Started on	Friday, 2 August 2019, 1:52 PM
State	Finished
Completed on	Monday, 12 August 2019, 12:00 AM
Time taken	9 days 10 hours
Marks	8.00/8.00
Grade	**10.00** out of 10.00 (**100**%)

Information

Accelerometer Events

In order to monitor the treatment received by parcels sent by post, a company occasionally sends test packages between its branches. The test packages contain a linear accelerometer to measure the accelerations to which the parcel is being subjected. The accelerometer is sampled 100 times per second and the data is written to an SD card. Only the absolute value of the acceleration vector is recorded; it is a floating point value in the range 0.0 to 100.0 with units of m/sec^2. For timing purposes the first sample is taken to be at time *t* = 0 secs.

You have been asked to analyse a stream of recorded accelerations, printing a message whenever a given critical acceleration value is exceeded. The program also records and prints the peak acceleration and the time at which it occurred.

Because the accelerometer readings are noisy, it is first necessary to smooth the data somewhat. The algorithm we will use is to replace each input value with a value obtained from a weighted average of it and its two neighbours, using weights of 1:2:1. If *d* is the input data array and *s* is the smoothed data array, the formula is:

$$s_i = (d_{i-1} + 2d_i + d_{i+1})/4$$

The two endpoints, *i* = 0 and *i* = *n* - 1 are special cases, dealt with by repeating the endpoint value:

$$s_0 = (3d_0 + d_1)/4$$

and

$$s_{n-1} = (d_{n-2} + 3d_{n-1})/4$$

[A possibly interesting aside for signal-processing enthusiasts: repeated applications of this function increase the smoothing in a well-behaved way, approximating wider and wider Gaussian filtering. The quality of smoothing is much better than that obtained by a simple running mean.]

Once the data has been smoothed, your program will search through the smoothed data and print the times at which the critical acceleration value, defined by the line

```
#define CRITICAL_HIGH 9.81
```

is exceeded. Once the limit has been exceeded, printing should stop until the smoothed data value drops below CRITICAL_LOW, defined by

```
#define CRITICAL_LOW 5.0
```

Steps

The program should be written in three steps, as follows. The steps are defined in detail in the following questions.

1. Write a function *readDoubles* that reads a sequence of white-space-separated double-precision floating point numbers from standard input into a given data array. The function returns when all data has been read, indicated by an EOF (End of File) indication from *scanf.*
2. Write a function *smoothData* that smooths the data according to the above specification.
3. Write the complete program, which reads the data, smooths it, then prints the times when the smoothed data exceeds the defined critical value, plus the peak recorded acceleration and the time at which it occurred.

Marks

This is the first of two assignment superquizzes for the C section of ENCE260. It contributes 4% towards your total grade for the course. The second one will contribute 6%. These numbers are slightly different from the values announced in the first lecture.

Style

Just because you're writing C doesn't mean you can forget all you learn in COSC121 about writing nice code! Please include a comment at the start of each function stating what it does. At the start of the final program (step 3) you must explain what the whole program does and also include your name and the date.

All code must be properly laid out in the 1TBS way, as in the labs. Choose good identifiers for variables and functions.

Important Note

By submitting your super-quiz answers you confirm that they are entirely your own work. Originality detection software will be used to compare your solution with other solutions. Dishonest practice, which includes:

- letting someone else create all or part of an item of work,
- copying all or part of an item of work from another person with or without modification, and
- allowing someone else to copy all or part of an item of work,

may lead to partial or total loss of marks, no grade being awarded and other serious consequences including notification of the University Proctor.

You are encouraged to discuss the general aspects of a problem with others. However, anything you submit for credit must be entirely your own work and not copied, with or without modification, from any other person. If you need help with specific details relating to your work, or are not sure what you are allowed to do, contact your tutors or lecturer for advice.

If you copy someone else's work or share details of your work with anybody else, you are likely to be in breach of university regulations and the Computer Science and Software Engineering department's policy. For further information please see

- Section J of the university's General Course and Examination Regulations in the University Calendar, and
- Academic Integrity Guidance for Staff and Students.

Question **1**

Correct

Mark 2.00 out of 2.00

Step 1: function *readDoubles*

In this step you are to write a general purpose function *readDoubles* that uses *scanf* to read a sequence of double-precision floating point numbers from standard input into a given array until the maximum number is reached or EOF occurs, whichever comes first. The function signature is:

```
int readDoubles(int n, double data[])
```

where *data* is an array of size *n* into which the numbers are read, *n* > 0. Each line of the input file can be assumed to contain zero or more valid floating point numbers separated by white space so they can be read by *scanf*. The definition of a valid floating point number is "anything that scanf will read as a double precision float", so it includes integers. You should not check the values you read against the limits of the value output by the accelerometer; this is a general-purpose function, not specific to the accelerometer exercise.

