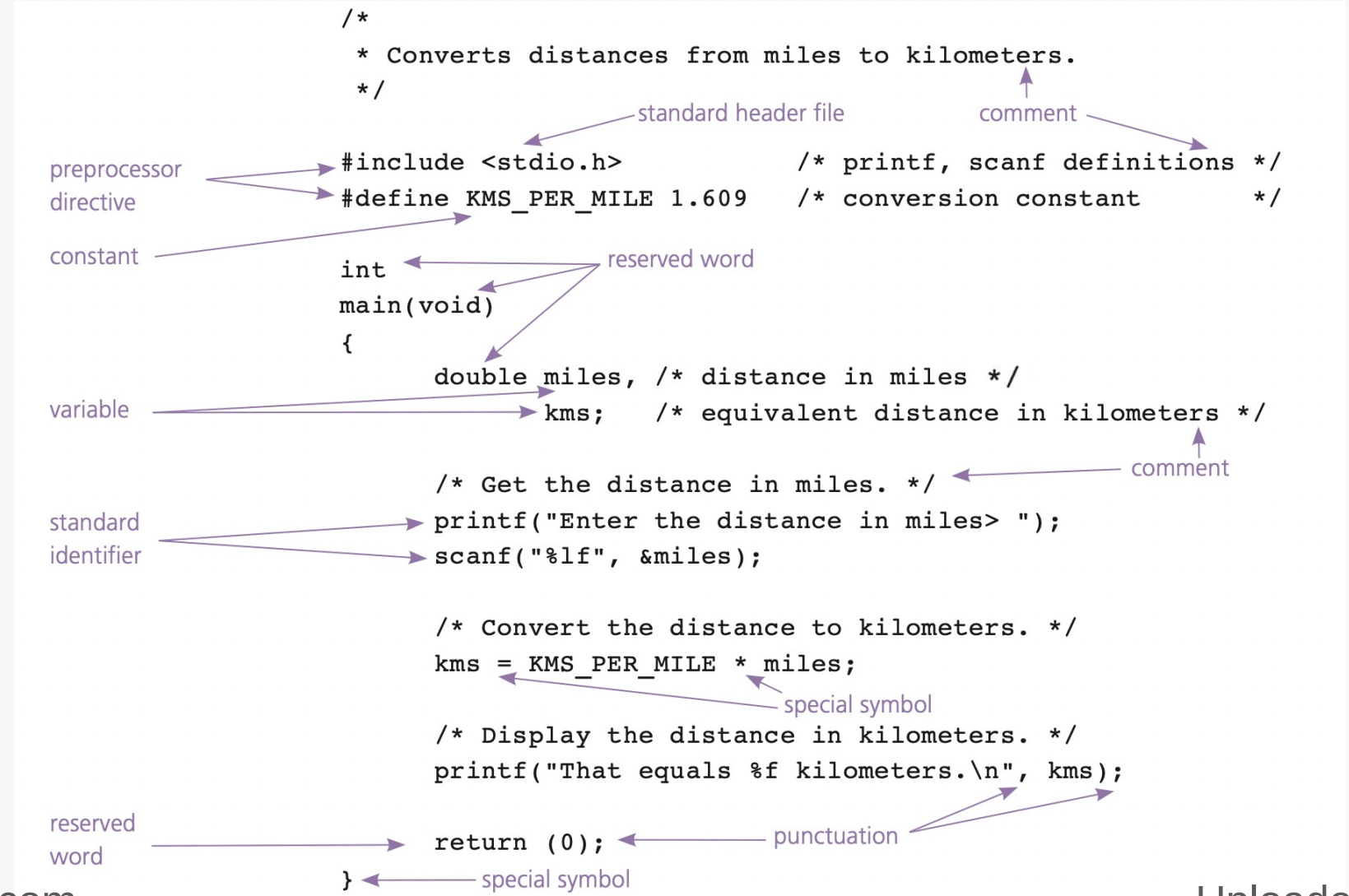
FACULTY OF ENGINEERING AND TECHNOLOGY

COMPUTER SCIENCE DEPARTMENT

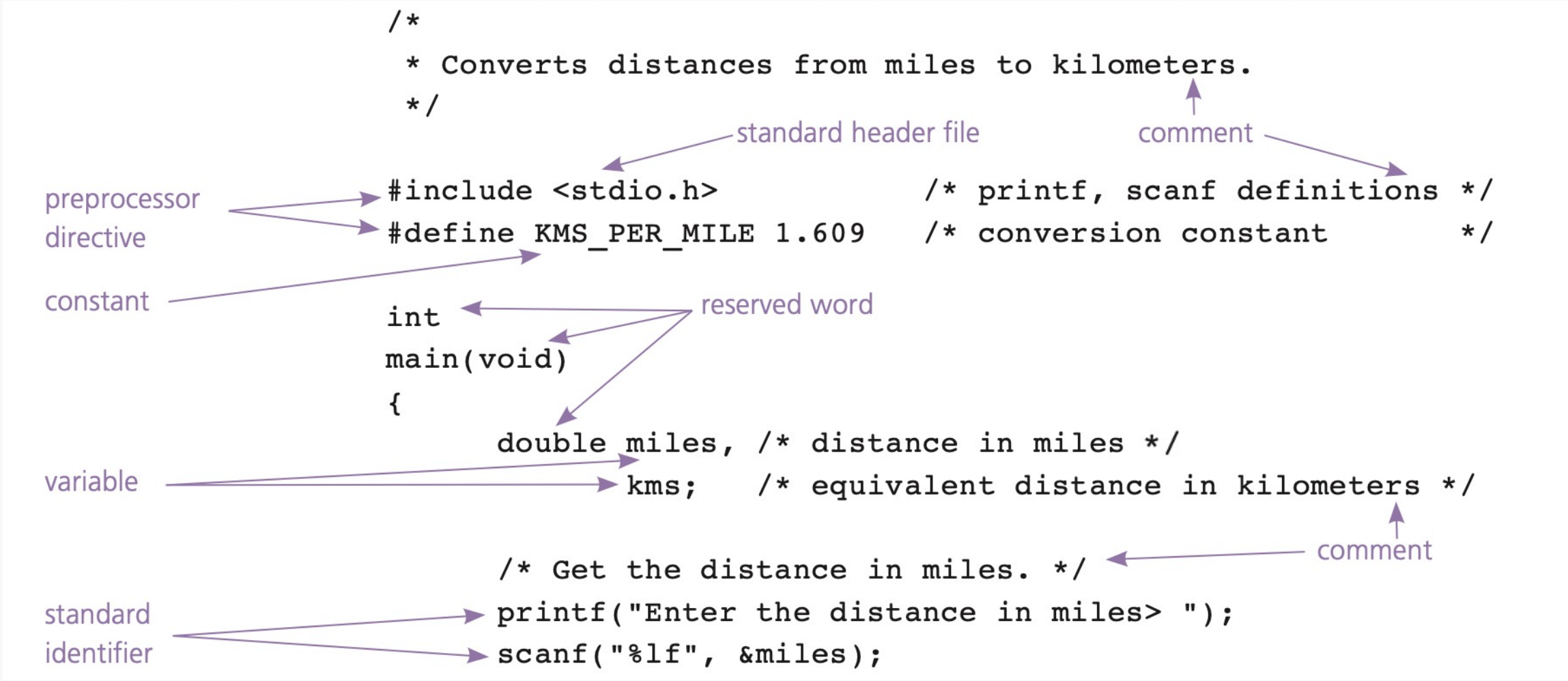
COMP1310
Introduction to Computer and Computing Ethics

