CSC 209H1 S 2012 Midterm Test
Duration — 50 minutes
Aids allowed: none

**Student Number:** |_|_|_|_|_|_|_|_|_|_|_|

**Last Name:** _____ **First Name:** _____

**Lecture Section:** L0101 **Instructor: Reid**

---

*Do **not** turn this page until you have received the signal to start.*
(Please fill out the identification section above, **write your name on the back of the test**, and read the instructions below.)
*Good Luck!*

---

This midterm consists of 4 questions on 6 pages (including this one). *When you receive the signal to start, please make sure that your copy is complete.* Comments are not required except where indicated, although they may help us mark your answers. They may also get you part marks if you can't figure out how to write the code.
If you use any space for rough work, indicate clearly what you want marked.

# 1: _____/ 6
# 2: _____/ 7
# 3: _____/ 7
# 4: _____/ 5

TOTAL: _____/25

---

## Question 1. [6 MARKS]

### Part (a) [5 MARKS]

The C program below has errors. When run, the output was:

```
123456789Anwar Patel: Anwar Patel  completed 10 courses with CGPA: 3.700000
123456789Anwar Patel: Anwar Patel  completed 10 courses with CGPA: 3.700000
```

Make changes by printing neatly directly on the code (or beside it) so that it will work as intended. Change as little as necessary. The problem is not the math in the calculation.

```
#include <stdio.h>
#include <string.h>

struct student {
    char id[9];
    char name[20];
    double CGPA;
    int courses;
};

void print(struct student s) {
    printf("%s: %s\t completed %d courses with CGPA: %f\n", s.id, s.name, s.courses, s.CGPA);
}

void complete_course(struct student s, double mark) {
    /* calculate the new CGPA  by weighting the old CGPA appropriately */
    double new_CGPA = ((s.CGPA * s.courses) + mark ) / (float) (s.courses + 1);
    s.CGPA = new_CGPA;
    s.courses++;
}

int main() {
  /* create a record for Anwar Patel (student number 123456789) who has a CGPA of 3.7 on
     his first 10 courses. */

  struct student s1;
  strncpy(s1.id, "123456789", 9);
  strncpy(s1.name, "Anwar Patel", strlen("Anwar Patel"));
  s1.CGPA = 3.7;
  s1.courses = 10;

  print(s1);

  complete_course(s1, 4.0);       /* Anwar just got a 4.0 in CSC209! */

  print(s1);
  return 0;
}
```

**Part (b)** [1 MARK]

Would your code above still work if the name of the student was "Clement Schiano de Collela""? Explain why or why not and any additional changes to the code that would be required.

## Question 2. [7 MARKS]

In each subquestion below, fill in the box with the declaration for an appropriate mystery function so that the following code would compile without error. Each subquestion is independent.

**Part (a)** [3 MARKS]

```
char ** students;
int class_size = 100;
if(mystery1(class_size, &students) > 0)
    printf("There were errors\n");
```

**Part (b)** [4 MARKS]

```
char ** authors;
char * most_famous;
int books[4] = {4,12,16,22};
int * p = books;

most_famous = mystery2(books[0], &p, *authors);
```

## Question 3. [7 MARKS]

Write a shell program that you might use to clean up your CSC209 directory. The script takes any number of filenames as command line arguments and moves all of the files with names ending in `.sh` to a subdirectory called `scripts`. If this subdirectory doesn't already exist, your script should create it. If your script is called with an argument that doesn't correspond to an existing regular file, you should just ignore that argument. You do not need to worry about further error checking. For example, you may assume that any regular files corresponding to the commandline arguments are readable.

## Question 4. [5 MARKS]

Consider the C program below with an array of strings `names`. Write a C function `zero_strings` that will set each name to an empty string and return the number of names that were changed in the process. Inside the box below, fill in the call to your function inside `main`. You need to determine the appropriate signature for your function.

```
int main() {
    int n =  XXXXX NOT SHOWN ON TEST XXXX
    char ** names = malloc(n * sizeof(char*));
    int i;
    for (i = 0; i < n; i++ ) {
       names[i] = malloc(NAMESIZE);
    }
    /* code not shown where some of names get set to non-empty strings */

   /* call your zero_strings function and print the return value */
```

```
    return 0;
}
```

**C function prototypes and structs:**

```
int fclose(FILE *stream)
char *fgets(char *s, int n, FILE *stream)
FILE *fopen(const char *file, const char *mode)
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
int fseek(FILE *stream, long offset, int whence)
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);
char *index(const char *s, int c)
void perror(const char *s)
int sprintf(char *s, const char *format, ...)
char *strchr(const char *s, int c)
size_t strlen(const char *s)
char *strncat(char *dest, const char *src, size_t n)
int strncmp(const char *s1, const char *s2, size_t n)
char *strncpy(char *dest, const char *src, size_t n)
char *strrchr(const char *s, int c)
char *strstr(const char *haystack, const char *needle)
```

**Shell comparison operators**

| Shell | Description |
|---|---|
| -d filename | Exists as a directory |
| -f filename | Exists as a regular file. |
| -r filename | Exists as a readable file |
| -w filename | Exists as a writable file. |
| -x filename | Exists as an executable file. |
| -z string | True if empty string |
| str1 = str2 | True if str1 equals str2 |
| str1 != str2 | True if str1 not equal to str2 |
| int1 -eq int2 | True if int1 equals int2 |
| -ne, -gt, -lt, -le | For numbers |
| !=, >, >=, <, <= | For strings |
| -a, -o | And, or. |

```
expr match STRING REGEXP
expr ARG1 + ARG2
```

**Useful Unix programs for shell programs:** `cat, cut, wc, grep, sort, sort -n (for numerical sorting), head, tail`

**Print your name in this box.**