Debugging

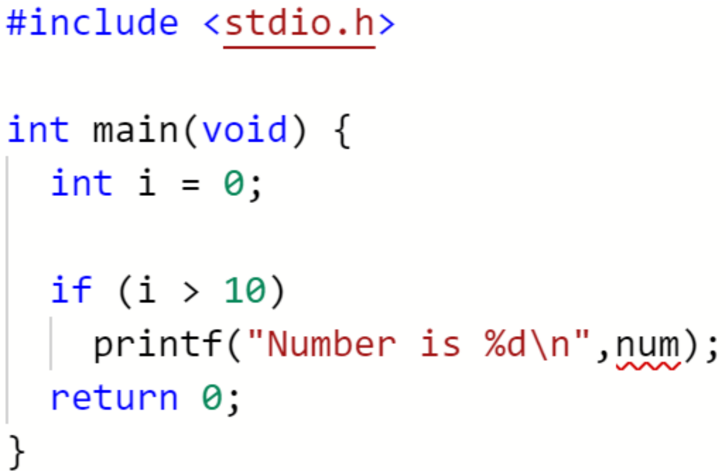
Outline

- Types of Errors
  - Compilation/Syntax errors
  - Run-Time errors
  - Logical errors

- Debugging techniques to locate logical errors

Types of Errors – Syntax Errors

- Type 1. **Compilation/Syntax Errors**
  - The errors that are detected during the compilation of the program
  - repl.it and other C environments can usually highlight these errors



```
#include <stdio.h>

int main(void) {
  int i = 0;

  if (i > 10)
    printf("Number is %d\n",num);
  return 0;
}
```

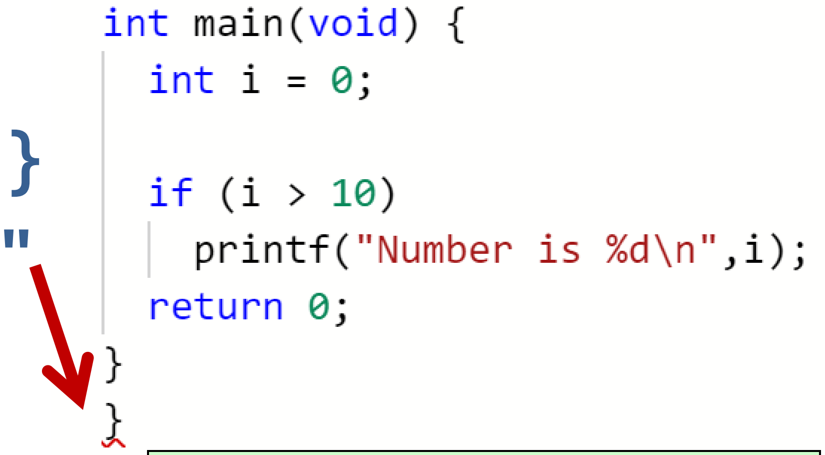
Move your mouse cursor over the squiggly lines to get more info about the error.

Note: Sometimes the errors may appear before the indicated line.

Types of Errors – Syntax Errors

- Type 1. **Compilation/Syntax Errors**

- Common syntax errors:
  - Duplicated variable names
  - Missing semi-colons **;**
  - Mismatched braces **{ }**
  - Mismatched quotes **" "**
    - **…("…) "…**
    - **…('…")…**
    - **…("…, "…)…**



```
int main(void) {
  int i = 0;

  if (i > 10)
    printf("Number is %d\n",i);
  return 0;
}
}
```

This is called a dangling brace.

Types of Errors – Run-time Errors

- Type 2. **Run-Time Errors**
  - The errors occur while the program is running and cause the program to crash.

