

Week 3- lecture 2

Operators

Edited by: Dr. Saeid Pourroostaei Ardakani
Autumn 2021



Smile More.®

Quiz!

Which one is False?

- A) `Printf()` prints until a `\0`.
- B) `getchar()` reads only one character from input.
- C) `scanf()` reads characters until it encounters a space.
- D) 2D array only uses `char`, but not strings.

Quiz!

Which one is False?

- A) `Printf()` prints until a `\0`.
- B) `getchar()` reads only one character from input.
- C) `scanf()` reads characters until it encounters a space.
- D) 2D array only uses char, but not strings.**

Overview

- **Operators**
- Meaningful names
- Encapsulation and Refactoring

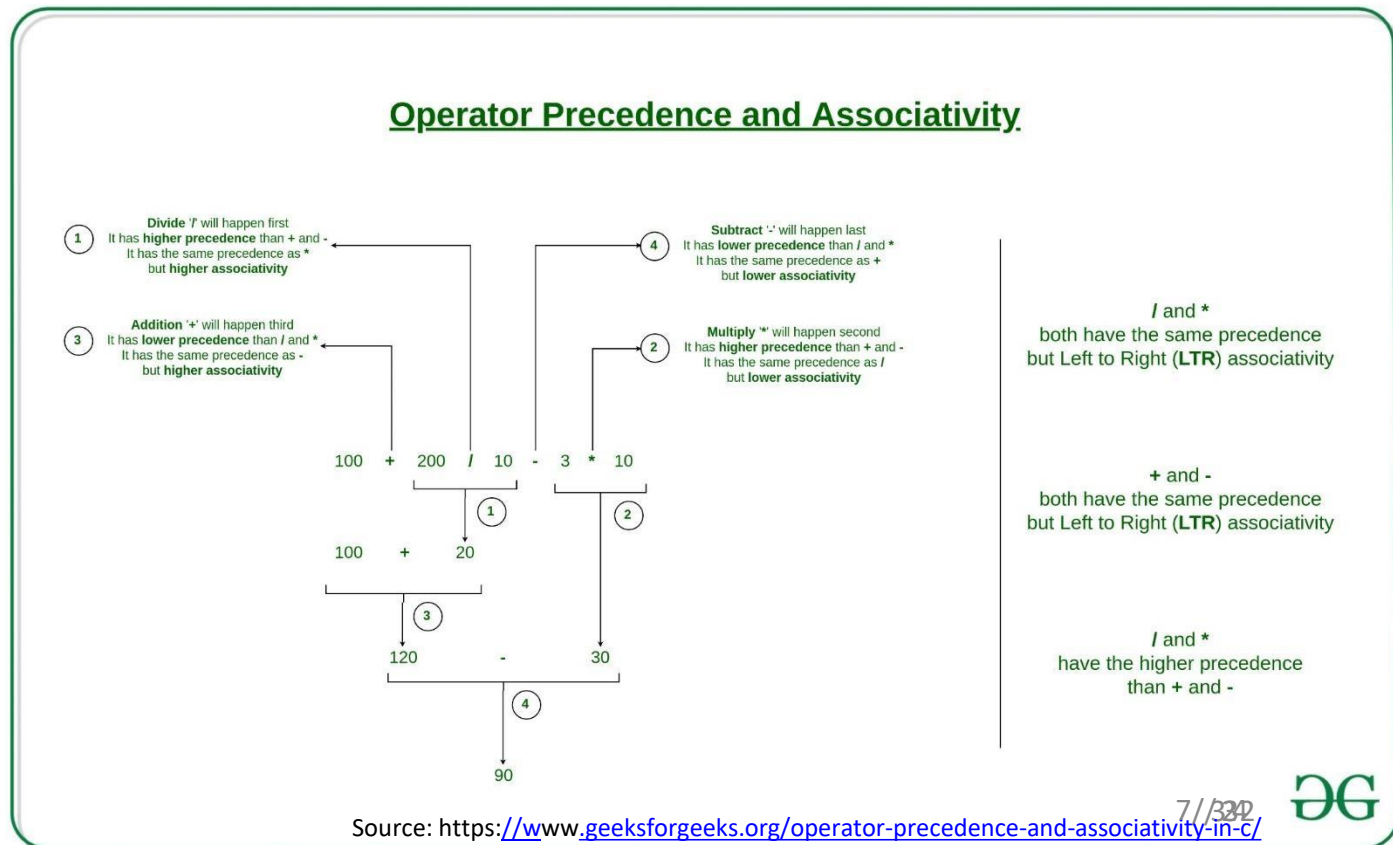


Operator Precedence

- $7 + 5 * 3 - 1 = 21$ $*$ has higher precedence
 $7 * 4 / 2 + 5 = 19$ $*$ and $/$ are left associative
- $a = 7 + (5 * 3) - 1;$
 rather than $(7 + 5) * (3 - 1);$
- $a = (((7 * 4) / 2) + 5);$

Operator Associativity

- Suggestion: ALWAYS use brackets and simply your statements!



Assignment and Arithmetic Operators

- Assignment (=)

```
int a, b, c;
```

```
a = b = c = 10;
```

From right to left

Watch out for types e.g.

```
int a;
```

```
float b;
```

```
b = a = 10.22;
```

MUST be int % int

- Arithmetic

```
+ - * / %
```


Increment and Decrement Operators

- `int a = 4;`

`a++;` `a = a + 1 = 5`

- `int a = 4, b;`

`b = a++;` `b = a; a = a + 1 = 5`
`BUT b = 4!!!`

- `int a = 1, b = 2, c = 3, d;`

`d = (++a)-(b--)+(--c);`

`d = 2 - 2 + 2 = 2`
`BUT a, b, c becomes 2, 1, 2`

Relational Operators

- $>$, $>=$, $<$, $<=$, $!=$, $==$

```
60     int a = 4;
