Structured Programming

Dr. Mohamed Khedr
Lecture 4
http://webmail.aast.edu/~khedr

Scientific Notation for floats

2.7E4 **means** 2.7×10^{4} =

2.7000 =

27000.0

2.7E-4 **means** 2.7×10^{-4} =

0002.7 =

0.00027

Output Formatting

Integer formatting

- **Very simple: Add a number between the % and the d in the placeholder to specify the “field length”.**
- **Numbers will appear right-justified with preceding blanks if needed.**

Integer Formatting

Example

```
int len = 234 ;
printf(“ Length is %5d “, len);
```

Output is:

```
Length is  △△234
```

Note: The △ stands for a blank

Integer Formatting

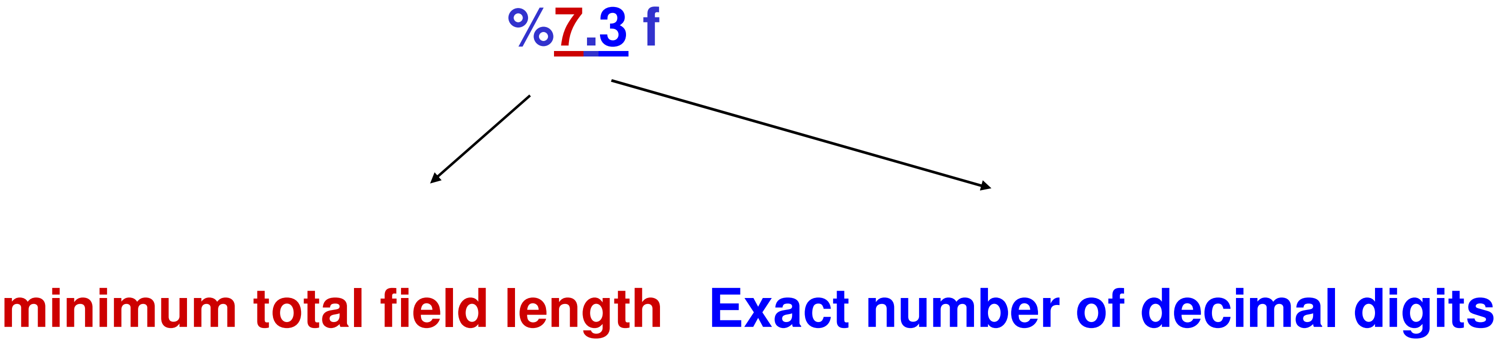
Displaying X using different %d placeholder

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

Output Formatting

Double formatting

We must indicate both the field width and the EXACT number of decimal places:



%7.3 f

minimum total field length **Exact number of decimal digits**

Note: The decimal part will be rounded

The whole part may be padded with blanks

REMEMBER: The value of the number does not change, only its appearance

Double Formatting

Displaying X using different %6.2f placeholder

Value of x	Displayed Output	Value of x	Displayed output
-99.42	-99.42	-25.554	-25.55
0.123	△△0.12	99.999	100.00
-9.536	△-9.54	999.4	999.40

Programming Examples

Example-1

- Write a program to ask the user for the width and length of a piece of land and then tell him how many orange trees he can grow on it. Given that each orange tree requires 4 m^2.

Programming Examples

Example-1

```
#include <stdio.h>
# define one_tree_space  4
int main(void)
{
    int   length,width, area, no_of_tree;
    printf(“Enter length of the land> ”);
   scanf(“%d”, &length);
   printf(“Enter width of the land> ”);
   scanf(“%d”, &width);
    area = length * width;
     no_of_tree = area / one_tree_space;
      printf(“The available number of trees is %d  tress\n”,
     no_of_tree);
      return(0);

}
```

Programming Examples

Example-2

- **Write a program to ask the user for the radius of a circle, and then display its area and circumference, displayed to 3 decimal digits.**

Programming Examples

Example-2

```
#include <stdio.h>
# define PI  3.141593
int main(void)
{
    double  radius, area, circumference;
    printf(“Enter radius of the circle> ”);
   scanf(“%lf”, &radius);
    area =  PI * radius * radius;
    circumference = 2 * PI * radius;
    printf(“The area of the circle =  %.3f\n”, area);
 printf(“The circumference of the circle =  %.3f\n”,circumference);
     return(0);

}
```

