Course: Programming Fundamentals – **ENCM 339**

Lab #: Lab 3

Instructor: S. Norman

Student Name: **Mitchell Sawatzky**

Lab Section: **B02**

Date Submitted: **Sept 29, 2015**

**Exercise C**

lab3exC.c

// ENCM 339 Fall 2015 Lab 3 Exercise C

// This program, as posted on D2L, is DEFECTIVE!

#include <stdio.h>

void time\_diff(int earlier\_h, int earlier\_min, int later\_h, int later\_min,

int \*diff\_h, int \*diff\_min);

// Computes difference between two times in the same day. Assumes use of

// a 24-hour clock.

int a[2];

int main(void)

{

int t1\_h = 23, t1\_min = 7, t2\_h = 10, t2\_min = 53;

int p\_h, p\_min;

printf("Difference in time between %02d:%02d and between %02d:%02d is ...\n",

t1\_h, t1\_min, t2\_h, t2\_min);

time\_diff(t2\_h, t2\_min, t1\_h, t1\_min, &p\_h, &p\_min);

printf("... %d hour(s) and %d minute(s).\n", p\_h, p\_min);

return 0;

}

void time\_diff(int earlier\_h, int earlier\_min, int later\_h, int later\_min,

int \*diff\_h, int \*diff\_min)

{

if (later\_min >= earlier\_min) {

\*diff\_h = later\_h - earlier\_h;

\*diff\_min = later\_min - earlier\_min;

}

else {

\*diff\_h = later\_h - earlier\_h - 1;

\*diff\_min = later\_min + 60 - earlier\_min;

}

}

Terminal Output

Mitchell@ttys003 04:14 {0} [lab3]$ ./test.out

Difference in time between 23:07 and between 10:53 is ...

... 12 hour(s) and 14 minute(s).

**Exercise D**

lab3exD.c

// ENCM 339 Fall 2015 Lab 3 Exercise D

#include <stdio.h>

void display\_array(const char\* label, const double\* x, size\_t n);

// REQUIRES

// label points to the beginning of a string.

// Elements x[0], ... x[n-1] exist.

// PROMISES

// label is printed, followed by values of x[0], ... x[n], all on

// one line, using %4.2f format for the doubles. If n == 0, the line

// of output points out that fact.

void reverse(double\* x, size\_t n);

// REQUIRES

// n > 0.

// Array elements x[0] ... x[n - 1] exist.

// PROMISES

// Order of elements x[0] ... x[n - 1] has been reversed.

// (So the new x[0] value is the old x[n - 1] value, and so on.)

void append(double\* dest, size\_t cap, size\_t dest\_count,

const double\* src, size\_t src\_count);

// Append a list of numbers in the src array to a list of

// numbers in the dest array, without overflowing the dest array.

// REQUIRES

// cap > 0. cap indicates the capacity of the dest array.

// dest\_count <= cap. dest\_count indicates how many elements

// of the dest\_array should be preserved.

// Elements src[0] ... src[src\_count-1] exist.

// PROMISES

// Elements have been copied to the dest array, such that

// dest[dest\_count] == src[0], and so on, until either

// all of src[0] ... src[src\_count-1] have been copied or

// dest[cap-1] has been updated, whichever happened first.

// In Step 3, add a function prototype and function interface comment

// for max\_element here.

double max\_element(const double\* x, size\_t n);

//REQUIRES

// n > 0

// Array elements x[0] ... x[n-1] exit.

// Array elements are all of type double.