61     int b = 5;
62     int c = 5;
63     int d = 6;
64
65     printf("%d > %d = %d\n", a, b, a > b);
66     printf("%d < %d = %d\n", a, b, a < b);
67
68     printf("%d <= %d = %d\n", a, b, a <= b);
69     printf("%d >= %d = %d\n", d, b, d >= b);
70     printf("%d >= %d = %d\n", c, b, c >= b);
71     printf("%d <= %d = %d\n", c, b, c <= b);
72
73     printf("%d == %d = %d\n", c, b, c == b);
74     printf("%d != %d = %d\n", c, b, c != b);
```

Not Operator

- Every non-zero number is true.
- Not operator (!) acts on a single operand
`int a = 4;`
`printf("%d\n", !a);`

↑
Display zero

`if(!a)` is equivalent to `if(a == 0)`

`if(a)` is equivalent to `if (a != 0)`

```
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121
```

```
int a = 4;  
  
if(!a)  
{  
    printf("a is equal to false\n");  
}  
else  
{  
    printf("a is equal to true\n");  
}  
  
if(a)  
{  
    printf("a is equal to true\n");  
}  
else  
{  
    printf("a is equal to false\n");  
}  
  
if(a == 0)  
{  
    printf("a is equal to false\n");  
}  
else  
{  
    printf("a is equal to true\n");  
}  
  
if(a != 0)  
{  
    printf("a is equal to true\n");  
}  
else  
{  
    printf("a is equal to false\n");  
}
```

11/32

Make Errors Obvious

- Try your best to avoid implicit statements.

```
.54     int a = 1;
.55     int b = 2;
.56     int c = 0;
.57
.58     if(a) // not recommended, it is best to make your purpose explicit
.59     {
.60         printf("a is true\n");
.61     }
.62
.63     if(a == 1)
.64     {
.65         printf("a is true\n");
.66     }
.67
```

