Identifying information will be printed here.

University of Waterloo
Midterm Examination
CS 136

Term: Winter Year: 2017

Date: Monday February 27
Time: 7:00 pm - 8:50 pm
Duration: 110 minutes
Sections: 001–011
Instructors: Akinyemi, Brecht, Fortes, Holtby, Irvine, Mondal, Tompkins

Student Signature: _____

UW Student ID Number: ANSWER KEY

Number of Exam Pages (including this cover sheet)	17 pages
Exam Type	Closed Book
Additional Material Allowed	NO ADDITIONAL MATERIALS ALLOWED

Instructions:

- All of your answers should be written in the version of C provided by the Seashell environment unless otherwise specified.
- **You may only use C language features discussed in Sections 01 through 08 (up to slide 18) of the course.**
- You may **NOT** use recursion unless otherwise specified.
- You do not need to consider overflow unless otherwise specified.
- When asked to provide an *interface*, you are **required** to provide the appropriate documentation.
- When you are only asked to *implement* (write) a function, you are not required to provide any additional documentation unless we ask you to do so, but any helper functions must include a brief purpose statement.
- Your functions do not have to check for invalid parameters or assert requirements unless we ask you to do so.
- For array and pointer parameters, you **must** use `const` when appropriate.
- You do not have to `#include` any of the standard libraries in your code (e.g., `stdio.h`) unless the question states otherwise.
- For any code we provide you may assume that the necessary libraries have been included even if the code provided does not explicitly do so.
- 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 you are confused about a question, state your assumptions and proceed to the best of your abilities.
- If you require more space to answer a question, there are blank pages at the end you may use instead, but you must **clearly indicate** in the provided answer space that you have done so.
- **Do not detach any pages and do not write on the QR codes**

Points on this page: 4

1. (9 points)

(a) (1 point) Explain why the following assertions are not necessary.

```
int f(int i) {
  assert(i <= INT_MAX);
  assert(i >= INT_MIN);
  // ...
}
```

No integer value can be outside of the range INT_MIN .. INT_MAX

(b) (1 point) Consider the following C statements:

```
float a = 3.75;
float b = 4.2344;
float *c = &a;
float *d = &b;
*d = *c;
//HERE
```

After the above has been executed (after control reaches the comment marked "HERE") list all the aliases of a and b

Aliases of a:

c or *c*

Aliases of b:

d or *d*

(c) (2 points) Given the following C program:

```
void shuffle(int *a, int b, int *c) {
     *c = *a;
     *a = b;
}

int main(void) {
     int x=1;
     int y=2;
     int z=3;
     shuffle(_____, _____, _____);
     printf("x=%d, y=%d, z=%d\n", x, y, z);
}
```

Fill in the three blanks in the above code so the main function prints out x=1, y=3, z=2.
You may not write any additional code or define any other variables.

shuffle (&z, y, &y); OR shuffle (&y, z, &z);

Points on this page: 5

(d) (1 point) Write a C function that has two **distinct** kinds of side effect.

```
void f(int *g) {
  *g += 1;
  printf("I'm a side effect!\n");
}
```

(e) (1 point) What is the difference between dynamic typing and static typing?

Whether or not variable types are determined before the program is run.

(f) (3 points) Write the C function `fact` that is passed a non-negative integer, and returns the factorial of that integer. You may use either recursion or iteration for this question.

Reminder, `fact(n) == 1 * 2 * ... * n` and `fact(0) == 1`

```
int fact(int n) {
  if (n <= 1) {
    return 1;
  }
  return n * fact(n-1);
}
```

Points on this page: 5

2. (5 points) Write a module (both the *interface* and *implementation*) that provides the function `add_n` and nothing else.

`add_n` has one parameter, an **`int`**. `add_n` returns this parameter plus the sum of all previous values that `add_n` has been called with. The first call will return the argument it was passed as there were no previous values.

For example, if `add_n` has not been called before:

```
int x = add_n(2);
int y = add_n(3);
int z = add_n(2);
assert(x == 2 && y == 5 && z == 7);
```

Clearly separate and identify your interface and implementation.

add.h:

```
// add_n(n) returns the parameter n plus the sum
//    of all previous values that add_n
//    has been called with.
// effects: updates the accumulative sum
int add_n(int n);
```

---

add.c:

```
#include "add.h"

static int g = 0;

int add_n(int n) {
  g += n;
  return g;
}
```

Points on this page: 3

3. (8 points)

(a) (3 points) Below is the start of the `colour.h` module interface.

Complete the interface by adding a declaration (and documentation) for the function `colour_add`. `colour_add` is passed pointers to two colour structures and modifies the **first** colour by adding the red, green, and blue values of the **second** colour to it.

