**Course: Programming Fundamentals - ENCM 339**

Lab 3

**Instructor:** M Moussavi

**Student Name:** William Ehman

**Lab Section:** B01

**Date submitted:** Oct 11, 2016

# **Exercise D**

## **Code Listing**

// ENCM 339 Fall 2016 Lab 3 Exercise D

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

//However, it has been fixed by William Ehman

#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 main(void)

{

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

int p\_hours = 0, p\_minutes = 0;

int \*p\_h = &p\_hours, \*p\_min = &p\_minutes;

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;

}

}

## **Sample Dialogue**

Will@Will-Laptop /cygdrive/c/users/will/documents/ENCM339/lab3

$ ./d.exe

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

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

# **Exercise E**

## **Code Listing**

// ENCM 339 Fall 2016 Lab 3 Exercise E

#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-1], 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.)

int increasing(const double\* x, size\_t n);

// REQUIRES

// n > 0.

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

// PROMISES

// Return value is 1 if n == 1.

// If n > 1, return value is 1 if all of a[0] < a[1] ... a[n-2] < a[n-1]

// are true, otherwise return value is 0.

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

// for min\_element here.

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

// REQUIRES

// n > 0.

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

// PROMISES

// Returns value (double) of minimum element in array x

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);

double test3[ ] = {1.5, 1.25, 2.5, 3.5, 4.5, 5.5, 5.25};

double test4[ ] = {1.0, 2.0, 3.0, 3.0, 4.0, 5.0};

int inc;

printf("\nSome tests of increasing ...\n");

display\_array(" for these numbers:", test3, 6);

inc = increasing(test3, 6);

printf(" expected return value is 0, actual value is %d\n\n", inc);

display\_array(" for these numbers:", &test3[1], 5);

inc = increasing(&test3[1], 5);

printf(" expected return value is 1, actual value is %d\n\n", inc);

display\_array(" for these numbers:", &test3[1], 6);

inc = increasing(&test3[1], 6);

printf(" expected return value is 0, actual value is %d\n\n", inc);

display\_array(" for these numbers:", test4, 6);

inc = increasing(test4, 6);

printf(" expected return value is 0, actual value is %d\n\n", inc);

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

double test5[ ] = {-1.5, -1.25, -2.5, -2.0, -4.5, -5.5, -5.25};

double test6[ ] = {1.0, 2.0, 3.0, 3.1, 0.5, 4.5};

double test7[ ] = {-5.0,-1.0,20.0};

double test8[ ] = {-5.0,5.0,-20.0};

double min;

printf("Four tests of minimum ...\n");

display\_array(" for these numbers:", test5, 7);

min = min\_element(test5,7);

printf(" expected return value is -5.500000, actual value is %f\n\n", min);

display\_array(" for these numbers:", test6, 6);

min = min\_element(test6,6);

printf(" expected return value is 0.500000, actual value is %f\n\n", min);

display\_array(" for these numbers:", test7, 3);

min = min\_element(test7,3);

printf(" expected return value is -5.000000, actual value is %f\n\n", min);

display\_array(" for these numbers:", test8, 3);

min = min\_element(test8,3);

printf(" expected return value is -20.000000, actual value is %f\n\n", min);

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)

{

double stored[n];

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

{

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

}

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

{

x[i]=stored[i];

}

}

int increasing(const double\* x, size\_t n)

{

if(n==1)

{

return 1;

}

int flag[n];

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

{

if (x[i]>x[i-1])

{

flag[i-1]=1;

}

else

{

flag[i-1]=0;

}

flag[n-1]=1;

}

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

{

if (flag[i] == 0) //if any flag array value is 0, break and return false

{

return 0;

}

else if (flag[i] == 1 && i == n-1) // if last flag array value is one, return true

{

return 1;

}

}

return 0;

}

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

{

double minimum = x[0]; //intialize minimum as first index of array (will change unless first index is minimum)

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

{

if (x[i] < minimum) //if next element is less than the first

{

minimum = x[i]; //replace minimum with this element and move on

}

}

return minimum; //return lowest element

}

//End of code

## **Sample Dialogue**

Will@Will-Laptop /cygdrive/c/users/will/documents/ENCM339/lab3

$ ./e.exe

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

Some tests of increasing ...

for these numbers: 1.50 1.25 2.50 3.50 4.50 5.50

expected return value is 0, actual value is 0

for these numbers: 1.25 2.50 3.50 4.50 5.50

expected return value is 1, actual value is 1

for these numbers: 1.25 2.50 3.50 4.50 5.50 5.25

expected return value is 0, actual value is 0

for these numbers: 1.00 2.00 3.00 3.00 4.00 5.00

expected return value is 0, actual value is 0

Four tests of minimum ...