OVERVIEW OF C

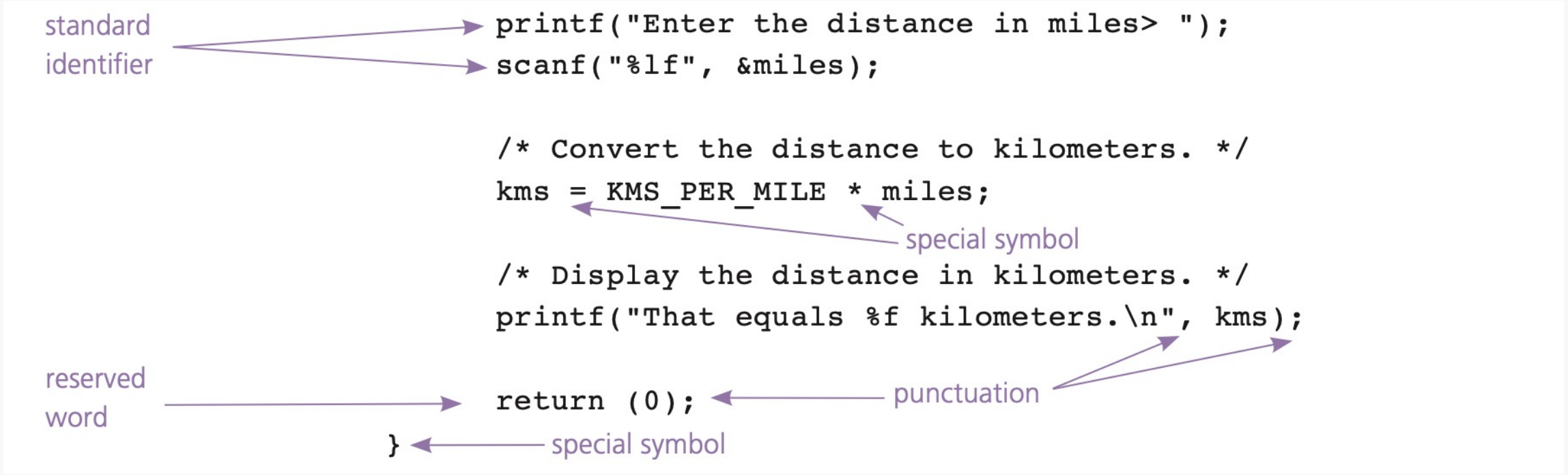
C Language Elements



C Language Elements



C Language Elements

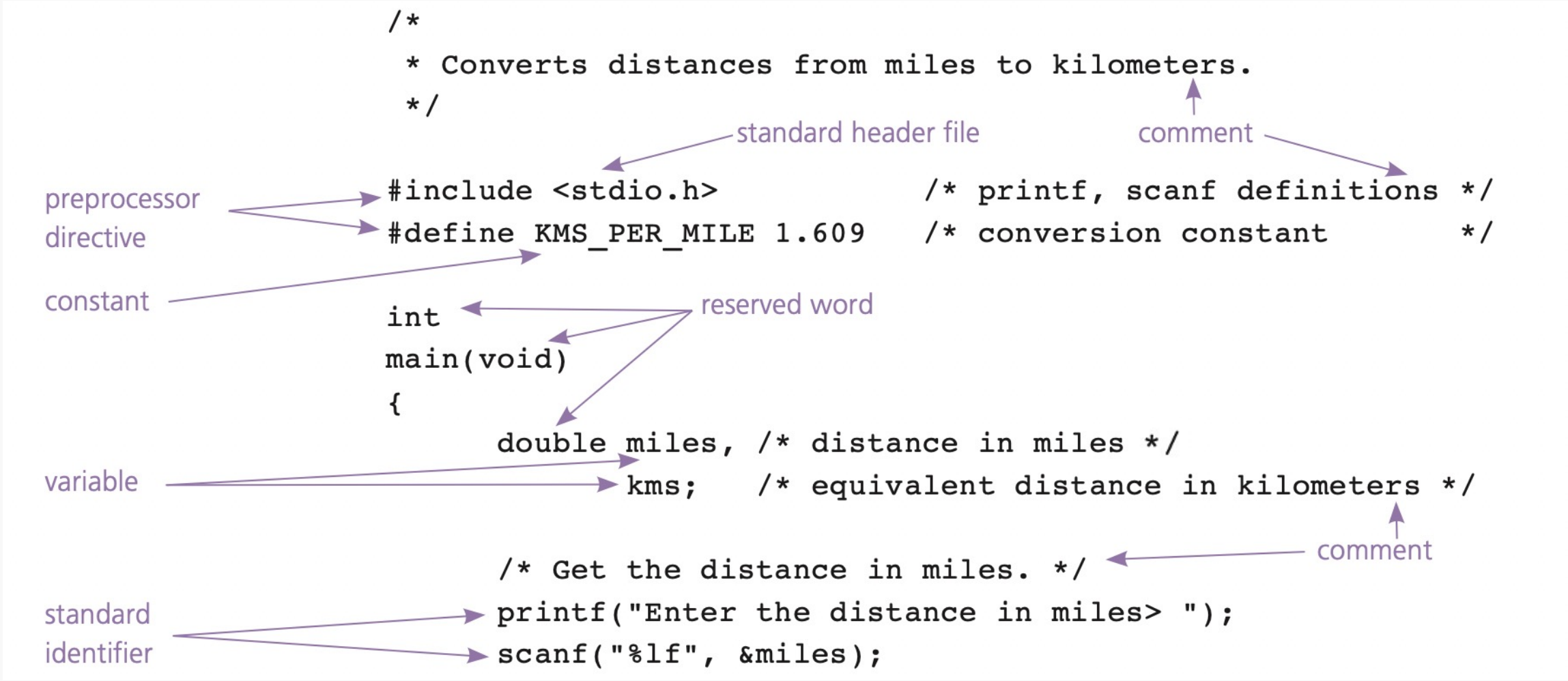


Preprocessor Directives

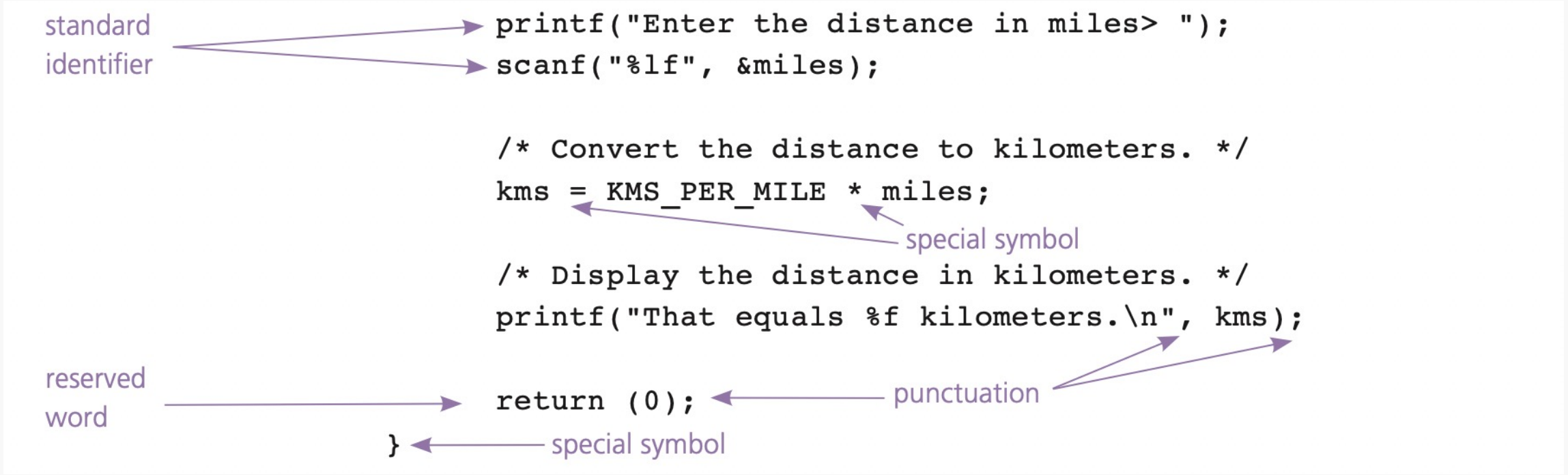
- The lines that begin with '#'.
- Commands that give instructions to the C preprocessor.
- The C preprocess modifies the C program prior to its compilation.

- `#include` gives a program access to a library.
  - *A library is a collection of useful functions and symbols that may be accessed by a program.*
- `#define` associates a constant macro with a specific value.
  - *This means that every occurrence of the constant macro in the C program text gets replaced with the specified value.*

C Language Elements



C Language Elements



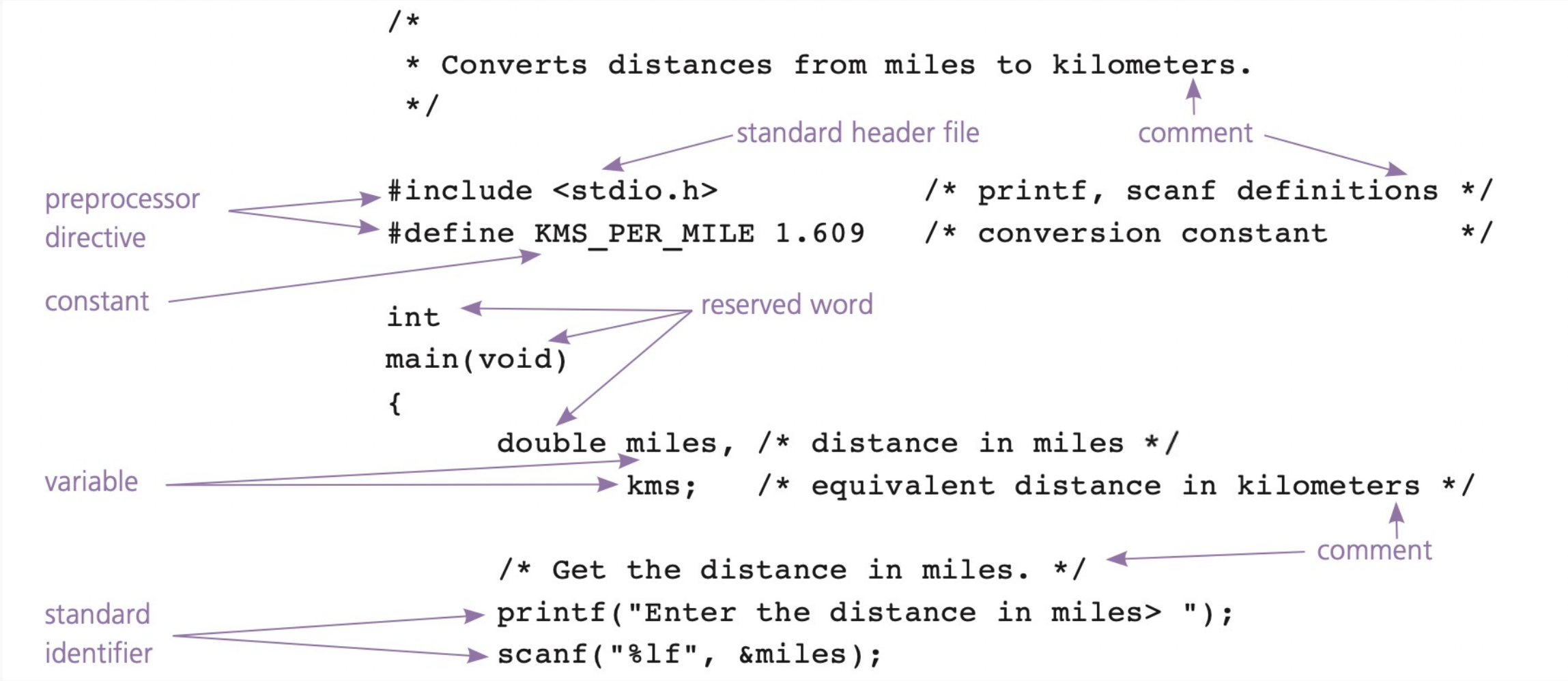
Comments

- Comments provide supplementary information making it easier for us to understand the program.
- Comments are ignored by the C preprocessor and compiler.

