UNIVERSITI
TEKNOLOGI
MALAYSIA
www.utm.my

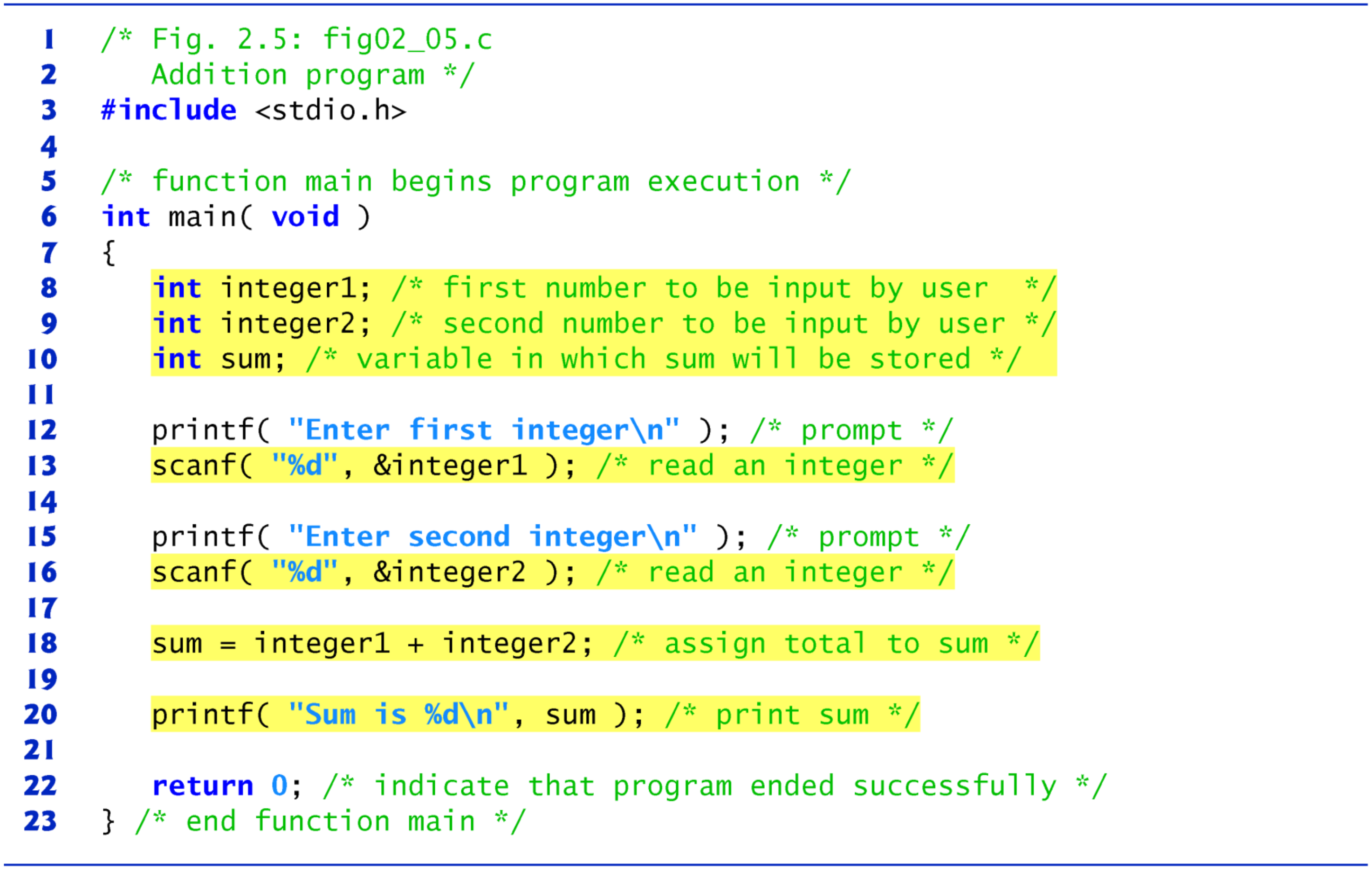
Week 5

Arithmetic Expressions

3.1 The `scanf`

The `scanf`

- **`scanf`**, requires **`stdio.h`** file
- Used to read input from keyboard
- The **`scanf`** has two arguments
- The first argument, the format control string, indicates the type of data that should be input by the user.
- The second argument begins with an ampersand (&)—called the address operator in C—followed by the variable name.



```c
/* Fig. 2.5: fig02_05.c
   Addition program */
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
   int integer1; /* first number to be input by user  */
   int integer2; /* second number to be input by user */
   int sum; /* variable in which sum will be stored */

   printf( "Enter first integer\n" ); /* prompt */
   scanf( "%d", &integer1 ); /* read an integer */

   printf( "Enter second integer\n" ); /* prompt */
   scanf( "%d", &integer2 ); /* read an integer */

   sum = integer1 + integer2; /* assign total to sum */

   printf( "Sum is %d\n", sum ); /* print sum */

   return 0; /* indicate that program ended successfully */
} /* end function main */
```

Fig. 2.5 | Addition program. (Part 1 of 2.)

The scanf

- This scanf has two arguments, "%d" and &integer1.
- The %d conversion specifier indicates that the data should be an integer (the letter d stands for “decimal integer”).
- The ampersand, when combined with the variable name, tells scanf the location (or address) in memory at which the variable integer1 is stored

Displaying a Prompt

- A prompt is a message that instructs the user to enter data.
- You should always use printf to display a prompt before each scanf statement.

```
printf( "Enter first integer\n" ); /* prompt */
scanf( "%d", &integer1 ); /* read an integer */
```

The `scanf`

- Can be used to input more than one value:

```
scanf ("%d %d", &integer1, &integer2);
```

- Order is important: first value entered goes to first variable, etc.