You do **not** need to write the *implementation*, only the *interface*.

```
// colour.h

struct colour {
  int red;
  int green;
  int blue;
};


// colour_add(c1,c2) adds the red, green,
// and blue components of c2 to c1
// requires: c1 and c2 point to valid colours
// effects: c1 is modified
void colour_add(struct colour *c1,
                const struct colour *c2);
```

Points on this page: 5

(b) (2 points)

Implement the function `colour_equal` that is passed pointers to two colour structures and returns true if they represent the same colour (have the same red, green, and blue values), false otherwise.

```
bool colour_equal(const struct colour *c1,
                  const struct colour *c2) {
  return c1->red == c2->red &&
         c1->green == c2->green &&
         c1->blue == c2->blue;
}
```

(c) (3 points)

Write an assertion-based test client that tests one (1) call of `colour_add`. You must use the `colour_equal` function from part (b).

```
#include "colour.h"

int main(void) {
  struct colour grey = {50,50,50};
  struct colour dark_blue = {0,0,25};
  struct colour pale_blue = {50,50,75}
  colour_add(&grey, &dark_blue);
  assert(colour_equal(&grey, &pale_blue));
}
```

Points on this page: 0

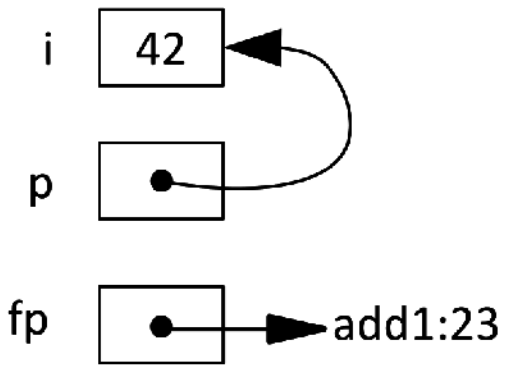
4. (7 points) On the following page, draw a memory diagram that illustrates the complete state of the program when line 9 is reached.

```
1  #include <stdio.h>
2
3  int g = 3;
4
5
6  int h(int *p, int n) {
7    int *p2 = &n;
8    *p += n;
9    // DRAW DIAGRAM AS IT WOULD APPEAR AT THIS POINT
10   int m = *p2 * *p;
11   return m - n;
12 }
13
14 int f(int *a, int *b) {
15   ++g;
16   int c = *a + *b;
17   int d = h(a, *a);
18   return d - c;
19 }
20
21 int main(void) {
22   int i = 5;
23   int j = f(&g, &i);
24   printf("%d\n", j);
25 }
```

Rules

- clearly label and separate all memory regions
- label and separate all stack frames
- represent a variable with a labelled box
- unknown values are written as a question mark (?)
- simple values (int, char, etc.) are written in the box
- pointer values pointing to a variable are drawn using arrows
- pointer values pointing to code are written "function name : line number"

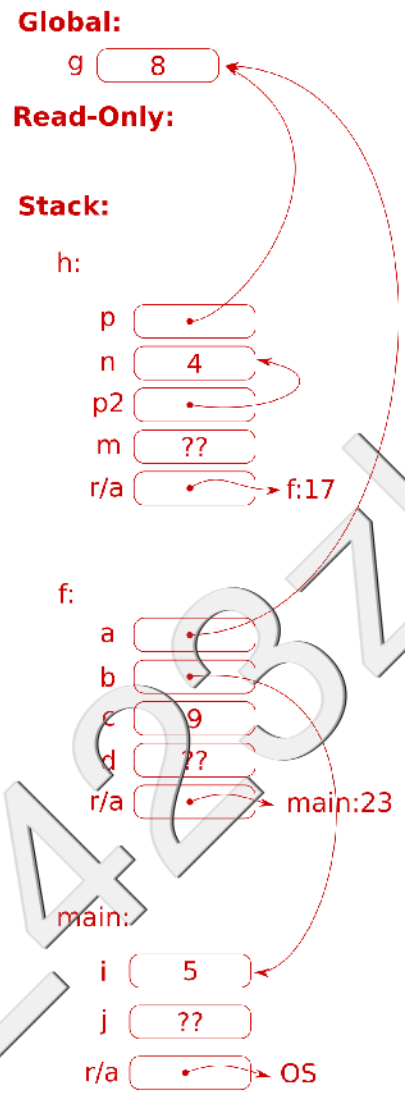
Pointer Examples



DRAW YOUR DIAGRAM ON THE FOLLOWING PAGE.

PRO TIP: YOU CAN FOLD YOUR EXAM SO THAT BOTH PAGES ARE VISIBLE.