Arithmetic Expression Assignment operator syntax

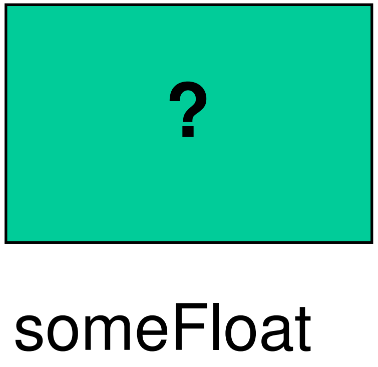
Variable = Expression

1) **first, Expression on right is evaluated**
2) **then the resulting value is stored in the memory location of Variable on left**

NOTE: An automatic type conversion occurs after evaluation but before the value is stored if the types differ for Expression and Variable

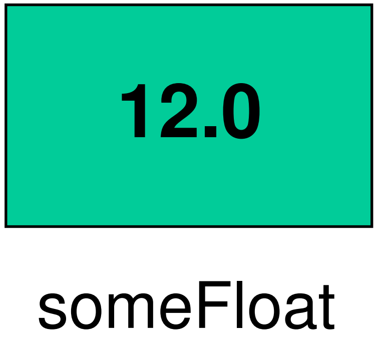
Arithmetic Expression What is stored?

```
float  someFloat;
```



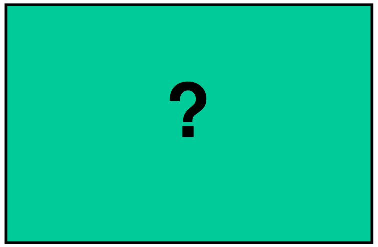
```
someFloat = 12;
```

// causes implicit type conversion



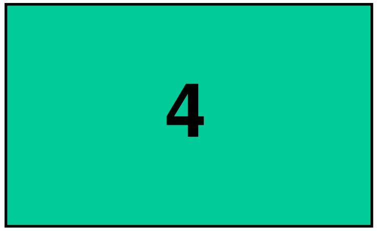
Arithmetic Expression What is stored?

```
int  someInt;



someInt = 4.8;
```



someInt

// causes implicit type conversion



someInt

Explicit Type Conversion

(int)4.8	has value	4
(float)5	has value	5.0
(float)(7/4)	has value	1.0
(float)7 / (float)4	has value	1.75

Using Casts to Prevent Integer Division (Example)

```
#include <stdio.h>
int main(void)
{
   int    total_score, num_students;
  double average;
  printf(“Enter sum of students’ scores> ”);
  scanf(“%d”, &total_score);
  printf(“Enter sum of students> ”);
  scanf(“%d”, &num_students);

   average = (double) total_score / (double) num_students;
    printf(“Average score is %.2f\n”, average);

    return(0);

}
```

Math in C

Math library

- The C math library provides a lot of useful predefined math functions
- Before you use them, remember to include the math library in your code:

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

- function sqrt:

```
y = sqrt ( x );
```

Math in C

Examples of Math functions

sin(x) cos(x) tan(x)

sqrt(x) **pow(x,y)**

log(x) log10(x) exp(x)

fabs(x) floor(x) ceil(x)

Math in C

complex math example-1

Write a program to get the roots of a quadratic equation, given the 3 coefficients a, b, and c,

$$a\,x^2 + b\,x + c = 0$$

$$\text{Root}_1 = \frac{-b-\sqrt{b^2-4ac}}{2a} \qquad \text{Root}_2 = \frac{-b+\sqrt{b^2-4ac}}{2a}$$

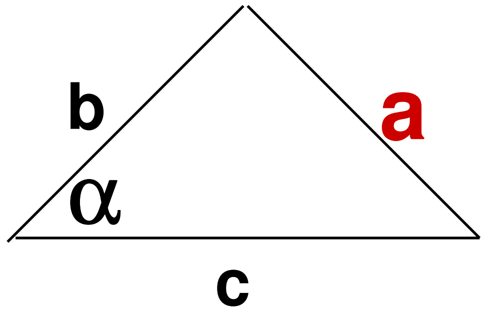
```
disc = pow(b,2) – 4 * a * c;

root_1 = (-b + sqrt(disc)) / (2 * a);

root_2 = (-b - sqrt(disc)) / (2 * a);
```