Example

```
#include "stdio.h”

int main()
{
      int minx, x, y, z;

      printf("Enter four ints: ");
      scanf( ”%d %d %d %d", &minx, &x, &y, &z);

      printf("You wrote: %d %d %d %d", minx, x, y, z);
      return 0;
}
```

Format specifiers

- **`%c`** for single characters
  - `scanf(" %c", &some_character);`

always put a space between " and % when reading characters

- **`%d`** for integers
  - `scanf ("%d", &some_integer);`
- **`%f`** for float
  - `scanf ("%f", &some_float);`
- **`%lf`** for double
  - `scanf ("%lf", &some_double);`

Reading Strings with `scanf`

- Can be used to read in a string
- Must first declare an array to hold characters in string:

```
char str[50];
```

- `str` is name of array, `50` is the number of characters that can be stored (the size of the array), including the NULL character at the end
- Can be used with `scanf` to assign a value:

```
scanf("%s", str);
```

Example

```
#include <stdio.h>
#include <conio.h>

int main(){
    // static declaration
    char str[50] = {0};
    // shorthand way to initialize all elements to 0
    int n;

    printf("Enter your First name and Age: ");
    scanf("%s%d", str, &n); //
    printf("Your name is %s and you are %d old\n", str, n);
    getch();
    return 0;
}
```

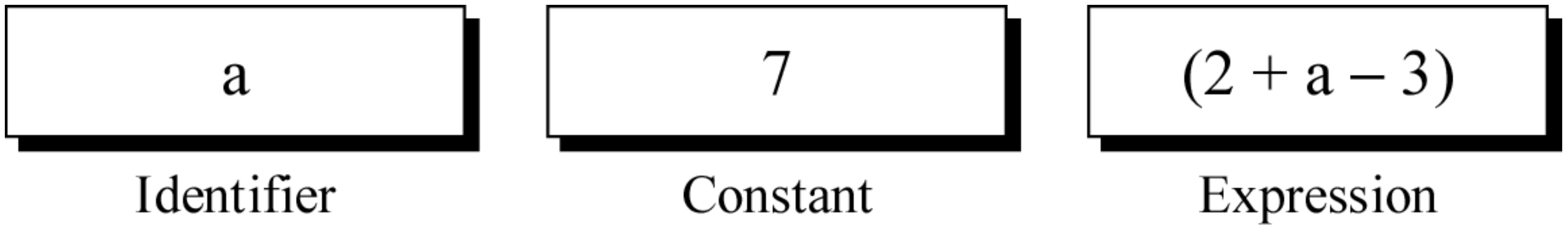
Exercise Week5_1

- Write a C program that asks the user to enter an integer, a floating point number, and a character, and write the results back out. All output must be in the format shown in the sample output.

Sample input: **Enter a single character : R** **Enter an array of characters: Hello** **Enter an integer: 7** **Enter a decimal number : 2.25**	**Sample output:** **You entered r** **You entered Hello** **You entered 7** **You entered 2.25**

