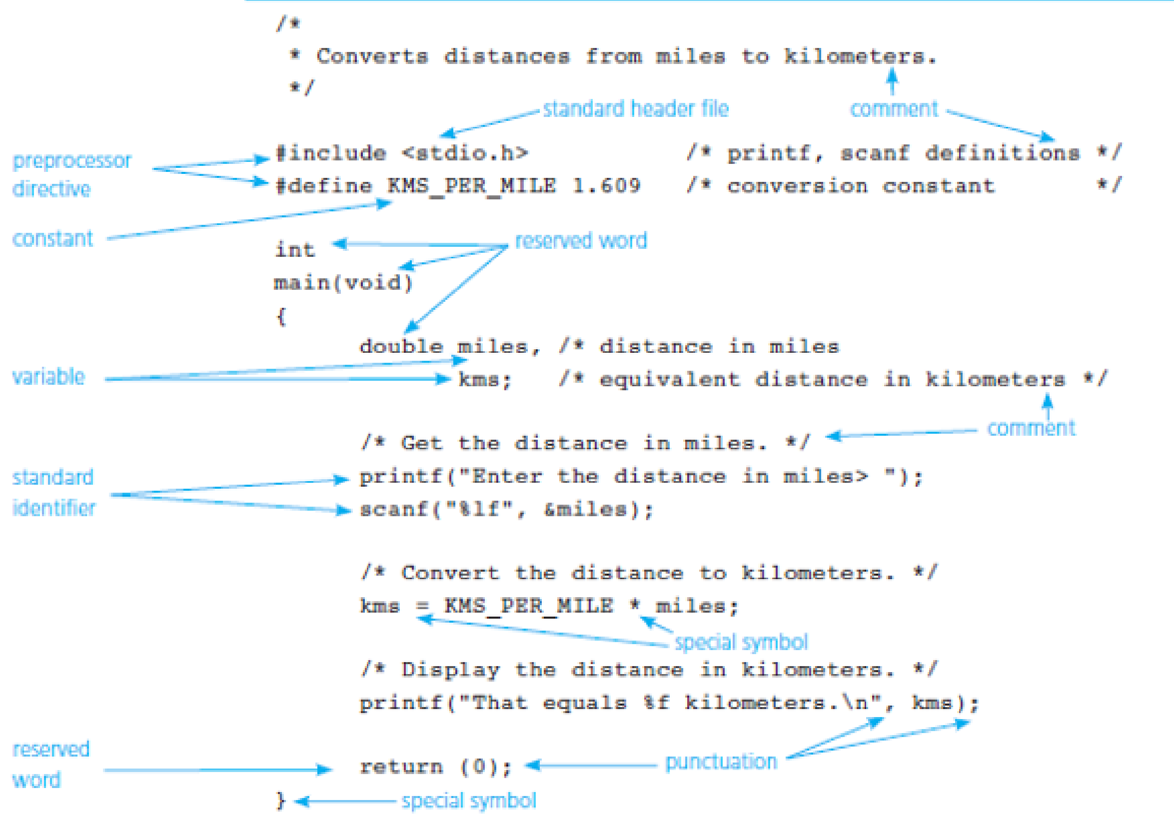
Engr 101 **August 23, 2019**
Day 2

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



Variables in C – Naming rules:

- 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.

Data Types:

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

Console I/O

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

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

- 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

Placeholders in format string

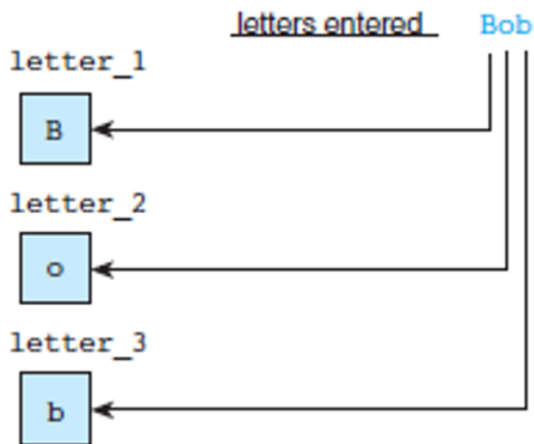
Placeholder	Variable Type	Function Use
% c	char	printf/scanf
% d	int	printf/scanf
% f	double	printf
% lf	double	scanf