Math in C

complex math example-2

Write a program to get the third side of a triangle (a), given the lengths of the other two sides (b, and c), and the angle α using the formula

$$a^2 = b^2 + c^2 - 2bc \cos \alpha$$



```
rad_angle = alpha * PI / 180;

a = sqrt(pow(b,2) + pow(c,2) – 2 * b * c * cos(rad_angle));
```

Conditional (ternary) Operator

- Syntax

```
expr1 ? expr2 : expr3
```

  - If **expr1** ≠ 0, then execute **expr2** and ignore **expr3**
  - If **expr1** = 0, then execute **expr3** and ignore **expr2**

Example: `x = i+j ? i+1 : j+1`

Example:

```
   x = 5 ? 4 : 2;          /* x = 4 */
```

Example:

```
   j = 4;
   i = 2
   x = i+j ? i+1 : j-1   /* x = 3 */
```

Example:

```
   l =a > b ? a : b;       /* the larger of  a  and  b */
```

Example:

```
   max =(a > b)?((a>c)?a:c):(b>c)?b:c);
    /*  the maximum number among  a,  b,  and c */
```

Example:

```
   x = a > 0 ? a: -a;      /*  the absolute value of  a */
```

sizeof Operator

- Syntax

  `sizeof(expr)`

  – The number of bytes occupied by **`expr`**

  – For most computers

    sizeof(3)		2 or 4 (bytes)

    (depending on16 bit CPU or 32 bit CPU), where 3 is an integer

    sizeof(3L)		4		(long int)

    sizeof(3.0) 8		(double float)

<u>Example:</u>

```
    double i;
    printf(“%d”,sizeof(i));     8
```

  – Usually, this operator is used to get the size of an organized variable (like **`struct`**, **`union`**, …)

  – This is one of a few functions that are *built-in*. No #include is required.

Address Operator

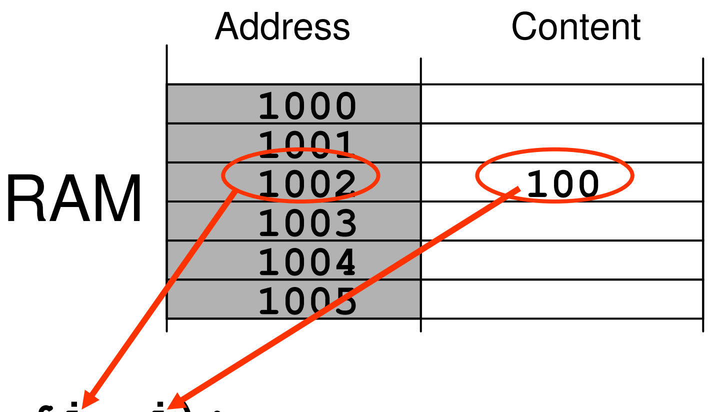
- Syntax

&var

  - Get the address of the variable
  - **&** means the address of **var**
  - Type of **var** may be