Compound Operators

- $\text{exp1 op} = \text{exp2};$
 $\text{exp1} = \text{exp1 op} (\text{exp2});$

Now ... let's suppose that $a = 4$ and $b = 2$

$a += 6;$

$$a = a + 6 = 10$$

$a *= b + 3;$

$$a = a * (b + 3) = 4 * (2 + 3) = 20$$

$a -= b + 8;$

$$a = a - (b + 8) = 4 - (2 + 8) = -6$$

$a /= b;$

$$a = a / b = 4 / 2 = 2$$

$a \% = b + 1;$

$$a = a \% (b + 1) = 4 \% (2 + 1) = 1$$

Logical && Operators

- && is left associative, returns 1 if all operands are true
- ```
int a = 10, b = 20, c;
c = (a > 15) && (++b > 15);
```
- ```
printf(“%d %d\n”, c, b);
```

Since **the first operand (a > 15) is false**, the second operand is not evaluated, the program displays 0 and 20;

Logical && Operators (2)

C:\ Command Prompt

```
C:\Users\z2017233\Desktop>operators
a = 10, b = 21, and c = 1
a = 10, b = 20, and c = 0
C:\Users\z2017233\Desktop>
```

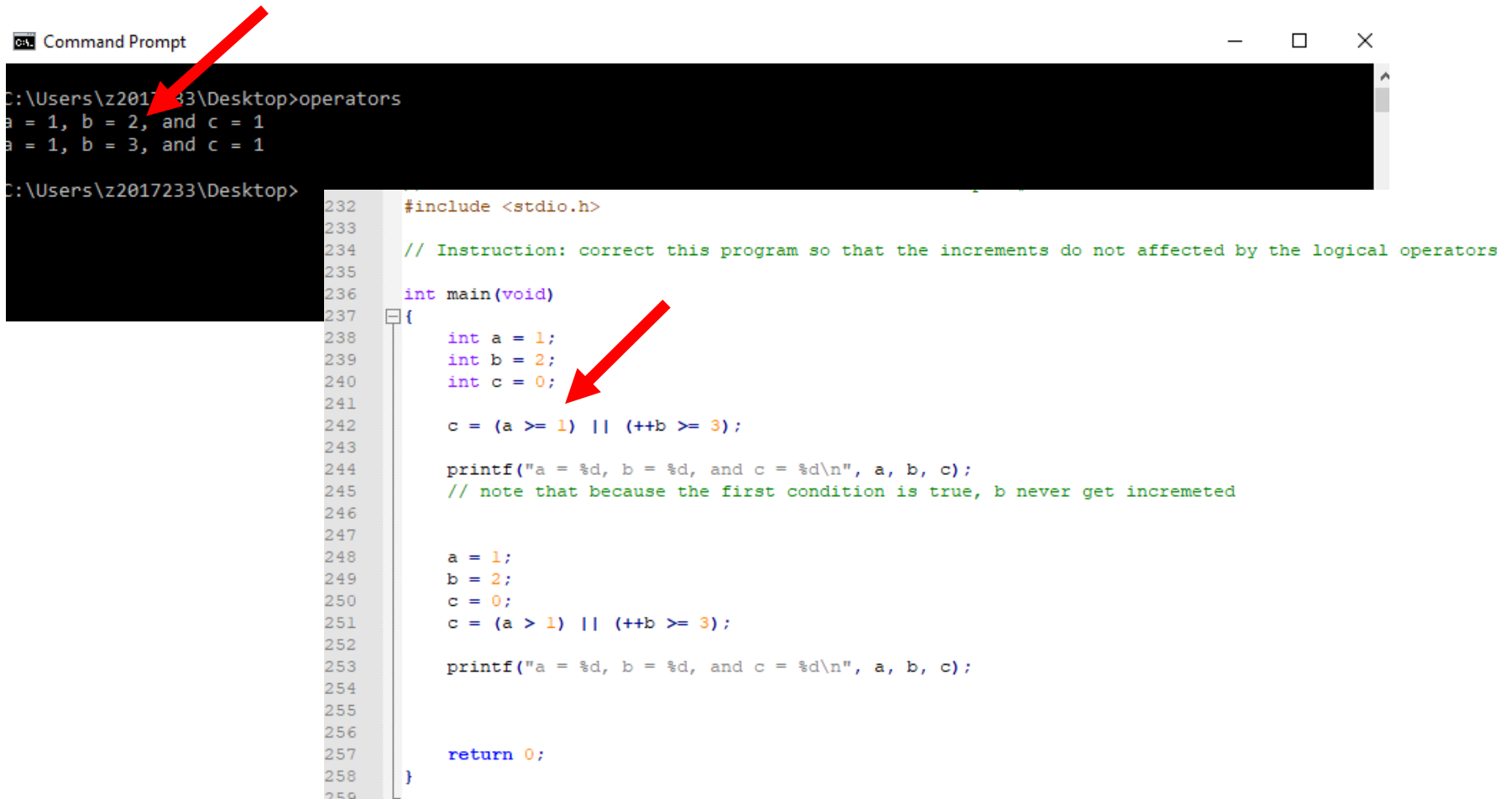
```
203 #include <stdio.h>
204
205 int main(void)
206 {
207     int a = 10;
208     int b = 20;
209     int c = 0;
210
211     c = (a >= 10) && (++b >= 15);
212
213     printf("a = %d, b = %d, and c = %d\n", a, b, c);
214
215
216     a = 10;
217     b = 20;
218     c = 0;
219     c = (a > 10) && (++b >= 15);
220
221     printf("a = %d, b = %d, and c = %d\n", a, b, c);
222     // note that because the first condition is false, b never get incremented
223
224
225     return 0;
226 }
227
```