3.2 Mathematical Expressions

Primary expression



a

Identifier

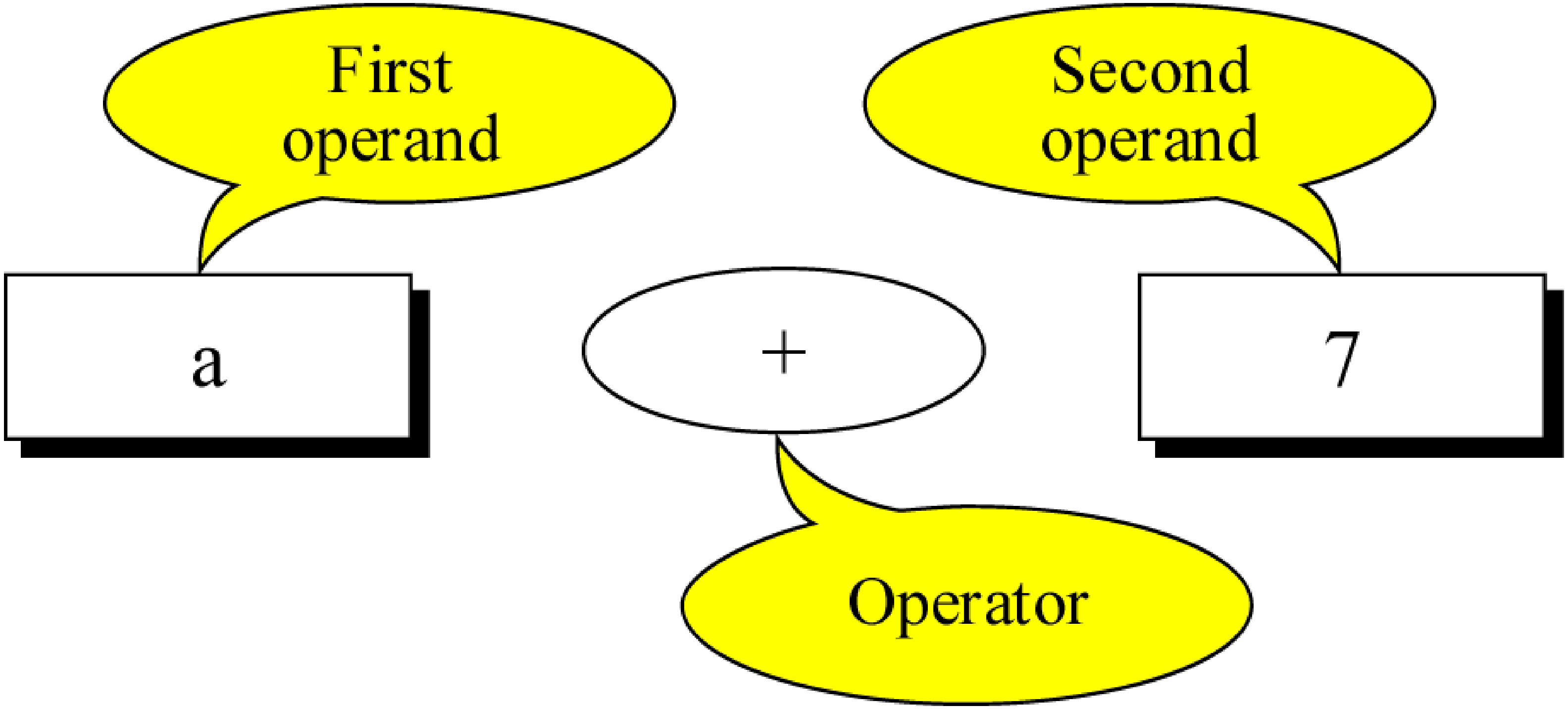
7

Constant

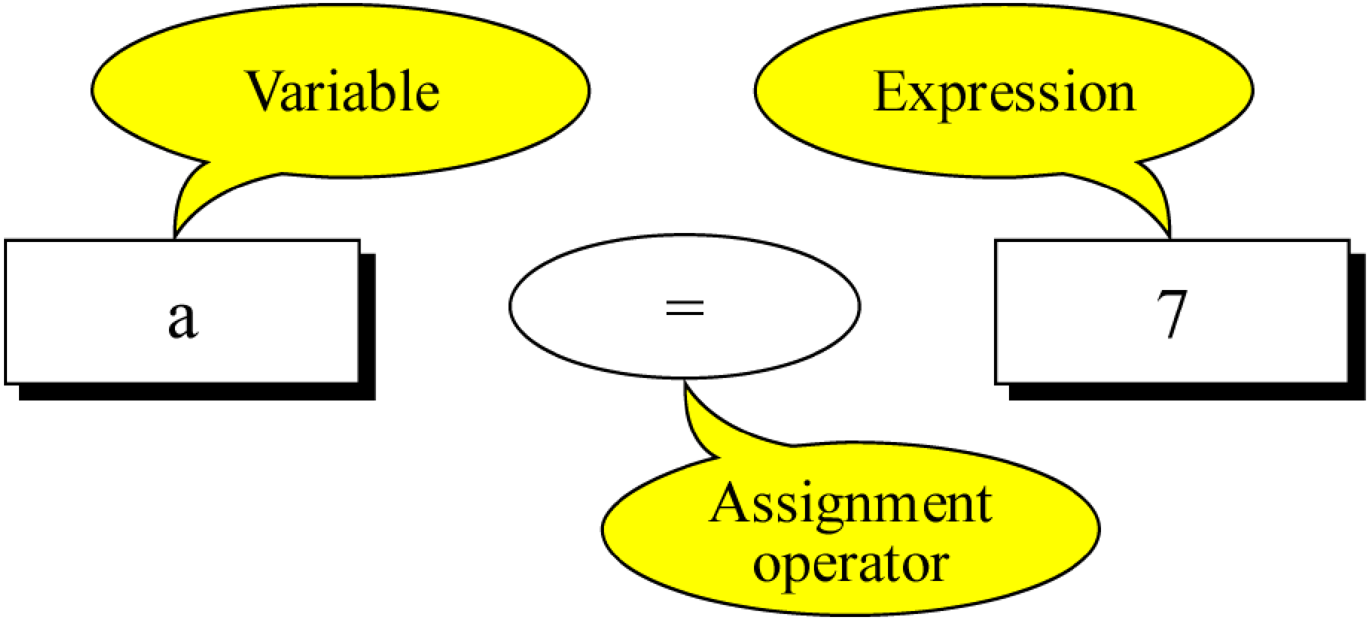
(2 + a – 3)