    (a) fundamental data type

    (b) organized data type



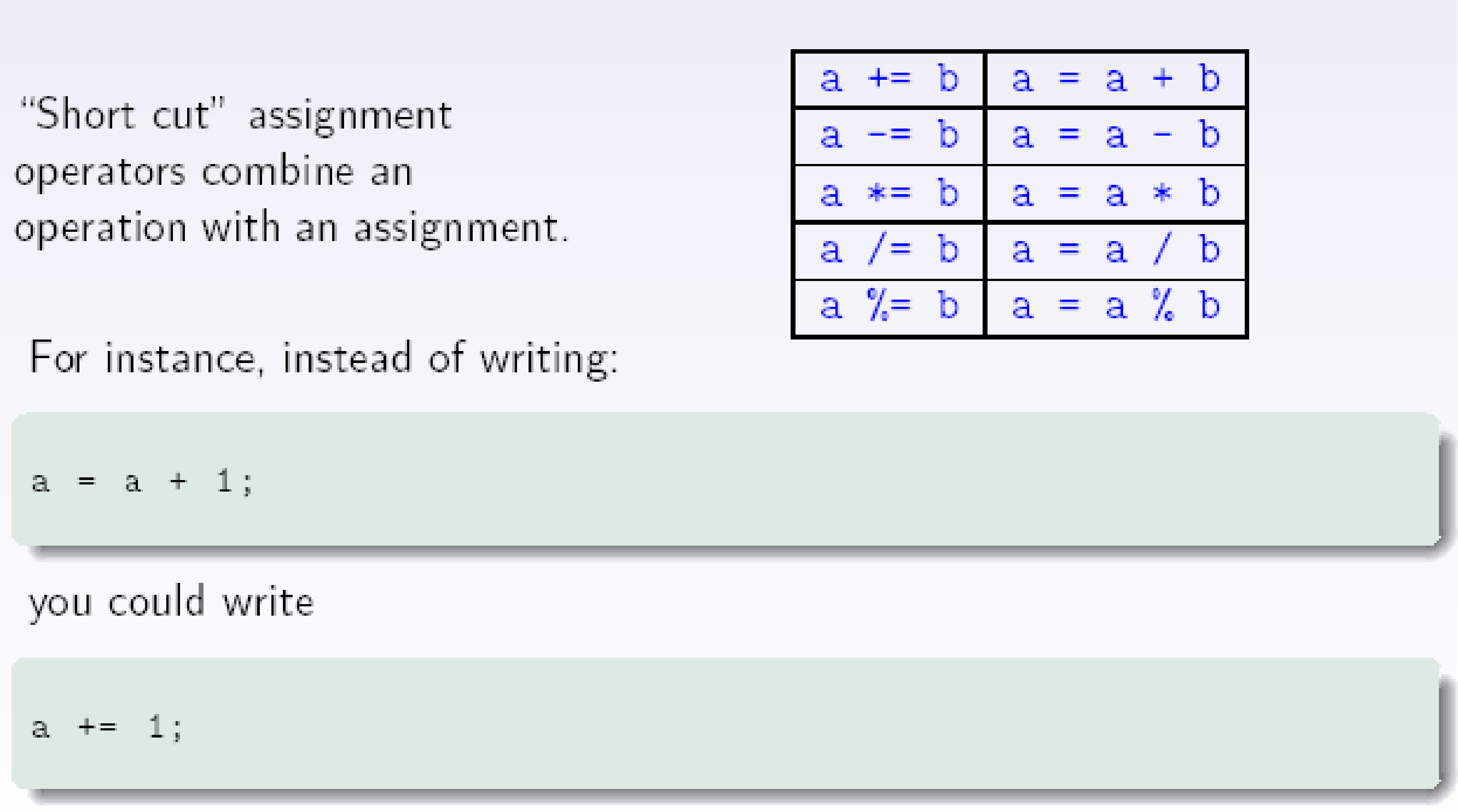
Address	Content
1000	
1001	
1002	100
1003	
1004	
1005	

Example:

```
int i=100;
printf("%d %d", &i, i);
```

Arithmetic Operators

Shortcut assignment



"Short cut" assignment operators combine an operation with an assignment.

`a += b`	`a = a + b`
`a -= b`	`a = a - b`
`a *= b`	`a = a * b`
`a /= b`	`a = a / b`
`a %= b`	`a = a % b`

For instance, instead of writing:

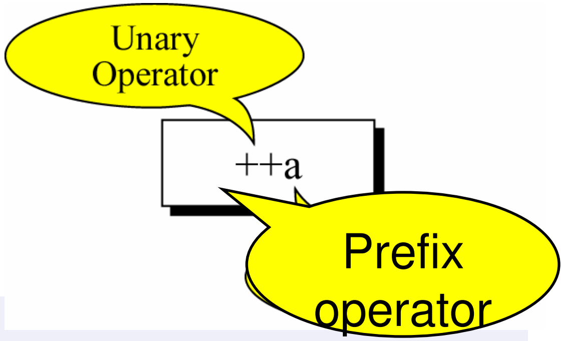
```
a = a + 1;
```

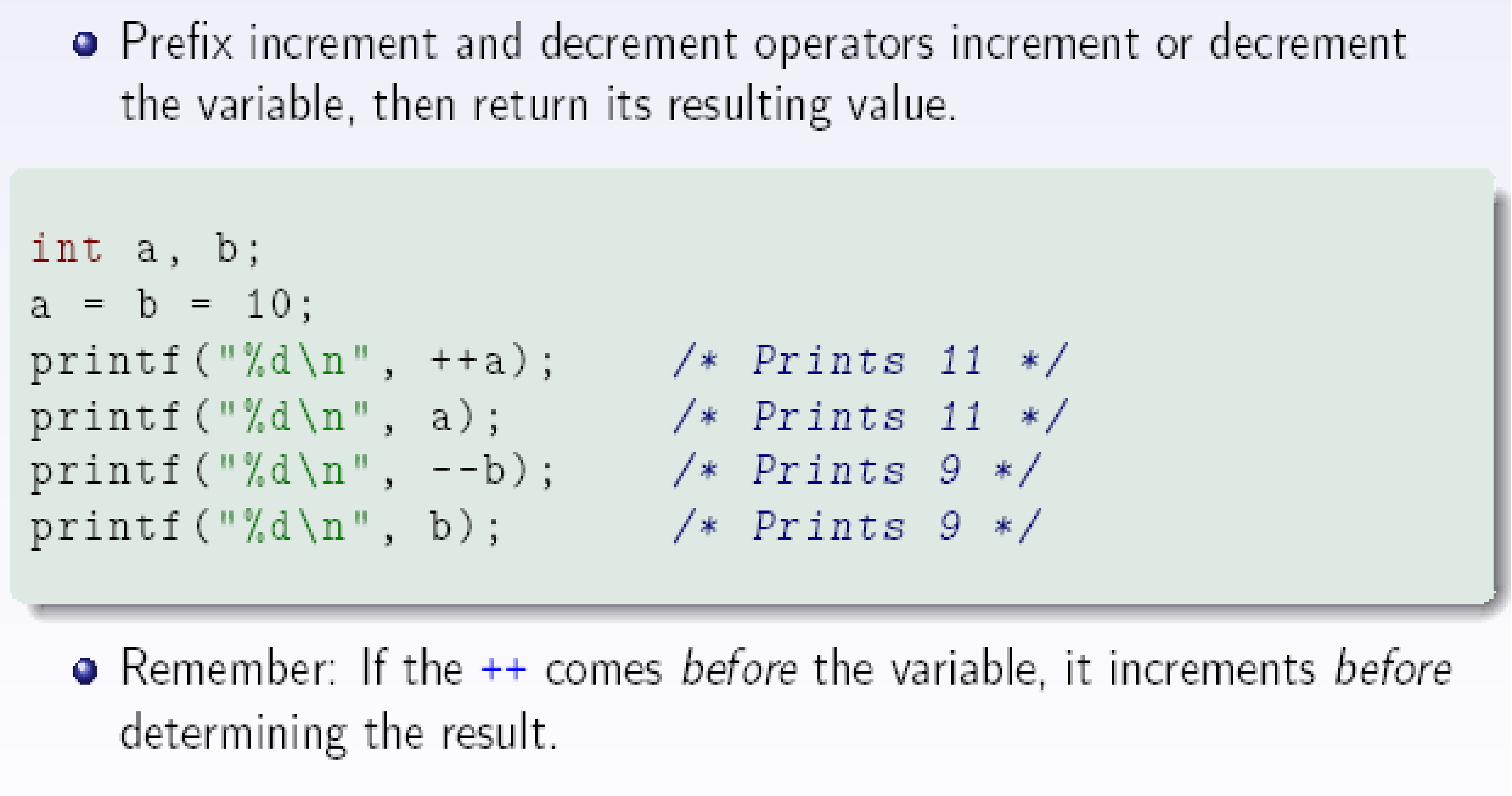
you could write

```
a += 1;
```

Arithmetic Operators

Prefix form





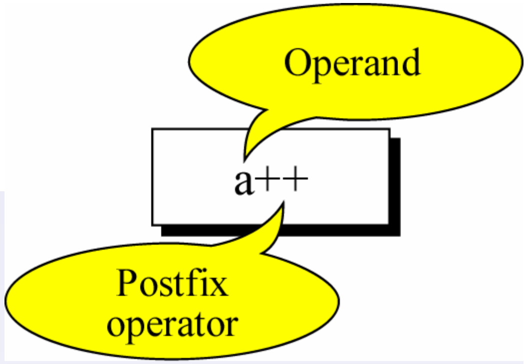
- Prefix increment and decrement operators increment or decrement the variable, then return its resulting value.

```
int a, b;
a = b = 10;
printf("%d\n", ++a);     /* Prints 11 */
printf("%d\n", a);       /* Prints 11 */
printf("%d\n", --b);     /* Prints 9 */
printf("%d\n", b);       /* Prints 9 */
```