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);
```



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

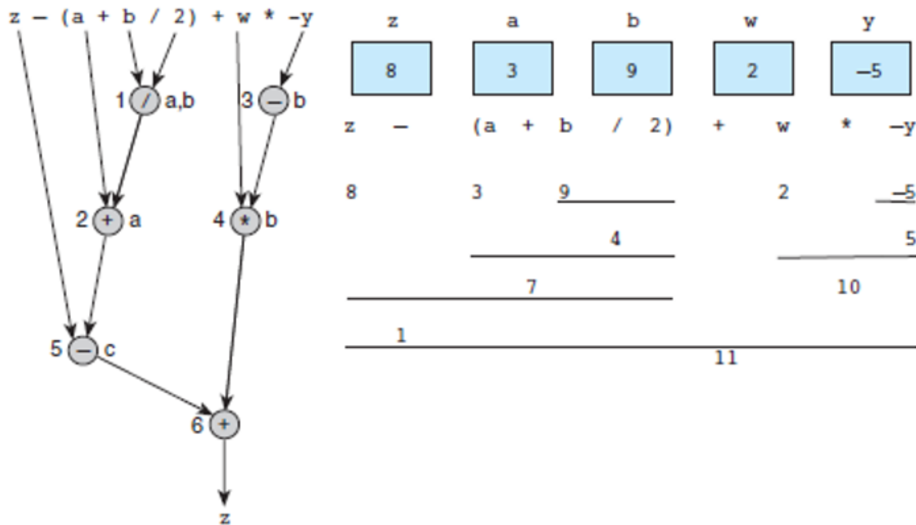
Data Type of an Expression

- 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
  - *, /, % next
  - binary +, - last
- 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 2.12
Evaluation Tree and Evaluation for z - (a + b / 2) + w * -y



Supermarket Coin Value Program

```
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 your 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> ");
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.     total_dollars = total_cents / 100;
40.     change = total_cents % 100;
41.
42.     /* Display the credit slip with value in dollars and change. */
```

(Continued)

```
43.     printf("\n\n%c%c%c Coin Credit\nDollars: %d\nChange: %d cents\n",
44.            first, middle, last, total_dollars, change);
45.
46.     return (0);
47. }

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 Values of Type int

Specifying the format of an integer value displayed by a C program is fairly easy. You simply add a number between the `%` and the `d` of the `%d` placeholder in the `printf` format string. This number specifies the **field width**—the number of columns to use for the display of the value. The statement

```
printf("Results: %3d meters = %4d ft. %2d in.\n",
       meters, feet, inches);
```

indicates that 3 columns will be used to display the value of `meters`, 4 columns will be used for `feet`, and 2 columns will be used for `inches` (a number between 0 and 11). If `meters` is 21, `feet` is 68, and `inches` is 11, the program output will be

```
Results:  21 meters =   68 ft. 11 in.
```

In this line, notice that there is an extra space before the value of `meters (21)` and two extra spaces before the value of `feet` (68). The reason is that the placeholder for `meters (%3d)` allows space for 3 digits to be printed. Because the value of `meters` is between 10 and 99, its two digits are displayed *right-justified*, preceded by one blank space. Because the placeholder for `feet` (`%4d`) allows room for 4 digits, printing its two-digit value right-justified leaves two extra blank spaces. We can use the placeholder `%2d` to display any integer value between -9 and 99. The placeholder `%4d` works for values in the range -999 to 9999. For negative numbers, the minus sign is included in the count of digits displayed.

Table 2.14 shows how two integer values are displayed using different format string placeholders. The character ▮ represents a blank character. The last line shows that C expands the field width if it is too small for the integer value displayed.

Formatting Values of Type double

To describe the format specification for a type `double` value, we must indicate both the total *field width* needed and the number of *decimal places* desired. The total field width should be large enough to accommodate all digits before and after the decimal point. There will be at least one digit before the decimal point because a

TABLE 2.14 Displaying 234 and −234 Using Different Placeholders

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

TABLE 2.15 Displaying x Using Format String Placeholder %6.2f

Value of x	Displayed Output	Value of x	Displayed Output
`-99.42`	`-99.42`	`-25.554`	`-25.55`
`.123`	`■■0.12`	`99.999`	`100.00`
`-9.536`	`■-9.54`	`999.4`	`999.40`

zero is printed as the whole-number part of fractions that are less than `1.0` and greater than `-1.0`. We should also include a display column for the decimal point and for the minus sign if the number can be negative. The form of the format string placeholder is `%`*n*`.`*m*`f` where *n* is a number representing the total field width, and *m* is the desired number of decimal places.

If `x` is a type `double` variable whose value will be between `-99.99` and `999.99`, we could use the placeholder `%6.2f` to display the value of `x` to an accuracy of two decimal places. Table 2.15 shows different values of `x` displayed using this format specification. The values displayed are rounded to two decimal places and are displayed right-justified in six columns. When you round to two decimal places, if the third digit of the value's fractional part is `5` or greater, the second digit is incremented by `1` (`-9.536` becomes `-9.54`). Otherwise, the digits after the second digit in the fraction are simply dropped (`-25.554` becomes `-25.55`).

Table 2.16 shows some values that were displayed using other placeholders. The last line shows it is legal to omit the total field width in the format string placeholder. If you use a placeholder such as `%.`*m*`f` to specify only the number of decimal places, the value will be printed with no leading blanks.

TABLE 2.16 Formatting Type double Values

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`