Testing, Profiling and Instrumentation

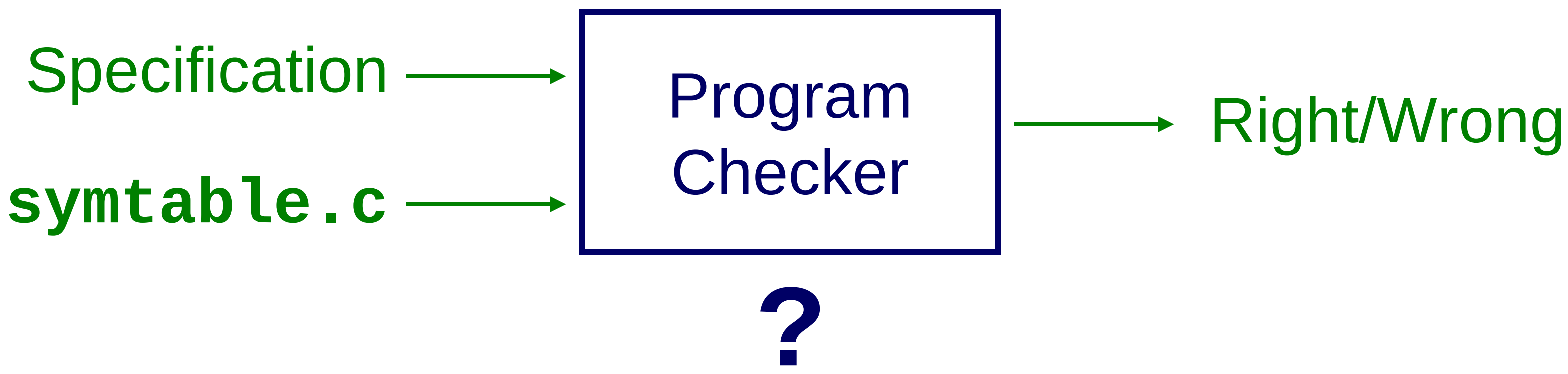
CS 217

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?

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

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

- Another boundary condition: EOF

```
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

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

struct timeval start_time, end_time;

gettimeofday(&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”

```
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 ...
```

Profiled execution

```
% 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
```

Format of gprof profile

```
                                  called/total       parents
index  %time    self descendents  called+self    name           index
                                  called/total       children