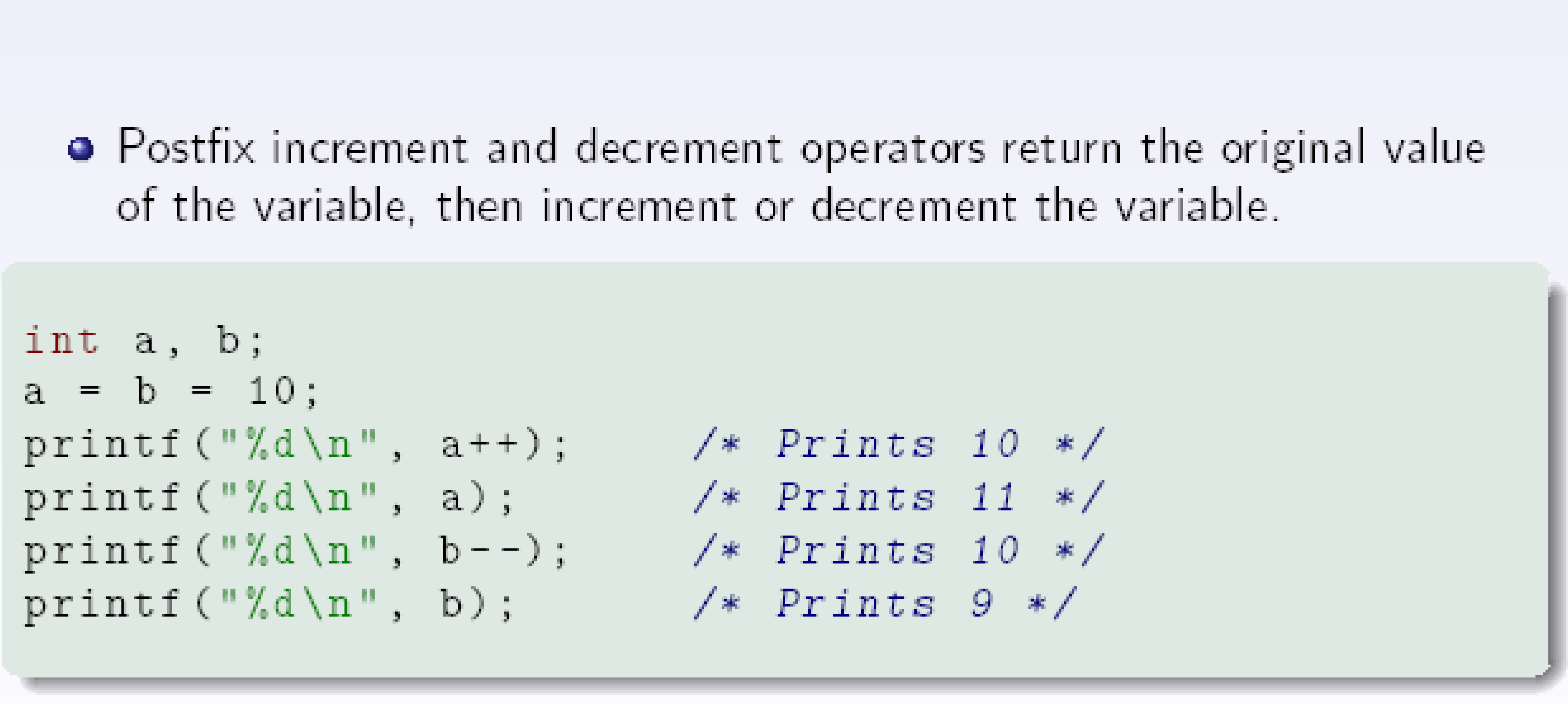
- Remember: If the ++ comes *before* the variable, it increments *before* determining the result.

