Course: Programming Fundamentals – **ENCM 339**

Lab #: Lab 4

Instructor: S. Norman

Student Name: **Mitchell Sawatzky**

Lab Section: **B02**

Date Submitted: **Oct 6, 2015**

**Exercise E, Part I**

Terminal Output

Mitchell@ttys000 17:06 {0} [lab4]$ ./test.out

Please enter a line of text. To quit, start it with q.

abc

Input line was "abc

"

Please enter a line of text. To quit, start it with q.

abcdef

Input line was "abcdef

"

Please enter a line of text. To quit, start it with q.

abcdefg

Input line was "abcdefg"

Please enter a line of text. To quit, start it with q.

Input line was "

"

Please enter a line of text. To quit, start it with q.

aaaaaaaaaaaaaaaaaaaaaaaaa

Input line was "aaaaaaa"

Please enter a line of text. To quit, start it with q.

Input line was "aaaaaaa"

Please enter a line of text. To quit, start it with q.

Input line was "aaaaaaa"

Please enter a line of text. To quit, start it with q.

Input line was "aaaa

"

Please enter a line of text. To quit, start it with q.

q

Reason for quitting: found q at beginning of line.

**Exercise E, Part II**

line-getter4E.c

// line-getter4E.c

// ENCM 339 Fall 2015 Lab 4 Exercise E

#include <stdio.h>

#include <string.h>

#define QUIT\_LETTER 'q'

// Again, in a practical program, this is a ridiculously small size

// for an array that is supposed to hold a line of text. But it's

// convenient for testing purposes.

#define LINE\_ARRAY\_SIZE 8

int eat\_til\_newline(FILE \*stream);

// REQUIRES: stream is open for input.

// PROMISES: Characters are read from stream and discarded until either a

// '\n' has been read or an input error has occurred.

// Return value is 0 if '\n' is read, and EOF for an error.

int get\_a\_line(char \*s, int size, FILE \*stream);

// Does what fgets does, using repeated calls to fgetc, but

// provides a more useful return value than fgets does.

//

// REQUIRES

// size > 1.

// s points to the start of an array of at least size bytes.

// stream is open for input.

// PROMISES

// Return value is EOF if input error occurred.

// Otherwise, return value gives the index of the '\0' that

// terminates the string in the array.

void reverse(char \*s);

//REQUIRES

// s points to a valid c-string

//PROMISES

// The c-string that s points to will be reversed in place

// up to but not including the first occurence of '\0'l

int main(void)

{

char line[LINE\_ARRAY\_SIZE];

int input\_error = 0, len;

while (1) {

printf("Please enter a line of text. To quit, start it with %c.\n", QUIT\_LETTER);

len = get\_a\_line(line, LINE\_ARRAY\_SIZE, stdin);

if (len == EOF) {

input\_error = 1;

break;

}

if (line[0] == QUIT\_LETTER)

break;

if (line[len-1] == '\n') {

line[len-1] = '\0';

printf("The line, newline removed, was \"%s\".", line);

reverse(line);

printf(" In reverse, that is \"%s\".\n", line);

} else if (len == LINE\_ARRAY\_SIZE-1) {

eat\_til\_newline(stdin);

fputs("Input line ignored because it was too long!\n", stdout);

} else {

input\_error = 1;

break;

}

} // while (1)

fputs("\nReason for quitting: ", stdout);

if (input\_error)

fputs("unexpected input error.\n", stdout);

else

printf("found %c at beginning of line.\n", QUIT\_LETTER);

return 0;

}

int eat\_til\_newline(FILE \* stream)

{

int c;

do {

c = fgetc(stream);

} while (c != EOF && c != '\n');

// Return EOF if c == EOF, otherwise return 0.

return (c == EOF) ? EOF : 0;

}

int get\_a\_line(char \*s, int size, FILE \*stream)

{

int c, readChar = 0;

while (readChar < size-1) {

c = fgetc(stream);

if (c == EOF) {

return EOF;

} else {

s[readChar] = c;

readChar++;

if (c == '\n')

break;

}

}

s[readChar] = '\0';

return readChar;

}

void reverse(char \*s)

{

int j = 0, i;

while (s[j] != '\0')

j++;

for (i = 0, j -= 1; i < j; i++, j--) {

char storage = s[i];

s[i] = s[j];

s[j] = storage;

}

}

Terminal Output

Mitchell@ttys000 17:32 {0} [lab4]$ ./test.out

Please enter a line of text. To quit, start it with q.

abc

The line, newline removed, was "abc". In reverse, that is "cba".

Please enter a line of text. To quit, start it with q.

abcdef

The line, newline removed, was "abcdef". In reverse, that is "fedcba".

Please enter a line of text. To quit, start it with q.

abcdefg

Input line ignored because it was too long!

Please enter a line of text. To quit, start it with q.

aaaaaaaaaaaaaaaaaaaaaaaaaa

Input line ignored because it was too long!

Please enter a line of text. To quit, start it with q.

q

**Exercise F**

array-utils4F.c

// array-utils-4F.c

// ENCM 339 Fall 2015 Lab 4 Exercise F

// ATTENTION: The definitions given below for is\_sorted and max\_el

// are DEFECTIVE!