Numbers should be read until either *n* values have been read or EOF has occurred. Note that EOF is signalled by the return value of *scanf*. **This is the scanf function's return value, not the value that it returns via the pointer parameter.**

The return value from the function is the number of double-precision floating point numbers read.

Hint:

You should test your function by writing a small program with a *main* function that calls *readDoubles* . You should create test data files, e.g. *test1.txt*, containing test data and run your program (called *testreaddoubles,* say) with a command of the form

```
./testreaddoubles <test1.txt
```

This will allow you to test whether you are handling the EOF situation correctly.

For example:

Test	Input	Result

`double data[5] = {0.0};`
`int numRead = 0;`
 	
`numRead = readDoubles(4, data);`
`printf("Read %d values:\n", numRead);`
`for (int i = 0; i < numRead; i++) {`
`  printf("%0.3lf\n", data[i]);`
`}`
	`11.51 -93.2 0`
	`-1`
	`123.56`
		`Read 4 values:`
		`11.510`
		`-93.200`
		`0.000`
		`-1.000`

`double data[6] = {0.0};`
`int numRead = 0;`
 	
`numRead = readDoubles(6, data);`
`printf("Read %d values:\n", numRead);`
`for (int i = 0; i < numRead; i++) {`
`  printf("%0.3lf\n", data[i]);`
`}`
	`11.51  -93.2  0  -1`
	
	`123.567`
		`Read 5 values:`
		`11.510`
		`-93.200`
		`0.000`
		`-1.000`
		`123.567`

`double data[1] = {0.0};`
`int numRead = 0;`
 	
`numRead = readDoubles(1, data);`
`printf("Read %d values:\n", numRead);`
`for (int i = 0; i < numRead; i++) {`
`  printf("%0.3lf\n", data[i]);`
`}`
	`11.51`
	
	`-93.2`
	
	`  0  -1`
	`123.567`
		`Read 1 values:`
		`11.510`

Answer: (penalty regime: 0, 10, 20, ... %)

Ace editor not ready. Perhaps reload page?
Falling back to raw text area.

```
int readDoubles( int n, double data[]) //take in integer and array data in format double
{
    //a for loop with max n iterations, fist check if input is a double type only if it
is a double, counter n
    // if n condition is broken or EOF terminate for loop
    //can EOF be stored as a value
    int counter = 0;
    for (counter = 0; (counter < n); counter++) {
        double readvalue = 0;
        int truth = 0;
        truth = scanf("%lf", &readvalue);
        if ( truth != EOF ) {
            //the input is invalid as it is EOF
            //the other exit condition will be hit by the for loop
            data[counter] = readvalue;
        } else {
            return counter;
        }
    }
    return counter;

}
```

	Test	Input	Expected	Got	

✔	`double data[5] = {0.0};`
	`int numRead = 0;`
	
	`numRead = readDoubles(4, data);`
	`printf("Read %d values:\n",`
	`numRead);`
	`for (int i = 0; i < numRead; i++) {`
	`  printf("%0.3lf\n", data[i]);`
	`}`
		`11.51 -93.2 0`
		`-1`
		`123.56`
			`Read 4`
			`values:`
			`11.510`
			`-93.200`
			`0.000`
			`-1.000`
				`Read 4`
				`values:`
				`11.510`
				`-93.200`
				`0.000`
				`-1.000`
					✔

✔	`double data[6] = {0.0};`
	`int numRead = 0;`
	
	`numRead = readDoubles(6, data);`
	`printf("Read %d values:\n",`
	`numRead);`
	`for (int i = 0; i < numRead; i++) {`
	`  printf("%0.3lf\n", data[i]);`
	`}`
		`11.51  -93.2  0`
		`-1`
		
		`123.567`
			`Read 5`
			`values:`
			`11.510`
			`-93.200`
			`0.000`
			`-1.000`
			`123.567`
				`Read 5`
				`values:`
				`11.510`
				`-93.200`
				`0.000`
				`-1.000`
				`123.567`
					✔

✔	`double data[1] = {0.0};`
	`int numRead = 0;`
	
	`numRead = readDoubles(1, data);`
	`printf("Read %d values:\n",`
	`numRead);`
	`for (int i = 0; i < numRead; i++) {`
	`  printf("%0.3lf\n", data[i]);`
	`}`
		`11.51`
		
		`-93.2`
		
		`0  -1`
		`123.567`
			`Read 1`
			`values:`
			`11.510`
				`Read 1`
				`values:`
				`11.510`
					✔

Passed all tests! ✔

Correct

Marks for this submission: 2.00/2.00.