Arithmetic Operators

Postfix form



Postfix Increment and Decrement



- Postfix increment and decrement operators return the original value of the variable, then increment or decrement the variable.

```
int a, b;
a = b = 10;
printf("%d\n", a++);     /* Prints 10 */
printf("%d\n", a);       /* Prints 11 */
printf("%d\n", b--);     /* Prints 10 */
printf("%d\n", b);       /* Prints 9 */
```

Assignment Operators

- Syntax:

```
var = expression;
```

– Assign the value of expression to variable (**var**)

Example:

```
int x, y, z;
 x = 5;
 y = 7;                ⇒   Z = (x = 5) + (y = 7)   much faster
 z = x + y;
```

```
int x, y, z;
 x = y = z = 0;        ⇒ same as   x = (y = (z = 0));
```

```
int x = y = z = 0;     ⇒ wrong !  int x = 0, y = 0, z = 0;
```

```
int i, j;
float f, g;
 i = f = 2.5;         ⇒ i = 2;       f = 2.5;
 g = j = 3.5;         ⇒ g = 3.0;     j = 3;
```

Short Hand Assignment

- Syntax

  `f = f op g` can be rewritten to be **`f op= g`**

such as: `a = a + 2 ⇒ a += 2,` `a = a - 2 ⇒ a -= 2,` `a = a * 2 ⇒ a *= 2,`
`a = a / 2 ⇒ a /= 2,` `a = a % 2 ⇒ a %= 2,` `a = a << 2 ⇒ a <<= 2,`
`a = a & 2 ⇒ a &= 2,` `a = a | 2 ⇒ a |= 2,` `a = a ^ 2 ⇒ a ^= 2`

- No blanks between **`op`** and **`=`**
- **`x *= y + 1`** is actually **`x = x * (y+1)`** rather than **`x = x * y + 1`**

  <u>Example</u>: **`q = q / (q+2)`** ⇒ **`q /= q+2`**
  `j = j << 2` ⇒ **`j <<= 2`**

- Advantage: help compiler to produce more efficient code

  <u>More complicated examples</u>:

```
int a=1, b=2, c=3, x=4, y=5;
a += b += c *= x + y - 6;
printf(“%d %d %d %d\n”,a,b,c,x,y); /*  result is 12 11 9 4 5 */
a += 5 + b += c += 2 + x + y;       /* wrong */
a += 5 + (b+= c += 2 + x + y);      /* result is  22 16 14 4 5  */
```

Increment / Decrement Operators

++ (increment) -- (decrement)

- Prefix Operator
  - Before the variable, such as **++n** or **--n**
  - Increments or decrements the variable before using the variable
- Postfix Operator
  - After the variable, such as **n++** or **n--**
  - Increments or decrements the variable after using the variable

❑ ++n

1. Increment **n** 2. Get value of **n** in expression

❑ --n

1. Decrement **n** 2. Get value of **n** in expression

❑ n++

1. Get value of **n** in expression 2. Increment **n**

❑ n--

1. Get value of **n** in expression 2. Decrement **n**

Increment / Decrement Operators (cont.)

– Simple cases

```
++i;
i++;      (i = i + 1; or i += 1;)
--i;
i--;      (i = i - 1; or i -= 1;)
```

Example:

```
i = 5;
i++; (or ++i;) ⇒ 6
```

```
i = 5;
i--; (or --i;)
printf("%d", i) ⇒ 4
```

– Complicated cases

```
i = 5;                    i    j
j = 5 + ++i;          6    11


i = 5;
                      6    10
j = 5 + i++;


i = 5;               4      9
j = 5 + --i;


                     4    10
i = 5;
j = 5 + i--;
```

Increment / Decrement Operators (cont.)

- Invalid cases

```
++3          3++           --3        3--
++(x+y+z)    (x+y+z)++     --(x+y+z)  (x+y+z)--
++x++        --x--         ++x--      --x++
```

Note: Can not increment or decrement constant and expression

```
i ++j        or       i --j  (WRONG)
i + ++j      i + --j       i - --j    i - ++j      (OK)
```

Other Input / Output

`puts(line)` Print a string to standard output and append a newline

Example: `puts("12345");`

`putchar(c)` Print a character to standard output

Example: `putchar('A');`

`gets(line)` Read a string from standard input (until a newline is entered)

Example:

```
char buf[128];
gets(buf);
```

/* space is OK, and the '\n' won't be read in */

– Newline will be replaced by '\0'

`getchar()` Get a character from standard input

Example:

```
int c;
c = getchar();
```

/* `c` must be `int` */

- In-memory Format Conversion

```
sprintf(string, control, variables);
```