Course: Programming Fundamentals - ENCM 339   
Lab #: Lab 4  
Instructor: M. Moussavi  
Student Name: Liam Wrubleski  
Lab Section: B01  
Date submitted: Oct 15, 2016

# Exercise D – Part 1

## Sample Dialogues

liam@Inspiron /cygdrive/c/Users/liam/Documents/GitHub/ENCM339Assignments

$ ./a

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.

aaaaaaaaaaaaaaaaaaaaaaaaaaa

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 "aaaaaa

"

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

q

Reason for quitting: found q at beginning of line.

# Exercise D – Part 2

## Code Listing

// line-getter4D.c

// ENCM 339 Fall 2016 Lab 4 Exercise D

#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: string at location s will be reversed in place.

int main(void)

{

char line[LINE\_ARRAY\_SIZE];

int input\_error = 0;

while (1) {

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

QUIT\_LETTER);

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

if (val == EOF) {

// Case 3 or 4 (Input error.)

input\_error = 1;

break;

}

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

break;

// If val is the size-1, then the string was too long. Otherwise,

// it's fine.

if(val == LINE\_ARRAY\_SIZE-1){

eat\_til\_newline(stdin);

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

} else {

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

reverse(line);

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

}

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

/\* Strips newline from string read, replaces with '\0'. Therefore,

the index of '\0' will be size-1 only for a string too long for

the given array. Otherwise, it will be size-2 or less.

\*/

char c;

int i;

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

c=fgetc(stream);

if(EOF == c){

s[i]='\0';

return EOF;

}

s[i] = c;

if(c == '\n') break;

}

if(s[i] == '\n') s[i]='\0';

s[i]='\0';

return i;

}

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;

}

void reverse(char \*s)

{

char \*f=s+strlen(s)-1, \*b=s;

char t;

while (f > b){

t=\*f;

\*f--=\*b;

\*b++=t;

}

}

## Sample Dialogues

liam@Inspiron /cygdrive/c/Users/liam/Documents/GitHub/ENCM339Assignments

$ ./a

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.

aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

Input line ignored because it was too long!

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

q

# Exercise E

## Explanation of Logic Errors

In decreasing, the logic error occurs in the setting of ‘result’. Each time the for loop runs, the value in result is overwritten. This means that the loop will only return a 1 if the last two values are decreasing (i.e. a[n-2] > a[n-1]). Similarly, it will only return a 0 if the last two values are non-decreasing (i.e. a[n-2] <= a[n-1]). Therefore this function will return an incorrect value if the array is non-decreasing before the last two values, but will be correct otherwise.

In all\_unique, the logic error occurs in the initialization of the second for loop. As the second counter is also initialized at 0, it will check each value against every value in the array, including itself. This means that this function will always return 0. This can be fixed by initializing the second counter at one greater than the first counter, thus checking every value against every value that comes after it. They don’t need to be checked against the values that came before, as that pair would already have been checked in an earlier loop.

## Code Listing

// array-utils-4E.c

// ENCM 339 Fall 2016 Lab 4 Exercise E

#include <assert.h>

#include "array-utils4E.h"

int decreasing(const int \*a, int n)

{

assert (n >= 1);

int i, result = 1;

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

if (!(a[i] > a[i + 1]))

result = 0;

}

return result;

}

int all\_unique(const int \*a, int n)

{

assert(n >= 1);

int result = 1, i, j;

for (i = 0; i < n; i++)

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

if (a[i] == a[j])

result = 0;

return result;

}

int in\_range(const int\* a, int n, int min, int max)

{

assert(n >= 1);

int i,result=1;

for (i=0;i<n;i++)

result=(a[i] >= min && a[i] <= max)?result:0;

return result;

}

#ifdef UNIT\_TESTS

#include <stdio.h>

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

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

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

void test\_decreasing(const char \*tag,

const int \*a, int n, int expected\_rv);

void test\_all\_unique(const char \*tag,

const int \*a, int n, int expected\_rv);

void test\_in\_range(const char \*tag,

const int \*a, int n, int min, int max, int expected\_rv);

int main(void)

