Duration: *50 minutes* (11:10am–12:00pm)

Aids Allowed: *One <u>non</u>-programmable calculator.*

Student Number: _____

Last Name: _____

First Name: _____

Tutorial Section: (circle one) SS 1083 (Bijan) SS 1074 (Kiran) SS 2111 (Hojjat) SS 2130 (Jerry) LM 155 (Mike)

---

*Do **not** turn this page until you have received the signal to start.*
(In the meantime, please fill out the identification section above, and read the instructions below *carefully*.)

This test consists of 4 questions on 5 pages (including this one). *When you receive the signal to start, please make sure that your copy of the test is complete.* Answer each question directly on the test paper, in the space provided, and *use the reverse side of the pages for rough work.* (If you need more space for one of your solutions, use the reverse side of the page and indicate **clearly** which part of your work should be marked.)

Be aware that concise, well thought-out answers will be rewarded over long rambling ones. Also, unreadable answers will be given zero (0) so write legibly. In your answers, you may use without justification any facts given during the course, *as long as you state them clearly*. You must justify any other facts needed for your solutions.

General Hint: We were careful to leave ample space on the test paper to answer each question, so if you find yourself using much more room than what is available, you're probably missing something. Also, remember that hints are just hints: you are not required to follow them!

1: _____/10

2: _____/15

3: _____/15

4: _____/10

TOTAL: _____/50

Good Luck!

Question 1. [10 MARKS]

Part (a) [2 MARKS]

Write your student number **legibly** at the top of every page of this test, *except page 1* (on page 1, write your student number only where we ask for it).

Part (b) [4 MARKS]

What is the *smallest* positive number that can be represented exactly with a floating-point system in base 10, with 2 digits for the exponent and 5 digits for the mantissa, if *all* floating-point numbers are normalized (irrespective of the value of the exponent)? Show your work.

Part (c) [4 MARKS]

What is the *largest* possible round-off error when representing a real number using the floating-point system described in part (b), if the number to be represented does not exceed the largest floating-point number? Give an explicit example of a real number that achieves this error.

Question 2. [15 MARKS]

In this question, you will use the *bisection* method to find a root of the function $f(x) = 16^x - 10$.

Part (a) [3 MARKS]

Find two real numbers $a < b$ such that $f(a) < 0$ and $f(b) > 0$. What can you conclude about the real number $\log_{16} 10$, and why?

Part (b) [4 MARKS]

Simulate the bisection method by hand for *two (2)* steps, starting from the interval $[a, b]$ you obtained in part (a) for the function $f(x)$. Show your work, including the interval obtained after each step.

Question 2. (CONTINUED)

Part (c) [4 MARKS]

After 2 steps, what value would the bisection method return as an approximate root of $f(x)$, starting from the interval $[a, b]$ you found in part (a)? Give a bound on the error for this value (justify briefly).

Part (d) [4 MARKS]

Given your initial interval $[a, b]$ from part (a), how many iterations of the bisection method would be necessary to guarantee that your answer is accurate to two (2) decimal places? Justify and show your work.

Question 3. [15 MARKS]

Complete the C function `bisection` in the program below. (The notation "`x^y`" is used to denote x^y in the comments.) Note that you will be marked on the correctness of your idea as well as on the correctness of your code.

```
#include <math.h>

/* Computes and returns the value of the function f(x) = 16^x - 10. */
float f(float x) { return pow(16,x) - 10; }

/*
** Computes and returns an approximate root of the function f(x) = 16^x - 10
** using the bisection method, starting from the given initial interval [a,b]
** and stopping when the error becomes smaller than the specified tolerance
** parameter tol.  It is *not* necessary to keep track of the number of
** iterations performed, and you can make calls to the C function 'f()'
** defined above where appropriate.
*/
float bisection(float a, float b, float tol) {









} /* END bisection() */
```

Question 4. [10 MARKS]

Complete the C program below by writing code where indicated.

```
#include <stdio.h>
#define N  10

/*
** Computes the sum and the product of the elements of array 'a'.  'n' is the
** number of elements in 'a', and '*s' and '*p' should be used to store the
** sum and product, respectively (the sum in '*s' and the product in '*p').
*/
void sum_prod(float a[], int n, float *s, float *p) {
   /* COMPLETE THE BODY OF THIS FUNCTION. */









} /* END sum_prod() */

int main() {
   int array[N], sum, product, i;

   printf("Enter %d numbers: ", N);
   for (i = 0; i < N; i++)  scanf("%d", &array[i]);

   /* WRITE A SINGLE LINE OF CODE THAT CALLS FUNCTION 'sum_prod' TO FIND THE
      SUM AND PRODUCT OF THE ELEMENTS IN 'array' AND STORE THE RESULTS IN THE
      TWO VARIABLES 'sum' AND 'product'. */




   printf("Sum = %d\nProduct = %d\n", sum, product);
   return 0;
} /* END main() */
```