Carleton University
Department of Systems and Computer Engineering
SYSC 2006 – Foundations of Imperative Programming – Summer 2014

Section A – Midterm Exam

Instructions

1. The TAs will not answer queries about this question paper. Exam questions will not be explained, and no hints will be given. If you think something is unclear or ambiguous, make a reasonable assumption (one that does not contradict the question), write it at the start of your solution, and answer the question.
2. You do not need to write header comments or inline comments for your functions. You do not need to copy any code from the question paper into your answer booklet.

PART A: Programming

Question 1 [9 marks]

Write a C function, `is_ascending`, which checks if an array of n integers is an ascending sequence (a sequence is ascending if each number in the sequence is larger than or equal to the number before it).

If the sequence is ascending, return true. If the sequence is not ascending, return false.

For example, if a caller passes the array [1,2,3,4,2,6], the function will return false. If a caller passes the array [2,4,5,7,7,7,8,12], the function will return true.

The function must have the following signature:

```
_Bool is_ascending(int seq[], int n)
```

Mark breakdown

Overall function stuff (declaration, arguments, return value)	/1
Proper use of loops (correct number of iterations)	/1
Algorithm determines ascendingness of sequence	/4
Demonstrate knowledge of arrays	/2
Demonstrate knowledge of basic C syntax	/1

Sample Solution

```
_Bool is_ascending(int seq[], int n)
{
	for (int i = 0; i < n-1; i++)
	{
		if (seq[i+1] < seq[i])
		{
			return 0;
		}
	}
	return 1;
}
```

Question 2 [8 marks]

Write a function which calculates the first *n* terms of the following formula, and returns the result of the calculation:

$$f(x,y,z) = 1 + \frac{x^{3y}}{z+1} - \frac{x^{4y}}{z+3} + \frac{x^{5y}}{z+5} - \frac{x^{6y}}{z+7} + \cdots$$

The function signature is:

```
double midterm_sum(int n, double x, double y, int z);
```

where arguments *x*, *y*, and *z* represent *x*, *y*, and *z* in the given formula. The first *n* terms of the formula are calculated by your function. For example, for n=3, you will calculate and return the following:

$$1 + \frac{x^{3y}}{z+1} - \frac{x^{4y}}{z+3}$$

C provides a function `double pow(double a, double b);` declared in `math.h`, which returns the result of the calculation a^b. You have access to this function.

You can assume that n will always be >= 1.

Mark breakdown

Overall function stuff (declaration, arguments, return value)	/1
Function call (to pow()) used properly	/1
Proper use of loops (correct number of iterations)	/1
Algorithm calculates correct value at each iteration	/3
Algorithm calculates alternating +/-	/1
Demonstrate knowledge of basic C syntax	/1

Sample Solution

```
double midterm_sum(int n, double x, double y, int z)
{
	double sum = 1;
	int a, b;
	double term;
	for (int i = 0; i < n-1; i++)
	{
		a = i+3;
		b = 2*i + 1;
		term = pow(x, y*a) / (z+b);
		if (i%2 == 0)
		{
			sum += term;
		}
		else
		{
			sum -= term;
		}
	}
	return sum;
}
```

Question 3 [13 marks]

Write a C function which calculates and returns the average value and the standard deviation of an array of double values.

In order to return both the average value and standard deviation from the same function, the two values must be grouped together as members of the following C structure:

```
struct avg_std {
     double avg;
     double std;
};
```

Therefore, your function should have the following signature:

```
struct avg_std average_and_std(double * data, int n);
```

Where `data` points to an array of double values, and `n` represents the number of values in the array pointed to by data.

During the lectures, we looked at a function which calculates the average value of an array of numbers. You can use this function within your `average_and_std` function, rather than calculating the average value of an array yourself. The average function from the lectures has the following signature:

```
double average(double *data, int n);
```

You will need to calculate the standard deviation yourself. The formula for standard deviation is as follows:

$$\sqrt{\frac{\sum_{i=0}^{n-1}[(x_i - \mu)^2]}{n}}$$

Where x_i is the ith element of the array, μ is the average value of the array, and n is the number of elements in the array.

You can use the `math.h` library function `double sqrt(double x)` in order to calculate the square root.

Mark breakdown

Overall function stuff (declaration, arguments, return value)	/1
Function completes overall goal (save average, save std, return both)	/3
Function calls (to average() and sqrt()) used properly	/1
Proper use of loops (correct number of iterations)	/1
Struct properly initialized and used	/3

Algorithm calculates correct standard deviation value	/3
Demonstrate knowledge of basic C syntax	/1

Sample Solution

```
struct avg_std average_and_std(double * data, int n)
{
	struct avg_std result;
	double std = 0.0;
	double avg = average(data, n);

	for (int i = 0; i < n; i++)
	{
		std += pow(data[i]-avg, 2);
	}

	std = std / n;
	std = sqrt(std);

	result.avg = avg;
	result.std = std;
	return result;
}
```

PART B: Tracing Code using Memory Diagrams

Question 4 [10 marks]

Rob wrote this program. It's pretty great.

```
////////////////////////////////////////////
// BEST_PROGRAM_EVER
// Does awesome things.
// By: Rob.
////////////////////////////////////////////

int best_function_ever(int * x, int * y)
{
      int result;
      if (*x > 6) /* POINT A */
      {
            *x = *x - 2;
      }
      else
      {
            *x = *x + 2;
      }
      result = *x - *y;
      return result;
}

// Continued on the next page…
```

```
int main()
{
      int a, b;
      int * pa, pb;
      a = 2;
      b = 5;
      pa = &a;
      pb = pa;
      pa = &b;
      best_function_ever(pa, pb); /* POINT B */
      return 0;
}
```

Using the notation presented in lectures, draw two separate memory diagrams, one for question a) and one for question b). Show all activation records for both a) and b); do not erase the activation records for functions which have returned.

a) Draw a memory diagram showing the state of memory immediately *before* the line at Point A is executed within `best_function_ever` (after the function `best_function_ever` has been called, right before the line `if (*x > 6);` is executed).

b) Draw a memory diagram showing the state of memory immediately *after* the line at Point B is executed within `main` (after `best_function_ever` has returned, but before the final return statement of the program).