{

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

int test\_02[] = { 10 };

int test\_03[] = { 99, 98, 97, 96, 96 };

int test\_04[] = { 10, 11, 10, 9, 8 };

int test\_05[] = { 20, 19, 18, 19, 17 };

test\_decreasing("test\_01", test\_01, COUNT(test\_01), 1);

test\_decreasing("test\_02", test\_02, COUNT(test\_02), 1);

test\_decreasing("test\_03", test\_03, COUNT(test\_03), 0);

test\_decreasing("test\_04", test\_04, COUNT(test\_04), 0);

test\_decreasing("test\_05", test\_05, COUNT(test\_05), 0);

fputc('\n', stdout);

int test\_06[] = { 10, 7, 8, 9, 10 };

int test\_07[] = { 11, 12, 13, 13, 14 };

int test\_08[] = { 15, 16, 17, 18, 16, 19 };

int test\_09[] = { 20 };

int test\_10[] = { 20, 21, 22, 23, 24 };

test\_all\_unique("test\_06", test\_06, COUNT(test\_06), 0);

test\_all\_unique("test\_07", test\_07, COUNT(test\_07), 0);

test\_all\_unique("test\_08", test\_08, COUNT(test\_08), 0);

test\_all\_unique("test\_09", test\_09, COUNT(test\_09), 1);

test\_all\_unique("test\_10", test\_10, COUNT(test\_10), 1);

fputc('\n', stdout);

int test\_11[] = { 10, 9, 8, 7, 11 }, min\_11=7, max\_11=11; //1

int test\_12[] = { -1, 12, 30, 7, 42 }, min\_12=0, max\_12=100; //0

int test\_13[] = { -20, 14, -7, 0, -2 }, min\_13=-100, max\_13=0; //0

int test\_14[] = { 4 }, min\_14=4, max\_14 = 4; // 1

int test\_15[] = { -1, -2, -3, 11, 12 }, min\_15=0, max\_15=10; //0

test\_in\_range("test\_11", test\_11, COUNT(test\_11), min\_11, max\_11, 1);

test\_in\_range("test\_12", test\_12, COUNT(test\_12), min\_12, max\_12, 0);

test\_in\_range("test\_13", test\_13, COUNT(test\_13), min\_13, max\_13, 0);

test\_in\_range("test\_14", test\_14, COUNT(test\_14), min\_14, max\_14, 1);

test\_in\_range("test\_15", test\_15, COUNT(test\_15), min\_15, max\_15, 0);

fputc('\n', stdout);

return 0;

}

void test\_decreasing(const char \*tag,

const int \*a, int n, int expected\_rv)

{

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

if (expected\_rv == decreasing(a, n))

printf(" Pass.\n");

else

printf(" FAIL!\n");

}

void test\_all\_unique(const char \*tag,

const int \*a, int n, int expected\_rv)

{

printf("Testing all\_unique for case with tag \"%s\":", tag);

if (expected\_rv == all\_unique(a, n))

printf(" Pass.\n");

else

printf(" FAIL!\n");

}

void test\_in\_range(const char \*tag,

const int \*a, int n, int min, int max, int expected\_rv)

{

printf("Testing in\_range for case with tag \"%s\", min %d, max %d:\n",

tag, min, max);

if(expected\_rv == in\_range(a, n, min, max))

printf("Pass.\n");

else

printf("FAIL!\n");

}

#endif // #ifdef UNIT\_TESTS

## Sample Dialogues

liam@Inspiron /cygdrive/c/Users/liam/Documents/GitHub/ENCM339Assignments

$ gcc -Wall -DUNIT\_TESTS=1 array-utils4E.c

liam@Inspiron /cygdrive/c/Users/liam/Documents/GitHub/ENCM339Assignments

$ ./a

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

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

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

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

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

Testing all\_unique for case with tag "test\_06": Pass.

Testing all\_unique for case with tag "test\_07": Pass.

Testing all\_unique for case with tag "test\_08": Pass.

Testing all\_unique for case with tag "test\_09": Pass.

Testing all\_unique for case with tag "test\_10": Pass.

Testing in\_range for case with tag "test\_11", min 7, max 11:

Pass.

Testing in\_range for case with tag "test\_12", min 0, max 100:

Pass.

Testing in\_range for case with tag "test\_13", min -100, max 0:

Pass.

Testing in\_range for case with tag "test\_14", min 4, max 4:

Pass.

Testing in\_range for case with tag "test\_15", min 0, max 10:

Pass.