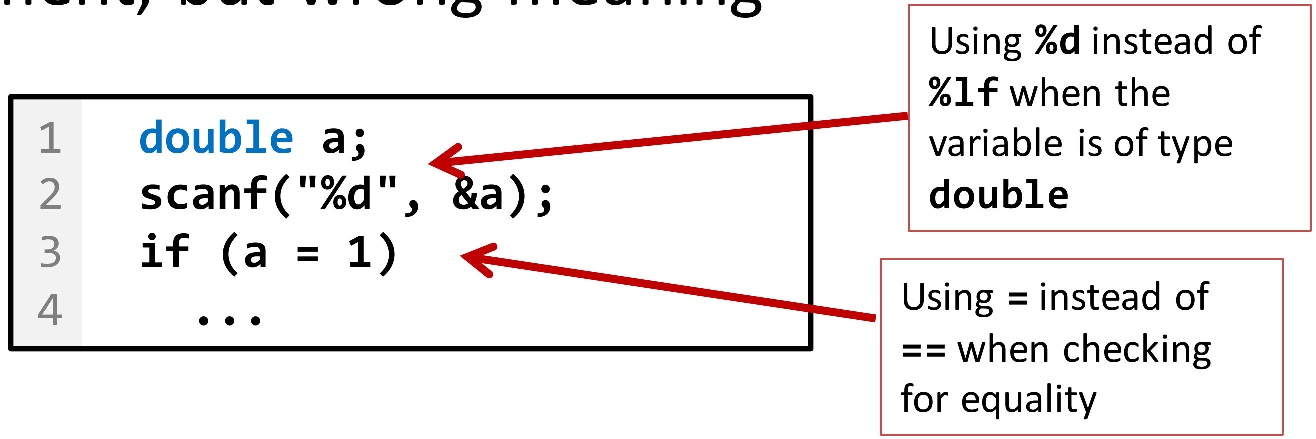
```
int a, b;
a = 3;
b = 0;
printf("%d\n", a / b);
```

- Common run-time errors:
  - Division by zero
  - Array index out of bound
    - The consequence of the array index out of bound error is unpredictable;
    - The program may crash (run-time errors), or
    - Some variables may get modified unknowingly (the program does not crash).

```
int array[10] = { 0 };
array[10] = 50;
printf("%d\n", array[1000]);
```

Types of Errors – Logical Errors

- Type 3. **Logical Errors**: the result is unexpected!
  - Not syntax errors or run-time errors.
    - i.e., the program can be compiled and executed successfully.
  - But, the program ***logic*** is wrong.
- Source of errors: (1) **Typo**.
  - Valid C statement, but wrong meaning



```
double a;
scanf("%d", &a);
if (a = 1)
  ...
```

Using **`%d`** instead of **`%lf`** when the variable is of type **`double`**

Using = instead of == when checking for equality

Types of Errors – Logical Errors

- Source of errors: (2) **Incorrect program logic**.
  - This is the most frustrating moment!
  - Because we usually spend **most of the programming time** in discovering where the error is.

- Don’t give up yet!
  - We have systematic ways to locate logical bugs.

How to Locate a Logical Error?

- The output of this program is incorrect.
- How should we approach to find the bug?

```
// A program to convert temperature in degree Fahrenheit
// to equivalent degrees in Celsius and Kelvin.

double F, C, K; // Fahrenheit, Celsius, Kelvin

scanf("%lf", &F);

C = 5 / 9 * (F - 32);

K = C + 273.15;

printf("%.2lfF = %.2lfC = %.2lfK\n", F, C, K);
```

The first bug in computer history

How to Locate a Logical Error?

- Every statement computes in the following manners
  - Base its computation on the value of some variable(s)
  - Update the value of some variable(s)

```c
// A program to convert temperature in degree Fahrenheit
// to equivalent degrees in Celsius and Kelvin.

double F, C, K; // Fahrenheit, Celsius, Kelvin

scanf("%lf", &F);

C = 5 / 9 * (F - 32);

K = C + 273.15;

printf("%.2lfF = %.2lfC = %.2lfK\n", F, C, K);
```

Use the value of **F** to compute, and update the value of **C**.

How to Locate a Logical Error?

- If a variable is assigned a wrongly computed value, subsequent computations will likely produce wrong results.

```c
// A program to convert temperature in degree Fahrenheit
// to equivalent degrees in Celsius and Kelvin.

double F, C, K; // Fahrenheit, Celsius, Kelvin

scanf("%lf", &F);

C = 5 / 9 * (F – 32);

K = C + 273.15;

printf("%.2lfF = %.2lfC = %.2lfK\n", F, C, K);
```

If **C** is assigned a wrong value here, then subsequently the value of **K** and the output will be affected.

How do we find out if **C**'s value is wrong?

