Identifying information will be printed here.

University of Waterloo
Midterm Examination
CS 136

Term: Winter Year: 2014

Date: March 3, 2014
Time: 7:00 pm - 8:50 pm
Duration: 110 minutes
Sections: 001–006
Instructors: Reaz Ahmed, Ahmed Hajyasien, Victoria Sakhnini, Dave Tompkins

Student Signature: _____

UW Student ID Number: ANSWER KEY

Number of Exam Pages (including this cover sheet)	19 pages
Exam Type	Closed Book
Additional Material Allowed	NO ADDITIONAL MATERIALS ALLOWED

Marking Scheme (for graders' use only)

Question	Max	Score	Grader
1	14		
2	6		
3	7		
4	10		

Question	Max	Score	Grader
5	8		
6	10		
7	10		

Maximum Total Score: 65

Total Score:

Instructions:

- **You may only use C language features discussed in Sections 01 through 08 (up to slide 26) of the course, inclusive.**
- For each function you are asked to write, you are not required to provide any additional documentation unless we ask you to do so.
- All helper functions must include a brief purpose statement
- Your functions do not have to check for invalid inputs or assert valid inputs unless we ask you to do so.
- Functions you write may call:
  - Any function you have written in another part of the same question.
  - Any function we asked you to write for a previous part of the same question (even if you didn't do that part).
  - Any other built-in or library function discussed in class or used in homework, unless specifically noted in the question.
- For this exam, a *valid* C statement is one that could not cause any errors in the Seashell environment, including memory access errors and syntax errors.
- If you believe there is an error in the exam, notify a proctor. An announcement will be made if a significant error is found.
- It is your responsibility to properly interpret a question. **Do not ask questions regarding the interpretation of a question**; they will not be answered and you will only disrupt your neighbours. If there is a non-technical term you do not understand you may ask for a definition.
- If, despite your best efforts, you are still confused about a question, state your assumptions and proceed to the best of your abilities.

1. (14 points) Short Answers: Write concise, short answers. Point form is acceptable.

(a) (3 points) List three key advantages to modularization, and briefly explain each one (no more than one sentence each).

- RE-USABILITY: can build programs faster OR build larger programs more easily.
- MAINTAINABILITY: easier to debug OR easier to make changes/improvements
- ABSTRACTION: do not need to understand it to use it (can write large programs without understanding every piece)

(b) (2 points) Briefly explain the special form `provide` in Racket, and why there is no equivalent in C.

provide: gives identifiers "program scope" OR specifies the identifiers or "bindings" that are available in the module.

In C, all functions and variables have program scope by default and are available to other modules.

(c) (2 points) Briefly explain the purpose of *requires* and *effects* function documentation in C.

Require: identifies conditions or "state" that must be true before the function can be called, including any restrictions on the parameters.
Effect: identifies what a function prints, reads or mutates.

(d) (3 points) Racket uses *dynamic typing*. What kind of typing does C use? Give one advantage of the typing C uses. Provide a brief example of some Racket code that demonstrates dynamic typing that would not be possible in C.

- static typing
- advantage: built-in contract OR can detect type errors before running OR no run-time type checking
- (many possible answers:
  ; dtype can be a number or a string
  (define dtype (if (>= x 0) x "invalid"))

(e) (2 points) Write the declaration for a C function *add* that takes two ints x and y, and returns an int $x+y$.

int add(int x, int y);

(f) (2 points) Give the output of the following C program. Explain your answer.

```
#include <stdio.h>
#include <limits.h>
//INT_MAX is the largest value that can be represented by an int
//This is 2^31 - 1 = 2147483647
void f(int i) {
  ++i;
  if (i > INT_MAX) {
    printf("OVERFLOW\n");
  } else {
    printf("NO OVERFLOW\n");
  }
}
int main(void) {
  f(INT_MAX);
  printf("DONE\n");
}
```

Output:
NO OVERFLOW
DONE
explanation: NO OVERFLOW is printed because no int can be greater than INT_MAX.

2. (6 points) TrueFalse: Circle the correct answer for each of the following statements.

(a) (1 point) **True** **False** For positive integers, the C modulo operator (%) behaves the same as the Racket quotient function.

False

(b) (1 point) **True** **False** Every C module must have a main function.

False

(c) (1 point) **True** **False** In C, `(a != 0) && (b/a == 2)` will produce an error if a is 0.

False

(d) (1 point) **True** **False** 11010 (in binary) = 16 (in base 10) = 1A (in hex)

False

(e) (1 point) **True** **False** `printf("hello!\n")` is a C expression with the value of 7.

True

(f) (1 point) **True** **False** In the following C code: `bool isfive = (n == 5);` the assignment operator appears only once.