                                                     <spontaneous>
[1]     59.7   12.97        0.00                 internal_mcount [1]
                0.00        0.00       1/3           atexit [35]
-----------------------------------------------
                                                     <spontaneous>
[2]     40.3    0.00        8.75                 _start [2]
                0.00        8.75       1/1           main [3]
                0.00        0.00       2/3           atexit [35]
-----------------------------------------------
                0.00        8.75       1/1           _start [2]
[3]     40.3    0.00        8.75       1         main [3]
                0.00        8.32       1/1           getBestMove [4]
                0.00        0.43       1/1           clock [20]
                0.00        0.00       1/747130      GameState_expandMove [6]
                0.00        0.00       1/1           exit [33]
                0.00        0.00       1/1           Move_read [36]
                0.00        0.00       1/1           GameState_new [37]
                0.00        0.00       6/747135      GameState_getStatus [31]
                0.00        0.00       1/747130      GameState_applyDeltas [25]
                0.00        0.00       1/1           GameState_write [44]
                0.00        0.00       1/1           sscanf [54]
                0.00        0.00       1/1698871     GameState_getPlayer [30]
                0.00        0.00       1/1           _fprintf [58]
                0.00        0.00       1/1           Move_write [59]
                0.00        0.00       3/3           GameState_playerToStr [63]
                0.00        0.00       1/2           strcmp [66]
                0.00        0.00       1/1           GameState_playerFromStr [68]
                0.00        0.00       1/1           Move_isValid [69]
                0.00        0.00       1/1           GameState_getSearchDepth [67]
-----------------------------------------------
                0.00        8.32       1/1           main [3]
[4]     38.3    0.00        8.32       1         getBestMove [4]
                0.27        8.05       6/6           minimax [5]
                0.00        0.00       6/747130      GameState_expandMove [6]
                0.00        0.00      35/4755325     Delta_free [10]
                0.00        0.00       1/204617      GameState_genMoves [17]
                0.00        0.00       5/945027      Move_free [23]
                0.00        0.00       6/747130      GameState_applyDeltas [25]
                0.00        0.00       6/747129      GameState_unApplyDeltas [27]
                0.00        0.00       2/1698871     GameState_getPlayer [30]
-----------------------------------------------
                                  747123             minimax [5]
                0.27        8.05       6/6           getBestMove [4]
[5]     38.3    0.27        8.05       6+747123  minimax [5]
                0.63        3.56  747123/747130      GameState_expandMove [6]
                0.25        1.82 4755290/4755325     Delta_free [10]
                0.07        0.69  204616/204617      GameState_genMoves [17]
                0.02        0.36  945022/945027      Move_free [23]
                0.23        0.00  747123/747130      GameState_applyDeltas [25]
                0.22        0.00  747123/747129      GameState_unApplyDeltas [27]
                0.10        0.00 1698868/1698871     GameState_getPlayer [30]
                0.09        0.00  747129/747135      GameState_getStatus [31]
                0.01        0.00  542509/542509      GameState_getValue [32]
                                  747123             minimax [5]
-----------------------------------------------
                0.00        0.00       1/747130      main [3]
                0.00        0.00       6/747130      getBestMove [4]
                0.63        3.56  747123/747130      minimax [5]
[6]     19.3    0.63        3.56  747130         GameState_expandMove [6]
                0.47        2.99 4755331/5700361     calloc [7]
                0.11        0.00 2360787/2360787     .rem [28]
-----------------------------------------------
                0.00        0.00       1/5700361     Move_read [36]
                0.00        0.00       1/5700361     GameState_new [37]
                0.09        0.59  945028/5700361     GameState_genMoves [17]
                0.47        2.99 4755331/5700361     GameState_expandMove [6]
[7]     19.1    0.56        3.58 5700361         calloc [7]
                0.32        2.09 5700361/5700362     malloc [8]
                0.64        0.00 5700361/5700361     .umul [18]
                0.43        0.00 5700361/5700361     _memset [22]
                0.10        0.00 5700361/5700363     .udiv [29]
-----------------------------------------------
                0.00        0.00       1/5700362     _findbuf [41]
                0.32        2.09 5700361/5700362     calloc [7]
[8]     11.1    0.32        2.09 5700362         malloc [8]
                0.62        0.62 5700362/5700362     _malloc_unlocked <cycle 1> [13]
                0.23        0.20 5700362/11400732    _mutex_unlock [14]
                0.22        0.20 5700362/11400732    mutex_lock [15]
```

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