Question **2**

Correct

Mark 2.00 out of 2.00

Function *smoothData*

In this step you are to write a function *smoothData* with signature

```
void smoothData(int n, double data[])
```

that smooths the given *n*-element array *data* **in place**, using the 1:2:1 filter explained in the introduction:

$$s_i = (d_{i-1} + 2d_i + d_{i+1})/4$$

$$s_0 = (3d_0 + d_1)/4$$

$$s_{n-1} = (d_{n-2} + 3d_{n-1})/4$$

Because the smoothing is done on the actual array itself, in place, you will need to be careful that you don't overwrite a data value that you need in the next step. For example, if the data is

... 3.0, 5.0, 4.0, 6.0 ...

the 5.0 needs to be replaced with (3.0 + 2 * 5.0 + 4.0) /4, which is 4.25. However, when computing the next value in the sequence to replace the value 4.0, the required expression is (5.0 + 2 * 4.0 + 6.0) / 4 which is 4.5, **not** (4.25 + 2 * 4.0 + 6.0) which is what you get if you have already replaced the 5.0 with 4.25.

You *could* avoid that overwritten-value problem by generating an entire new array, then copying it back over the original array, but that is not the C way! Please find a solution that does not involve generating a new array.

You may assume that *n* will be greater than or equal to 3.

The variable names s0, si, d0, and di are allowed in this question.

For example:

Test	Result

`double data[] = {1.0, 2.0, 4.0, 4.0};`
`smoothData(4, data);`
`printf("Smoothed data: %.3lf", data[0]);`
`for (int i = 1; i < 4; i++) {`
`  printf(", %.3lf", data[i]);`
`}`
`puts("");`
	`Smoothed data: 1.250, 2.250, 3.500, 4.000`

Answer: (penalty regime: 0, 10, 20, ... %)

Ace editor not ready. Perhaps reload page?

Falling back to raw text area.

```
void smoothData(int n, double data[])
{
    double current = data[0];
    double previous = 0;
    for (int counter = 0; counter < n; counter++) {
        current = data[counter];
        if (counter == 0) {
            previous = current;
            data[counter] = (((3 * current) + data[counter + 1]) / 4);
        } else if (counter == (n - 1)) {
            //last number
            data[counter] = ((previous + (3 * data[n - 1])) / 4);
            previous = current;
        } else {
            //normal case
            data[counter] = ((previous + (2 * data[counter]) + data[counter + 1]) / 4);
            previous = current;
        }
    }


}
```

	Test	Expected	Got	

✔	`double data[] = {1.0, 2.0, 4.0, 4.0};`
	`smoothData(4, data);`
	`printf("Smoothed data: %.3lf", data[0]);`
	`for (int i = 1; i < 4; i++)`
	`{`
	`  printf(", %.3lf", data[i]);`
	`}`
	`puts("");`
		`Smoothed data: 1.250, 2.250, 3.500, 4.000`	`Smoothed data: 1.250, 2.250, 3.500, 4.000`	✔

Passed all tests! ✔

Correct

Marks for this submission: 2.00/2.00.

Question **3**

Correct

Mark 4.00 out of 4.00

The final program

In this final step, you must write a program that calls the functions *readDoubles* and *smoothData* and which then prints the following:

1. A sequence of lines showing the times of all critical threshold crossings and their times, in the following format. The definition of a critical threshold crossing is given below.
   ```
   Acceleration of 9.81 m/sec^2 exceeded at t = 23.54 secs.
   ```
2. A blank line.
3. The maximum acceleration, and the time at which it **first** occurred, in the following format:
   ```
   Maximum acceleration: 23.18 m/sec^2 at t = 39.35 secs.
   ```

Paste your entire working program including support functions into the answer box.

Notes:

1. Notice the full stops at the ends of the output lines.
2. All printed output relates to the data *after smoothing.*
3. All floating point numbers are printed to two decimal place.
4. There are *two* floating point numbers to be formatted into the maximum acceleration line but it is acceptable to hard code the 9.81 into the other acceleration lines if you wish (though explicitly formatting the value of CRITICAL_HIGH would be more elegant).
5. The accelerometer is sampled 100 times per second.
6. You may assume that the input file contains only valid data and that there are at most 100,000 data points. [Unless you use dynamic memory - taught in the last week of term - you need to know the maximum input data array size in order to reserve space for it.]

Critical threshold crossing

A critical threshold crossing in the smoothed data, referred to as the *signal s,* is defined by two constants:

```
#define CRITICAL_HIGH 9.81
```

and

```
#define CRITICAL_LOW 5.0
```

A critical threshold crossing occurs whenever the signal first exceeds the critical high level after having previously been below the critical low value. You may assume that s_0 is less than the critical low value. In other words, once the critical high value is exceeded, no further critical high crossings can occur until the signal has dropped below the critical low value.

