Testing, Profiling and Instrumentation

CS 217
Testing, Profiling, and Instrumentation

• How do you know if your program is correct?
  ◦ Will it ever core dump?
  ◦ Does it ever produce the wrong answer?
    – Testing

• How do you know what your program is doing?
  ◦ How fast is your program?
  ◦ Why is it slow for one input but not for another?
  ◦ Does it have a memory leak?
    – Timing
    – Profiling
    – Instrumentation
Program Verification

• How do you know if your program is correct?
  ◦ Can you **prove** that it is correct?
  ◦ Can you **prove** properties of the code?
    – e.g., it terminates
Program Testing

- Convince yourself that your program probably works

How do you write a test program?

Specification → Test Program → Probably Right/Wrong

symtable.c
Test Programs

• Properties of a good test program
  ◦ Tests boundary conditions
  ◦ Exercise as much code as possible
  ◦ Produce output that is known to be right/wrong

How do you achieve all three properties?
Test Boundary Conditions

• Most bugs occur at boundary conditions
  ○ What is the most common boundary condition bug?

• What are the boundary conditions of this code?
  ```
  int I;
  char s[MAXLINE];

  for ( i=0; (s[i] = getchar()) != '\n' && I < MAXLINE-1; i++ )
    ;
  s[--i] = '\0';
  ```

• Boundary conditions
  ○ Input starts with \n
  ○ End of file
Test Boundary Condition, cont’d

• Rewrite the code
  
  ```c
  for ( i=0; i<MAXLINE-1; i++)
    if ((s[i] = getchar()) == ‘\n’)    
      break;
  S[i] = ‘\0’;
  ```

• Another boundary condition: EOF
  
  ```c
  for ( i=0; i<MAXLINE-1; i++)
    if ((s[i] = getchar()) == ‘\n’ || s[i] == EOF)
      break;
  S[i] = ‘\0’;
  ```

• What are other boundary conditions?
  - Nearly full
  - Exactly full
  - Over full
Test As Your Write Code

• Recall using “assert” in previous lecture
• Check pre- and post-conditions for each function
  ◦ Boundary conditions
• Check invariants
• Check error returns
• What is the typical percentage of code doing error-checking?
Systematic Testing

• Test plan
  ◦ Unit tests (for each module)
  ◦ System tests

• Design test cases
  ◦ Know what input gives what output, according to the spec
  ◦ Know conservation properties
  ◦ Compare independent implementations
  ◦ What legal inputs to test
    – Boundary conditions and inductions
    – Multi-dimensional inputs and combinations
      • Assembler: instructions, comments, directives
      • Numerical program: operations, legal combinations
  ◦ What illegal inputs to test
    – Common illegal inputs
    – Possible security holes
    – Multi-dimensional illegal inputs
A Test Case Example

• “De-comment” test
  ◦ Test single line comment
  ◦ Test very long line
  ◦ Multiple line comment
  ◦ Test many lines
  ◦ Nested comment
  ◦ String literal in comment
  ◦ Character literal in comment
  ◦ Comment in string literal
  ◦ Comment in character literal
  ◦ Unterminated comment
  ◦ . . .
Test Automation

• Automation can provide better test coverage

• Test program
  ◦ Client code to test modules
  ◦ Scripts to test inputs and compare outputs

• QA test is an iterative process
  ◦ Initial automated test program or scripts
  ◦ Test simple parts first
  ◦ Unit tests before system tests
  ◦ Add tests as new cases created

• Regression test
  ◦ Test all cases to compare the new version with the previous one
  ◦ A bug fix often create new bugs in a large software system

• What tests cannot be done automatically?
Stress Tests

• Motivations
  ◦ Use computer to generate inputs to test
  ◦ High-volume tests often find bugs

• What to generate
  ◦ Very long inputs
  ◦ Random inputs (binary vs. ASCII)
  ◦ Fault injection

• How much test
  ◦ Exercise all data paths
  ◦ Test all error conditions
  ◦ Run whenever possible
Who Test What

• Implementers
  ◦ White-box testing
  ◦ Pros: An implementer knows all data paths
  ◦ Cons: An implementer tests the same way

• Quality Assurance (QA) engineers
  ◦ Black-box testing
  ◦ Pros: No knowledge about the implementation
  ◦ Cons: Unlikely to test all data paths