Expression

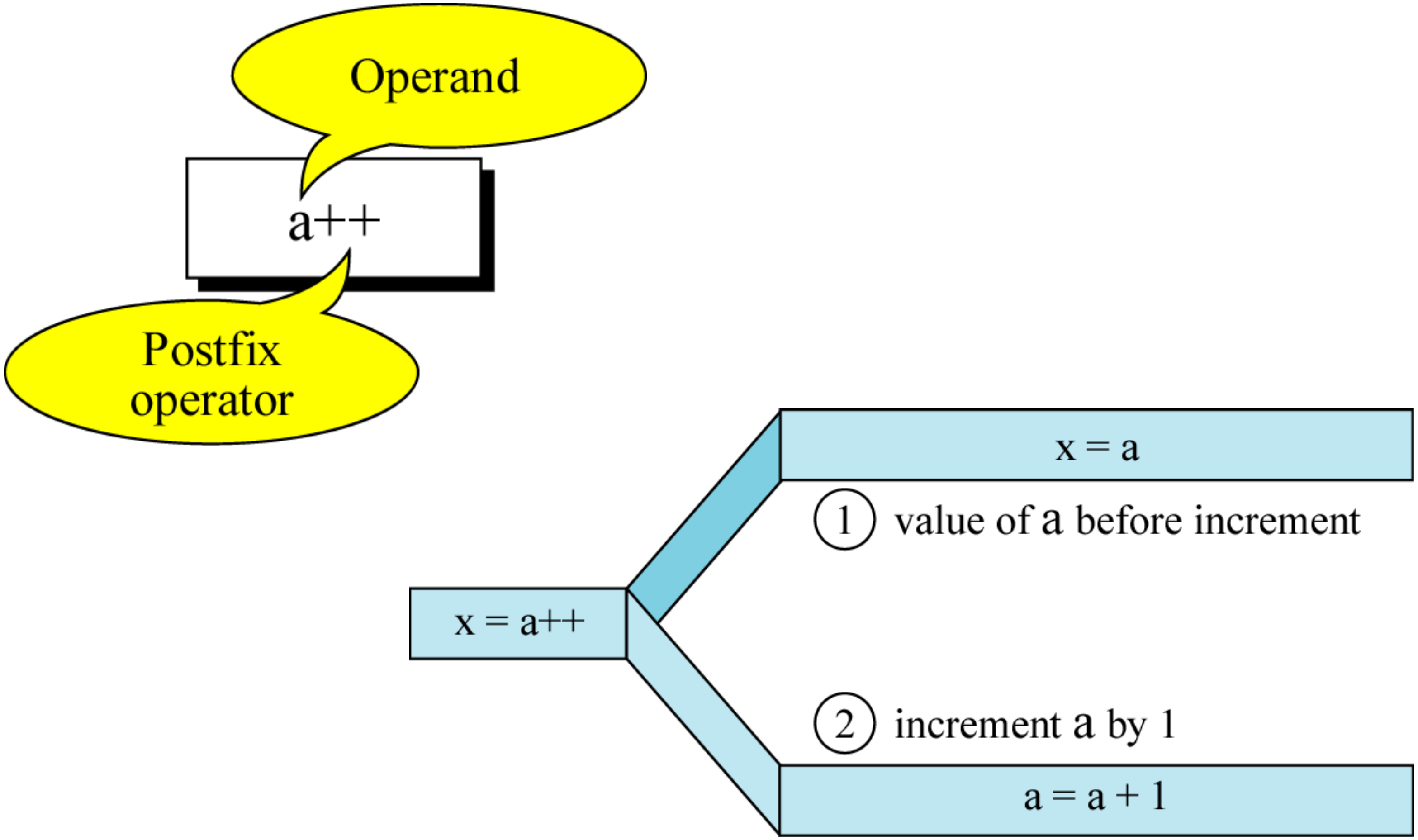
Binary expression



Assignment expression



Postfix expression



Operand

a++

Postfix operator

x = a++

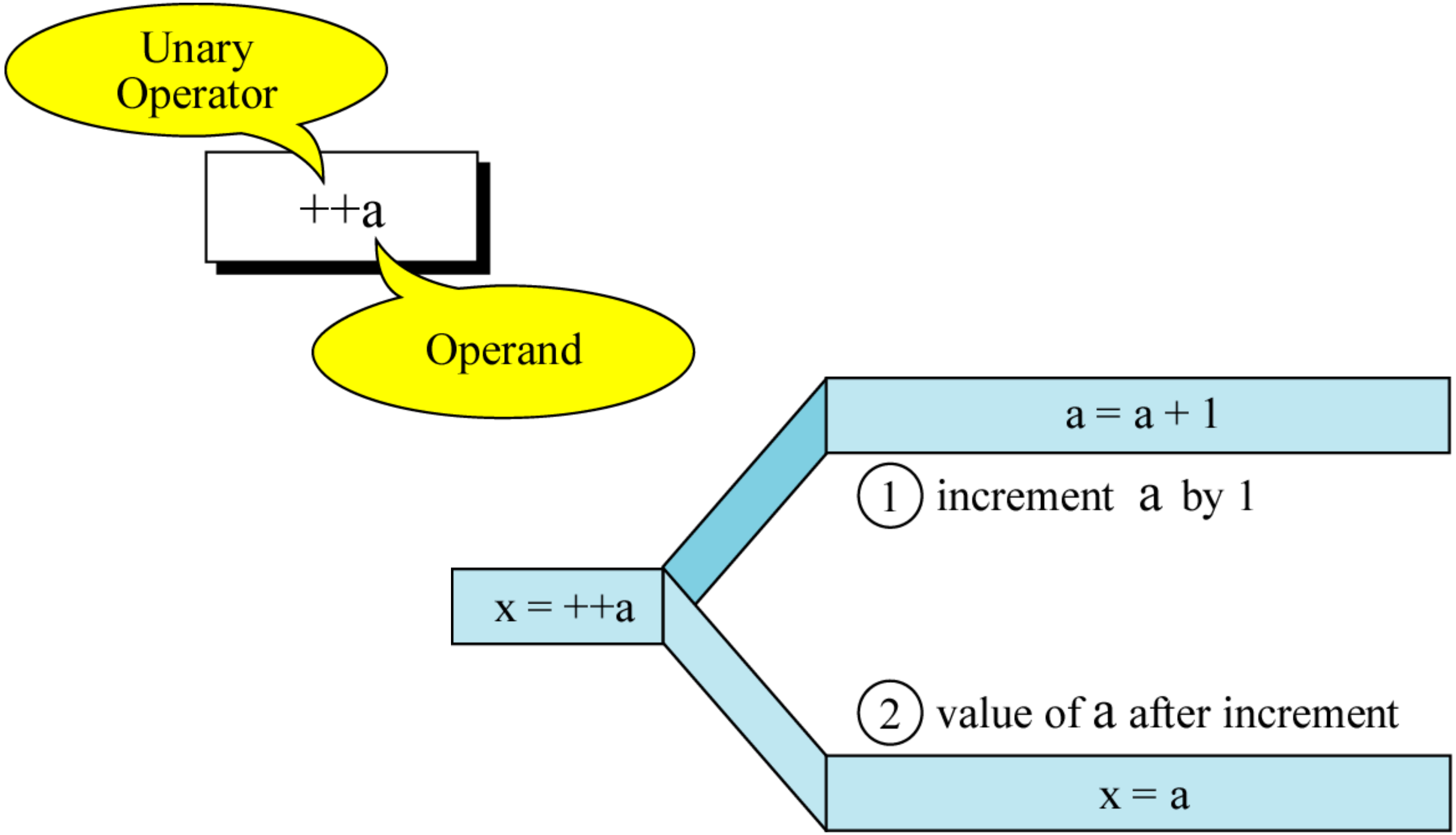
x = a

1 value of a before increment

2 increment a by 1

a = a + 1

Prefix expression



Mathematical Expressions

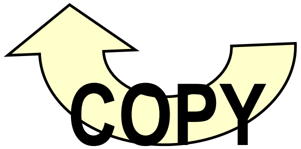
- Can create complex expressions using multiple mathematical operators
- An expression can be a literal, a variable, or a mathematical combination of constants and variables
- Can be used in assignment, `printf`, other statements:

```
area = 2 * PI * radius;
printf ("border is: %d",  2*(l+w));
```

Assignment operator =