False

3. (7 points) For this question, you will become the Seashell environment. The program below contains at least 7 (seven) mistakes that won't let your code "run" (or would lead to unexpected results). Find seven (unique) mistakes with this code. *Note: poor style is not considered to be a mistake.*

For each unique mistake, circle the mistake code and/or draw an arrow at it, and number the mistake (1,2,3...). In the table on the next page, briefly describe how you would *fix* each mistake. Note: there are no two unique mistakes in the same line, but you might find the same mistake occurs more than once in the same line which is considered one mistake.

We have identified a sample mistake (#0) on line 1 for you.

```
1  #include "stdio.h"      <------- 0
2
3  struct Circle{
4          int x_centre;
5          int y_centre;
6          int r;
7  };
8  struct Point{
9          int x;
10         int y;
11 }
12
13 bool point_in_circle(struct Point *p, struct Circle *c){
14   return (((c->x_centre - p->x)*(c->x_centre - p->x)+
15            (c->y_centre - p->y)*(c->y_centre - p->y)) < (c->r * c->r));
16 }
17
18 bool compare_circles(struct Circle c1, struct Circle c2){
19   return c1==c2;
20 }
21
22 bool compare_points(struct Point *p1, struct Point *p2){
23   return *p1.x == *p2.x  && *p1.y == *p2.y;
24 }
25
26 void update_r(const struct Circle * c, int new_r){
27   c->r = new_r;
28 }
29
30 int main(void){
31   struct Circle c1={0,0,10};
32   struct Circle c2={0,10,10};
33   struct Circle c3={0,10,20};
34   struct Point  p1={2,2};
35   struct Point  p2={12,23};
36   printf("%d\n", point_in_circle(&p1, &c1));
37   printf("%d\n", compare_circles(&c1, &c2));
38   printf("%d\n", compare_points(&p1, &p2));
39   p2={2,2};
40   printf(compare_points(&p1, &p2));
41   update_r(&c2, 20);
42   printf("%d\n", compare_circles(c2, c3));
43 }
```

Mistake	Line #	Brief Explanation & how you would fix it
0.	1	should be `<stdio.h>`, not `"stdio.h"`.
1.		
2.		
3.		
4.		
5.		
6.		
7.		

Mistake	Line #	Brief Explanation & how you would fix it
0.	1	should be `<stdio.h>`, not `"stdio.h"`.
1.	2	missing `include <stdbool.h>`
2.	11	missing ;
3.	19	`c1.x_centre==c2.x_centre && c1.y_centre==c2.y_centre && c1.r==c2.r;`
4.	23	`(*p1).x == (*p2).x && (*p1).y == (*p2).y ;`
5.	37	`printf("%d\n", compare_circles(c1, c2));`
6.	26	remove `const`
7.	39	`p2=p1; or p2.x=2; p2.y=2;`
8.	40	`printf("%d\n", compare_points(&p1, &p2));`

4. (10 points) Write a C function `count_digits(int num);` where `num` is a positive integer. The function counts how many times each of the digits 0..9 appears in `num`, and prints the results (see the following example below).
Example: Calling `count_digits(347213);` will print the following:

```
The digit 0 appeared 0 time(s) in 347213
The digit 1 appeared 1 time(s) in 347213
The digit 2 appeared 1 time(s) in 347213
The digit 3 appeared 2 time(s) in 347213
The digit 4 appeared 1 time(s) in 347213
The digit 5 appeared 0 time(s) in 347213
The digit 6 appeared 0 time(s) in 347213
The digit 7 appeared 1 time(s) in 347213
The digit 8 appeared 0 time(s) in 347213
The digit 9 appeared 0 time(s) in 347213
```

Note: You are not allowed to define 10 counter variables. You must use iteration to solve the problem, and you can not use recursion. Do not define any additional helper functions.

```
void count_digits(int num) {
  for (int i=0; i<=9; ++i) {
    int occurs = 0;
    int temp = num;
    while(temp > 0) {
      int rem = temp % 10;
      if (rem == i) {
        occurs++;
      }
      temp = temp / 10;
    }
    printf("The digit %d appeared %d
            time(s) in %d\n", i, occurs, num);
  }
}
```

5. (8 points) Write the C function `pyramid(int n);` that prints a pyramid of numbers with n lines as illustrated below. A call to the function `pyramid(7)` prints the following:

```
1
2  3
3  4  5
4  5  6  7
3  4  5
2  3
1
```