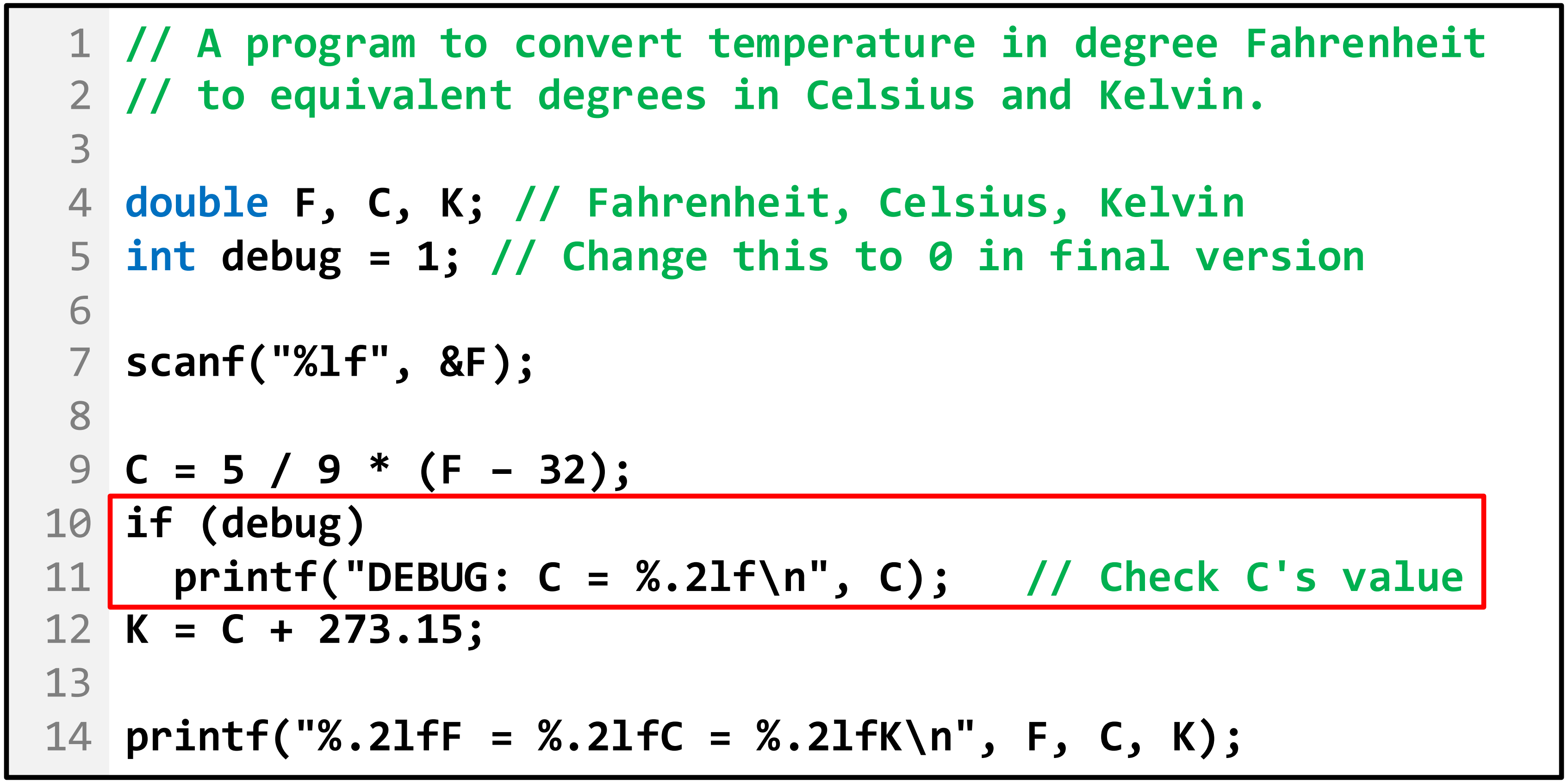
How to Locate a Logical Error?

- Variables usually hold some clues to the bug.
  - One way to inspect variables is to output their values.

```
// A program to convert temperature in degree Fahrenheit
// to equivalent degrees in Celsius and Kelvin.

double F, C, K; // Fahrenheit, Celsius, Kelvin

scanf("%lf", &F);

C = 5 / 9 * (F – 32);
printf("DEBUG: C = %.2lf\n", C);    // Check C's value
K = C + 273.15;

printf("%.2lfF = %.2lfC = %.2lfK\n", F, C, K);
```

How to Locate a Logical Error?