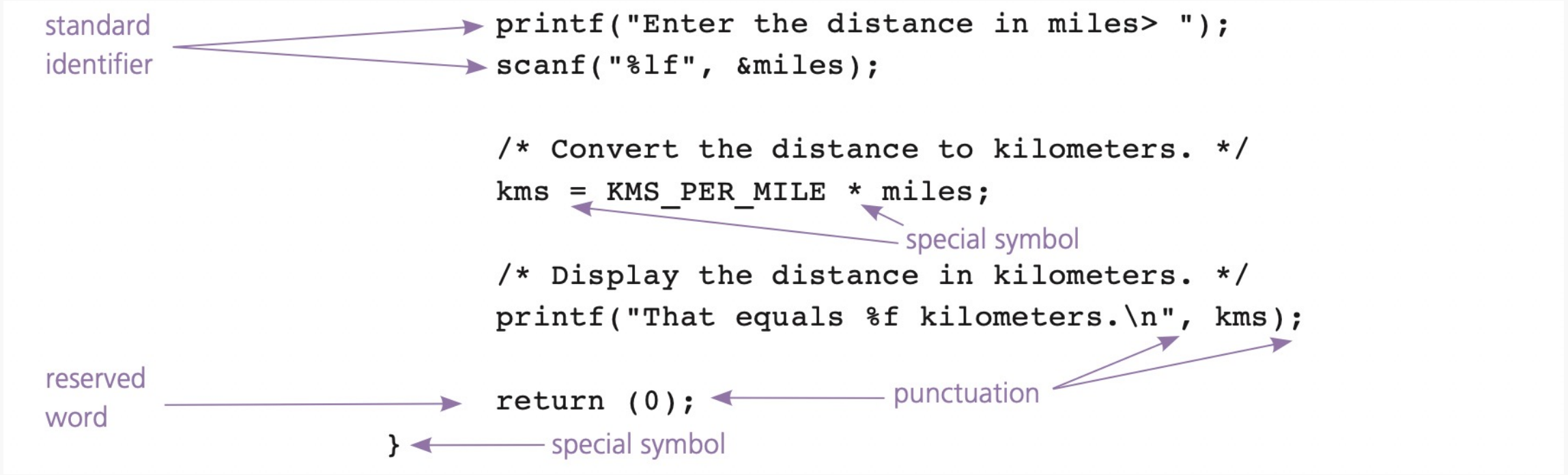
- Comments can be:
  - *One-line comments:* `// This is a one-line comment`
  - *Multi-line comments:*
```
/* This is a multi-line comment
   I can include more information here */
```

- It is a very good practice to document your work with comments, and you are expected to do it in every piece of code you write and submit.

C Language Elements



C Language Elements

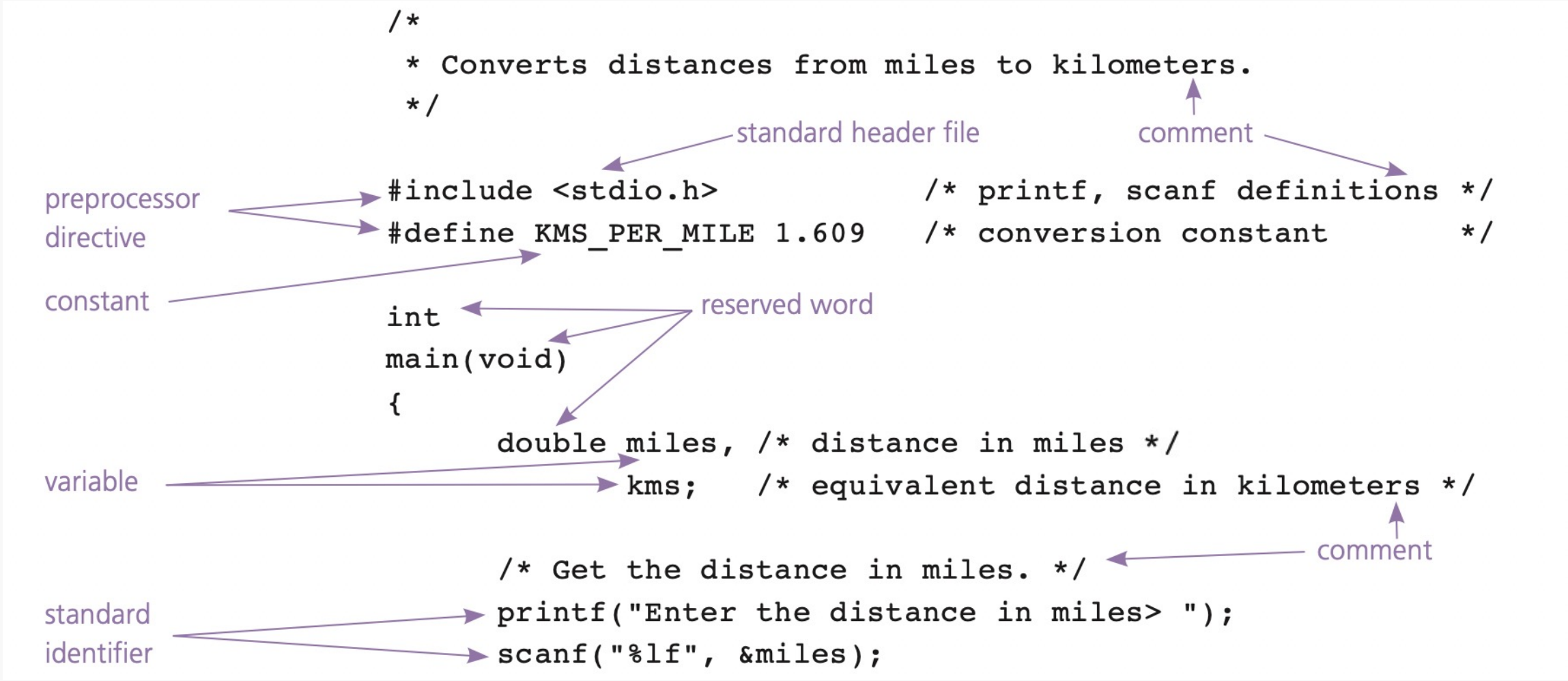


Reserved Words

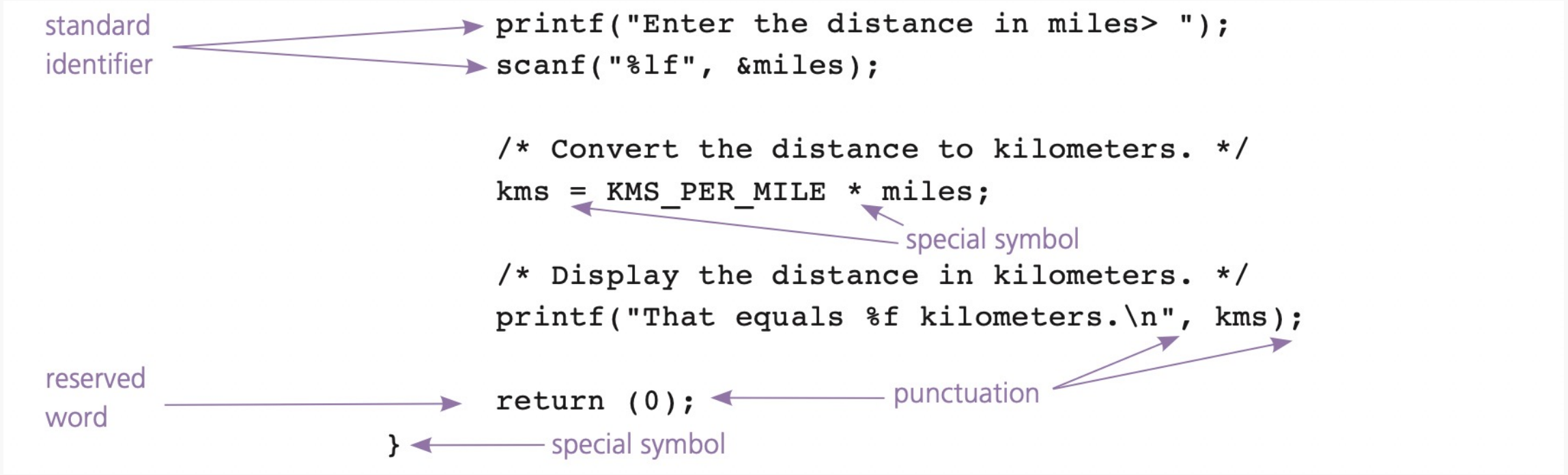
- Words that have special meaning in C.