You may assume that n is an odd integer in the range [3-99] (inclusive).
Use the placeholder `"%3d"` to `printf` each integer, for example: `printf("%3d", x);`

```
// solution 1
void pyramid(int n) {
  for (int i=1; i<=n/2+1; ++i) {
    for (int j=i; j<=2*i-1; ++j) {
      printf("%3d",j);
    }
    printf("\n");
  }
  for (int i=n/2; i>0; --i) {
    for (int j=i; j<=2*i-1; ++j) {
      printf("%3d",j);
    }
    printf("\n");
  }
}
```

```
// solution 2

void pyramid(int n) {
  int line = 1;
  while (1) {
    int col = 1;
    int printed;
    while (col <= line) {
      printed = col + line - 1;
      printf("%3d", printed);
      col++;
    }
    printf("\n");
    if (printed == n) {
      break;
    }
    line++;
  }
  line--;
  while (line >= 1) {
    int col = 1;
    int printed;
    while (col <= line) {
      printed = col + line - 1;
      printf("%3d", printed);
      col++;
    }
    printf("\n");
    line--;
  }
}
```

```
// solution 3

void print_line(int line) {
  int col = 1;
  while(col <= line) {
    printf("%3d", line+col-1);
    col++;
  }
  printf("\n");
}

void print_pyramid(int line, int n) {
  if (line == n/2 +1) {
    print_line(line);
  } else {
    print_line(line);
    print_pyramid(line+1, n);
    print_line(line);
  }
}

void pyramid(int n) {
  print_pyramid(1,n);
}
```

6. (10 points) Consider the following program.

```
1  #include <stdio.h>
2
3  void exchange (int *a, int *b);
4
5  int main(void) {
6    int a, b;
7    a = 5;
8    b = 7;
9    printf("In main: ");
10   printf("a = %d, b = %d\n", a, b);
11   exchange(&a, &b);
12   printf("a = %d, b = %d\n", a, b);
13 }
14
15 void exchange (int *pa, int *pb) {
16   int temp;
17   temp = *pa;
18   *pa = *pb;
19   *pb = temp;
20   printf("In exchange: ");
21   printf("a = %d, b = %d\n", *pa, *pb);
22 }
```

(a) (3 points) Write the output from running this program:

a)
In main: a=5, b=7
In exchange: a=7, b=5
a=7, b=5

(b) (7 points) Draw the call stack immediately after "In exchange: " is printed.

Stack frames have the following format:

```
<Function_name>:
  <parameter1_name>: <parameter1_value>
  <parameter2_name>: <parameter2_value>
  ...
  <local_variable1_name>:<local_variable1_value>
  ...
  return addr: <function name> : <line number>
```

For pointer values, draw an arrow that points at the variable (value) it "points at".

b)

```
exchange:
   pa: Addr_1
   pb: Addr_2
   temp: 5
   return addr: main : 11
```

```
main:
   a: 7 [Addr_1]
   b: 5 [Addr_2]
   return addr: OS
```

7. (10 points) Write a Racket module `games.rkt` that will keep track of the hockey game scores. Initially, the played games record contains no games. Your module must provide the following three functions:

```
(define-struct game (home-team  visiting-team
                     home-score visiting-score) #:transparent)

;; A game is a structure (game h v hs vs), where
;;    h is a String (name of home team),
;;    v is a String (name of visiting team),
;;    hs is a Nat (score of home team)
;;    vs is a Nat (score of visiting team).
;; Note: no games end in a tie - there is always a winning team.

;; add-game: game -> Void
;;   Effect: record/add the game gm to all recorded games so far
;; (add-game gm)

;; total-score: String -> Void
;;   Effect: prints the total points scored by team tname in
;;         all its games recorded so far as:
;;   "The total scored points for **tname** is : **total points**\n"
;; (total-score tname)

;; reset-games Void -> Void
;;   Effect: deletes all previously recorded games.
;; (reset-games)
```

Here is an example of a client:

```
(require "games.rkt")
(add-game (game "Canada" "Switzerland" 10 1))
(define g1 (game "Slovakia" "Canada" 13 18))
(define g2 (game "Switzerland" "Sweden" 1 3))
(add-game g1)
(add-game g2)
(total-score "Canada")
    ;; prints: The total scored points for Canada is : 28
(reset-games)
(total-score "Slovakia") ;; prints:
    ;; prints: The total scored points for Slovakia is : 0
```

```
(provide total-score add-game reset-games)
(define glst empty)
//  global list can be initialized with
// anything as an indicator for
// "no games recorded"

(define (total-score  tname)
  ;; calc: game Nat -> Nat
  ;; produces the total scored points by tname,
  ;; by adding gm result into total
  (define (calc gm total)
   (cond
     [(equal? (game-home-team gm) tname)
       (+ (game-home-score gm) total)]
     [(equal? (game-visiting-team gm) tname)
       (+ (game-visiting-score gm) total)]
     [else total]))
   ;; main function body
  (printf "The total scored points for ~a is :
   ~a \n" tname (foldr calc 0 glst)))

;; add-game: game -> Void
;; record/add  the consumed  gm to all recorded
;; games so far
(define (add-game gm)
  (set! glst (cons gm glst)))

;; reset-games: Void -> Void
;; deleting all the past recorded games.
(define (reset-games)
  (set! glst empty))
```

Write your answer for Question 7 here.

This page is intentionally left blank for your use. Do not remove it from your booklet. If you require more space to answer a question, you may use this page, but you must **clearly indicate** in the provided answer space that you have done so.