#include <assert.h>

#include "array-utils4F.h"

int is\_sorted(const int \*a, int n)

{

assert (n >= 1);

if (n == 1)

return 1;

int k, result;

for (k = 0; k < n - 1; k++) {

if (a[k] <= a[k + 1])

result = 1;

else {

result = 0;

break;

}

}

return result;

}

int max\_el(const int \*a, int n)

{

assert(n >= 1);

int result = a[0], i;

for (i = 1; i < n; i++)

if (a[i] > result)

result = a[i];

return result;

}

int is\_arith\_seq(const int \*a, int n)

{

assert (n >= 1);

if (n == 1)

return 1;

int i, sum = a[1]-a[0];

for (i=1; i < n; i++) {

if (a[i]-a[i-1] != sum)

return 0;

}

return 1;

}

#ifdef UNIT\_TESTS

#include <stdio.h>

// This macro works for variables declared to be arrays. (DON'T try to

// use for function parameters declared to be arrays!)

#define COUNT(x) (sizeof(x)/sizeof(x[0]))

void test\_is\_sorted(const char \*tag, const int \*a, int n, int expected\_rv);

void test\_max\_el(const char \*tag, const int \*a, int n, int expected\_rv);

void test\_is\_arith\_seq(const char \*tag, const int \*a, int n, int expected\_rv);

int main(void)

{

int test\_01[] = { 10, 20, 30, 40, 50 };

int test\_02[] = { 10, 10, 10, 10 };

int test\_03[] = { 10, 20, 30, 40, 35 };

int test\_04[] = { 10, 20, 30, 25, 40 };

int test\_05[] = { 10, 5, 15, 25 };

test\_is\_sorted("test\_01", test\_01, COUNT(test\_01), 1);

test\_is\_sorted("test\_02", test\_02, COUNT(test\_02), 1);

test\_is\_sorted("test\_03", test\_03, COUNT(test\_03), 0);

test\_is\_sorted("test\_04", test\_04, COUNT(test\_04), 0);

test\_is\_sorted("test\_05", test\_05, COUNT(test\_05), 0);

fputc('\n', stdout);

int test\_06[] = { 100, 1, 2, 3 };

int test\_07[] = { 1, 2, 100, 3 };

int test\_08[] = { 1, 2, 3, 100 };

int test\_09[] = { -1, -2, -3, -4 };

int test\_10[] = { -8, -7, -6, -7, -8 };

test\_max\_el("test\_06", test\_06, COUNT(test\_06), 100);

test\_max\_el("test\_07", test\_07, COUNT(test\_07), 100);

test\_max\_el("test\_08", test\_08, COUNT(test\_08), 100);

test\_max\_el("test\_09", test\_09, COUNT(test\_09), -1);

test\_max\_el("test\_10", test\_10, COUNT(test\_10), -6);

fputc('\n', stdout);

int test\_11[] = { 4, 3, 2, 1 };

int test\_12[] = { 1, 2, 3, 4 };

int test\_13[] = { 1, 2, 3, 100 };

int test\_14[] = { -1, -2, -3, -4 };

int test\_15[] = { 2, 4, 6, 8, 10 };

test\_is\_arith\_seq("test\_11", test\_11, COUNT(test\_11), 1);

test\_is\_arith\_seq("test\_12", test\_12, COUNT(test\_12), 1);

test\_is\_arith\_seq("test\_13", test\_13, COUNT(test\_13), 0);

test\_is\_arith\_seq("test\_14", test\_14, COUNT(test\_14), 1);

test\_is\_arith\_seq("test\_15", test\_15, COUNT(test\_15), 1);

fputc('\n', stdout);

return 0;

}

void test\_is\_sorted(const char \*tag, const int \*a, int n, int expected\_rv)

{

printf("Testing is\_sorted for case with tag \"%s\":", tag);

if (expected\_rv == is\_sorted(a, n))

printf(" Pass.\n");

else

printf(" FAIL!\n");

}

void test\_max\_el(const char \*tag, const int \*a, int n, int expected\_rv)

{

printf("Testing max\_el for case with tag \"%s\":", tag);

if (expected\_rv == max\_el(a, n))

printf(" Pass.\n");

else

printf(" FAIL!\n");

}

void test\_is\_arith\_seq(const char \*tag, const int \*a, int n, int expected\_rv)

{

printf("Testing is\_arith\_seq for case with tag \"%s\":", tag);

if (expected\_rv == is\_arith\_seq(a, n))

printf(" Pass.\n");

else

printf(" FAIL!\n");

}

#endif // #ifdef UNIT\_TESTS

Terminal Output:

Mitchell@ttys000 17:59 {0} [lab4]$ ./test.out

Testing is\_sorted for case with tag "test\_01": Pass.

Testing is\_sorted for case with tag "test\_02": Pass.

Testing is\_sorted for case with tag "test\_03": Pass.

Testing is\_sorted for case with tag "test\_04": Pass.

Testing is\_sorted for case with tag "test\_05": Pass.

Testing max\_el for case with tag "test\_06": Pass.

Testing max\_el for case with tag "test\_07": Pass.

Testing max\_el for case with tag "test\_08": Pass.

Testing max\_el for case with tag "test\_09": Pass.

Testing max\_el for case with tag "test\_10": Pass.

Testing is\_arith\_seq for case with tag "test\_11": Pass.

Testing is\_arith\_seq for case with tag "test\_12": Pass.

Testing is\_arith\_seq for case with tag "test\_13": Pass.

Testing is\_arith\_seq for case with tag "test\_14": Pass.

Testing is\_arith\_seq for case with tag "test\_15": Pass.

The function max\_el is defective because it initializes the result variable to 0. The array of integers may have a maximum that is less than 0, and if this is the case then the existing definition of max\_el will return 0. To fix this, simply initialize the result variable to an element of a.

The function is\_sorted only functionally checks if the last 2 elements in the array are sorted, which is to say that any element before the last two could be unsorted, and the function would still return 1 as long as the last 2 were sorted. To fix this, simply stop checking if elements are sorted as soon as an unsorted pair of elements is reached.