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.