```
   %  cumulative    self              self    total
 time   seconds   seconds    calls  ms/call  ms/call name
 57.1     12.97    12.97                             internal_mcount [1]
  4.8     14.05     1.08  5700352     0.00     0.00  _free_unlocked [12]
  4.4     15.04     0.99                             _mcount (693)
  3.5     15.84     0.80 22801464     0.00     0.00  _return_zero [16]
  2.8     16.48     0.64  5700361     0.00     0.00  .umul [18]
  2.8     17.11     0.63   747130     0.00     0.01  GameState_expandMove [6]
  2.5     17.67     0.56  5700361     0.00     0.00  calloc [7]
  2.1     18.14     0.47 11400732     0.00     0.00  _mutex_unlock [14]
  1.9     18.58     0.44 11400732     0.00     0.00  mutex_lock [15]
  1.9     19.01     0.43  5700361     0.00     0.00  _memset [22]
  1.9     19.44     0.43        1   430.00   430.00  .div [21]
  1.8     19.85     0.41  5157853     0.00     0.00  cleanfree [19]
  1.4     20.17     0.32  5700366     0.00     0.00  _malloc_unlocked  [13]
  1.4     20.49     0.32  5700362     0.00     0.00  malloc [8]
  1.3     20.79     0.30  5157847     0.00     0.00  _smalloc          [24]
  1.2     21.06     0.27        6    45.00  1386.66  minimax [5]
  1.1     21.31     0.25  4755325     0.00     0.00  Delta_free [10]
  1.0     21.54     0.23  5700352     0.00     0.00  free [9]
  1.0     21.77     0.23   747130     0.00     0.00  GameState_applyDeltas [25]
  1.0     21.99     0.22  5157845     0.00     0.00  realfree [26]
  1.0     22.21     0.22   747129     0.00     0.00  GameState_unApplyDeltas [27]
  0.5     22.32     0.11  2360787     0.00     0.00  .rem [28]
  0.4     22.42     0.10  5700363     0.00     0.00  .udiv [29]
  0.4     22.52     0.10  1698871     0.00     0.00  GameState_getPlayer [30]
  0.4     22.61     0.09   747135     0.00     0.00  GameState_getStatus [31]
  0.3     22.68     0.07   204617     0.00     0.00  GameState_genMoves [17]
  0.1     22.70     0.02   945027     0.00     0.00  Move_free [23]
  0.0     22.71     0.01   542509     0.00     0.00  GameState_getValue [32]
  0.0     22.71     0.00      104     0.00     0.00  _ferror_unlocked [357]
  0.0     22.71     0.00       64     0.00     0.00  _realbufend [358]
  0.0     22.71     0.00       54     0.00     0.00  nvmatch [60]
  0.0     22.71     0.00       52     0.00     0.00  _doprnt [42]
  0.0     22.71     0.00       51     0.00     0.00  memchr [61]
  0.0     22.71     0.00       51     0.00     0.00  printf [43]
  0.0     22.71     0.00       13     0.00     0.00  _write [359]
  0.0     22.71     0.00       10     0.00     0.00  _xflsbuf [360]
  0.0     22.71     0.00        7     0.00     0.00  _memcpy [361]
  0.0     22.71     0.00        4     0.00     0.00  .mul [62]
  0.0     22.71     0.00        4     0.00     0.00  ___errno [362]
  0.0     22.71     0.00        4     0.00     0.00  _fflush_u [363]
  0.0     22.71     0.00        3     0.00     0.00  GameState_playerToStr [63]
  0.0     22.71     0.00        3     0.00     0.00  _findbuf [41]
```

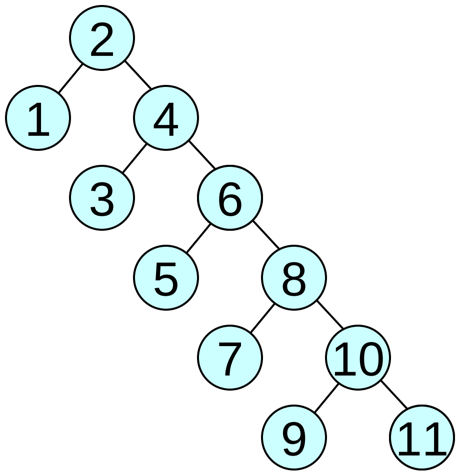
Second part of profile looks like this; it's the simple (i.e.,useful) part; corresponds to the "prof" tool

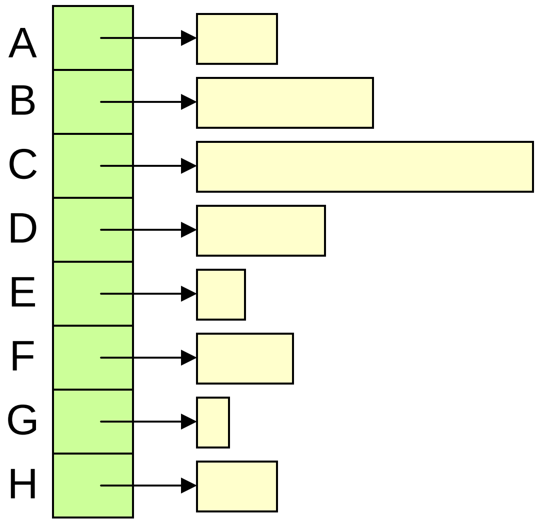
Don't even *think* of optimizing these