- Binary operator used to assign a value to a variable.
- Its left operand is the destination variable
- Its right operand is an expression.

```
int var;
var = 10;
```



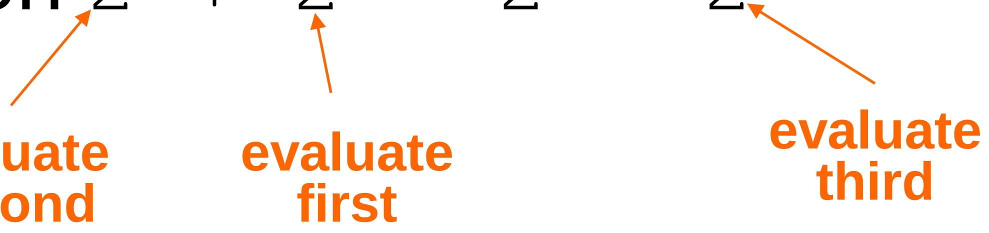
COPY

Order of Operations

In an expression with more than one operator, evaluate in this order:

- ()
- – (unary negation), in order, left to right
- `*  /  %`, in order, left to right
- `+  –`, in order, left to right

In the expression `2 + 2 * 2 – 2`



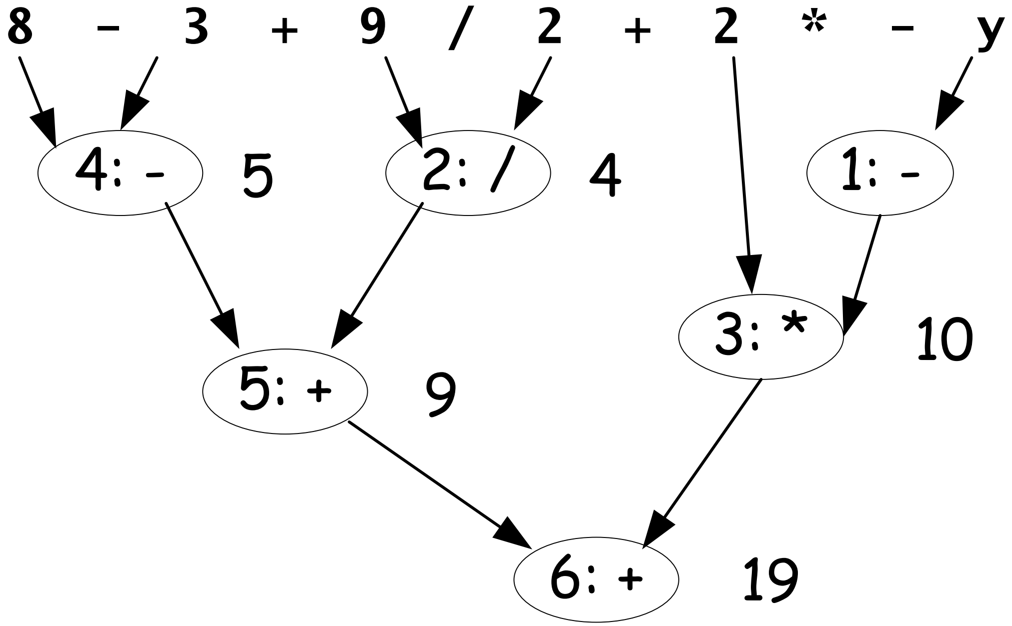
evaluate second

evaluate first

evaluate third

Example

```
int z, y=-5;
z= 8 - 3 + 9 / 2 + 2 * - y;
z= 8 - (3 + 9 / 2) + 2 * - y;// try this
```