for these numbers: -1.50 -1.25 -2.50 -2.00 -4.50 -5.50 -5.25

expected return value is -5.500000, actual value is -5.500000

for these numbers: 1.00 2.00 3.00 3.10 0.50 4.50

expected return value is 0.500000, actual value is 0.500000

for these numbers: -5.00 -1.00 20.00

expected return value is -5.000000, actual value is -5.000000

for these numbers: -5.00 5.00 -20.00

expected return value is -20.000000, actual value is -20.000000

# **Exercise F**

## **Code Listing**

// ENCM 339 Fall 2016 Lab 3 Exercise F

#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] = 40;

buffer[1] = 52;

buffer[2] = 50;

buffer[3] = 32;

buffer[4] = 124;

buffer[5] = 124;

buffer[6] = 32;

buffer[7] = 48;

buffer[8] = 41;

buffer[9] = 32;

buffer[10] = 38;

buffer[11] = 38;

buffer[12] = 32;

buffer[13] = 45;

buffer[14] = 51;

buffer[15] = 32;

buffer[16] = 104;

buffer[17] = 97;

buffer[18] = 115;

buffer[19] = 32;

buffer[20] = 97;

buffer[21] = 32;

buffer[22] = 118;

buffer[23] = 97;

buffer[24] = 108;

buffer[25] = 117;

buffer[26] = 101;

buffer[27] = 32;

buffer[28] = 111;

buffer[29] = 102;

buffer[30] = 32;

buffer[31] = 49;

buffer[32] = 46;

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

buffer[33] = 0;

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

return 0;

}

## **Sample Dialogue**

Will@Will-Laptop /cygdrive/c/users/will/documents/ENCM339/lab3

$ ./f.exe

The string in buffer is "(42 || 0) && -3 has a value of 1."

# **Exercise G**

## **Code Listing**

// ENCM 339 Fall 2016 Lab 3 Exercise G

#include <stdio.h>

#include <string.h>

int safecat(char \*dest, const char\* src, size\_t dest\_size);

// REQUIRES

// Array elements dest[0] ... dest[dest\_size-1] exist.

// dest points to the beginning of a C string.

// src points to the beginning of a C string.

// PROMISES

// If the sum of the length of the two strings are less than

// dest\_size, the string in the dest array is the concatenation

// of the original string in dest and the string from src, and

// the return value is the length of the new string.

//

// If not, the string in dest is the concatenation of the original

// string in dest and as many characters from src as possible,

// while leaving room for '\0' in dest[dest\_size-1], and the return

// value is -1.

int main(void)

{

int rv;

char buf[10];

buf[0] = '\0';

rv = safecat(buf, "", 10);

printf("buf contains \"%s\" and rv is %d (expect \"\" and 0).\n",buf, rv);

rv = safecat(buf, "abcde", 10);

printf("buf contains \"%s\" and rv is %d (expect \"abcde\" and 5).\n", buf, rv);

rv = safecat(buf, "fghi", 10);

printf("buf contains \"%s\" and rv is %d (expect \"abcdefghi\" and 9).\n", buf, rv); // Note correction in above line.

buf[0] = '\0';

rv = safecat(buf, "jkl", 10);

printf("buf contains \"%s\" and rv is %d (expect \"jkl\" and 3).\n",buf, rv);

rv = safecat(buf, "mnopqrs", 10);

printf("buf contains \"%s\" and rv is %d (expect \"jklmnopqr\" and -1).\n", buf, rv);

buf[0] = '\0';

rv = safecat(buf, "01", 10);

printf("buf contains \"%s\" and rv is %d (expect \"01\" and 2).\n",buf, rv); // Note correction in above line.

rv = safecat(buf, "2345", 10);

printf("buf contains \"%s\" and rv is %d (expect \"012345\" and 6).\n",buf, rv);

rv = safecat(buf, "6789ABCDEF", 10);

printf("buf contains \"%s\" and rv is %d (expect \"012345678\" and -1).\n", buf, rv);

return 0;

}

int safecat(char \*dest, const char\* src, size\_t dest\_size)

{

int destlen = strlen(dest);

int i = strlen(dest);

for (int j = 0;/\*src[j] != '\0' || \*/i<dest\_size-1;++j, ++i)

{

dest[i]=src[j];

}

dest[i]='\0';

//Return values work as intended

if (dest\_size < strlen(src)+destlen)

{

return -1;

}

else if (dest\_size >= strlen(src)+destlen)

{

return strlen(dest);

}

return 0;

}

## **Sample Dialogue**

Will@Will-Laptop /cygdrive/c/users/will/documents/ENCM339/lab3

$ ./g.exe

buf contains "" and rv is 0 (expect "" and 0).

buf contains "abcde" and rv is 5 (expect "abcde" and 5).

buf contains "abcdefghi" and rv is 9 (expect "abcdefghi" and 9).

buf contains "jkl" and rv is 3 (expect "jkl" and 3).

buf contains "jklmnopqr" and rv is 9 (expect "jklmnopqr" and -1).

buf contains "01" and rv is 2 (expect "01" and 2).

buf contains "012345" and rv is 6 (expect "012345" and 6).

buf contains "012345678" and rv is -1 (expect "012345678" and -1).