//PROMISES

// Returns the value of the element in array x with the maximum

// value.

int main(void)

{

double test1[ ] = {1.1, 2.2, 3.3, 4.4, 5.5};

double test2[ ] = {-0.5, -1.0, -1.5, -2.0, -2.5, -3.0};

printf("Some quick checks of display\_array ...\n");

display\_array(" all of test1:", test1, 5);

display\_array(" none of test1:", test1, 0);

display\_array(" last 3 elements of test1:", &test1[2], 3);

display\_array(" all of test2:", test2, 6);

printf("\nTwo tests of reverse ...\n");

reverse(test1, 5);

display\_array(" test1 after reversing:", test1, 5);

reverse(test2, 6);

display\_array(" test2 after reversing:", test2, 6);

// In testing append, we'll give the destination arrays 11 elements,

// then pretend that their capacities are only 10 elements. That

// way we can check to make sure that the value of element 10

// hasn't been changed from the 0.0 it gets initialized to.

printf("\nTwo tests of append ...\n");

double test3[11] = {1.0, 2.0, 3.0, 4.0};

double test3src[ ] = {5.0, 6.0, 7.0};

append(test3, 10, 4, test3src, 3);

display\_array(" test3 after append:", test3, 11);

double test4[11] = {1.1, 2.1, 3.1, 4.1, 5.1};

double test4src[ ] = {6.1, 7.1, 8.1, 9.1, 10.1, 11.1, 12.1};

append(test4, 10, 5, test4src, 7);

display\_array(" test4 after append:", test4, 11);

// In Step 3, add some tests for max\_element here.

printf("\nFour tests of max\_element ...\n");

double test5[5] = {-6.9, -2.0, -3.0, -1.0, -4.0};

display\_array(" test5 src: ", test5, 5);

printf(" test5 max: %lf\n", max\_element(test5, 5));

double test6[6] = {1.1, 2.2, 3.3, 5.5, 100.7, 8.0};

display\_array(" test6 src: ", test6, 6);

printf(" test6 max: %lf\n", max\_element(test6, 6));

double test7[7] = {13.37, 4.2, -5.0, -6.0, 10.0, -10.0, 0.0};

display\_array(" test7 src: ", test7, 7);

printf(" test7 max: %lf\n", max\_element(test7, 7));

double test8[8] = {13.37, 4.2, -5.0, -6.0, 10.0, -10.0, 0.0, 42.0};

display\_array(" test8 src: ", test8, 8);

printf(" test8 max: %lf\n", max\_element(test8, 8));

return 0;

}

void display\_array(const char\* label, const double \* x, size\_t n)

{

size\_t i ;

printf("%s", label);

if (n == 0)

printf(" [no contents to print]\n");

else {

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

printf(" %4.2f", x[i] );

printf("\n");

}

}

void reverse(double\* x, size\_t n)

{

int i;

double swapValue;

for (i = 0; i < n/2; i++) {

swapValue = x[i];

x[i] = x[n-1-i];

x[n-1-i] = swapValue;

}

return;

}

void append(double\* dest, size\_t cap, size\_t dest\_count,

const double\* src, size\_t src\_count)

{

int i;

for (i = 0; i + dest\_count < cap && i < src\_count; i++) {

dest[dest\_count + i] = src[i];

}

return;

}

// In Coding Step 3, add a function definition max\_element here.

double max\_element(const double\* x, size\_t n)

{

double max = x[0];

int i;

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

if (max < x[i])

max = x[i];

}

return max;

}

Terminal Output

Mitchell@ttys003 13:29 {0} [lab3]$ ./test.out

Some quick checks of display\_array ...

all of test1: 1.10 2.20 3.30 4.40 5.50

none of test1: [no contents to print]

last 3 elements of test1: 3.30 4.40 5.50

all of test2: -0.50 -1.00 -1.50 -2.00 -2.50 -3.00

Two tests of reverse ...

test1 after reversing: 5.50 4.40 3.30 2.20 1.10

test2 after reversing: -3.00 -2.50 -2.00 -1.50 -1.00 -0.50

Two tests of append ...

test3 after append: 1.00 2.00 3.00 4.00 5.00 6.00 7.00 0.00 0.00 0.00 0.00

test4 after append: 1.10 2.10 3.10 4.10 5.10 6.10 7.10 8.10 9.10 10.10 0.00

Four tests of max\_element ...

