

# Chapter 15

## Debugging

Original slides from Gregory Byrd, North  
Carolina State University

Modified by C. Wilcox, Y. Malaiya  
Colorado State University

# Debugging with High Level Languages

- **Same goals as low-level debugging**
  - Examine and set values in memory
  - Execute portions of program
  - Stop execution when (and where) desired
- **Want debugging tools to operate on high-level language constructs**
  - Examine and set variables, not memory locations
  - Trace and set breakpoints on statements and function calls, not instructions
  - ...but also want access to low-level tools when needed

# Types of Errors

## ● Syntactic Errors

- Input code is not legal
- Caught by compiler (or other translation mechanism)

## ● Semantic Errors

- Legal code, but not what programmer intended
- Not caught by compiler, because syntax is correct

## ● Algorithmic Errors

- Problem with the logic of the program
- Program does what programmer intended, but it doesn't solve the right problem

# Syntactic Errors

- Common errors:
  - missing semicolon or brace
  - mis-spelled type in declaration
- One mistake can cause an avalanche of errors
  - because compiler can't recover and gets confused

```
main () {  
    int i  
    int j;  
    for (i = 0; i <= 10; i++) {  
        j = i * 7;  
        printf("%d x 7 = %d\n", i, j);  
    }  
}
```

**missing semicolon**



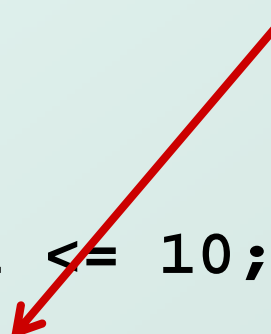
# Semantic Errors

## ● Common Errors

- Missing braces to group statements together
- Confusing assignment with equality
- Wrong assumptions about precedence/associativity
- Wrong limits on for-loop counter
- Uninitialized variables

```
main () {  
    int i  
    int j;  
    for (i = 0; i <= 10; i++)  
        j = i * 7;  
        printf("%d x 7 = %d\n", i, j);  
}
```

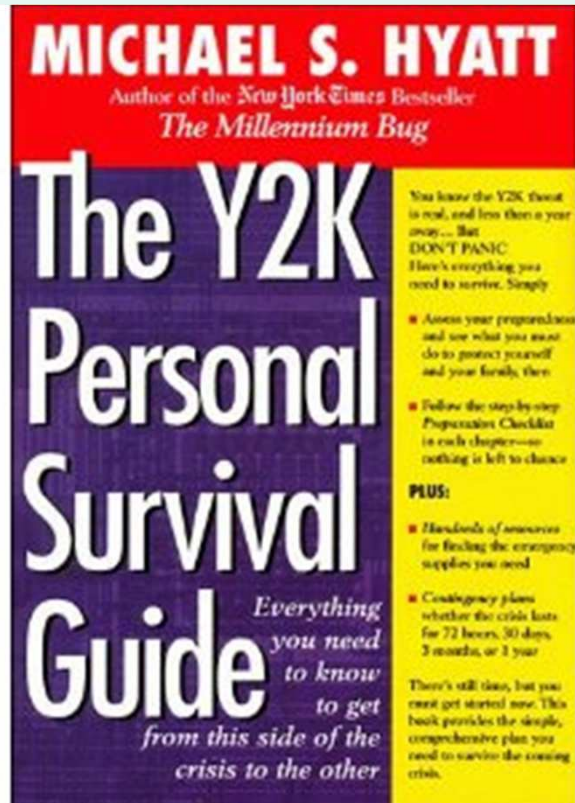
**missing braces,  
so printf not part of if**



# Algorithmic Errors

- Design is wrong, so program does not solve the correct problem
- Difficult to find
  - Program does what we intended
  - Problem might not show up until after many runs
- Maybe difficult to fix
  - May have to redesign
  - May have large impact on program code
- Classic example: Y2K bug
  - only allow 2 digits for year, assuming 19\_\_

# Y2K BUG



# Debugging Techniques

## ● Ad-Hoc

- Insert printf statements to track control flow and display values
- Add code to explicitly check for values out of expected range, incorrect branches, etc.
- Advantage:
  - No special debugging tools needed
- Disadvantages:
  - Frequent recompile and execute cycles makes this method time-consuming
  - Requires intimate knowledge of code
  - Inserted code can be buggy



# Source-Level Debugger

```
1 #include <stdio.h>
2
3 int AllSum(int n);
4
5 int main()
6 {
7     int in;                /* Input value */
8     int sum;              /* Value of 1+2+3+...+n */
9
10    do {
11        printf("Input a number: ");
12        scanf("%d", &in);
13
14    if (in > 0) {
15        sum = AllSum(in);
16        printf("The AllSum of %d is %d\n", in, sum);
17    }
18    } while (in > 0);
19 }
20
21 int AllSum(int n)
22 {
23     int i;                /* Iteration count */
24     int result;          /* Result to be returned */
25
26     for (i=1; i<=n; i++) /* This loop calculates sum */
27         result = result + i;
```

Program stopped at line 14

AllSum.c main SOURCE

main window  
of Cygwin  
version of gdb

# Source-Level Debugging Techniques

## ● Breakpoints

- Stop when a particular statement is reached
- Stop at entry or exit of a function
- **Conditional breakpoints:**  
Stop if a variable is equal to a specific value, etc.
- **Watchpoints:**  
Stop when a variable is set to a specific value

## ● Single-Stepping

- Execute one statement at a time
- Step "into" or step "over" function calls
  - **Step into:** next statement is first inside function call
  - **Step over:** execute function without stopping
  - **Step out:** finish executing function, stop on exit

# Source-Level Debugging Techniques

## ● Displaying Values

- Show value consistent with declared type of variable
- Dereference pointers (variables that hold addresses)
  - See Chapter 16
- Inspect parts of a data structure
  - See Chapters 19