Testing Modules

Module versus system testing

- When a software system is developed as a collection of modules, it pays to test the modules by themselves before testing the system. Testing the modules by themselves permits better testing at lower cost and also makes debugging simpler.
- **Example**: LineStorage
  `LSGetWord(lineNum,wordNum)` checks that `lineNum` and `wordNum` are in range.
  In KWIC, `LSGetWord` is *always* called with parameters that are in range.
  Thus, in KWIC, the range checking code of `LSGetWord` cannot be tested.
- **Bottom line**: isolate the module under test.
- In KWIC: 5 of the 6 modules are tested completely standalone. There is a test suite for the system as well. All the tests run completely automatically.

Two driving questions in designing module tests

Does the MUT do any I/O?

Does the MUT call functions from other modules?

MUT: Module Under Test

Input module testing: overview

Module service

Input's job is to load the lines in stdin into the LineStorage module.
Calls LSAddLine for each line in stdin
Calls LSAddWord for each word in each line

Test harness

Sample input files are needed.
A driver is needed to invoke INInit and INLoad.
A stub is needed for LSAddLine and LSAddWord.
A csh script is needed to invoke the driver for each input file.

WordTable module testing: overview

Module service

WordTable's job is to determine whether a given word is a noise word.
Does no I/O.
Does not call functions from any other KWIC module.

Test harness

A driver is needed to invoke the WordTable functions and check the return values.

LineStorage module testing: overview

Module service

LineStorage's job is to store the lines, providing access to the ith word in the jth line.
Does no I/O.
Does not call functions from any other KWIC module.

Test harness

A driver is needed to invoke the LineStorage functions and check the return values.

ShiftSort module testing: overview

Module service

ShiftSort's job is to provide access to the ith word in the jth line in a shifted and sorted version of the lines stored in LineStorage.
Does no I/O.
Does call LineStorage functions: LSNumLines, LSNumWords, LSGetWord.
Does call the WordTable function WTIsMember

Test harness

A driver is needed to invoke the ShiftSort functions and check the return values.
A stub is needed to simulate the 3 LineStorage functions.
A stub is needed to simulate WTIsMember.

Output module testing: overview

Module service

Output's job is to format the shifted, sorted lines offered by ShiftSort.
Writes to stdout.
Calls ShiftSort: SSNumLines, SSNumWords, SSGetWord, SSGetShiftNum.

Test harness

A driver is needed to invoke the Output functions.
A stub is needed to simulate ShiftSort.
A csh script is needed to invoke driver and compare act and exp.

Input module testing: details

- MIS sketch

```
void INInit(void);
KWStatus INLoad(void);
```

- Driver

Trivial: simply calls INInit and INLoad.

- Stubs: LineStorage.c
  Only the LineStorage functions called by Input are implemented: LSAddLine, LSAddWord, LSNumLines. The stubs write the function name and parameter values to stdout.

- Files:
  input/: simulated input to KWIC
  exp/: expected output from LineStorage stubs
  act/: actual output from LineStorage stubs

- Makefile contains shell commands with the following pseudocode

```
foreach f in input/
        run driver on f, saving output in act/f
        compare act/f with exp/f; report differences
```

- example: contents of test0 in input and exp

```
input/test0:
        The C Programming Language
        The Cat in the Hat
exp/test0:
        LSAddLine()
        LSAddWord(The)
        LSAddWord(C)
        LSAddWord(Programming)
        LSAddWord(Language)
        LSAddLine()
        LSAddWord(The)
        LSAddWord(Cat)
        LSAddWord(in)
        LSAddWord(the)
        LSAddWord(Hat)
```

Test case selection

- The Interval Rule
  AKA: "boundary value testing", "domain testing"

- Given the integer interval [L..U], L <= U

- Base rule: test the end points and at least one interior point
  for L = 0, U = 100: Test on {0,50,100}

- Error values: test the end points and at least one interior point
  Base rule applied to -infinity..L-1 and L+1..infinity
  for L = 0, U = 100: test on {-1000,-1,101,1000}

- Indirect application: applies to non-scalars with scalar properties
  For a list of maximum size 100: test on list sizes {0,50,100}

Example: LineStorage/driver.C

- Top level pseudocode
  initialize lineList
  invoke runTest
  invoke LSReset
  invoke runTest

- runTest pseudocode
  load lineList into LineStorage
  check that every line and word has been stored
  check that line number range errors are detected
  check that word number range errors are detected

- Use of the interval rule
  Number of words on a line
  Number of characters in a word
  Line number parameters: normal and error values
  Word number parameters: normal and error values

LineStorage module testing: details

- Function prototypes

```
void LSInit(void);
KWStatus LSAddLine(void);
KWStatus LSAddWord(char* word);
const char* LSGetWord(int lineNum,int wordNum);
int LSNumWords(int lineNum);
int LSNumLines(void);
```

- driver `struct`

```
#define NUMLINES 5
#define MAWORDS 6
struct {
        int numWords;
        char* wordList[MAXWORDS];
} lineList[NUMLINES] = {
        {5,"The","Cat","in","the","Hat"},
        {4,"The","C","Programming","Language"}
        ...
};
```

- driver pseudocode

  Check that LSAddWord fails when there are no lines
  Load all the lines in lineList, checking return codes
  Check that LineList contains the lines in lineList
  Check that line, word number range errors are detected

- driver `for` loop: loads all the lines in lineList (simplified to remove error checking).

```
for (i = 0; i < NUMLINES; i++) {
        LSAddLine();
        for (j = 0; j < lineList[i].numWords; j++) {
                LSAddWord(lineList[i].wordList[j]);
        }
}
```