Carleton University
Department of Systems and Computer Engineering
SYSC 2006 - Foundations of Imperative Programming - Fall 2014

Section A - Midterm Exam

Instructions

1. This exam is closed book. Calculators are permitted.
2. 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. Please, do not ask the TAs questions during the exam.
3. Answer the questions in your answer booklet. You can write your solutions in pen or pencil.
4. You do not need to copy any code from the question paper to your answer booklet. You do not have to write comments in your C code; however, feel free to add comments to clarify anything you think requires further explanation. Please don't write comments that state the obvious; for example, there is no need for comments similar to this one:

   ```
   x = x + 1; // Add 1 to x
   ```

5. You can take this question paper with you when you leave the exam room.

PART A: Programming

Question 1 [4 marks]

Here is the declaration of a C structure that stores the surface area and volume of a 3-dimensional object::

```
struct area_vol {
    double area;
    double volume;
};
typedef struct area_vol area_vol_t;
```

A right cone has radius *r* and height *h*. The volume and surface area of a right cone are:

$$volume = \frac{1}{3}\pi r^2 h$$

$$area = \pi r\sqrt{r^2 + h^2}$$

Write the definition of a function named `cone_area_volume` that is passed the radius and height of a right cone. The function returns a structure containing the cone's surface area and volume. The function prototype is:

```
area_vol_t cone_area_volume(double radius, double height);
```

Use 3.1415 as the value of π. C's math library (`math.h`) has two functions that may be of use:

- `double pow(double x, double y);` returns the value of `x` raised to the power of `y`.
- `double sqrt(double x);` returns the square root of `x`.

You do not have to write `#include` statements in your solution.

Question 2 [11 marks]

(a) (1 mark) Here is the C statement that outputs an integer variable `k`, followed by the newline character (`'\n'`), which moves the cursor to the start of the next line:

```
printf("%d\n", k);
```

Here is a C statement that initializes an array named `samples` with six integers:

```
int samples[] = {5, 3, 9, 1, 7, 12};
```

Write the C statement that outputs the last integer (`12`) from array `samples`, followed by newline. Don't write a complete function; instead, just write the one C statement that is required.

(b) (2 marks) Assume that an array named `elems` has been initialized with `n` integers.Write the C statements that output all the integers in the array on one line. Don't write a complete function. Don't write the statements that declare and initialize `elems`.

(c) (2 marks) An array named `values` contains `n` integers. Write the C statements that calculate and output the number of even integers in the array. For example, if `values` and `n` are initialized this way:

```
int values[] = {6, 3, 9, 2, 2, 7, 10};
n = sizeof(values) / sizeof(int); // n will be assigned 7
```

your code should output: `4`. Your code can assume that `values` will always contain at least one integer. Don't write a complete function; instead, just write the C statements that are required. Hint: the expression `a % b` yields the remainder when `a` is divided by `b`.

(d) (6 marks) Write a function named `even_odd` that is passed an array containing `n` integers. The function prototype is:

```
_Bool even_odd(int arr[], int n);
```

The function returns `true` if every even integer in the array is followed by one or more odd integers; otherwise it returns `false`. Assume that parameter `n` will always be positive. (Your function does not have to check this).

For example:

- If `arr` contains `{6, 3, 4, 7}` the function returns `true`.
- If `arr` contains `{9, 6, 3, 4, 7}` the function returns `true`.
- If `arr` contains `{6, 3, 4, 8}` the function returns `false` (`4` is followed by `8`).
- If `arr` contains `{6, 3, 1, 4, 7}` the function returns `true`.
- If `arr` contains `{6, 3, 4}` the function returns `false` (`4` is not followed by an odd integer).

PART B: Tracing Code/Memory Diagrams

Question 3 [7 marks]

```
#include <stdlib.h>

int mystery(int *ptr, int f)
{
    int k;
    k = 3 + *ptr;
    *ptr = k + f;    /* Point A */
    f = f + 2;
    return k;

}

int main(void)
{
    int f = 4;
    int k = 2;

    int result = mystery(&f, k);  /* Point B */
    exit(0);
}
```

Using the notation presented in lectures, draw two separate memory diagrams, one each for parts (a) and (b). *Do not combine your solutions into a single diagram.* Do not use arrows to depict the flow of data into and out of variables. Remember, arrows are only used to depict pointers.

(a) Draw a diagram that depicts the program's activation frame(s) immediately **after** the statement at Point A is executed; that is, immediately after `*ptr = k + f;` is executed, but before `f = f + 2;` is executed.

(b) Draw a diagram that depicts the program's activation frame(s) immediately **after** the statement at Point B is executed; that is, immediately after `int result = mystery(&f, k);` has been executed, but before `exit` is called.

Question 4 [8 marks]

```
#include <stdlib.h>

void strange(int list[], int n)
{
    int *p = list;
    int i;

    /* Point A */
    for (i = 0; i < n; i = i + 2) {
        if (p[1] < *(list + i)) {
            p[1] = p[1] * 2;
        }
        p = p + 2;
    }
    /* Point B */
    return;
}

int main(void)
{
    int arr[] = {5, 3, 6, 7, 9, 2};
    strange(arr, 6);
    exit(0);
}
```

Using the notation presented in lectures, draw two separate memory diagrams, one each for parts (a) and (b). *Do not combine your solutions into a single diagram.* Do not use arrows to depict the flow of data into and out of variables. Remember, arrows are only used to depict pointers.

(a) Draw a diagram depicting this program's activation frame(s) when execution reaches Point A; that is, just **before** the `for` loop is executed.

(b) Draw a diagram depicting this program's activation frame(s) when execution reaches Point B; that is, just **after** execution of the `for` loop has finished, but before the `return` statement is executed.

When answering this part, make sure that you clearly separate your rough work from your final solution. Consider sketching a sequence of diagrams (one for each iteration of the loop) on the unlined (rough work) pages of your answer booklet, and then draw the final diagram (showing the state of the program at Point B) on a lined page.