Order of Operations

Show prove for the following expression

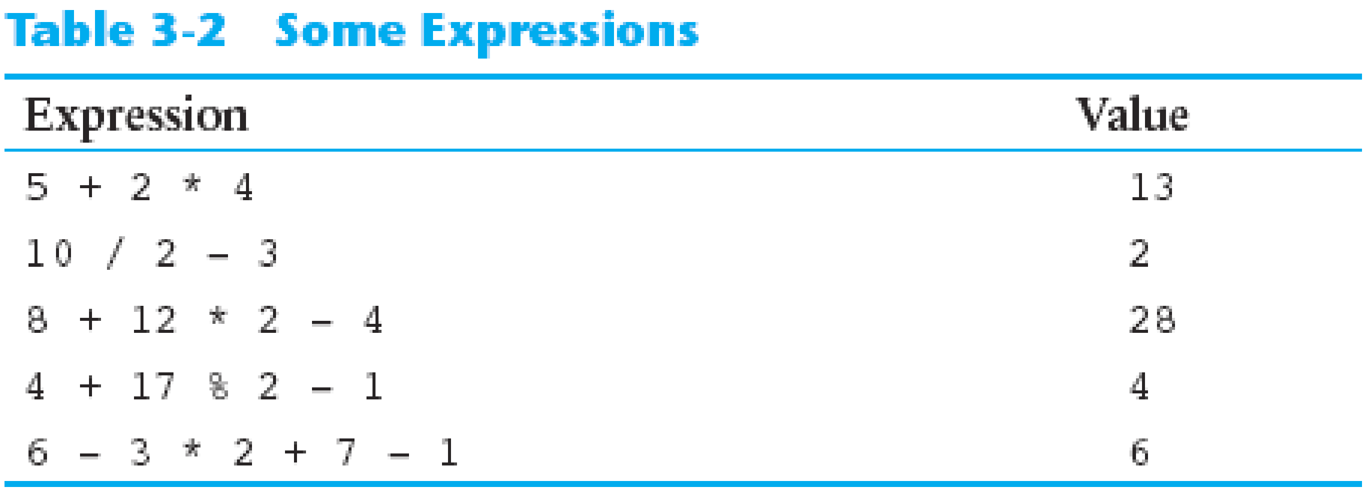


Table 3-2 Some Expressions

Expression	Value
`5 + 2 * 4`	13
`10 / 2 - 3`	2
`8 + 12 * 2 - 4`	28
`4 + 17 % 2 - 1`	4
`6 - 3 * 2 + 7 - 1`	6

Associativity of Operators

- – (unary negation) associates right to left
- *, /, %, +, – associate left to right
- parentheses ( ) can be used to override the order of operations:

```
 2 + 2  *  2 - 2  = 4
(2 + 2) *  2 - 2  = 6
 2 + 2  * (2 - 2) = 2
(2 + 2) * (2 - 2) = 0
```

Grouping with Parentheses

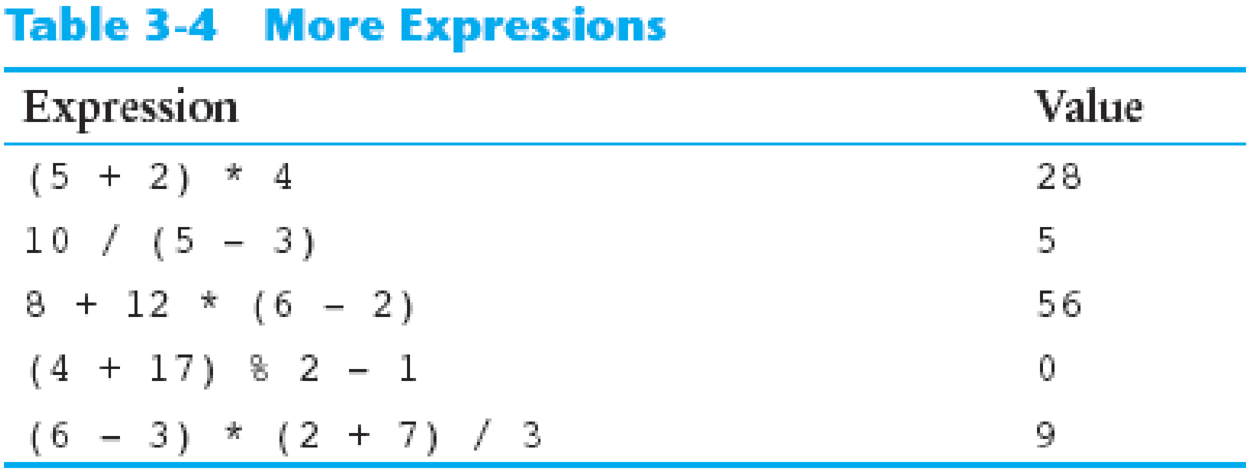


Table 3-4 More Expressions

Expression	Value
`(5 + 2) * 4`	28
`10 / (5 - 3)`	5
`8 + 12 * (6 - 2)`	56
`(4 + 17) % 2 - 1`	0
`(6 - 3) * (2 + 7) / 3`	9

Algebraic Expressions

- Multiplication requires an operator:

  $Area = lw$ is written as `Area = l * w;`

- There is no exponentiation operator:

  $Area = s^2$ is written as `Area = pow(s, 2);`

- Parentheses may be needed to maintain order of operations:

  $$m = \frac{y_2 - y_1}{x_2 - x_1}$$ is written as

  `m = (y2-y1) /(x2-x1);`

Algebraic and C Multiplication Expressions

Algebraic Expression	Operation	C Equivalent
6B	6 times B	6 * B
(3)(12)	3 times 12	3 * 12
4xy	4 times x times y	4 * x * y

Exercise Week5_2

- Write the formula in C statement.

$$b^2 - 4ac$$

$$\frac{a + b}{c + d}$$

$$\frac{1}{1 + x^2}$$

3.3 When You Mix Apples and Oranges: *Type Conversion*

When You Mix Apples and Oranges: *Type Conversion*

- Operations are performed between operands of the same type.
- If not of the same type, C will convert one to be the type of the other
- The type of the result depends on the types of the operands.
- If the types of the operands differ (e.g. an integer added to a floating point number), one is "promoted" to other.
  - The "smaller" type is promoted to the "larger" one.