• Customers
  ◦ Field test
  ◦ Pros: Unexpected ways of using the software
  ◦ Cons: No enough cases
Timing, Profiling, and Instrumentation

- How do you know what your code is doing?
  - How slow is it?
    - How long does it take for certain types of inputs?
  - Where is it slow?
    - Which code is being executed most?
  - Why am I running out of memory?
    - Where is the memory going?
    - Are there leaks?
  - Why is it slow?
    - How imbalanced is my binary tree?
Timing

• Most shells provide tool to time program execution
  ◦ e.g., bash "time" command

```
bash> tail -1000 /usr/lib/dict/words > input.txt
bash> time sort5.pixie < input.txt > output.txt
real 0m12.977s
user 0m12.860s
sys  0m0.010s
```
Timing

• Most operating systems provide a way to get the time
  ◦ e.g., UNIX “gettimeofday” command

```c
#include <sys/time.h>

struct timeval start_time, end_time;

gmtimeofday(&start_time, NULL);
  <execute some code here>
gettimeofday(&end_time, NULL);

float seconds = end_time.tv_sec - start_time.tv_sec +
  1.0E-6F * (end_time.tv_usec - start_time.tv_usec);
```
Timing

- Some CPU provides access to CPU “ticks”

```c
unsigned long long int getCPUTicks(void)
{
    unsigned long long int x;

    asm volatile (".byte 0x0f, 0x31":"=A" (x));
    return x;
}
```
Profiling

• Gather statistics about your program’s execution
  ◦ How much time did execution of a function take?
  ◦ How many times was a particular function called?
  ◦ How many times was a particular line of code executed?
  ◦ Which lines of code used the most time?

• Most compilers come with profilers
  ◦ e.g., pixie and prof
Profiling with gcc+gprof

• Apparently, `prof` doesn't work with `gcc`, must use `gprof`

```
PROFFLAGS = -Wall -ansi -pedantic -O4 -NDEBUG -pg

CFLAGS= ${PROFFLAGS}

profile: player testinput
    -player MIN <testinput
    gprof player >profile
```

`minus sign means “keep going even if errors”`

```
player: player.c minimax.c gamestate.c ... 
gcc ${CFLAGS} player.c minimax.c ...
```
% make profile

gcc -Wall -O4 -DNDEBUG -pg -o player . . .

player MIN  <testinput

5
  12 11 10  9  8  7
|-------------------------||     4  4  4  5  5  5    ||  0                    1 ||     4  4  4  4  4  0    ||-------------------------|
  0  1  2  3  4  5

MIN player reports invalid move by MAX player

make: *** [profile] Error 1 (ignored)

gprof player >profile
First part of gprof profile looks like this; it's for sophisticated users (i.e. more sophisticated than your humble professor) and I will ignore it.
## Format of gprof profile