- Examples:
  - `int`
  - `void`
  - `continue`
  - `return`
  - `case`

C Language Elements



C Language Elements

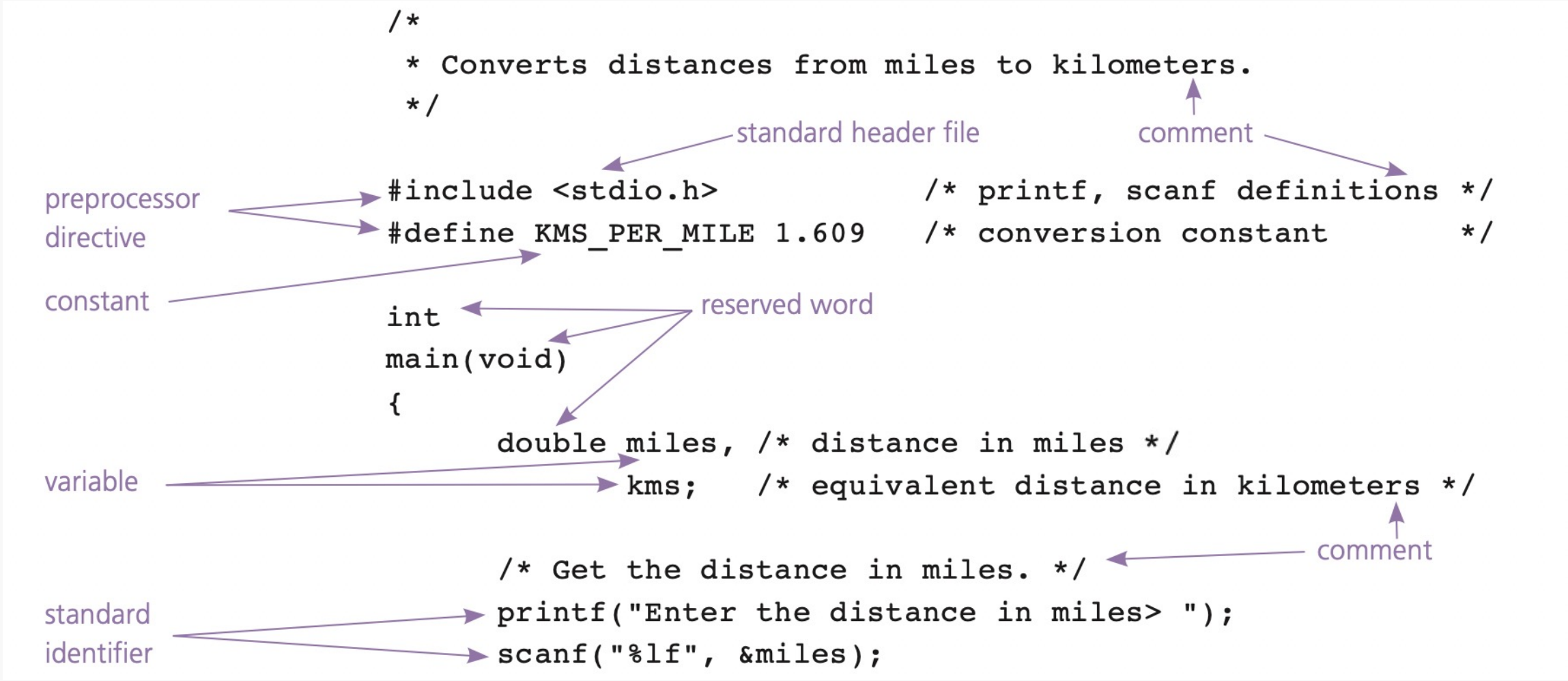


Standard Identifiers

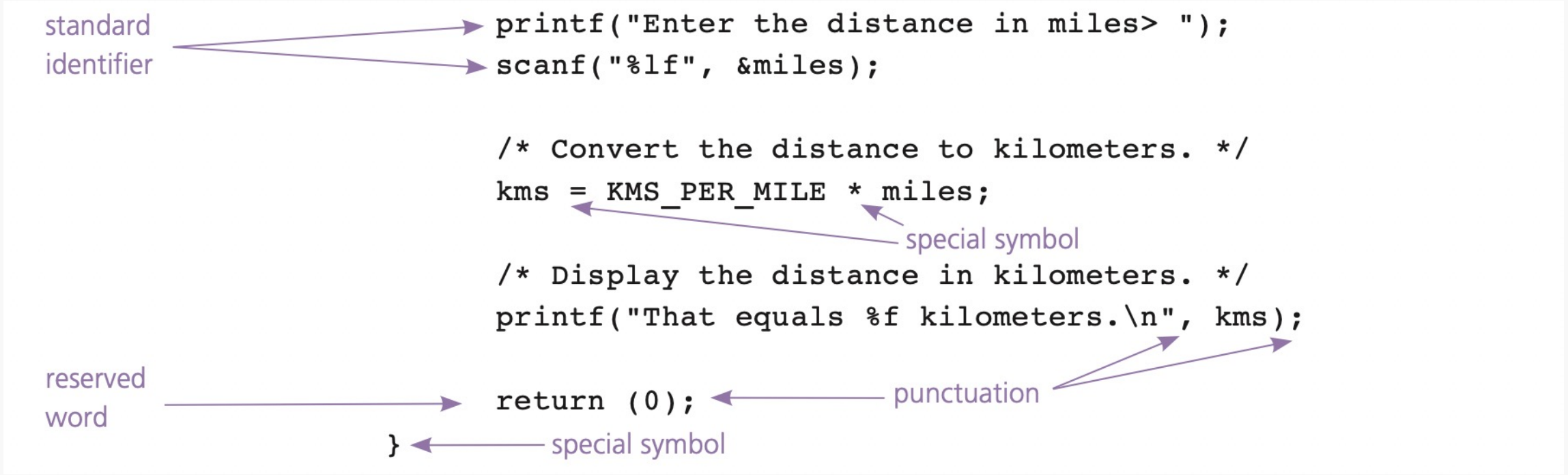
- Like reserved words: words that have a special meaning.
- Unlike reserved words: they can be redefined and used by the programmer for other purposes.

- Example: Names of operations defined in a library.
  - `printf`
  - `scanf`

C Language Elements



C Language Elements



Variables

- Variables are defined through declarations.
- A declaration is the part of a program that tells the compiler the name of a memory cell in the program.

- Variables are declared using a variable type and a variable name.
  - *A variable type defines the kind of information stored in the variable*
  - *A variable name is how this variable will be referenced throughout the program.*

```
variable_type variable_name;
```

Variables – cont.

- Variables can be declared in lists:

```
int x, y, z;
```

- Or separately:

```
int x;
int y;
int z;
```

- Variables can be given initial values when they are declared:

```
int x = 6, y = 8, z = 11;
```

Variables – cont.

- Data types:
  - `int`:
    - Integers, whole numbers
    - e.g. 2, 942, 0, -7
    - Stored in 16 bits

Variables – cont.

- Data types – cont.:
  - `float` *and* `double`*:*
    - A number with an integral part and a fractional part that are separated by a decimal point.
    - e.g. 4.14159, 0.0005, 150.0
    - A float is stored in 32 bits, while a double is stored in 64 bits.

Variables – cont.