More formally, if we let C_H and C_L denote the critical high and low values respectively, a critical threshold crossing occurs at time t_i if and only if $s_i > C_H$ and there exists $j < i$ such that $s_j < C_L$ and $C_L \leq s_k \leq C_H$ for all $j < k < i$.

For example:

Input	Result
0.0 0.0 0.0 9.82 9.82 9.82 0.0 0.0 0.0 2.0 1.75 2.0 22.5 0.0 19.125 23.875 19.5 11.25	Acceleration of 9.81 m/sec^2 exceeded at t = 0.04 secs. Acceleration of 9.81 m/sec^2 exceeded at t = 0.12 secs. Maximum acceleration: 21.59 m/sec^2 at t = 0.15 secs.

Answer: (penalty regime: 0, 10, ... %)

Ace editor not ready. Perhaps reload page?
Falling back to raw text area.

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

#define CRITICAL_HIGH 9.81
#define CRITICAL_LOW 5.0

void smoothData(int n, double data[])
{
    double current = data[0];
    double previous = 0;
    for (int counter = 0; counter < n; counter++) {
```

```
        current = data[counter];
        if (counter == 0) {
            previous = current;
            data[counter] = (((3 * current) + data[counter + 1]) / 4);
        } else if (counter == (n - 1)) {
            //last number
            data[counter] = ((previous + (3 * data[n - 1])) / 4);
            previous = current;
        } else {
            //normal case
            data[counter] = ((previous + (2 * data[counter]) + data[counter + 1]) / 4);
            previous = current;
        }
    }
}

int readDoubles( int n, double data[]) //take in integer and array data in format
double
{
    //a for loop with max n iterations, fist check if input is a double type only if it
is a double, counter n
    // if n condition is broken or EOF terminate for loop
    //can EOF be stored as a value
    int counter = 0;
    for (counter = 0; (counter < n); counter++) {
        double readvalue = 0;
        int truth = 0;
        truth = scanf("%lf", &readvalue);
        if ( truth != EOF ) {
            //the input is invalid as it is EOF
            //the other exit condition will be hit by the for loop
            data[counter] = readvalue;
        } else {
            return counter;
        }
    }
    return counter;

}


int main(void)
{
    //have switch = true if acceleration exceeded --> print message --> do nothing till
drop below critical low
    //then enable printing message again
    //have 2 checks if critical low hit and is critical high
    //prevent message spamming
    //run smoothing in for loop for every 3 values given
    //use readdoubles to check for eof?
    double data[100000];
    int ValidInput = 0;

    ValidInput = readDoubles(100000, data);
    smoothData(ValidInput, data);
    //int High_hit = 0;
    int Low_hit = 1;
    double MaxValue = 0;
    double MaxTime = 0;
    for (int i = 0; i < ValidInput; i++)

    {
        double smoothed_value;
        smoothed_value = data[i];
        if (MaxValue < data[i]) {
            MaxValue = data[i];
            MaxTime = i;
        }
        //case1 hit critical high
```

```
        if ((smoothed_value > CRITICAL_HIGH) && (Low_hit == 1)) {
            double j = i;
            printf("Acceleration of 9.81 m/sec^2 exceeded at t = %.2lf secs.\n",
j/100.0);
            //change Low_hit to false
            Low_hit = 0;
            //High_hit = 1;
        } else if ((smoothed_value > CRITICAL_HIGH) && (Low_hit == 0)) {
            //i have already exceeded but havent hit low so just continue
        } else if((smoothed_value < CRITICAL_LOW) && (Low_hit == 0)) {
            //i hit low so reset switch
            Low_hit = 1;
            //High_hit = 0;
        }
        //int High_hit = 0;
        //int Low_hit = 1;)

    }
    printf("\nMaximum acceleration: %0.2lf m/sec^2 at t = %0.2lf secs.", MaxValue,
MaxTime/100.0);
}
```

	Input	Expected	Got	
✔	0.0 0.0 0.0 9.82 9.82 9.82 0.0 0.0 0.0 2.0 1.75 2.0 22.5 0.0 19.125 23.875 19.5 11.25	Acceleration of 9.81 m/sec^2 exceeded at t = 0.04 secs. Acceleration of 9.81 m/sec^2 exceeded at t = 0.12 secs. Maximum acceleration: 21.59 m/sec^2 at t = 0.15 secs.	Acceleration of 9.81 m/sec^2 exceeded at t = 0.04 secs. Acceleration of 9.81 m/sec^2 exceeded at t = 0.12 secs. Maximum acceleration: 21.59 m/sec^2 at t = 0.15 secs.	✔

Passed all tests! ✔

Correct

Marks for this submission: 4.00/4.00.

◄ Quiz6: Dynamic Memory Jump to... Superquiz 2 ►