Logical II Operators

- `||` returns 1 if one of more operands are true
- ```
int a = 10, b = 20, c;
c = (a > 5) || (++b > 15);
```
- ```
printf("%d %d\n", c, b);
```

Since **the first operand (a > 5) is true**, the second operand is not evaluated, the program displays 1 and 20;

Logical II Operators (2)



```
CA: Command Prompt
C:\Users\z2017233\Desktop>operators
a = 1, b = 2, and c = 1
a = 1, b = 3, and c = 1
C:\Users\z2017233\Desktop>

#include <stdio.h>
// Instruction: correct this program so that the increments do not affected by the logical operators
int main(void)
{
    int a = 1;
    int b = 2;
    int c = 0;

    c = (a >= 1) || (++b >= 3);

    printf("a = %d, b = %d, and c = %d\n", a, b, c);
    // note that because the first condition is true, b never get incremented

    a = 1;
    b = 2;
    c = 0;
    c = (a > 1) || (++b >= 3);

    printf("a = %d, b = %d, and c = %d\n", a, b, c);

    return 0;
}
```

Suggestions: Sequence Statements

- Put only one statement per line
- Avoid statements that rely on **side-effect order** e.g. ++, -- put the variables on lines by themselves
- Use blank lines to organise statements into paragraphs and to separate logically related statements
- Use indentations

Source: <http://homepages.inf.ed.ac.uk/dts/pm/Papers/nasa-c-style.pdf>



Recommended coding style

- blank lines, spaces and indentation

```
1 #include <stdio.h>
2
3 #define LOWER 0
4 #define UPPER 300
5 #define STEP 20
6
7 int main() /* Fahrenheit - Celsius table */
8 {
9     int fahr = 0;
10
11     for (fahr = LOWER; fahr <= UPPER; (fahr = (fahr + STEP)))
12         printf("%4d %6.1f\n", fahr, ((5.0 / 9.0) * (fahr - 32)));
13
14     return 0;
15 }
```

```
1 #include <stdio.h>
2 #define LOWER 0
3 #define UPPER 300
4 #define STEP 20
5 int main()
6 {
7     int fahr;
8     for (fahr=LOWER;fahr<=UPPER;fahr=fahr+STEP)
9         printf("%4d %6.1f\n",fahr,(5.0/9.0)*(fahr-32));
10    return 0;
11 }
```

Your compiler doesn't check your comments !!