```
  %  cumulative    self              self    total
 time   seconds   seconds    calls  ms/call  ms/call name
 57.1     12.97    12.97                             internal_mcount [1]
  4.8     14.05     1.08  5700352     0.00     0.00  _free_unlocked [12]
  4.4     15.04     0.99                             _mcount (693)
  3.5     15.84     0.80 22801464     0.00     0.00  _return_zero [16]
  2.8     16.48     0.64  5700361     0.00     0.00  .umul [18]
  2.8     17.11     0.63   747130     0.00     0.01  GameState_expandMove [6]
  2.5     17.67     0.56  5700361     0.00     0.00  calloc [7]
  2.1     18.14     0.47 11400732     0.00     0.00  _mutex_unlock [14]
  1.9     18.58     0.44 11400732     0.00     0.00  mutex_lock [15]
  1.9     19.01     0.43  5700361     0.00     0.00  _memset [22]
  1.9     19.44     0.43        1   430.00   430.00  .div [21]
  1.8     19.85     0.41  5157853     0.00     0.00  cleanfree [19]
  1.4     20.17     0.32  5700366     0.00     0.00  _malloc_unlocked  <cycle 1> [13]
  1.4     20.49     0.32  5700362     0.00     0.00  malloc [8]
  1.3     20.79     0.30  5157847     0.00     0.00  _smalloc          <cycle 1> [24]
  1.2     21.06     0.27        6    45.00  1386.66  minimax [5]
  1.1     21.31     0.25  4755325     0.00     0.00  Delta_free [10]
  1.0     21.54     0.23  5700352     0.00     0.00  free [9]
  1.0     21.77     0.23   747130     0.00     0.00  GameState_applyDeltas [25]
  1.0     21.99     0.22  5157845     0.00     0.00  realfree [26]
  1.0     22.21     0.22   747129     0.00     0.00  GameState_unApplyDeltas [27]
  0.5     22.32     0.11  2360787     0.00     0.00  .rem [28]
  0.4     22.42     0.10  5700363     0.00     0.00  .udiv [29]
  0.4     22.52     0.10  1698871     0.00     0.00  GameState_getPlayer [30]
  0.4     22.61     0.09   747135     0.00     0.00  GameState_getStatus [31]
  0.3     22.68     0.07   204617     0.00     0.00  GameState_genMoves [17]
  0.1     22.70     0.02   945027     0.00     0.00  Move_free [23]
  0.0     22.71     0.01   542509     0.00     0.00  GameState_getValue [32]
  0.0     22.71     0.00      104     0.00     0.00  _ferror_unlocked [357]
  0.0     22.71     0.00        4     0.00     0.00  _thr_main [367]
  0.0     22.71     0.00        3     0.00     0.00  GameState_playerToStr [63]
  0.0     22.71     0.00        2     0.00     0.00  strcmp [66]
  0.0     22.71     0.00        1     0.00     0.00  GameState_getSearchDepth [67]
  0.0     22.71     0.00        1     0.00     0.00  GameState_new [37]
  0.0     22.71     0.00        1     0.00     0.00  GameState_playerFromStr [68]
  0.0     22.71     0.00        1     0.00     0.00  GameState_write [44]
  0.0     22.71     0.00        1     0.00     0.00  Move_isValid [69]
  0.0     22.71     0.00        1     0.00     0.00  Move_read [36]
  0.0     22.71     0.00        1     0.00     0.00  Move_write [59]
  0.0     22.71     0.00        1     0.00     0.00  check_nlspath_env [46]
  0.0     22.71     0.00        1     0.00   430.00  clock [20]
  0.0     22.71     0.00        1     0.00     0.00  exit [33]
  0.0     22.71     0.00        1     0.00  8319.99  getBestMove [4]
  0.0     22.71     0.00        1     0.00     0.00  getenv [47]
  0.0     22.71     0.00        1     0.00  8750.00  main [3]
  0.0     22.71     0.00        1     0.00     0.00  mem_init [70]
  0.0     22.71     0.00        1     0.00     0.00  number [71]
  0.0     22.71     0.00        1     0.00     0.00  scanf [53]
```

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?





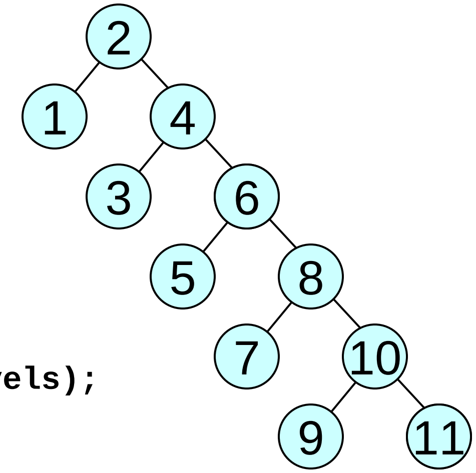
`2,1,4,3,6,5,8,7,10,9,11`

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);
}
```



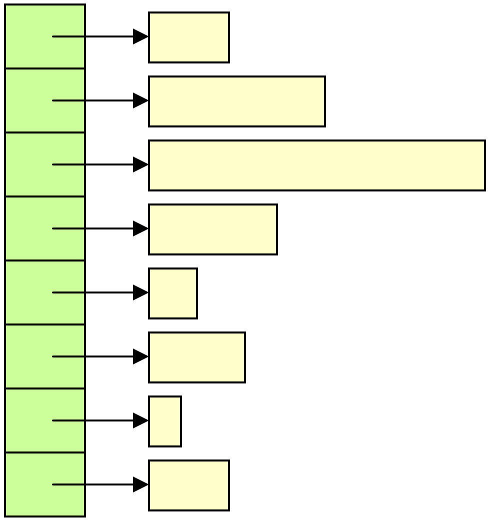

```
void Tree_PrintHistogram(Tree_T oTree, FILE *fp)
{
  /* Define histogram */
  int ipHistogram[MAX_LEVELS];
  int i;

  /* 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”);
}
```

Instrumentation Example

Hash table implemented as array of sets



```
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