test5 src: -6.90 -2.00 -3.00 -1.00 -4.00

test5 max: -1.000000

test6 src: 1.10 2.20 3.30 5.50 100.70 8.00

test6 max: 100.700000

test7 src: 13.37 4.20 -5.00 -6.00 10.00 -10.00 0.00

test7 max: 13.370000

test8 src: 13.37 4.20 -5.00 -6.00 10.00 -10.00 0.00 42.00

test8 max: 42.000000

**Exercise E**

lab3exE.c

// ENCM 339 Fall 2015 Lab 3 Exercise E

#include <stdio.h>

int main(void)

{

char buffer[80]; // enough space for a string of length <= 79

// THIS IS A GOOD WAY TO LEARN SOMETHING ABOUT C STRINGS, BUT IT'S

// NOT A GOOD EXAMPLE OF READABLE OR PRACTICAL CODE!

// Put characters into the string using ASCII codes.

buffer[0] = 64;

buffer[1] = 35;

buffer[2] = 36;

buffer[3] = 37;

buffer[4] = 33;

buffer[5] = 32;

buffer[6] = 51;

buffer[7] = 51;

buffer[8] = 57;

buffer[9] = 32;

buffer[10] = 105;

buffer[11] = 115;

buffer[12] = 110;

buffer[13] = 39;

buffer[14] = 116;

buffer[15] = 32;

buffer[16] = 97;

buffer[17] = 108;

buffer[18] = 119;

buffer[19] = 97;

buffer[20] = 121;

buffer[21] = 115;

buffer[22] = 32;

buffer[23] = 102;

buffer[24] = 117;

buffer[25] = 110;

buffer[26] = 33;

// Put the end-of-string character at the end of the string.

buffer[27] = 0;

printf("The string in buffer is \"%s\"\n", buffer);

return 0;

}

**Exercise F**

lab3exF.c

// ENCM 339 Fall 2015 Lab 3 Exercise F

#include <stdio.h>

#include <string.h>

size\_t my\_strlen(const char \*s);

// Duplicates the library strlen function.

// REQUIRES

// s points to the first char in a C string.

// PROMISES

// Return value is the number of characters in the string up to but

// not including the terminating '\0'.

char \*my\_strcat(char \*dest, const char \*src);

// Duplicates the library strcat function.

// REQUIRES

// dest and src to point to the beginnings of C strings.

// dest points to the beginning of an array with at least this

// many chars: strlen(dest) + strlen(src) + 1.

// PROMISES

// C string pointed to dest is modified by appending all the characters

// from the src string.

// Return value is the address of the first char in the dest string.

int main(void)

{

// Use strlen a few times.

printf("strlen(\"\") returns %zu.\n", my\_strlen(""));

printf("strlen(\"A\") returns %zu.\n", my\_strlen("A"));

printf("strlen(\"ABCDE\") returns %zu.\n\n", my\_strlen("ABCDE"));

// Now demonstrate a few uses of strcat.

// Start with an empty string, followed by junk characters.

char s[16] = {'\0', 'X', 'X', 'X', 'X', 'X', 'X', 'X',

'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X'};

char \*p;

p = my\_strcat(s, ""); // Append empty string to empty string.

printf("First strcat result: \"%s\"\n", p);

p = my\_strcat(s, "E"); // Append a string of length 1.

printf("Second strcat result: \"%s\"\n", p);

p = my\_strcat(s, "NCM"); // Append a longer string.

printf("Third strcat result: \"%s\"\n", p);

p = my\_strcat(s, " 339!"); // Append another longer string.

printf("Fourth strcat result: \"%s\"\n", p);

p = my\_strcat(s, ""); // Append another empty string.

printf("Fifth strcat result: \"%s\"\n", p);

return 0;

}

size\_t my\_strlen(const char \*s)

{

size\_t len = 0;

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

len++;

return len;

}

char \*my\_strcat(char \*dest, const char \*src)

{

size\_t len = strlen(dest);

int index = 0;

while (src[index] != '\0') {

dest[len+index] = src[index];

index++;

}

dest[len+index] = '\0';

return dest;

}