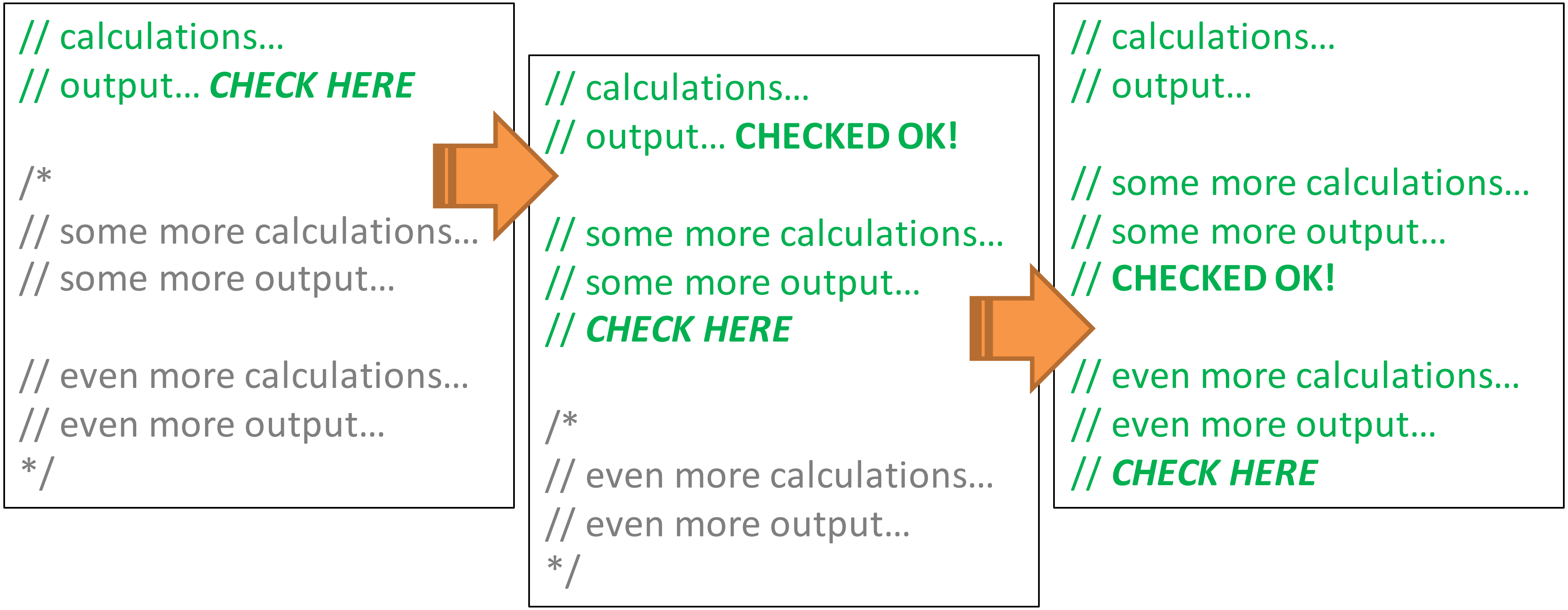
- (Even better) Use a variable to control whether debug messages should be printed
  - Can suppress debug messages in final version (e.g., Replit submission)



```c
// A program to convert temperature in degree Fahrenheit
// to equivalent degrees in Celsius and Kelvin.

double F, C, K; // Fahrenheit, Celsius, Kelvin
int debug = 1; // Change this to 0 in final version

scanf("%lf", &F);

C = 5 / 9 * (F - 32);
if (debug)
  printf("DEBUG: C = %.2lf\n", C);   // Check C's value
K = C + 273.15;

printf("%.2lfF = %.2lfC = %.2lfK\n", F, C, K);
```

Narrowing Down the Range

- If you find that you have too many lines of code and not sure where to insert **`printf()`**, you may comment out later parts of the program first