Points on this page: 7



Points on this page: 5

5. (5 points)

(a) (4 points) Write a C program that prints all permutations that are exactly 3 characters long where each character can be one of 'A', 'B', 'C', 'D', and 'E'. The order of the output must be printed in reverse alphabetical order (i.e., the first line of output should be EEE). Each permutation should be printed on a separate line.

```
int main() {
  for (char c1 = 'E'; c1 >= 'A'; --c1) {
    for (char c2 = 'E'; c2 >= 'A'; --c2) {
      for (char c3 = 'E'; c3 >= 'A'; --c3) {
        printf("%c%c%c\n", c1, c2, c3);
      }
    }
  }
}
```


(b) (1 point) How many permutations will be printed?

$5^3 = 125$

Points on this page: 8

6. (8 points) Write a C program that reads integer values from the user using `scanf`. When the user enters a negative value or when scanf fails to read an integer, the program then prints out all previously read numbers that are less than the most recently entered natural number, in the order that they were entered. Each number printed is followed by a newline character. If there were no natural numbers entered, or if the most recent number is also the smallest, then no output is printed.

An example has been provided below in the form of files `simple.in` and `simple.expect`.

`simple.in`	`simple.expect`
1	1
2	2
3	2
2	
3	

You may assume the user enters no more than 64 values.

For this question you must write a *complete program*, and must include all needed interface files.

```
#include <stdio.h>
const int MAX_INPUT = 64;
int main(void) {
  int input[MAX_INPUT];
  int values;
  for (values=0 ;values < MAX_INPUT; ++values) {
     int rv = scanf("%d", &input[values]);
     if (rv != 1 || input[values] < 0) break;
  }
  for (int i = 0; i < values - 1; ++i) {
    if (input[i] < input[values-1]) {
       printf("%d\n", input[i]);
    }
  }
}
```

Points on this page: 4

7. (7 points) Consider the following Racket function

```
(define (mystery n)
  (cond [(zero? n) 0]
        [(odd? (remainder n 10)) (add1 (mystery (quotient n 10)))]
        [else (mystery (quotient n 10))]))
```

(a) (4 points) Write an equivalent function in C that uses recursion. Include appropriate documentation.

```
// mystery(n) counts the number of odd digits in n
int mystery(int n) {
  if (n == 0) {
    return 0;
  }
  if ((n % 10) % 2) {   // or simply n % 2
    return 1 + mystery(n / 10);
  }
  return mystery(n / 10);
}
```

Points on this page: 3

(b) (3 points) Write an equivalent C function that uses *iteration* instead of recursion. You do not need documentation this time.

```
int mystery(int n) {
  int num_odd_digits = 0;
  while (n) {
    if ((n % 10) % 2) {
      num_odd_digits += 1;
    }
    n = n / 10;
  }
  return num_odd_digits;
}
```

Points on this page: 7

8. (7 points) Write a C function `isosceles(r)` that prints a triangle pattern using numbers (see examples) with `r` rows, where `r` is a positive integer that is less than 10.

Sample Output:

`isosceles(2)`	`isosceles(3)`	`isosceles(4)`

`  1`
`121`
	`  1`
	` 121`
	`12321`
		`   1`
		`  121`
		` 12321`
		`1234321`

Note that there is a newline immediately after the last digit on each line, and that there are no spaces before the bottom row.

```
void isosceles(int r) {
  for (int row = 1; row <= r; ++row) {
    for (int c = 1; c <= r - row; ++c) {
      printf(" "); // print padding
    }
    for (int c = 1; c <= row; ++c) {
      printf("%d", c);
    }
    for (int c = row-1; c >= 1; --c) {
      printf("%d", c);
    }
    printf("\n");
  }
}
```

Points on this page: 7

9. (7 points) Implement the C function `missing`.

```
// missing(a, len) prints all numbers in the range 1 to 100 that
//   do not appear in the array a, and returns the total number
//   of missing numbers (i.e. the count of the numbers printed)
//   The numbers are printed in ascending order, one per line.
// requires: a is an array with length len, len > 0
// effects: text may be printed
```

```
int missing(const int a[], int len) {
  bool present[100] = {false}; // inits to 0 (false)
  int count = 0;
  for (int i = 0; i < len; ++i) {
    if (a[i] >= 1 && a[i] <= 100) {
      present[a[i]-1] = true;
    }
  }

  for (int i = 0; i < 100; ++i) {
    if (!present[i]) {
      ++count;
      printf("%d\n", i+1);
    }
  }
  return count;
}
```

Alternative answer

```
int missing(const int a[], int len) {
  int count = 0;
  for (int i = 1; i <= 100; ++i) {
    bool is_missing = true;
    for (int j = 0; j < len; ++j) {
      if (a[j] == i) {
        is_missing = false;
        break;
      }
    }
    if (is_missing) {
      printf("%d\n", i);
      ++count;
    }
  }
  return count;
}
```

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.