Debugging Techniques

Outline

- Three major types of Program Errors
  1. Compilation/syntax errors
  2. Run-time errors
  3. Logical errors

- Debugging techniques to locate logical errors

Type 1. **Compilation/Syntax Errors**

- Errors that are detected during the program compilation
- Most of the integrated development environment (IDE) will highlight these errors

```
int main(void) {
    int i;
    for (i = 0; i < 10; i++) {
        printf("%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.

Type 1. **Compilation/Syntax Errors**

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

```
int main(void) {
    int i;
    for (i = 0; i < 10; i++) {
        printf("%d\n", i);
    }
    return 0;
}
}
```

This is called a dangling brace.

Type 2. **Run-Time Errors**

- Errors that occur when the program is running and it causes the program to crash.
- Common run-time errors:
  - Division by zero

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

  - 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).

(See week 5 notes)

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

Type 3. **Logical Errors**

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

```
1   double a;
2   scanf("%d", &a);
3   if (a = 1)
4     ...
```

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

Using **=** instead of **==** when checking for equality

Type 3. **Logical Errors**

- Source of errors: (2) **Incorrect program logic**.
  - This is the most frustrating moment that you may have experienced in previous lab. exercises!
  - 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.
  - Also, **GOOD programming style** helps!!!

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

It's a moth!

Reference: https://www.nationalgeographic.org/thisday/sep9/worlds-first-computer-bug/

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)

```
1  // A program to convert temperature in degree Fahrenheit
2  // to equivalent degrees in Celsius and Kelvin.
3
4  double F, C, K; // Fahrenheit, Celsius, Kelvin
5
6  scanf("%lf", &F);
7
8  C = 5 / 9 * (F - 32);
9
10 K = C + 273.15;
11
12 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.

```
1  // A program to convert temperature in degree Fahrenheit
2  // to equivalent degrees in Celsius and Kelvin.
3
4  double F, C, K; // Fahrenheit, Celsius, Kelvin
5
6  scanf("%lf", &F);
7
8  C = 5 / 9 * (F - 32);
9
10 K = C + 273.15;
11
12 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.

```
1  // A program to convert temperature in degree Fahrenheit
2  // to equivalent degrees in Celsius and Kelvin.
3
4  double F, C, K; // Fahrenheit, Celsius, Kelvin
5
6  scanf("%lf", &F);
7
8  C = 5 / 9 * (F - 32);
9  printf("DEBUG: C = %.2lf\n", C);   // Check C's value
10 K = C + 273.15;
11
12 printf("%.2lfF = %.2lfC = %.2lfK\n", F, C, K);
```

Narrowing Down the Range

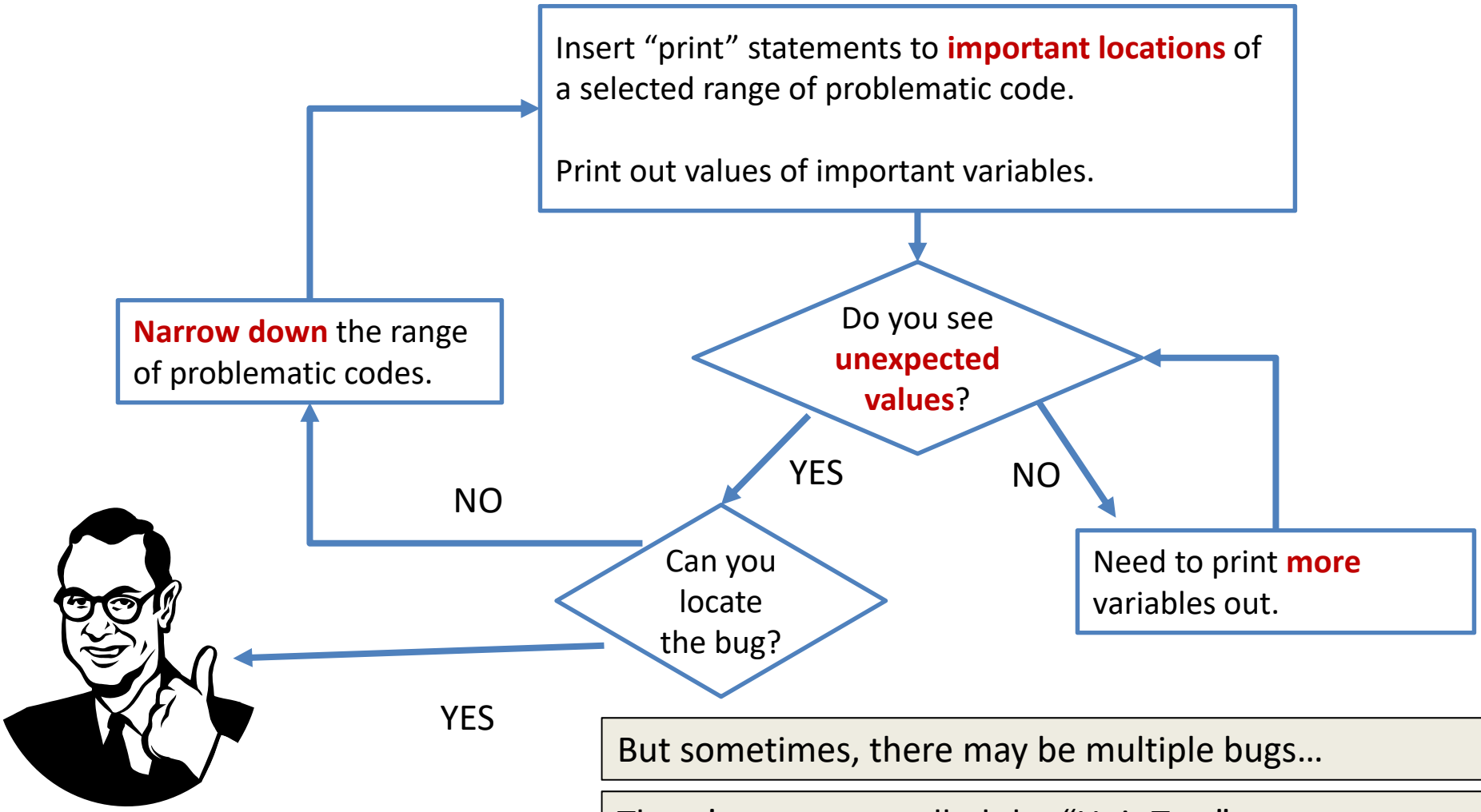
- If you find that you have too many lines of code and not sure where to insert **`printf()`**, you may comment out later parts of the program first

```
// 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



But sometimes, there may be multiple bugs…

There's a concept called the "Unit Test"

Steps in Locating Logical Errors

General hints:

- **Divide and Conquer**:
  - As mentioned in previous slide
- **Unit Test**:
  - Test after you finished even a small piece of code
- **Tools**:
  - Use printf() – your friend
  - Later, learn "debugger" – more powerful friend
- **GOOD programming style** helps!!!
  - https://en.wikibooks.org/wiki/C_Programming/Structure_and_style