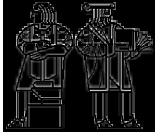


6.170 Lecture 3a Debugging, Again



John Guttag
MIT EECS

6.170 Pruning the Test Data

- Find simplest input that will provoke bug
Usually not the input that revealed existence of bug
- Start with data that revealed bug
Keep paring it down
Often leads directly to bug
- Binary search is a good thing
On data
On program text

6.170 An Interactive Example (Lect3Ex2)

Read in a set of integers and decide which of them are prime numbers

First invocation `java Lect3Ex2`

`arrayIndexOutOfBoundsException`

Try easy thing first `jdb Lect3Ex2` to get location of error

```
public static void main(String[] args) throws IOException {
```

...

```
File inputFile = new File(args[0]);
```

Expecting an argument to main, which I did not supply

6.170 Fix and Try Again

Put in defensive programming code before offending line

```
if (!(args.length == 1)) {
    System.out.println("Wrong number of inputs to Lect3Ex2");
    return;
}
```

`java Lect3Ex3`

Note nice diagnostic

`java Lect3Ex3 primes`

Too much input before error

Elect not to use debugger
Instead think

Formulate a theory based on test and program

Choose data to test theory

6.170 Structure of Program

Read number

Check cache of recently seen numbers

Implemented using a ring buffer

If in cache return with answer

If not in cache

Compute answer

Insert in cache

3	4	9	5	11	2	15	7	8
t	f	f	t	t	t	f	t	f

4

6.170 Structure of Program

Read number

Check cache of recently seen numbers

Implemented using a ring buffer

If in cache return with answer

If not in cache

Compute answer

Insert in cache

3	4	9	5	12	2	15	7	8
t	f	f	t	f	t	f	t	f

5

6.170 **Structure of Program**


Read number

Check cache of recently seen numbers
Implemented using a ring buffer

If in cache return with answer

If not in cache
Compute answer
Insert in cache

3	4	9	5	12	2	15	7	8
t	f	f	t	f	t	f	t	f



Now, look at data and formulate a theory and test

© John Chapin/John Guttag Spring 2000 Slide 7

6.170 **A Few Theories**

Always wrong the second time
Not consistent with existing data
Gets many numbers right the second time through

Gets wrong answer if numbers sufficiently close in list
Because of stale cache entry
Try 3 4 3 4
Refutes hypothesis

Wrong if non-primes sufficiently close in list
Try 4 4
Suggests that cache gets stale for non-primes
Look at relevant code

© John Chapin/John Guttag Spring 2000 Slide 8

6.170 **The Errant Code**

```
public void insertPrime(int i) {
    if (ind == cacheSize) ind = 0;
    members[ind] = i;
    val[ind] = true;
    ind++;
}

public void insertNonPrime(int i) {
    if (ind == cacheSize) ind = 0;
    members[ind] = i;
    ind++;
}
```

If we look at initialization code, we would see that val initialized to false

© John Chapin/John Guttag Spring 2000 Slide 9

6.170 **Fixing Bugs**

Don't rush into anything
Haste makes waste

Does bug explain all observed symptoms?
If not, are other symptoms independent?

Should it be fixed in concert with other changes?
Think about Y2K remediation

What are the ramifications of the proposed "fix?"
Will it break other things?
Does it allow you tidy other things up?
Code should not always grow

Make sure that you can get back to where you are

© John Chapin/John Guttag Spring 2000 Slide 10


6.170 **Key Concepts**

Testing and debugging are different
Testing reveals existence of bugs
Debugging pinpoints location of bugs

Goal is to get program to work
Not to find bugs

Debugging should be a systematic process
Use the "scientific method"

It's important to understand source of bugs
To decide on appropriate repair



© John Chapin/John Guttag Spring 2000 Slide 11