Comma Operator

- Comma (,) is left associative

```
int b;
```

```
b = 20, b = b + 30, printf("Num = %d\n", b);
```

Num = 50 will be displayed

- The most common use of comma is in **for** statement

```
int a, b;
```

```
for(a = 1, b = 2; b < 10; a++, b++)
```

This loop will be executed 8 times

Suggestions: Compound Statements

- Lists of statements enclosed in braces aka blocks, i.e.
Braces around single statements can sometime help improve the readability.
- If a for loop will not fit on one line, **do three!!**

```
for (curr = *listp, trail = listp;  
    curr != NULL;  
    trail = &(curr->next), curr = curr->next)  
{  
    statement_1;  
    ...  
    statement_n;  
}
```

Source: <http://homepages.inf.ed.ac.uk/dts/pm/Papers/nasa-c-style.pdf>



Suggestions: Compound Statements (cont.)

- For large block, comment closing braces

```
for (sy = sytable; sy != NULL; sy = sy->sy_link)
{
    if (sy->sy_flag == DEFINED)
    {
        ...
    }          /* if defined          */
    else
    {
        ...
    }          /* if undefined        */
}             /* for all symbols      */
```

- Look at the following example

```
/* Locate end of string */
for (char_p = string; *char_p != EOS; char_p++)
    ;    /* do nothing */
```

Suggestion: Limit the Complexity

Command Prompt

```
C:\Users\z2017233\Desktop>controls
10

C:\Users\z2017233\Desktop>
```

```
249 #include <stdio.h>
250
251 // Instruction: correct the program so that it will display the correct message
252
253 int main(void)
254 {
255     int i = 10;
256
257     printf("%d\n", (i==10)?i++:(i>10)?i++:(i>10)?i--:(i>10)?i--:0);
258
259     return 0;
260 }
```

Overview

- Operators
- **Meaningful names**
- Encapsulation and Refactoring



Meaningful Names

- Precise and consistent
- Not too long names
- Follow uniform scheme when abbreviation
- C is case sensitive!

Example: standard short names

c	characters
i, j, k	indices
n	counters
p, q	pointers
s	strings

Example: standard suffixes for variables

_ptr	pointer
_file	variable of type file*
_fd	file descriptor

Examples - Commenting

Example: boxed comment prolog

```
/* ****  
 * FILE NAME  
 *  
 * PURPOSE  
 *  
 **** */
```

Example: section separator

```
/* **** */
```

Example: block comment

```
/*  
 * Write the comment text here, in complete sentences.  
 * Use block comments when there is more than one  
 * sentence.  
 */
```

Example: short comments

```
double ieee_r[];          /* array of IEEE real*8 values */
```

Overview

- Operators
- Meaningful names
- **Encapsulation and Refactoring**



Encapsulation and Information Hiding

- Grouping related elements into:
 - Files e.g., header files
 - Data sections and function sections
 - Groups of logically related functions
 - Groups of logically related data e.g., structure
- Controlling the visibility or scope of program elements:
 - Include only needed header files
 - External variable is only visible to a function when declared by the external declaration



Example

```

1 #include <stdio.h>
2
3 int main(){
4     printf("Hello World!!\n");
5     return 0;
6 }
7

```

```

int printf(const char *format, ...)
{
    va_list args;

    va_start( args, format );
    return print( 0, format, args );
}

int sprintf(char *out, const char *format, ...)
{
    va_list args;

    va_start( args, format );
    return print( &out, format, args );
}

```

```

static int print(char **out, const char *format, va_list args )
{
    register int width, pad;
    register int pc = 0;
    char scr[2];

    for ( ; *format != 0; ++format) {
        if (*format == '%') {
            ++format;
            width = pad = 0;
            if (*format == '\\0') break;
            if (*format == '%') goto out;
            if (*format == '-') {
                ++format;
                pad = PAD_RIGHT;
            }
            while (*format == '0') {
                ++format;
                pad |= PAD_ZERO;
            }
            for ( ; *format >= '0' && *format <= '9'; ++format) {
                width *= 10;
                width += *format - '0';
            }
            if( *format == 's' ) {
                register char *s = (char *)va_arg( args, int );
                pc += prints (out, s?s:"(null)", width, pad);
                continue;
            }
            if( *format == 'd' ) {
                pc += printi (out, va_arg( args, int ), 10, 1, width, pad, 'a');
                continue;
            }
            if( *format == 'x' ) {
                pc += printi (out, va_arg( args, int ), 16, 0, width, pad, 'a');
                continue;
            }
            if( *format == 'X' ) {
                pc += printi (out, va_arg( args, int ), 16, 0, width, pad, 'A');
                continue;
            }
            if( *format == 'u' ) {
                pc += printi (out, va_arg( args, int ), 10, 0, width, pad, 'a');
                continue;
            }
            if( *format == 'c' ) {
                /* char are converted to int then pushed on the stack */
                scr[0] = (char)va_arg( args, int );
                scr[1] = '\\0';
                pc += prints (out, scr, width, pad);
                continue;
            }
            else {
                out:
                printchar (out, *format);
                ++pc;
            }
        }
        if (out) **out = '\\0';
        va_end( args );
        return pc;
    }
}