- Data types – cont.:
  - `char`:
    - An individual character value: a letter, a digit, or a special symbol.
    - Enclosed in apostrophes.
      - *e.g. 'A', 'z', '2', '*', ' '*
    - Characters are stored as their Unicode value
      - *This means that the character and its Unicode value are interchangeable*

```
char x = 'A';
```

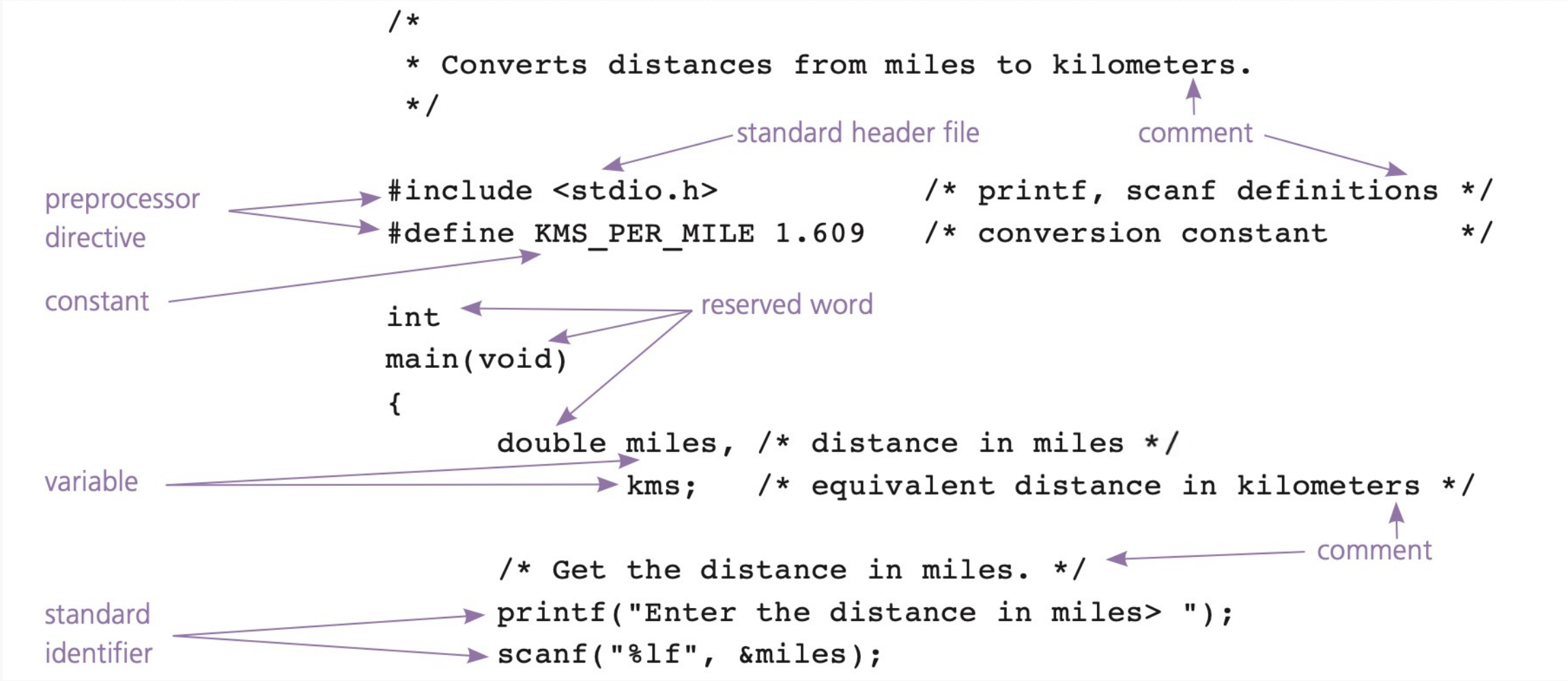
is equivalent to

```
char x = 65;
```

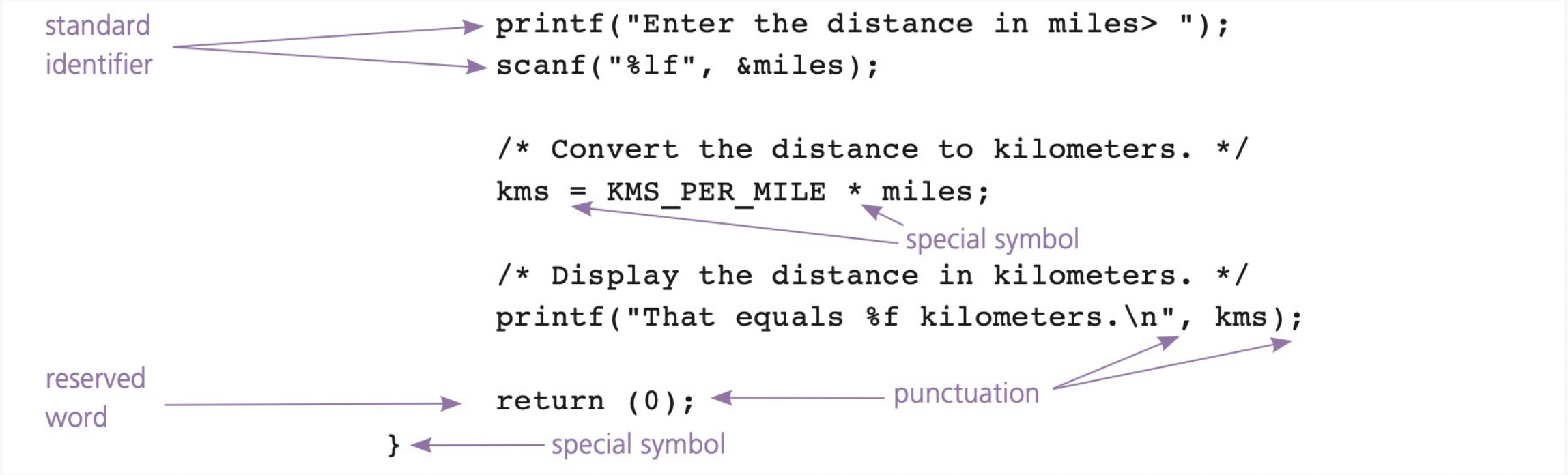
Variables – cont.

- Naming variables:
  - *Names must consist only of letters, digits, and underscores.*
  - *Names must not begin with a digit.*
  - *Names cannot be a C reserved word.*
  - *It is best to avoid using variable names that are already identified in a C standard library.*
  - *It is best to select names that clearly identify the purpose of the variable.*
  - *Names are case sensitive.*

C Language Elements



C Language Elements



Executable Statements

- C statements used to write the algorithm.
- They tell the compiler to do something.

- Some types of executable statements:
  1. *Assignment statements:*

```
variable = expression;
```

The expression can be another *variable*, a *constant*, or a *combination* of these connected by operators such as +, -, /, and *.

e.g.

```
x = y;
x = 2;
x = y + z + 2.0;
```

Executable Statements – cont.

- Some types of executable statements:
  1. *Assignment statements.*
  2. *Input/Output Operations*
     - Output:
       - In output statements, anything between double quotations is printed as is:

         `printf(format string);`

         e.g. `printf("Enter the object mass in grams > ");`

       - To print the value of variables, we need to use placeholders within our double quotations:

         `printf(format string, print list);`

         e.g. `printf("I am %d years old, and my gpa is %f\n", age, gpa);`

Executable Statements – cont.

- Some types of executable statements:
  1. *Assignment statements.*
  2. *Input/Output Operations*
     - Output:
     - Input:
       - Input statements have a list of one or more placeholders in double quotations, followed by a list of one or more variables where the placeholders' values will be stored.

         `scanf(format string, input list);`

         e.g. `scanf("%c%d", &first_initial, &age);`

         - `&` *is the C address-of operator that tells the function where to find each variable*

Placeholders

- Begin with the symbol `%`
- For printing, they indicate where within the string to display the output value.
- For reading, they indicate the type of the variable that will be read.

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

Arithmetic Expressions

Arithmetic Operator	Meaning	Examples
+	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 / 2 is 2 5.0 / 2.0 is 2.5
%	remainder of division	5 % 2 is 1 5.0 % 2.0 is ?

not allowed

Arithmetic Expressions – cont.

Notice the differences:

- int *op* int = int ← this is the cause of many errors that can be difficult to catch
- double *op* int = double
- int *op* double = double
- double *op* double = double

Arithmetic Expressions – cont.

Examples with Division:

- int /int = int → 9 / 8 = 1
- double / int = double → 9.0 / 8 = 1.125000
- int / double = double → 9 / 8.0 = 1.125000
- double / double = double → 9.0 / 8.0 = 1.125000

<u>Division by zero is undefined</u>

Arithmetic Expressions – cont.

- What is the value stored in each of the following variables?

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

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

Arithmetic Expressions – cont.

- What is the value stored in each of the following variables?

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

x = 4.5

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

Arithmetic Expressions – cont.

- What is the value stored in each of the following variables?

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

x = 4.5

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

n = 4

Arithmetic Expressions – cont.

- Printing with arithmetic expressions:

```
double k, m;
k = 9 / 6;
m = 9 / 6.0;
printf("k = %f \nm = %f", k, m);
```

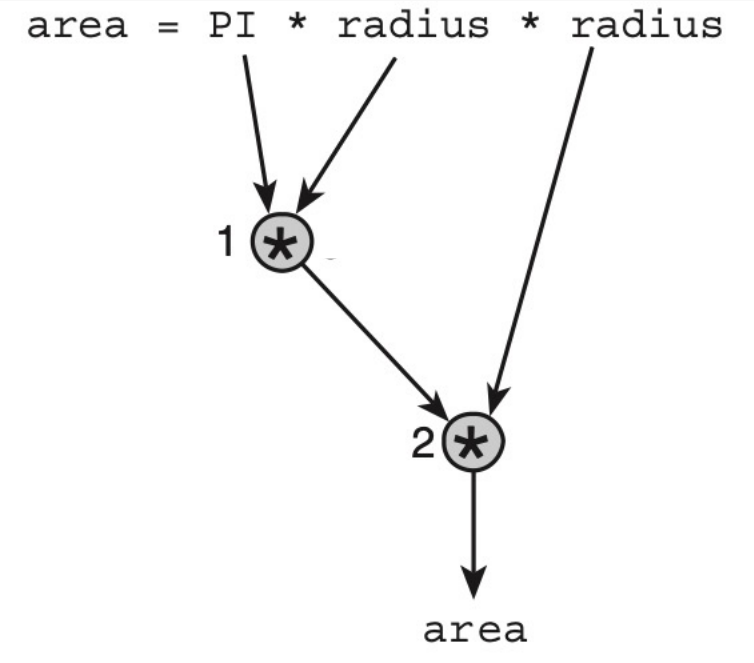
```
k = 1.000000
m = 1.500000
```

Arithmetic Expressions – cont.

