

# Testing & Debugging

- ✱ Fixing Compiler Errors
- ✱ Fixing Run Time Errors
- ✱ Desk Checks
- ✱ Creating a Test Plan
  - Black Box & White Box Testing
  - Categories of test values

CS1A

© Michele Rousseau

Testing & Debugging

1

# Testing & Debugging Strategies



Image: Courtesy of © Christine Jopling

© Michele Rousseau

Testing & Debugging

2

## Testing & Debugging

- Testing is as important as coding
  - Buggy code is costly
- Testing and debugging go together like peas in a pod:
  - Testing finds errors;
  - debugging localizes and repairs them.
  - Together these form the “testing/debugging cycle”: we test, then debug, then repeat.
- Debug then retest!
  - This avoids (reduces) the introduction of new bugs when debugging.

© Michele Rousseau

Testing & Debugging

3

## Fixing Compiler Errors

When you get a compiler error...

- Read the message and think about what it is trying to tell you
  - If it says something like “missing ;”  
Look at the line above
- Fix your errors in order from top to bottom
- If you are getting a “binaries” error or can’t open output file
  - Go to Project → clean
- As a habit you should close your previous projects
  - Right click on the project folder and go to close project

© Michele Rousseau

Testing & Debugging

4

## Make Debugging Easier

- Using proper style helps
  - Making your code more readable makes it easier to find your errors.
  - Little things like spaces between operators and operands
    - Such as... << and >> operators for cout and cin
- Make sure you name your variables with names that make sense

© Michele Rousseau

Testing & Debugging

5

## Fixing Runtime errors

Runtime errors are tricky because the compiler isn’t telling you where the error is or what is causing the problem.

The key to fixing runtime problems is first to figure out which part of the code is causing you issues.

- Try to isolate the problem
  - Use ctrl-/ to comment out code and ctrl-/ to uncomment it
- Insert cerr statements and output your variables to make sure they have the values you expect
  - You can delete them after you’ve found your problem
  - cerr works similarly to cout - except it should flush the output buffer and output in red
- Desk check your code!

© Michele Rousseau

Testing & Debugging

6

## Avoiding runtime errors

- Don't try to sit down and write the whole program at once → then debug.
- Write one section at a time and test it.
  - For example:
 

```
cout << "Enter your annual income: ";
cin >> income;

cout << "Enter your pay increase rate: ";
cin >> increaseRate;

cerr << "TESTING: income: " << income;
cerr << "\tincrease rate: " << increaseRate << endl << endl;
```

© Michele Rousseau

Testing & Debugging

7

## Check Dependent Variables

- If one calculation is dependent upon another check the dependent variables.

```
salesTax = salesTaxRate * retailPrice;
totalPrice = retailPrice + salesTax;
```

```
cerr << "TESTING: sales tax: " << salesTax;
cerr << "\tsales tax rate: " << salesTaxRate;
cerr << "\t retail price: " << retailPrice;
cerr << "\ttotalPrice: " << endl << endl;
```

- Testing Loops
  - Check the LCV and each variable at each iteration
  - USE `cin.ignore(10000, '\n')` ← to hold the cursor
    - This way you trace the code through each iteration

© Michele Rousseau

Testing & Debugging

8

## Code Walkthroughs

- Perform a desk check
  - Write down all the variables used in the code segment
    - At this point it is best to walk through every line of code
  - Don't skip steps
  - Don't assume values
    - if there is no value in your variable and you are using it make sure you are initializing it in the code

© Michele Rousseau

Testing & Debugging

9

## Black Box Testing

- Black box testing
  - You don't see the code
  - Test based on the specifications of the program
  - The purpose is to test the **functionality of the code**
  - Determine the **test cases** and **expected output**
  - Compare the **actual output** to the **expected output**