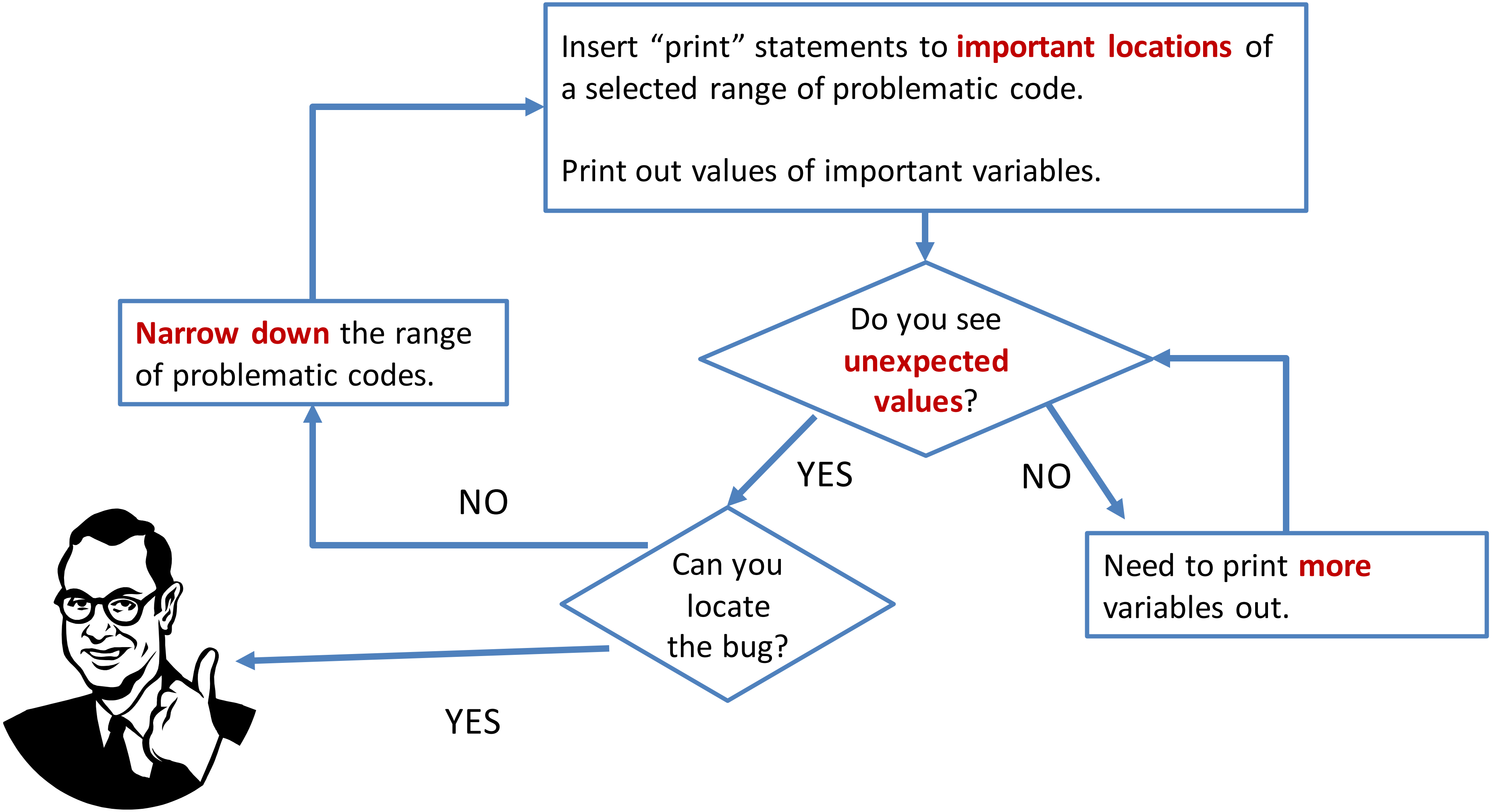


```
// calculations…
// output… CHECK HERE

/*
// some more calculations…
// some more output…

// even more calculations…
// even more output…
*/
```

```
// calculations…
// output… CHECKED OK!

// some more calculations…
// some more output…
// CHECK HERE

/*
// even more calculations…
// even more output…
*/
```

```
// calculations…
// output…

// some more calculations…
// some more output…
// CHECKED OK!

// even more calculations…
// even more output…
// CHECK HERE
```

Steps in Locating Logical Errors



Insert "print" statements to **important locations** of a selected range of problematic code.

Print out values of important variables.

Do you see **unexpected values**?

YES

NO

Need to print **more** variables out.

Can you locate the bug?

NO

Narrow down the range of problematic codes.

YES