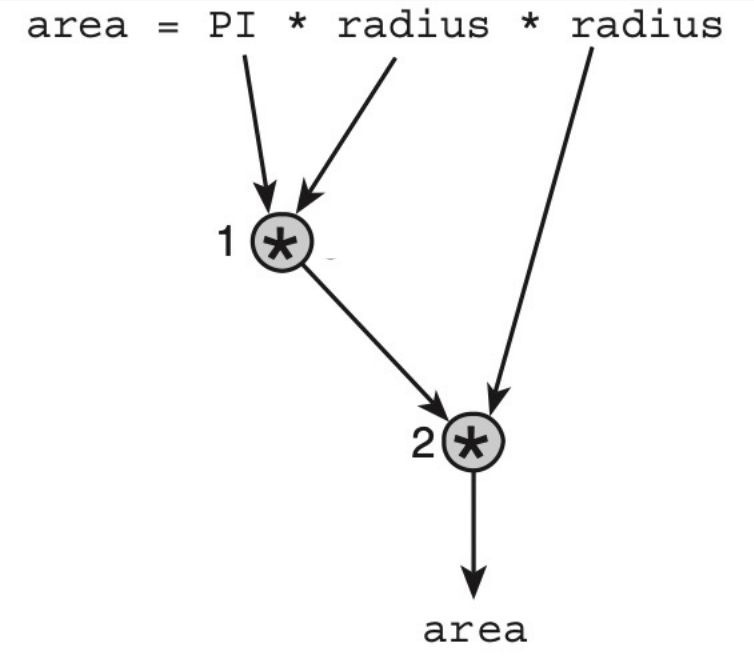
- Precedence Rules: what happens when we have multiple operators?

1. ( )
2. + - (unary = positive/negative)
3. * / %
4. + - (binary = addition/subtraction)

Arithmetic Expressions – cont.



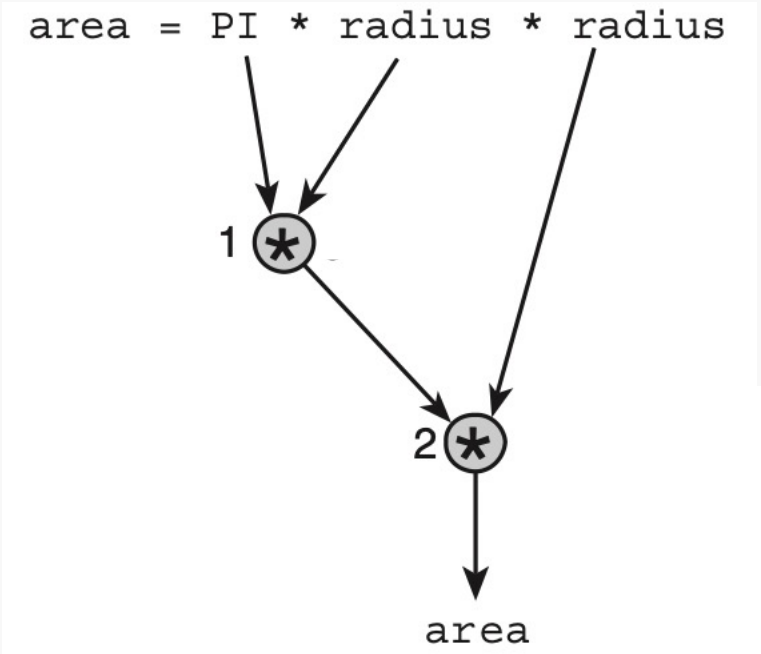
Arithmetic Expressions – cont.



If we suppose:
PI = 3.14159
radius = 2.0

How would this expression be resolved?

Arithmetic Expressions – cont.

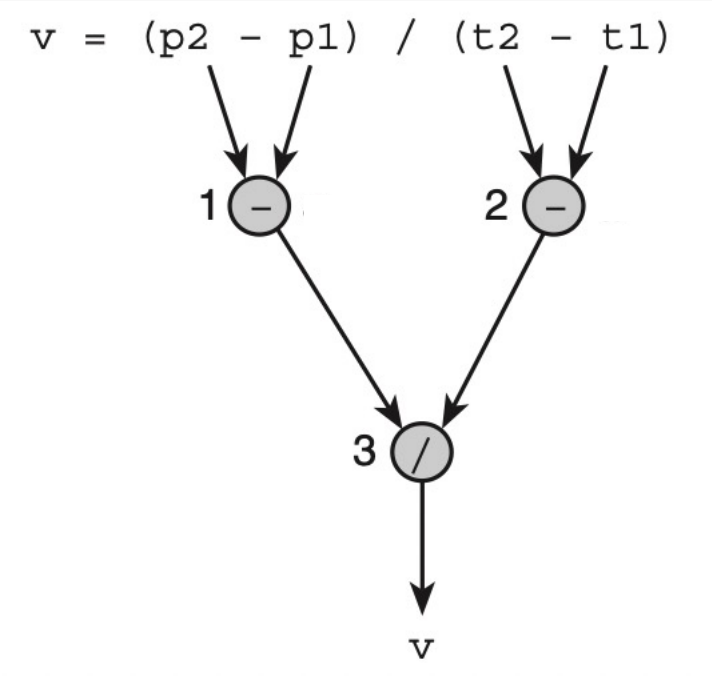


If we suppose:
PI = 3.14159
radius = 2.0

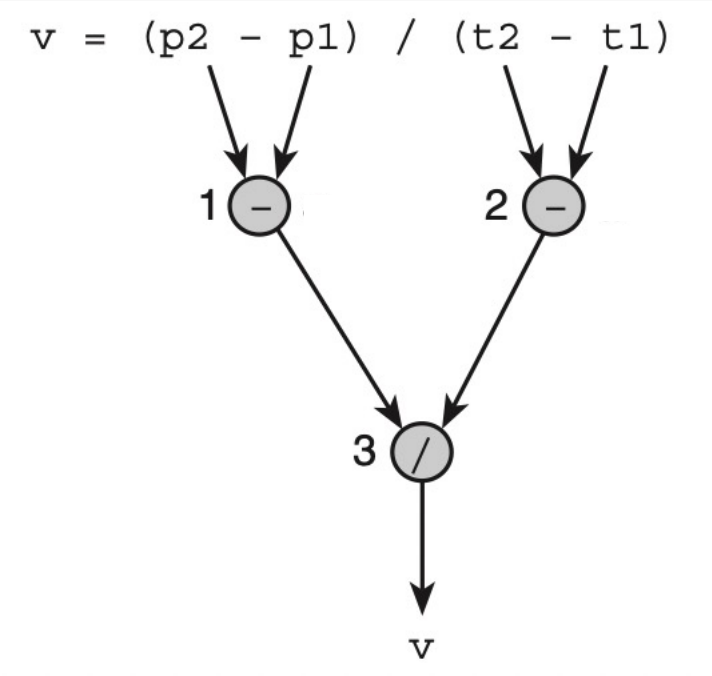
How would this expression be resolved?

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

Arithmetic Expressions – cont.



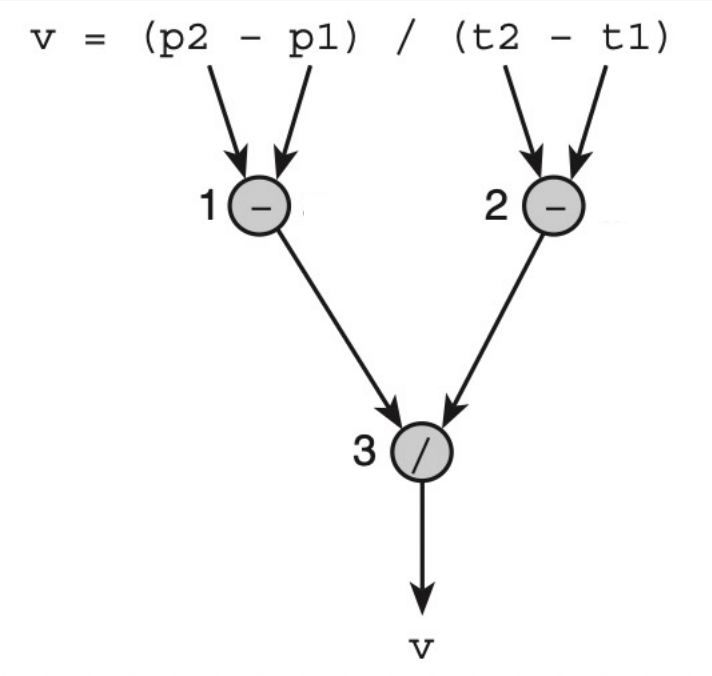
Arithmetic Expressions – cont.



If we suppose:
p1 = 4.5
p2 = 9.0
t1 = 0.0
t2 = 60.0

How would this expression be resolved?

Arithmetic Expressions – cont.

