CSC 209H1 S 2017 Midterm Test
Duration — 50 minutes
Aids allowed: none

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

Last Name: ______________________ **First Name:** ______________________

Instructor: Reid
Section: L0101 (10:10-11:00am)

---

*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 6 questions on 8 pages (including this one). *When you receive the signal to start, please make sure that your copy is complete.*
Comments are not required, 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.
No error checking is required unless you are specifically requested to do it for an individual question.
You do not need to provide the include statements for your programs.
If you use any space for rough work, indicate clearly what you want marked.

# 1:	______/10
# 2:	______/ 1
# 3:	______/ 3
# 4:	______/ 2
# 5:	______/ 4
# 6:	______/ 6
TOTAL:	______/26

Question 1. [10 MARKS]

Part (a) [8 MARKS]

Complete the following program according to the instructions in the comments. Assume that all system calls succeed and that the arguments are passed correctly.

```
struct graph {
    char *title;
    int size;
    int *data;
};

int main(int argc, char **argv) {
    if(argc != 4){
        fprintf(stderr, "Usage: mkgraph TITLE NUM_POINTS FILE\n");
    }

    // Allocate space on the heap for a struct graph and save the pointer to
    // this memory in a variable called graph




    // Set the graph's size field to the integer value of the second command
    // line argument




    // Set graph's title to the first command line argument without copying it




    // Initialize the graph's data field to refer to heap-allocated space for
    // size number of elements.




    // Open the file given by file name in the third command line argument
    // The file will be opened for reading.








    // CONTINUED on NEXT PAGE
    // The open file is a text file containing  numbers separated by spaces
    // Read each number from the file and store it in the data field of graph
    // in the order they are read
    // Compute the average of the elements and print it in the statment below









    printf("%f\n", ____________________________________________);
    return 0;
}
```

Part (b) [2 marks]

Check the box beside each of the free statements below that correctly free memory used in the above program. For each line that is not checked, briefly explan why the statement is incorrect.

☐ `free(graph);`

☐ `free(graph->title);`

☐ `free(graph->data);`

☐
```
for(int i = 0; i < graph->size; i++) {
    free(graph->data[i]);
}
```

Question 2. [1 mark]

Write a line to execute `compute_hash` with a blocksize argument of 4 and redirect standard input so that the program reads from the file `test.in` rather than from standard input.

Question 3. [3 marks]

Your current working directory has two files prog.c and helper.c. prog.c has a main function in it and calls functions from helper.c

Write a Makefile with one rule so that when you type `make prog`, the executable prog will be created only if one or both of the files prog.c and helper.c have been modified. Note that the program should be compiled with the -Wall and -g flags.

Question 4. [2 marks]

I spent a long time working on my `print_ftree` program on cdf and it works perfectly, but when I try it on my Linux machine at home, once it a while it gives me a segmentation fault.

Part (a) [1 mark] Is there a bug in my code? Explain your answer.

Part (b) [1 mark] When does the segmentation fault message appear? Check the appropriate answer or answers.

- ☐ Compile time
- [✓ Run time]
- ☐ When runnning `make print_ftree`

Question 5. [4 marks]

For each of the code fragments below, there is missing code. At the very least, the line (or lines) that declare and possibly initialize the variable `x` are missing. If the code will not compile no matter what you put for the missing code, check `COMPILE ERROR` and explain why. If the code will compile, but is not guaranteed to run without an error, check `RUN-TIME ERROR` and explain why. Otherwise, check `NO ERROR` and give the correct declaration for x. You don't need to show any other missing code. The first one is done for you.

Code Fragment	ERROR	Declaration for `x` or explanation

`int y;`
`// missing code`
`x = y;`
	☑ NO ERROR
	☐ COMPILE ERROR
	☐ RUN-TIME ERROR
		`int x;`

`char input[6];`
`// missing code`
`strcpy(input, x);`
	☐ NO ERROR
	☐ COMPILE ERROR
	☐ RUN-TIME ERROR	

`char *film = "Moonlight";`
`// missing code`
`x = film;`
`x[0] = 'L';`
	☐ NO ERROR
	☐ COMPILE ERROR
	☐ RUN-TIME ERROR	

`char *args[3];`
`// missing code`
`x = *args[2];`
	☐ NO ERROR
	☐ COMPILE ERROR
	☐ RUN-TIME ERROR	

`// struct location is defined`
`struct location src;`
`// missing code`
`src = x;`
	☐ NO ERROR
	☐ COMPILE ERROR
	☐ RUN-TIME ERROR	

`int *mkpoint(int x, int y) {`
`    int pt[2] = {x, y};`
`    return pt;`
`}`
`x = mkpoint(3, 4);`
	☐ NO ERROR
	☐ COMPILE ERROR
	☐ RUN-TIME ERROR	

Question 6. [6 marks]

Complete the function below, so that the following examples will work correctly and the minimum amount of memory is used. You may only use the string library functions `strlen` and `strncpy`. (You are not required to use these functions.)

```
    char *path1 = "/usr/include";
    char *result1 = basename(path1);

    char *path2 = "/usr/include/";
    char *result2 = basename(path2);

    char *path3 = "file.txt";
    char *result3 = basename(path3);

    char *path4 = "./a2/test/test1.in";
    char *result4 = basename(path4);
    printf("%s, %s, %s, %s\n",
        result1, result2, result3, result4);
```

Prints:

```
include, , file.txt, test1.in
```

```
/* Returns the component of path following the final '/'
 * path will not be NULL
 * Do not modify path
*/
char *basename(char *path) {
```

C function prototypes:

```
int fclose(FILE *stream)
char *fgets(char *s, int n, FILE *stream)
FILE *fopen(const char *file, const char *mode)
int fprintf(FILE *stream, const char *format, ...)
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream)
void free(void *ptr)
int fscanf(FILE *restrict stream, const char *restrict format, ...)
int fseek(FILE *stream, long offset, int whence)
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream)
void *malloc(size_t size)
DIR *opendir(const char *name)
void perror(const char *s)
int printf(const char *format, ...)
struct dirent *readdir(DIR *dir)
int scanf(const char *restrict format, ...)
int lstat(const char *file_name, struct stat *buf)
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)
```

Excerpt from fgets man page:

```
    fgets() reads in at most one less than size characters from stream  and
    stores  them  into  the buffer pointed to by s.  Reading stops after an
    EOF or a newline.  If a newline is read, it is stored into the  buffer.
    A  terminating  null  byte ('\0') is stored after the last character in
    the buffer.
```

Excerpt from scanf/fscanf man page:

```
RETURN VALUES
     scanf and fscanf return the number of input items assigned.  This can be
     fewer than provided for, or even zero, in the event of a matching fail-
     ure. The value EOF is returned if an input failure occurs before any
     conversion such as an end- of-file occurs.
```

Makefile variables: `$@` is the target, `$^` is the list of prerequisites `$<` is the first prerequisite.

Print your name in this box.