```

Code Refactoring

```
#include<stdio.h>
void main()
{
    int a,b,result;
    printf("\nGoing to calculate the sum :");
    printf("\nEnter two numbers:");
    scanf("%d %d",&a,&b);
    result = a+b;
    printf("\nThe sum is %d",result);
}
```

```
#include<stdio.h>
void sum(int, int);
void main()
{
    int a,b,result;
    printf("\nGoing to calculate the sum :");
    printf("\nEnter two numbers:");
    scanf("%d %d",&a,&b);
    sum(a,b);
}
void sum(int a, int b)
{
    printf("\nThe sum is %d",a+b);
}
```

To improves non-functional attributes of the software

Code Refactoring (2)

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     /* square root of n with Newton-Raphson approximation */
7
8     double r = 10.0;
9     double n = 20.0;
10    double t = 30.0;
11
12    r = (n / 2);
13    while ( abs( r - (n / r) ) > t )
14    {
15        r = 0.5 * ( r + (n / r) );
16    }
17    printf( "r = %.2f\n", r );
18    return 0;
19 }
```

Function Declaration

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 double square_root_approx(double n);
5
6 int main()
7 {
8     double approx = 0.0;
9
10    approx = square_root_approx(20.0);
11    printf( "r = %.2f\n", approx );
12
13    return 0;
14 }
15
16 double square_root_approx(double n)
17 {
18     /* square root of n with Newton-Raphson approximation */
19
20     double r = 10.0;
21     double t = 30.0;
22
23     r = (n / 2);
24     while ( abs( r - (n / r) ) > t )
25     {
26         r = 0.5 * ( r + (n / r) );
27     }
28
29     return r;
30 }
```

Function Definition

Nested If Statements

- Do not use nested if statements when only if clause contains actions

Example: absence of braces produces undesired result

```
if (n > 0)
    for (i = 0; i < n; i++)
        if (s[i] > 0)
        {
            printf("...");
            return(i);
        }
else /* WRONG -- the compiler will match to closest */
    /* else-less if */
    printf("error - n is zero\n");
```

Without the braces, else will be paired with the nearest If.

Example: braces produce desired result

```
if (n > 0)
{
    for (i = 0; i < n; i++)
    {
        if (s[i] > 0)
        {
            printf("...");
            return(i);
        }
    }
}
else /* CORRECT -- braces force proper association */
    printf("error - n is zero\n");
```

Source: <http://homepages.inf.ed.ac.uk/dts/pm/Papers/nasa-c-style.pdf>

Style Guides

- NASA

https://mechatronics.me.wisc.edu/labresources/DataSheets/NASA-GSFC_C_Programming_Styles-94-003.pdf

- IPA

<https://www.ipa.go.jp/files/000065271.pdf>

- In-house, etc.

Summary

- Operators
- Meaningful names
- Encapsulation and Refactoring



Quiz!

Which one is True?

- A) “%” can use float operand.
- B) “a++” and “++a” act same.
- C) “for(a = 1, b = 2; b < 2; a++, b++)” addresses no repeat.
- D) Encapsulation uncovers information.

Quiz!

Which one is True?

- A) “%” can use float operand.
- B) “a++” and “++a” act same.
- C) “for(a = 1, b = 2; b < 2; a++, b++)” addresses no repeat.
- D) Encapsulation uncovers information.