**Testing**

When I started the project, I planned to use unit testing, and perform test driven development; due to the project making heavy use of the compiler compilers lex and yacc, however, I could not come up with a good way of performing unit testing on my programs. In light of this, I’ve decided to carry out testing of key functions in isolated source files with tailored test cases. The testing of the lexing and parsing elements of the programs will be done via white box testing.

|  |  |
| --- | --- |
| Function prototype | **int** swapIntEndianness(**int** input |
| Purpose | Swap the endianness of the integer “input” and return it. |
| Testing code | #include <stdio.h>  **struct** testCase {  **int** input;  **int** expectedOutput;  };  **int** swapIntEndianness(**int** input) {  **int** output = 0;    **for** (**int** i = 0; i < 4; i++) {  \*((**char** \*) &output - i + 3) = \*((**char** \*) &input + i);  }    **return** output;  }  **int** main(**int** argc, **char** \*argv[]) {  **struct** testCase tests[] = {{0x00000000, 0x00000000},  {0x000000FF, 0xFF000000},  {0x0000FF00, 0x00FF0000},  {0x00FF0000, 0x0000FF00},  {0xFF000000, 0x000000FF},  {0x12345678, 0x78563412}};    **int** actualOutput;    printf("Input Expected Actual \n");    **for** (**int** i = 0; i < **sizeof**(tests) / **sizeof**(**struct** testCase); i++) {  actualOutput = swapIntEndianness(tests[i].input);    printf("%08X %08X %08X", tests[i].input, tests[i].expectedOutput, actualOutput);    **if** (actualOutput != tests[i].expectedOutput) {  printf(" FAIL\n");    **continue**;  }    printf(" PASS\n");  }    **return** 0;  } |
| Outcome |  |

|  |  |
| --- | --- |
| Function prototype | **int** writeVariableLengthQuantity(**char** \*outputPtr, **int** input) |
| Purpose | Write the integer “input” as a big endian variable length quantity at the pointer “outputPtr”. Return the length of the variable length quantity written. |
| Testing code | #include <stdio.h>  **struct** testCase {  **int** input;  **int** expectedVLQ;  **int** expectedReturn;  };  **int** writeVariableLengthQuantity(**char** \*outputPtr, **int** input) {  **if** (input == 0) {  \*outputPtr = 0;  **return** 1;  }    **int** length = 5;    **for** (**int** i = 4; i >= 0; i--) {  **if** (input >> i \* 7) {  **break**;    } **else** {  length--;  }  }    **for** (**int** i = length - 1; i >= 0; i--) {  **if** (i != 0) {  \*(outputPtr + length - i - 1) = ((input >> i \* 7) & 0x7F) + 0x80;    } **else** {  \*(outputPtr + length - i - 1) = (input >> i \* 7) & 0x7F;  }  }    **return** length;  }  **int** main(**int** argc, **char** \*argv[]) {  **struct** testCase tests[] = {{0x00000000, 0x00000000, 0},  {0x00000040, 0x00000040, 1},  {0x0000007F, 0x0000007F, 1},  {0x00000080, 0x00000081, 2},  {0x00002000, 0x000000C0, 2},  {0x00003FFF, 0x00007FFF, 2},  {0x00004000, 0x00008081, 3},  {0x00100000, 0x000080C0, 3},  {0x001FFFFF, 0x007FFFFF, 3},  {0x00200000, 0x00808081, 4},  {0x08000000, 0x008080C0, 4},  {0x0FFFFFFF, 0x7FFFFFFF, 4}};    **int** actualVLQ;  **int** actualReturn;    printf("Input Expected Expected Actual Actual \n");  printf(" VLQ Return VLQ Return \n");    **for** (**int** i = 0; i < **sizeof**(tests) / **sizeof**(**struct** testCase); i++) {  actualVLQ = 0;  actualReturn = writeVariableLengthQuantity((**char** \*) &actualVLQ, tests[i].input);    printf("%08X %08X %08X %08X %08X", tests[i].input, tests[i].expectedVLQ, tests[i].expectedReturn, actualVLQ, actualReturn);    **if** (actualVLQ != tests[i].expectedVLQ) {  printf(" FAIL\n");    **continue**;  }    printf(" PASS\n");  }    printf("\nNOTE: All VLQ values corrected for endianness\n");    **return** 0;  } |
| Outcome |  |

|  |  |
| --- | --- |
| Function prototype | **int** readVariableLengthQuantity(**char** \*inputPtr) |
| Purpose | Read a big endian variable length quantity from pointer “inputPtr”, and return it as a little endian integer. |
| Testing code | #include <stdio.h>  **struct** testCase {  **char** input[4];  **int** expectedOutput;  };  **int** readVariableLengthQuantity(**char** \*inputPtr) {  **char** \*workingPtr = inputPtr;    **while** (\*workingPtr & 0x80) {  workingPtr++;  }    **int** output = 0;  **int** outputShift = 0;  **do** {  output |= (\*workingPtr & 0x7F) << outputShift;    outputShift += 7;    } **while** (workingPtr-- != inputPtr);  **return** output;  }  **int** main(**int** argc, **char** \*argv[]) {  **struct** testCase tests[] = {{{0x00, 0x00, 0x00, 0x00}, 0},  {{0x40, 0x00, 0x00, 0x00}, 0x40},  {{0x7F, 0x00, 0x00, 0x00}, 0x7F},  {{0x81, 0x00, 0x00, 0x00}, 0x80},  {{0xC0, 0x00, 0x00, 0x00}, 0x2000},  {{0xFF, 0x7F, 0x00, 0x00}, 0x3FFF},  {{0x81, 0x80, 0x00, 0x00}, 0x4000},  {{0xC0, 0x80, 0x00, 0x00}, 0x100000},  {{0xFF, 0xFF, 0x7F, 0x00}, 0x1FFFFF},  {{0x81, 0x80, 0x80, 0x00}, 0x200000},  {{0xC0, 0x80, 0x80, 0x00}, 0x8000000},  {{0xFF, 0xFF, 0xFF, 0x7F}, 0xFFFFFFF}};    **int** actualOutput;    printf("Input Expected Actual \n");    **for** (**int** i = 0; i < **sizeof**(tests) / **sizeof**(**struct** testCase); i++) {  actualOutput = readVariableLengthQuantity(tests[i].input);    **for** (**char** c = 0; c < 4; c++) {  printf("%02X", (**unsigned** **char**) tests[i].input[c]);  }    printf(" %08X %08X", tests[i].expectedOutput, actualOutput);    **if** (actualOutput != tests[i].expectedOutput) {  printf(" FAIL\n");    **continue**;  }    printf(" PASS\n");  }    **return** 0;  } |
| Outcome |  |