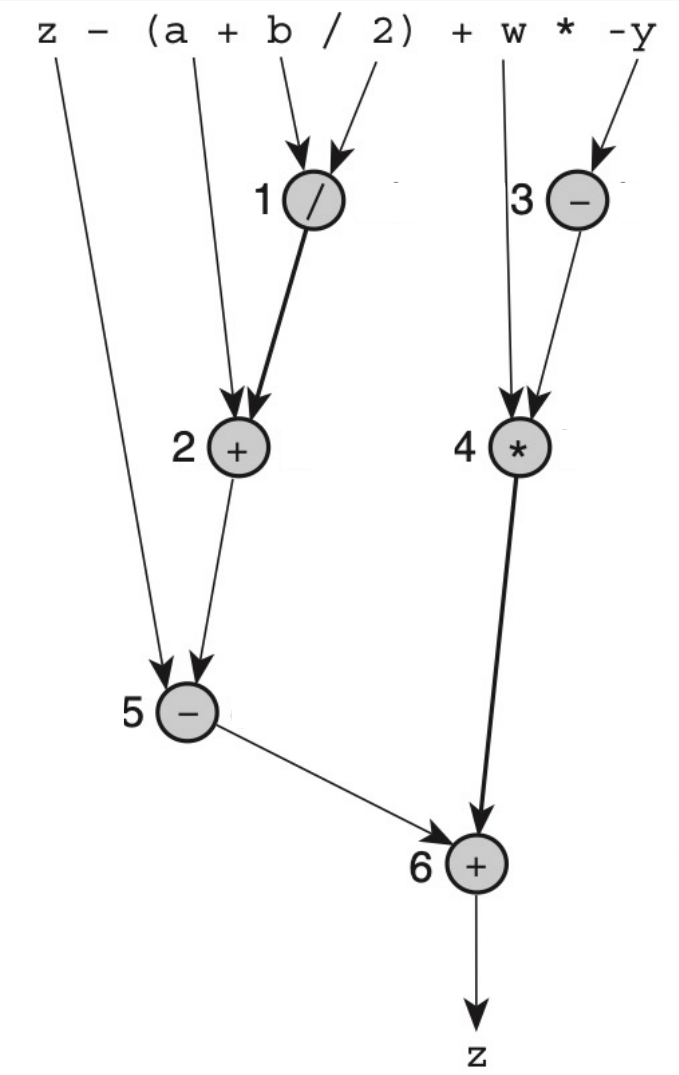


If we suppose:
p1 = 4.5
p2 = 9.0
t1 = 0.0
t2 = 60.0

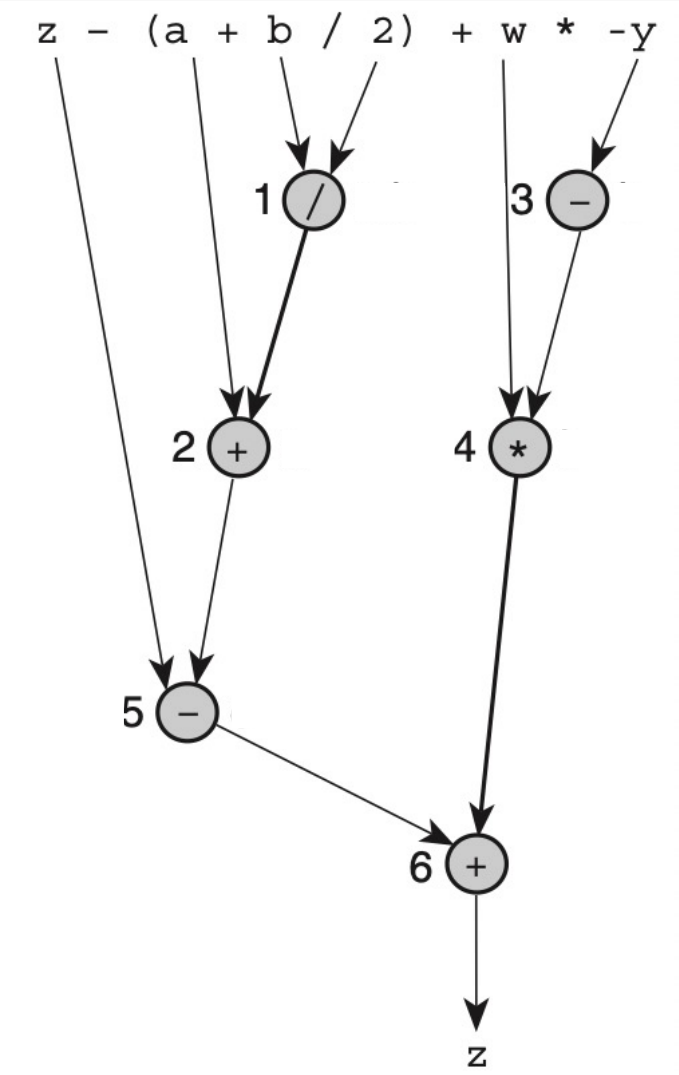
How would this expression be resolved?

```
v  =     (p2  -  p1)   /    (t2  -  t1)

         9.0     4.5         60.0     0.0

              4.5                  60.0

                      0.075
```

Arithmetic Expressions – cont.



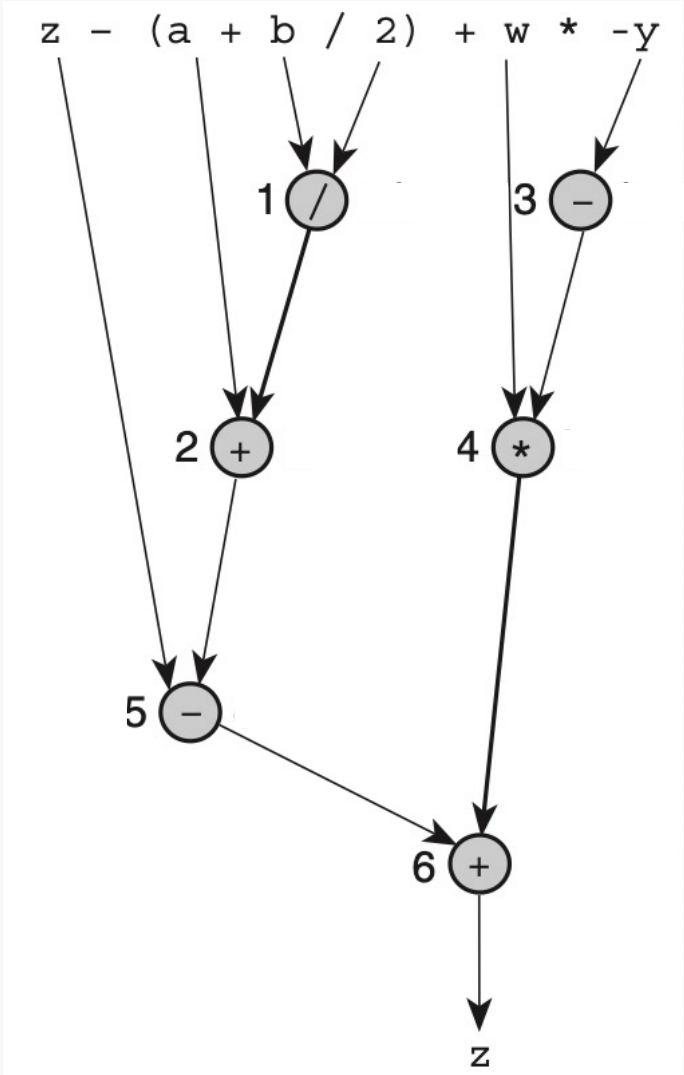
Arithmetic Expressions – cont.



If we suppose:
z = 8
a = 3
b = 9
w = 2
y = -5

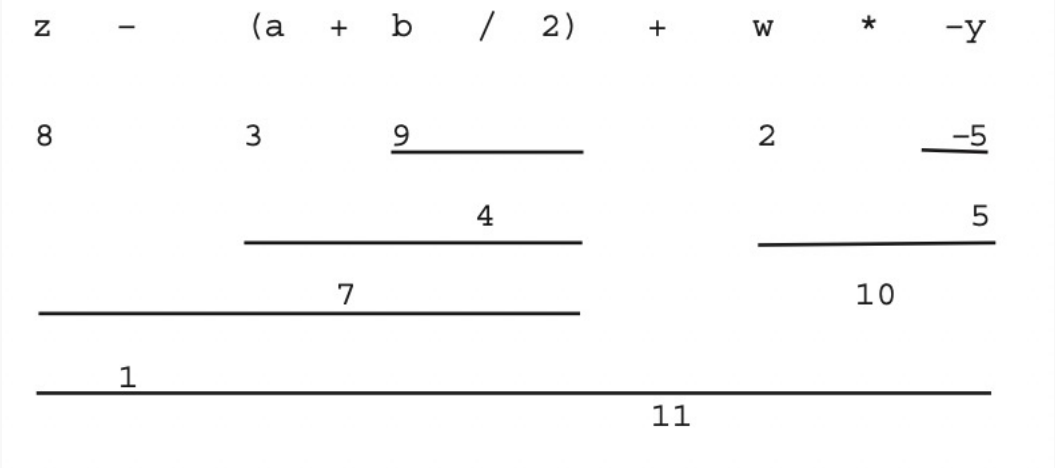
How would this expression be resolved?