<table>
<thead>
<tr>
<th>% cumulative</th>
<th>self seconds</th>
<th>calls</th>
<th>self ms/call</th>
<th>total ms/call</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>57.1</td>
<td>12.97</td>
<td></td>
<td></td>
<td></td>
<td>internal_mcount [1]</td>
</tr>
<tr>
<td>4.8</td>
<td>14.05</td>
<td>5700352</td>
<td>0.00</td>
<td>0.00</td>
<td>_free_unlocked [12]</td>
</tr>
<tr>
<td>4.4</td>
<td>15.04</td>
<td></td>
<td></td>
<td></td>
<td>__mcount (693)</td>
</tr>
<tr>
<td>3.5</td>
<td>15.84</td>
<td>22801464</td>
<td>0.00</td>
<td>0.00</td>
<td>_return_zero [16]</td>
</tr>
<tr>
<td>2.8</td>
<td>16.48</td>
<td>5700361</td>
<td>0.00</td>
<td>0.00</td>
<td>__umul [18]</td>
</tr>
<tr>
<td>2.8</td>
<td>17.11</td>
<td>747130</td>
<td>0.01</td>
<td></td>
<td>GameStateExpandMove [6]</td>
</tr>
<tr>
<td>2.5</td>
<td>17.67</td>
<td>5700361</td>
<td>0.00</td>
<td>0.00</td>
<td>malloc [7]</td>
</tr>
<tr>
<td>2.1</td>
<td>18.14</td>
<td>11400732</td>
<td>0.00</td>
<td>0.00</td>
<td>mutex_unlock [14]</td>
</tr>
<tr>
<td>1.9</td>
<td>18.58</td>
<td>11400732</td>
<td>0.00</td>
<td>0.00</td>
<td>mutex_lock [15]</td>
</tr>
<tr>
<td>1.9</td>
<td>19.01</td>
<td>5700361</td>
<td>0.00</td>
<td>0.00</td>
<td>_memset [22]</td>
</tr>
<tr>
<td>1.9</td>
<td>19.44</td>
<td>1</td>
<td>430.00</td>
<td>430.00</td>
<td>.div [21]</td>
</tr>
<tr>
<td>1.8</td>
<td>19.85</td>
<td>5157853</td>
<td>0.00</td>
<td>0.00</td>
<td>cleanfree [19]</td>
</tr>
<tr>
<td>1.4</td>
<td>20.17</td>
<td>5700366</td>
<td>0.00</td>
<td>0.00</td>
<td>_malloc_unlocked [13]</td>
</tr>
<tr>
<td>1.4</td>
<td>20.49</td>
<td>5700362</td>
<td>0.00</td>
<td>0.00</td>
<td>malloc [8]</td>
</tr>
<tr>
<td>1.3</td>
<td>20.79</td>
<td>5157847</td>
<td>0.00</td>
<td>0.00</td>
<td>_smalloc [24]</td>
</tr>
<tr>
<td>1.2</td>
<td>21.06</td>
<td>1</td>
<td>45.00</td>
<td>1386.66</td>
<td>minimax [5]</td>
</tr>
<tr>
<td>1.1</td>
<td>21.31</td>
<td>4755325</td>
<td>0.00</td>
<td>0.00</td>
<td>Delta_free [10]</td>
</tr>
<tr>
<td>1.0</td>
<td>21.54</td>
<td>5700352</td>
<td>0.00</td>
<td>0.00</td>
<td>free [9]</td>
</tr>
<tr>
<td>1.0</td>
<td>21.77</td>
<td>747130</td>
<td>0.00</td>
<td>0.00</td>
<td>GameState_applyDeltas [25]</td>
</tr>
<tr>
<td>1.0</td>
<td>21.99</td>
<td>5157845</td>
<td>0.00</td>
<td>0.00</td>
<td>realfree [26]</td>
</tr>
<tr>
<td>1.0</td>
<td>22.21</td>
<td>747129</td>
<td>0.00</td>
<td>0.00</td>
<td>GameState_unApplyDeltas [27]</td>
</tr>
<tr>
<td>0.5</td>
<td>22.32</td>
<td>2300787</td>
<td>0.00</td>
<td>0.00</td>
<td>.udiv [29]</td>
</tr>
<tr>
<td>0.4</td>
<td>22.42</td>
<td>5700363</td>
<td>0.00</td>
<td>0.00</td>
<td>_umul [18]</td>
</tr>
<tr>
<td>0.4</td>
<td>22.52</td>
<td>1698871</td>
<td>0.00</td>
<td>0.00</td>
<td>GameState_getPlayer [30]</td>
</tr>
<tr>
<td>0.4</td>
<td>22.61</td>
<td>747135</td>
<td>0.00</td>
<td>0.00</td>
<td>GameState_getStatus [31]</td>
</tr>
<tr>
<td>0.3</td>
<td>22.68</td>
<td>204617</td>
<td>0.00</td>
<td>0.00</td>
<td>GameState_genMoves [17]</td>
</tr>
<tr>
<td>0.1</td>
<td>22.70</td>
<td>945027</td>
<td>0.00</td>
<td>0.00</td>
<td>Move_free [23]</td>
</tr>
<tr>
<td>0.0</td>
<td>22.71</td>
<td>542509</td>
<td>0.00</td>
<td>0.00</td>
<td>GameState_getValue [32]</td>
</tr>
<tr>
<td>0.0</td>
<td>22.71</td>
<td>104</td>
<td>0.00</td>
<td>0.00</td>
<td>__ferror_unlocked [357]</td>
</tr>
<tr>
<td>0.0</td>
<td>22.71</td>
<td>64</td>
<td>0.00</td>
<td>0.00</td>
<td>_realbufend [358]</td>
</tr>
<tr>
<td>0.0</td>
<td>22.71</td>
<td>54</td>
<td>0.00</td>
<td>0.00</td>
<td>_nvmatch [60]</td>
</tr>
<tr>
<td>0.0</td>
<td>22.71</td>
<td>52</td>
<td>0.00</td>
<td>0.00</td>
<td>_doprint [42]</td>
</tr>
<tr>
<td>0.0</td>
<td>22.71</td>
<td>51</td>
<td>0.00</td>
<td>0.00</td>
<td>memchr [61]</td>
</tr>
<tr>
<td>0.0</td>
<td>22.71</td>
<td>51</td>
<td>0.00</td>
<td>0.00</td>
<td>printf [43]</td>
</tr>
<tr>
<td>0.0</td>
<td>22.71</td>
<td>13</td>
<td>0.00</td>
<td>0.00</td>
<td>_write [359]</td>
</tr>
<tr>
<td>0.0</td>
<td>22.71</td>
<td>10</td>
<td>0.00</td>
<td>0.00</td>
<td>_xFIobuf [360]</td>
</tr>
<tr>
<td>0.0</td>
<td>22.71</td>
<td>7</td>
<td>0.00</td>
<td>0.00</td>
<td>memccpy [361]</td>
</tr>
<tr>
<td>0.0</td>
<td>22.71</td>
<td>4</td>
<td>0.00</td>
<td>0.00</td>
<td>_mul [62]</td>
</tr>
<tr>
<td>0.0</td>
<td>22.71</td>
<td>4</td>
<td>0.00</td>
<td>0.00</td>
<td>__errno [362]</td>
</tr>
<tr>
<td>0.0</td>
<td>22.71</td>
<td>4</td>
<td>0.00</td>
<td>0.00</td>
<td>_fflush_u [363]</td>
</tr>
<tr>
<td>0.0</td>
<td>22.71</td>
<td>3</td>
<td>0.00</td>
<td>0.00</td>
<td>GameState_playerToStr [63]</td>
</tr>
<tr>
<td>0.0</td>
<td>22.71</td>
<td>3</td>
<td>0.00</td>
<td>0.00</td>
<td>__findbuf [41]</td>
</tr>
</tbody>
</table>

