PRINCE OF SONGKLA UNIVERSITY

FACULTY OF ENGINEERING

Department of Computer Engineering

Midterm Examination: Semester 1 **Academic Year**: 2007-2008

Date: 5th August 2007 **Time**: 9:00 – **11:00 (2 hours)**

Subject Number: 240-304 **Room**: R200

Subject Title: Mathematics for Computer Engineering

Lecturer: Aj. Andrew Davison

Exam Duration: 2 hours

This paper has 4 pages.

Authorized Materials:

- Writing instruments (e.g. pens, pencils).
- Books (e.g. dictionaries) and calculators are **not** permitted.

Instructions to Students:

- *Answer questions in English.* Perfect English is **not** required.
- Attempt all questions.
- Write your answers in an answer book.
- Start your answer to each question on a new page
- Clearly number your answers.
- Any unreadable parts will be considered wrong.
- When writing programs, use good layout, and short comments; marks will not be deducted for minor syntax errors.
- The marks for each part of a question are given in brackets (...).

Question 1 (25 minutes; 25 marks)

Use induction to show that each equation is true:

a) $n! >= 2^n$, when $n >= 4$ (10)

b) $1 + 4 + 7 + ... + (3n - 2) = n(3n - 1)/2$, when $n >= 1$ (15)

Question 2 (15 minutes; 15 marks)

Consider the following C fragment:

```
scanf("%d", &n);
sum = 0;
for (i = 1; i <= n; i++)
  sum = sum + i;
```

The loop invariant S(k) is $sum_k = i_k(i_k - 1)/2$, where sum_k and i_k are the values of `sum` and `i` after k iterations of the loop. Assume that `n` is a positive integer.

a) Prove that the loop invariant is correct, by induction on k. (10)

b) What is the value of `sum` after the loop terminates? Explain your answer. (5)

Question 3 (45 minutes; 45 marks)

A *Menger Sponge* is a fractal shape based around squares.

The sponge begins as a single square, with its lower left corner drawn at the point (x,y) with sides of length size.

In the first step of a sponge's creation, a small square is drawn in the center of the large one, with its lower left corner at (x+size/3, y+size/3), with sides of length size/3 (see Figure 1).

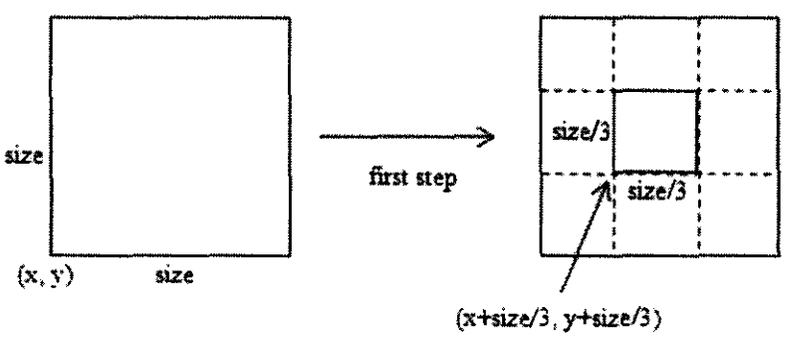

Figure 1. First Step in Generating a Menger Sponge.

The dotted squares in Figure 1 are **not** part of the drawing, but show how the large square is divided up.

The next step applies the Menger algorithm to the eight dotted squares around the center square, drawing a smaller square in the center of each one, as in Figure 2.

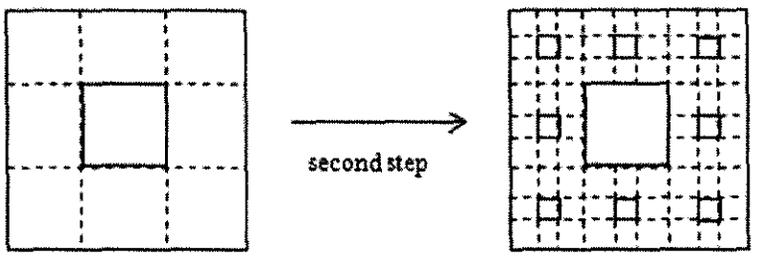

Figure 2. Second Step in Generating a Menger Sponge.

The dotted squares in Figure 2 are **not** part of the drawing, but show how the dotted squares are divided up into smaller squares.

The recursive drawing continues by drawing smaller squares in the centers of the dotted squares.

Assume that you have two drawing functions available in C:

```
void setPen(double x, double y);
void drawLine(double xDistance, double yDistance);
```

setPen() moves the drawing pen to (x,y) without drawing anything. drawLine() draws a line from the current pen position, moving the pen xDistance in the x- direction, yDistance in the y- direction. The x- axis is across the screen, the y-axis is straight up.

a) Write a C function:

```
void drawSquare(double x, double y, double size);
```

It draws a square with its lower left corner at the point (x,y) with sides of length size. Use **only** setPen() and drawLine() to implement the function.
Do **not** implement setPen() or drawLine(). (10)

b) Write the C function:

```
void menger(double x, double y, double size);
```

It uses **recursion** to draw the squares making up the Menger Sponge. It uses drawSquare() from part(a) to draw each square. (25)

menger() does not draw a square if its side length is less than 1 unit.

The recursive drawing is started from main() with:

```
void main()
{ double x = 0, y = 0, size = 80;
  drawSquare(x, y, size);    // initial large square
  menger(x, y, size);        // smaller squares
}
```

c) Draw a *simple* diagram showing how menger() executes when it is drawing the first two steps in the Menger Sponge. The diagram should include the memory used by each function call. Explain the diagram in words. (10)

Question 4 (35 minutes; 35 marks)

The i^{th} prefix average of a data[] array is the average of the first (i+1) elements of data[]. The average is stored in the i^{th} cell of the prefixAvgs[] array:

prefixAvgs[i] = (data[0] + data[1] + ... + data[i]) / (i+1);

A C function which fills in the prefixAvgs[] array:

```
void prefixAverages(int data[], int n, double prefixAvgs[])
{
  double sum;
  int i, j;

  for(i=0; i < n; i++) {
    sum = data[0];
    for(j=1; j <= i; j++)
      sum += data[j];
    prefixAvgs[i] = ((double) sum) / (i+1);
  }
}
```

a) Work out the worst case big-oh running time for prefixAverages(). Show all your working. (15)

b) Rewrite the prefixAverages() function to have a **linear** worse case big-oh running time (i.e. O(n)). Write down the function **and** calculate the worst case big-oh running time. Show all your working. (15)

c) Explain in words why the code in part (a) is slower than the version you wrote in part (b). You should refer to the parts of prefixAverages() which cause the slowness. (5)

--- End of Examination ---