    `char → int → float → double`
- This can impact the results of calculations.

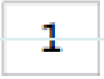
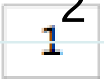
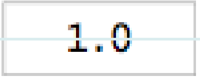
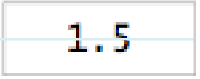
`int` and `double`

If all operands are integer, the output will be integer, otherwise, the output will be double

Example

```
main()
{
    int i1=3, i2=2, output1, output2;
    double d=2.0, output3, output4;

    output1 = i1/i2; /* 3/2   */
    output2 = i1/d; /* 3/2.0 */
    output3 = i1/i2; /* 3/2   */
    output4 = i1/d; /* 3/2.0 */
}
```

output1	output2	output3	output4
 1	 1	 1.0	 1.5

3.4 Type Casting

Type Casting

- Used for manual data type conversion
- Useful for floating point division using int
- Format

```
(data type)variable
```

Example 1

```
double a=3.0, b=2.0, output;
output = a % b; /*syntax error!!!*/
```

Solution:

```
output = (int)a % (int)b; /*free from error!*/
```

Example 2

```
main()
{
    int total_marks = 456, num_studs = 5;
    double ave_marks1, ave_marks2;

    ave_marks1 = total_marks/num_studs; ave_marks2 =
    (double) total_marks / num_studs;
}
```

ave_marks1	ave_marks2
91.0	91.2

3.6 Named Constants

Named Constants

- Named constant (constant variable): variable whose content cannot be changed during program execution
- Used for representing constant values with descriptive names:

```
const double TAX_RATE = 0.0675;
const int NUM_STATES = 50;
```

- Often named in uppercase letters

Named Constants - example

What will be the output?

```
void main()
{
        const int a=5;
        a++;

        printf(“%d”, a);
}
```

const vs. #define

- #define

```
#define NUM_STATES 50
```

  - Note no ; at end
- Interpreted by pre-processor rather than compiler
- Does not occupy memory location like const

Exercise Week5_3

- Write a program that will convert Malaysian Ringgit (RM) amounts to Japanese Yen and to Euros. The conversion factors to use are:

  1 RM = 0.21734 Euros

  1 RM = 36.0665 Yen

- Solve the problems using constant values.

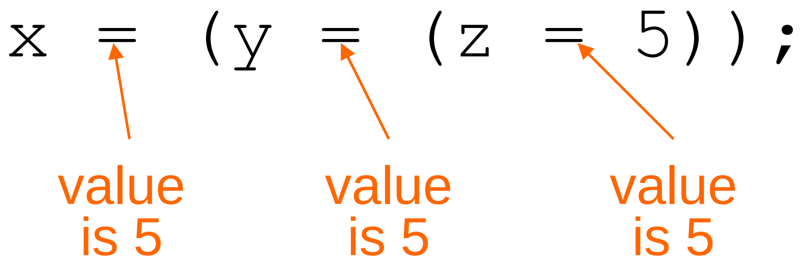
3.7 Multiple Assignment and Combined Assignment

Multiple Assignment and Combined Assignment

- The = can be used to assign a value to multiple variables:

```
x = y = z = 5;
```

- Value of = is the value that is assigned
- Associates right to left:



```
x = (y = (z = 5));
```

value is 5

value is 5

value is 5

Combined Assignment

- Look at the following statement:

```
sum = sum + 1;
```

This adds 1 to the variable **`sum`**.