**Second part of profile looks like this; it's the simple (i.e., useful) part; corresponds to the "prof" tool**
Don’t even **think** of optimizing these

<table>
<thead>
<tr>
<th>% cumulative</th>
<th>self time</th>
<th>self seconds</th>
<th>calls</th>
<th>ms/call</th>
<th>total ms/call</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>57.1</td>
<td>12.97</td>
<td>12.97</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.8</td>
<td>14.05</td>
<td>1.08</td>
<td>5700352</td>
<td>0.00</td>
<td>0.00</td>
<td>internal_mcount [1]</td>
</tr>
<tr>
<td>4.4</td>
<td>15.04</td>
<td>0.99</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.5</td>
<td>15.84</td>
<td>0.80</td>
<td>22801464</td>
<td>0.00</td>
<td>0.00</td>
<td>_free_unlocked [12]</td>
</tr>
<tr>
<td>2.8</td>
<td>16.48</td>
<td>0.64</td>
<td>5700361</td>
<td>0.00</td>
<td>0.00</td>
<td>_mcount [693]</td>
</tr>
<tr>
<td>2.8</td>
<td>17.11</td>
<td>0.63</td>
<td>747130</td>
<td>0.00</td>
<td>0.00</td>
<td>GameState_expandMove [6]</td>
</tr>
<tr>
<td>2.6</td>
<td>17.67</td>
<td>0.56</td>
<td>5700361</td>
<td>0.00</td>
<td>0.00</td>
<td>calloc [7]</td>
</tr>
<tr>
<td>2.1</td>
<td>18.14</td>
<td>0.47</td>
<td>11400732</td>
<td>0.00</td>
<td>0.00</td>
<td>_mutex_unlock [14]</td>
</tr>
<tr>
<td>1.9</td>
<td>18.58</td>
<td>0.44</td>
<td>11400732</td>
<td>0.00</td>
<td>0.00</td>
<td>_return_zero [16]</td>
</tr>
<tr>
<td>1.9</td>
<td>19.01</td>
<td>0.43</td>
<td>5700361</td>
<td>0.00</td>
<td>0.00</td>
<td>_memset [22]</td>
</tr>
<tr>
<td>1.9</td>
<td>19.44</td>
<td>0.43</td>
<td></td>
<td>430.00</td>
<td>430.00</td>
<td>.div [21]</td>
</tr>
<tr>
<td>1.8</td>
<td>19.85</td>
<td>0.41</td>
<td>5157853</td>
<td>0.00</td>
<td>0.00</td>
<td>cleanfree [19]</td>
</tr>
<tr>
<td>1.4</td>
<td>20.17</td>
<td>0.32</td>
<td>5700366</td>
<td>0.00</td>
<td>0.00</td>
<td>malloc_unlocked &lt;cycle 1&gt; [13]</td>
</tr>
<tr>
<td>1.4</td>
<td>20.49</td>
<td>0.32</td>
<td>5700362</td>
<td>0.00</td>
<td>0.00</td>
<td>_malloc [78]</td>
</tr>
<tr>
<td>1.3</td>
<td>20.79</td>
<td>0.30</td>
<td>5157847</td>
<td>0.00</td>
<td>0.00</td>
<td>malloc &lt;cycle 1&gt; [24]</td>
</tr>
<tr>
<td>1.2</td>
<td>21.06</td>
<td>0.27</td>
<td></td>
<td>1386.66</td>
<td>1386.66</td>
<td>_Minimax [5]</td>
</tr>
<tr>
<td>1.1</td>
<td>21.31</td>
<td>0.25</td>
<td>4755325</td>
<td>0.00</td>
<td>0.00</td>
<td>Delta_free [10]</td>
</tr>
<tr>
<td>1.0</td>
<td>21.54</td>
<td>0.23</td>
<td>5700352</td>
<td>0.00</td>
<td>0.00</td>
<td>free [9]</td>
</tr>
<tr>
<td>1.0</td>
<td>21.77</td>
<td>0.23</td>
<td>747130</td>
<td>0.00</td>
<td>0.00</td>
<td>GameState_applyDeltas [25]</td>
</tr>
<tr>
<td>1.0</td>
<td>21.99</td>
<td>0.22</td>
<td>5157845</td>
<td>0.00</td>
<td>0.00</td>
<td>realloc_T26</td>
</tr>
<tr>
<td>1.0</td>
<td>22.21</td>
<td>0.22</td>
<td>747129</td>
<td>0.00</td>
<td>0.00</td>
<td>GameState_unApplyDeltas [27]</td>
