

# Static Detection of Dynamic Memory Errors

David Evans

- ## Outline
- Problem
  - Example (simple)
  - Solution
  - LCLint
  - Solution to Example
  - Analysis
  - Case Studies
  - Conclusion
  - Discussion

10/17/2003

Static Detection of Dynamic Memory Errors

2

## The Problem

- “Many important classes of bugs result from invalid assumptions about the results of functions and the values of parameters and global variables.”
- These bugs cannot be detected efficiently at compile-time
- They may be platform dependent
- Why not use run-time checking?

10/17/2003

Static Detection of Dynamic Memory Errors

3

## Sample Memory Errors

- null pointer misuse
- Lack of memory allocation or deallocation
- Undefined or deallocated storage
- Aliasing

10/17/2003

Static Detection of Dynamic Memory Errors

4

## Example

```
1 extern char *gname;
2
3 void setName (char *pname) {
4     gname = pname;
5 }
6
7 void main() {
8     char x;
9     char *bname;
10    gname = malloc(sizeof(char)*10);
11    bname = "x";
12    setName(bname);
13 }
```

- Memory Leak
  - gname is the only reference to a block of memory
  - That block of memory will be leaked when setName sets gname
- Aliasing
  - gname and bname become aliased after the call to setName
  - bname must not be deallocated if any later code uses gname

10/17/2003

Static Detection of Dynamic Memory Errors

5

## The Solution

- Introduce annotations to make assumptions explicit
- Extend LCLint to statically detect the errors

10/17/2003

Static Detection of Dynamic Memory Errors

6

## LCLint

- Background: MIT, UVA
- Splint – Secure Programming Lint
- Not just for memory error detection
- The process:
  - Checks each procedure individually
  - Outputs a warning message indicating the function that caused the anomaly

10/17/2003

Static Detection of Dynamic  
Memory Errors

7

## Annotations

- Express assumptions
- “Stylized comments”
- Must be placed in one of the following:
  - Type declaration
  - Function parameter
  - Function return value
  - Global and static declarations

10/17/2003

Static Detection of Dynamic  
Memory Errors

8

## Solution to Example

```
1 extern char *pname;  
2  
3 void setName (/*@null@*/ char *pname) {  
4     pname = pname;  
5 }
```

- The null annotation indicates that pname may have the value NULL
- At the exit point, there are still errors:
  - pname could be assigned NULL in the function
  - The function is not declared as having a possible NULL return

10/17/2003

Static Detection of Dynamic  
Memory Errors

9

## Other Solutions

```
1 extern char *pname;  
2  
3 void setName (char /*@null@*/ *pname)  
4 {  
5     pname = pname;  
6 }
```

- Remove null annotation from char \*pname
  - setName can still be called with a possibly null value
- Add a null annotation to pname
  - pname must be checked for null at every dereferencing

10/17/2003

Static Detection of Dynamic  
Memory Errors

10

## Another Solution

```
extern char *pname  
extern (/*@truenull@*/  
isNull (/*@null@*/ char *x));  
void setName (/*@null@*/ char *pname)  
{  
    if (!isNull (pname)) {pname = pname;}  
}
```

- A truenull function returns true if the input is NULL
- isNull checks to see if pname is NULL
- pname = pname executes only for non-null values

10/17/2003

Static Detection of Dynamic  
Memory Errors

11

## Case Study #1

- Simple employee database program
  - 1000 lines of code, 300 lines of interface specs
- Started with no annotations
- Applied annotations by adjusting default interpretations in LCLint

10/17/2003

Static Detection of Dynamic  
Memory Errors

12

## Case Study #1

- 15 annotations were needed to resolve the detected anomalies
  - 13 of these were only annotations
  - Only two would have been necessary if implicit annotations had been used
- With minimal effort in adding annotations:
  - Errors were found
  - Documentation was improved

10/17/2003

Static Detection of Dynamic  
Memory Errors

13

## Case Study #2

- LCLint
  - 100,000 lines of code
- Initial run produced ~1000 messages
- Bugs found
- New features added
- Memory annotations allowed efficiency improvements too risky to use without them
- Documentation was much improved

10/17/2003

Static Detection of Dynamic  
Memory Errors

14

## Case Study #2

- The entire program took fewer than four minutes to check
- Adding all the annotations required a few days over the course of a few weeks
- Adding the annotations was a methodical process, similar to iterating

10/17/2003

Static Detection of Dynamic  
Memory Errors

15

## Case Study #2

- Problem with spurious messages
- LCLint overestimated the importance of some anomalies
- Danger of suppressing messages

10/17/2003

Static Detection of Dynamic  
Memory Errors

16

## Conclusions

- Annotations are a quick and easy method of finding bugs
  - A 5000 line module was checked in under 10 seconds
- Annotations help find bugs other methods miss
- Annotations improve documentation
- Annotations, like typing, can be strict or relaxed

10/17/2003

Static Detection of Dynamic  
Memory Errors

17

## Conclusions

- The process is systematic
- Static checking is meant to complement run-time checking
- Annotations can also be added as a program is being developed

10/17/2003

Static Detection of Dynamic  
Memory Errors

18

## Discussion

- Can assertions be used in place of annotations?
- Compare annotations to types for variables.
- Are annotations the same as comments?