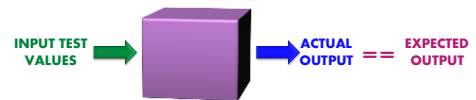


Image: © Michele Rousseau

© Michele Rousseau

Testing & Debugging

10

## Black Box Testing

- Plans should be written ahead of time
- Testing Inputs and outputs
- Example:
  - This program calculates how much pocket money a user has left over. The user gets \$20 a week for spending money.
  - Input
    - Name,
    - previous amount of money and
    - Amount Spent
  - Output
    - Name, Pocket Money

© Michele Rousseau

Testing & Debugging

11

## Testing

- Before you run your code
  - Write a test plan
- Think about the **expected inputs** and **expected outputs** → make a chart

Inputs			Expected Outputs	
Name	prevAmt	amtSpent	Name	Pocket Money
Jean Cyr	12.50	23.00	Jean Cyr	9.50
J.R.	23.57	15.00	J.R.	11.43

© Michele Rousseau

Testing & Debugging

12

## Also think about boundary tests

- Test **expected input**
- Test **boundary values**
  - Try 0 (zero)
  - if you have a selection statement
    - Calculate overtime if hours worked is greater than 40
      - Test these values: 40, 39, 41
- Test for **unexpected inputs**

Inputs			Expected Outputs	
Name	prevAmt	amtSpent	Name	Pocket Money
Jean Cyr	-10.00	23.00	Jean Cyr	-13.00

© Michele Rousseau

Testing & Debugging

13

## White Box Testing

- White Box Testing
  - Look at the code
  - Define your test cases based on the code
  - Testing the **Structure of the code**
    - Internal Behavior
    - Ensure that it meets the design

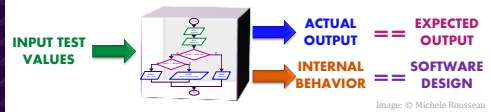


Image: © Michele Rousseau

© Michele Rousseau

Testing & Debugging

14

## White Box Testing

### TEST YOUR CONDITIONAL STATEMENTS

- Test the paths of your code
  - Nested statements
- If statements
  - test true
  - test false
- Loops
  - While loops
  - what happens if first input causes the loop to exit
- Test the boundaries
  - if (hours > 40)
  - Test these values: 40, 39, 41

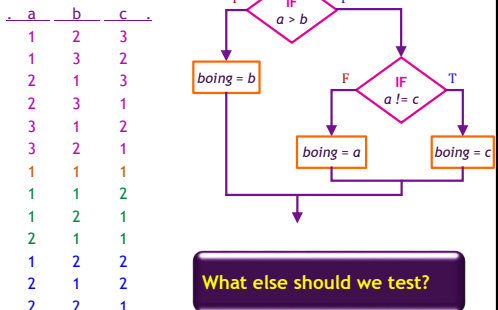
© Michele Rousseau

Testing & Debugging

15

## Testing Selection Statements

- If a, b, & c are integers.
- What do we test?



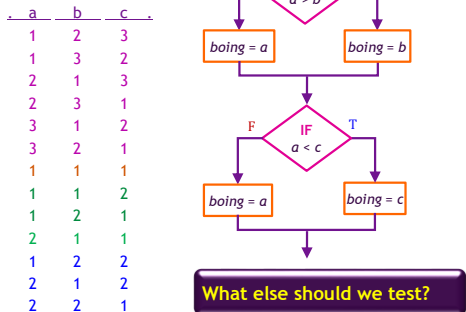
© Michele Rousseau

Testing & Debugging

16

## Testing the Branches

- If a, b, & c are integers.
- What do we test?



© Michele Rousseau

Testing & Debugging

17

## Summary

- Black Box Testing
  - Tests the functionality of the code
  - Think about what the code should do and write test cases before coding
    - What are the expected inputs and outputs
- White Box Testing
  - Tests the structure of the code
  - Think about your loops and selection statements
    - Do your test cases cover all the statements?
    - Do your test cases follow each branch?
- For all testing have test cases that check...
  - Expected inputs
  - Boundary values
  - Unexpected Values

© Michele Rousseau

Testing & Debugging

18