</tr>
<tr>
<td>0.5</td>
<td>22.32</td>
<td>0.11</td>
<td>2360787</td>
<td>0.00</td>
<td>0.00</td>
<td>.rem [28]</td>
</tr>
<tr>
<td>0.4</td>
<td>22.42</td>
<td>0.10</td>
<td>5700363</td>
<td>0.00</td>
<td>0.00</td>
<td>.udiv [29]</td>
</tr>
<tr>
<td>0.4</td>
<td>22.52</td>
<td>0.10</td>
<td>1698871</td>
<td>0.00</td>
<td>0.00</td>
<td>GameState_getPlayer [30]</td>
</tr>
<tr>
<td>0.4</td>
<td>22.61</td>
<td>0.09</td>
<td>747135</td>
<td>0.00</td>
<td>0.00</td>
<td>GameState_getStatus [31]</td>
</tr>
<tr>
<td>0.3</td>
<td>22.68</td>
<td>0.07</td>
<td>204617</td>
<td>0.00</td>
<td>0.00</td>
<td>GameState_genMoves [17]</td>
</tr>
<tr>
<td>0.1</td>
<td>22.70</td>
<td>0.02</td>
<td>945027</td>
<td>0.00</td>
<td>0.00</td>
<td>Move_free [23]</td>
</tr>
<tr>
<td>0.0</td>
<td>22.71</td>
<td>0.01</td>
<td>542509</td>
<td>0.00</td>
<td>0.00</td>
<td>GameState_getValue [32]</td>
</tr>
<tr>
<td>0.0</td>
<td>22.71</td>
<td>0.00</td>
<td>104</td>
<td>0.00</td>
<td>0.00</td>
<td>_ferror_unlocked [357]</td>
</tr>
<tr>
<td>0.0</td>
<td>22.71</td>
<td>0.00</td>
<td>4</td>
<td>0.00</td>
<td>0.00</td>
<td>_thr_main [367]</td>
</tr>
<tr>
<td>0.0</td>
<td>22.71</td>
<td>0.00</td>
<td>3</td>
<td>0.00</td>
<td>0.00</td>
<td>GameState_playerToStr [63]</td>
</tr>
<tr>
<td>0.0</td>
<td>22.71</td>
<td>0.00</td>
<td>2</td>
<td>0.00</td>
<td>0.00</td>
<td>strcmp [66]</td>
</tr>
<tr>
<td>0.0</td>
<td>22.71</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>GameState_getSearchDepth [67]</td>
</tr>
<tr>
<td>0.0</td>
<td>22.71</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>GameState_new [37]</td>
</tr>
<tr>
<td>0.0</td>
<td>22.71</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>GameState_playerFromStr [68]</td>
</tr>
<tr>
<td>0.0</td>
<td>22.71</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>GameState_write [44]</td>
</tr>
<tr>
<td>0.0</td>
<td>22.71</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>Move_isValid [36]</td>
</tr>
<tr>
<td>0.0</td>
<td>22.71</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>Move_read [59]</td>
</tr>
<tr>
<td>0.0</td>
<td>22.71</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>check_nlspath_env [46]</td>
</tr>
<tr>
<td>0.0</td>
<td>22.71</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>clock [20]</td>
</tr>
<tr>
<td>0.0</td>
<td>22.71</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>exit [53]</td>
</tr>
<tr>
<td>0.0</td>
<td>22.71</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>main [3]</td>
</tr>
<tr>
<td>0.0</td>
<td>22.71</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>mem_init [70]</td>
</tr>
<tr>
<td>0.0</td>
<td>22.71</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>number [71]</td>
</tr>
<tr>
<td>0.0</td>
<td>22.71</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>scanf [53]</td>
</tr>
</tbody>
</table>
Instrumentation