Other Similar Statements

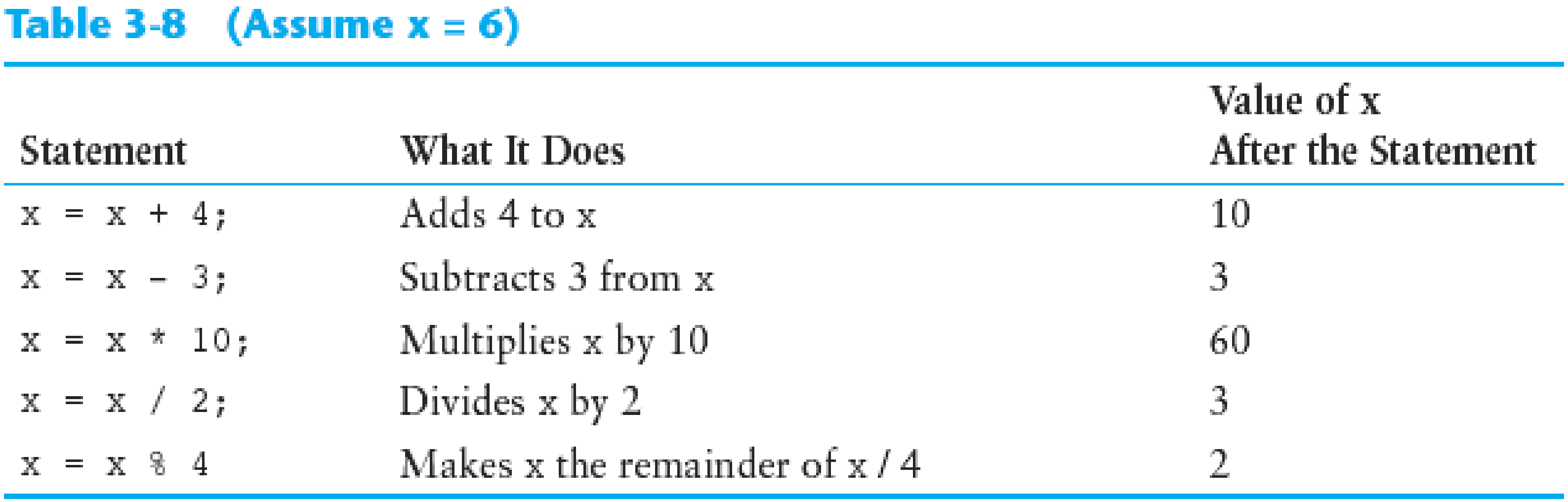


Table 3-8 (Assume x = 6)

		Value of x
Statement	What It Does	After the Statement
x = x + 4;	Adds 4 to x	10
x = x - 3;	Subtracts 3 from x	3
x = x * 10;	Multiplies x by 10	60
x = x / 2;	Divides x by 2	3
x = x % 4	Makes x the remainder of x / 4	2

Combined Assignment

- The combined assignment operators provide a shorthand for these types of statements.
- The statement

```
sum = sum + 1;
```

is equivalent to

```
sum += 1;
```

Combined Assignment Operators

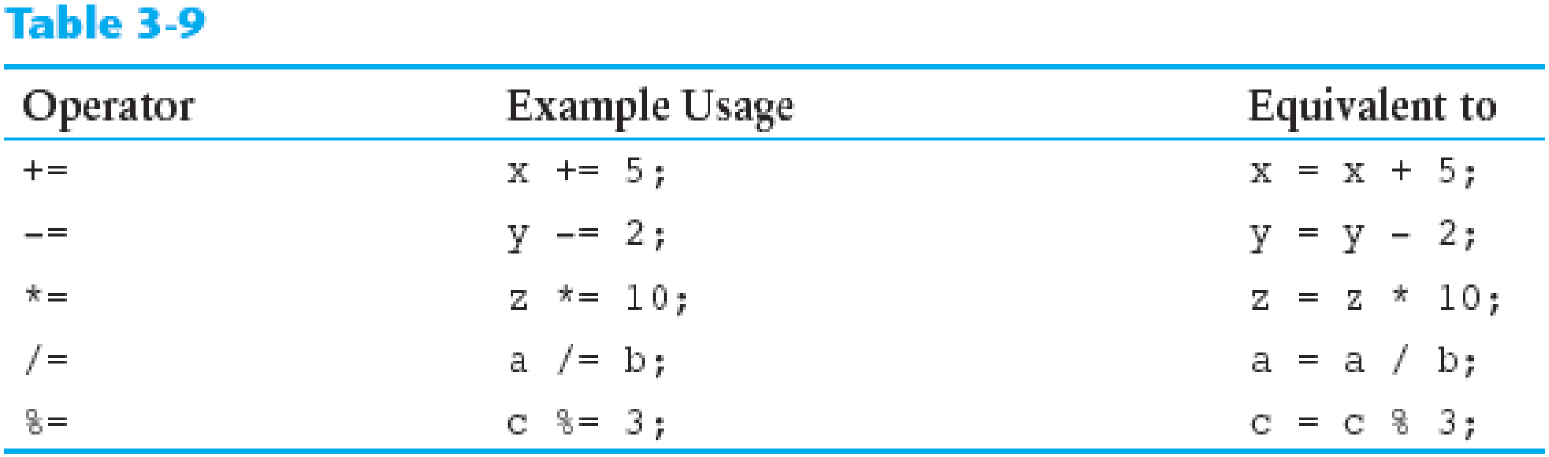


Table 3-9

Operator	Example Usage	Equivalent to
`+=`	`x += 5;`	`x = x + 5;`
`-=`	`y -= 2;`	`y = y - 2;`
`*=`	`z *= 10;`	`z = z * 10;`
`/=`	`a /= b;`	`a = a / b;`
`%=`	`c %= 3;`	`c = c % 3;`

Increment, decrement operators: ++, --

- Increment, decrement operators: ++, --
  - Instead of **`a = a + 1`** you can write **`a++`** or **`++a`**
  - Instead of **`a = a - 1`** you can write **`a--`** or **`--a`**
- What is the difference?

post-increment

```
  num = 10;
ans = num++;
```

First assign num to ans,
then increment num.
In the end,
num is 11
ans is 10

pre-increment

```
  num = 10;
ans = ++num;
```

First increment num,
then assign num to ans.
In the end,
num is 11
ans is 11

Mathematic Library Functions

- Available in C
- Can be called upon during pre-processing #include

```
#include <math.h>
#include <stdlib.h>
```

Some functions from Maths Library

Function	Library Func.	Purpose and example	Argument	Output
abs(x)	stdlib.h	x abs(-5) output 5	int	int
exp (x)	math.h	e^x exp(1.0) output 2.71828	double	double
log(x)	math.h	$\log_e(x)$ log((2.71828) output 1.0	double	double
pow(x, y)	math.h	X^y pow(0.16, 0.5) output 0.4	double, double	double
sqrt(x)	math.h	√x and x>=0.0 sqrt(2.25) output 1.5	double	double

Thank You

Q & A