Arithmetic Expressions – cont.



If we suppose:
z = 8
a = 3
b = 9
w = 2
y = -5

How would this expression be resolved?



Type Cast

- **Type casting** is converting an expression to a different type by writing the desired type in parentheses in front of the expression.

```
int x = 9, y = 2;
double z = x / y;
double w = (double) x / y;
```

Type Cast

- **Type casting** is converting an expression to a different type by writing the desired type in parentheses in front of the expression.

```
int x = 9, y = 2;
double z = x / y;                    z = 4.0
double w = (double) x / y;
```

Type Cast

- **Type casting** is converting an expression to a different type by writing the desired type in parentheses in front of the expression.

```
int x = 9, y = 2;
double z = x / y;                    z = 4.0
double w = (double) x / y;           w = 4.5
```

Type Case – cont.

- Type cast can also be used with characters.

- If you type cast a character as integer, you can find the code that your program is using to store this character.

```
char a = 'A';
int a_code  = (int)a;
printf("Code for A is %d.", a_code);
```

This will print

Code for A is 65.

Numerical Inaccuracies

- Can a computer make a mistake in calculations?
  - *Yes, in the following cases:*

1. If a number has an infinite number of binary digits, we have a ***representational error*** from rounding the number.
   - *For example, rounding PI to 3.14, or rounding 1/3 to 0.333333.*

2. If we apply an arithmetic operation to operands of vastly different magnitudes, the effect of smaller operand is lost, which is a ***cancellation error***.
   - *For example, 1000.0 + 0.0000001234 = 1000.0 in some computers*

Numerical Inaccuracies – cont.

3. If we multiply two very small numbers, the result maybe too small to represent in the memory, and thus is represented as 0, which is an ***arithmetic underflow***.

4. If we multiply two very large numbers, the result maybe too large to represent in the memory, which is an ***arithmetic overflow***. The way arithmetic overflow is resolved depends on the compiler you are using.

Formatting Numbers in Program Output

- Formatting Value of Type `int`
  - *We can specify the field width: the number of columns to use for the display of the value.*

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

Results: X21 meters = XX68 ft. 11 in.

  - *If the number was negative, the negative sign will take up one of the columns from the field width.*

```
int i = -123;
printf("i = %6d.", i);
```

i = XX-123.

Formatting Numbers in Program Output – cont.

- Formatting Value of Type double
  - *We can specify the field width and the number of decimal places.*
  - *The decimal point takes up one column from the field width.*

```
double a = 38.56, b = 201.117;
printf("a = %6.1f and b = %9.4f.", a, b);
```

a = XX38.6 and b = X201.1170.

Formatting Numbers in Program Output – cont.

- If the number used for the formatting of an integer or a double is negative, the number will be aligned to the left. This means the spaces will be added to the right of the printed variable.

```
int i = -123;
printf("i = %-6d.", i);
```

i = -123XX.

```
double a = 38.56, b = 201.117;
printf("a = %-6.1f and b = %-9.4f.", a, b);
```

a = 38.6XX and b = 201.1170X.

Escape Sequences

- Special sequences of characters that are usually used to format an output.
- Always start with a backslash '\'

Escape Sequence	Meaning
\n	New line
\t	Tab
\f	Form feed (new page)
\r	Return (go back to column 1 in the current line)
\b	Backspace

Escape Sequences – cont.

- If all escape sequences start with '\', then how do we represent a backslash character?
  - *We use two consecutive backslashes* '\\'

- The backslash can also be used to escape brackets and quotation marks.
  - *To print a bracket '(', we must use* '\(*'. Otherwise, the compiler will think it is an open bracket and raise a syntax error.*

```
printf("It\'s a cold day!");
```

This will print:

It's a cold day!

Common Programming Errors

- *“If something can go wrong, it will.” – Murphy’s Law.*

- A program rarely runs correctly the first time it is executed.
- Programming errors are often referred to as bugs.

- Debugging is the process of finding and correcting bugs in a program.

Common Programming Errors – cont.

- Syntax Errors:
  - *An error that occurs when there is a violation of the C grammar rules.*
  - *This error is detected during compilation, before the program runs.*

  - *Examples of syntax errors:*
    - Missing semicolons at the end of statements
    - Undeclared variables
    - Brackets that are not closed

Common Programming Errors – cont.

- Run-time Errors:
  - *An error that happens because of an attempt to perform an invalid operation.*
  - *This error is detected during the program execution.*
  - *A run-time error stops the execution of the program.*

  - *An example of a run-time error:*
    - Dividing by zero.

Common Programming Errors – cont.

- Logical Errors:
  - *An error that is caused by following an incorrect algorithm.*
  - *This error is not detected by the computer, but through an incorrect program output.*
  - *When you test your program thoroughly, you can better detect logical errors.*

  - *Examples of logical errors:*
    - Starting a counter from 1 instead of 0
    - Using AND instead of OR

Common Programming Errors – cont.

- Undetected Errors:
  - *Errors that are not detectable by the computer and are not caused by the algorithm.*
  - *You need to pay attention to your code to prevent these errors.*
  - *An example of an undetected error:*
    - Using the wrong placeholder to read a variable

    The compiler can handle an internal conversion between different data types, and so it will not detect this error.

    The logic of the algorithm is not affected by the way you read the variable, and so it is not a logical error.

Common Programming Errors – cont.

Find the errors in this code

```
#include <stdio.h>

int main (void) {
        int first, second;
        double temp, ans

        printf("Enter two integer> );
        scanf("%d%d", &first, &Second);
        temp = second / first;
        ans = first / temp;
        printf("The result is %.3f\n", ans);

        return(0)
}
```

Common Programming Errors – cont.

Find the errors in this code

```
#include <stdio.h>

int main (void) {
        int first, second;
        double temp, ans;                                   //syntax

        printf("Enter two integer> ");                      //syntax
        scanf("%d%d", &first, &second);                     //syntax
        temp = second / first;
        ans = first / temp;
        printf("The result is %.3f\n", ans);

        return(0);;                                         //syntax
}
```

Common Programming Errors – cont.

Find the errors in this code

```
#include <stdio.h>

int main (void) {
        int first, second;
        double temp, ans;                                   //syntax

        printf("Enter two integer> ");                      //syntax
        scanf("%d%d", &first, &second);                     //syntax
        temp = second / first;
        ans = first / temp;
        printf("The result is %.3f\n", ans);

        return(0);                                          //syntax
}
```

Suppose the user enters:
14 3

Common Programming Errors – cont.

Find the errors in this code

```
#include <stdio.h>

int main (void) {
        int first, second;
        double temp, ans;                                   //syntax

        printf("Enter two integer> ");                      //syntax
        scanf("%d%d", &first, &second);                     //syntax
        temp = second / first;
        ans = first / temp;                                 //run-time error
        printf("The result is %.3f\n", ans);

        return(0);                                          //syntax
}
```

Suppose the user enters:
14 3

Text Files

- File are declared using pointers:

```
FILE *inp;
FILE *outp;
```

- Files are opened using functions:

```
inp = fopen("input_data.txt", "r");
outp = fopen("output.txt", "w");
```

- File are closed using functions:

```
fclose(inp);
fclose(outp);
```

Text Files – cont.

- Reading from a file:

```
fscanf(inp,"%lf",&miles);
```

- Writing to a file:

```
fprintf(ftp_out, "The distance in miles is %.2f. \n",
                 miles);
```

Text Files Example

```
/* Inputs each number from an input file and writes it
 * rounded to 2 decimal places on a line of an output file.
 */
#include <stdio.h>

int
main(void)
{
     FILE *inp;          /* pointer to input file */
     FILE *outp;         /* pointer to ouput file */
     double item;
     int input_status;  /* status value returned by fscanf */

     /* Prepare files for input or output */
     inp = fopen("indata.txt", "r");
     outp = fopen("outdata.txt", "w");
```

Text Files Example – cont.

```
    /* Input each item, format it, and write it */
    input_status = fscanf(inp, "%lf", &item);
    while (input_status == 1) {
        fprintf(outp, "%.2f\n", item);
        input_status = fscanf(inp, "%lf", &item);
    }

    /* Close the files */
    fclose(inp);
    fclose(outp);

    return (0);
}
```

Text Files Example – cont.

```
    /* Input each item, format it, and write it */
    input_status = fscanf(inp, "%lf", &item);
    while (input_status == 1) {
        fprintf(outp, "%.2f\n", item);
        input_status = fscanf(inp, "%lf", &item);
    }

    /* Close the files */
    fclose(inp);
    fclose(outp);

    return (0);
}
```

File indata.txt

```
344 55 6.3556 9.4
43.123 47.596
```

Text Files Example – cont.

```
    /* Input each item, format it, and write it */
    input_status = fscanf(inp, "%lf", &item);
    while (input_status == 1) {
        fprintf(outp, "%.2f\n", item);
        input_status = fscanf(inp, "%lf", &item);
    }

    /* Close the files */
    fclose(inp);
    fclose(outp);

    return (0);
}
```

File indata.txt

```
344 55 6.3556 9.4
43.123 47.596
```

File outdata.txt

```
344.00
55.00
6.36
9.40
43.12
47.60
```