• Gather statistics about your data structures
  ◦ e.g., how many nodes are at each level of my binary tree?
  ◦ e.g., how many elements are in each bucket of my hash table?
  ◦ e.g., how much memory is allocated from the heap?

2, 1, 4, 3, 6, 5, 8, 7, 10, 9, 11
Instrumentation Example

static void Tree_FillHistogram(Tree_T oTree, TreeNode_T oNode, int *ipHistogram, int iLevel, int iMaxLevels)
{
    int i;

    /* Increment histogram entry */
    ipHistogram[iLevel]++;

    /* Recurse to children */
    if (iLevel < iMaxLevels)
        for (i = 0; i < oNode->nchildren; i++)
            Tree_FillHistogram(oTree, oNode->child[i], iLevel+1, iMaxLevels);
}

void Tree_PrintHistogram(Tree_T oTree, FILE *fp)
{
    /* Define histogram */
    int ipHistogram[MAX_LEVELS];
    int i;

    /* Load histogram recursively */
    Tree_FillHistogram(oTree, oTree->root, ipHistogram, 0, MAX_LEVELS);

    /* Print histogram */
    for (i = 0; i < MAX_LEVELS; i++)
        fprintf(fp, "%d ", ipHistogram[i];
    fprintf(fp, "\n");
}
typedef struct Hash *Hash_T;

struct Hash {
    Set_T *buckets;
    int nbuckets;
};

void Hash_PrintBucketCounts(Hash_T oHash, FILE *fp)
{
    int i;

    /* Print number of elements in each bucket */
    for (i = 0; i < oHash->nbuckets; i++)
        fprintf(fp, "%d ", Set_getLength(oHash->buckets[i]), fp);
    fprintf(fp, "\n");
}
### Interate...

1. Develop program
2. Test; modify program
3. Test again; if bugs, back to step 2
4. Is it fast enough? If not,
5. Profile; modify program; back to step 3

- Typically, reprofile several times until no more performance improvement is justified
Summary & Guidelines

• Test your code as you write it
  ○ It is very hard to debug a lot of code all at once
  ○ Isolate modules and test them independently
  ○ Design your tests to cover boundary conditions
  ○ Test modules bottom-up

• Instrument your code as you write it
  ○ Include asserts and verify data structure sanity often
  ○ Include debugging statements (e.g., #ifdef DEBUG and #endif)
  ○ You’ll be surprised what your program is really doing!!!

• Time and profile your code only when you are done
  ○ Don’t optimize code unless you have to (you almost never will)
  ○ Fixing your algorithm is almost always the solution
  ○ Otherwise, running optimizing compiler is usually enough