CSC 209H1 S 2019 Midterm Test
Duration — 50 minutes
Aids allowed: none

UTORID: |_|_|_|_|_|_|_|_|_|_|

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

Instructor: Campbell
Section: L0101

---

*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.
No error checking is required.
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: _____/ 4

2: _____/ 3

3: _____/ 3

4: _____/ 8

5: _____/ 2

6: _____/ 5

TOTAL: _____/25

Question 1. [4 MARKS]

Assume you have a terminal open, and the current working directory contains a C program file called `args.c`. The contents of the file are shown below:

```
#include <stdio.h>

int main(int argc, char *argv[]) {
    printf("%d\n", argc);
    return 0;
}
```

Part (a) [1 MARK] Write a command to compile `args.c` into an executable called `args`, using the `gnu99` standard and including the flag to display all warning messages.

Part (b) [1 MARK] Write the output of the program for each of the following invocations:

```
./args
```

```
./args abc 123 xyz
```

Part (c) [1 MARK]

Write a command that invokes `args` redirecting the program's standard output to a file called `data`.

Part (d) [1 MARK]

Write a single unix command to set the permissions of the file `args` to `rwxr-xrw-`.

Question 2. [3 MARKS]

Suppose that the current working directory contains only the files:

- `helpers.c`: contains helper functions
- `helpers.h`: contains the prototypes for those helper functions, and
- `life.c`: contains a `main` function that calls on the helper functions.

Write the commands needed to compile the code to produce object files `helpers.o` and `life.o`, and then use the object files to produce an executable named `mylife`.

Question 3. [3 MARKS]

The following program runs without errors. Print its output neatly in the box provided.

```
int func(int a, int *b) {
    int *ptr = &a;
    *ptr += 5;

    ptr = b;
    *ptr -= 3;

    return a;
}

int main() {
    int x = 2;
    int y = 8;
    int ret = func(x, &y);

    printf("x: %d\n", x);
    printf("y: %d\n", y);
    printf("ret: %d\n", ret);

    return 0;
}
```

Answer:

Question 4. [8 MARKS]

Consider the code and memory diagram below.

Part (a) [6 MARKS]

Fill in the memory diagram to show the current state of the program exactly before the return statement on **line 12** is executed. If there are uninitialized blocks of memory at that point in the program, write their values as `???`.

```
1  char **split(char *s) {
2      char *ptr = strchr(s, '.') + 1;
3
4      char **tokens = malloc(2 * sizeof(char *));
5      tokens[0] = malloc(ptr - s);
6      strncpy(tokens[0], s, ptr-s-1);
7      tokens[0][ptr-s-1] = '\0';
8
9      tokens[1] = malloc(strlen(ptr) + 1);
10     strcpy(tokens[1], ptr);
11
12     return tokens;
13 }
14
15 int main(void) {
16     char **arr = split("out.txt");
17     printf("%s\n", arr[1]);
18
19     // TODO: Free the allocated memory.
20
21     return 0;
22 }
```

Section	Address	Value	Label
Read-only	0x100		
	0x104		
	0x108		
	0x10c		
	0x110		
	0x114		
	0x118		
	0x11c		
	⋮	⋮	
Heap	0x23c		
	0x240		
	0x244		
	0x248		
	0x24c		
	0x250		
	0x254		
	0x258		
	0x25c		
	⋮	⋮	
Stack	0x454		
	0x458		
	0x45c		
	0x460		
	0x464		
	0x468		
	0x46c		
	0x470		
	0x474		
	0x478		
	0x47c		

Part (b) [2 MARKS]

Add the necessary statement(s) that would follow line 19 to properly free the memory allocated by the program:

Question 5. [2 MARKS]

The following code snippet runs without errors. Print its output neatly in the box provided.

```
struct Car {
    char *color;
    int mileage;
};

void update_mileage(struct Car c, struct Car *c_ptr) {
    c_ptr->mileage += 500;
    c.mileage += 200;
}


int main() {
    struct Car car;
    char *color_ptr = "Green";
    car.color = color_ptr;
    car.mileage = 1000;

    color_ptr = "Blue";
    struct Car *car_ptr = &car;
    car_ptr->mileage = 1500;

    printf("(%s, %d)\n", car.color, car.mileage);

    update_mileage(car, &car);

    printf("(%s, %d)\n", car_ptr->color, car_ptr->mileage);

    return 0;
}
```

Question 6. [5 MARKS]

The question is based on the following linked list definition:

```
struct node {
    int ID;
    char *name; // Points to a dynamically allocated string.
    struct node *next;
};
```

Considering that the name of each linked list node has the form `"lastname, firstname"`, for each node starting at the specified `head`, reorder the two names and convert them into the following form: `"firstname-lastname"`. Write your code so that it does not have a memory leak.

```
void format_name(struct node *head) {




}
```

C function prototypes:

```
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)
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)
void perror(const char *s)
int scanf(const char *restrict format, ...)
char *strchr(const char *s, int c)
size_t strlen(const char *s)
char *strcat(char *dest, const char *src)
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 *strstr(const char *haystack, const char *needle)
long int strtol(const char *nptr, char **endptr, int base);
```

Excerpt from strcpy/strncpy man page:

```
The strcpy() functions copy the string src to dst (including the terminating ‘\0’ character).
The strncpy() function copies at most n characters from src into dst.  If src is less than n
characters long, the remainder of dst is filled with ‘\0’ characters.  Otherwise, dst is not terminated.
```

Excerpt from strstr man page:

```
The strstr() function finds the first occurrence of the substring needle
in the string haystack. It returns a pointer to the beginning of the
substring, or NULL if the substring is not found.
```

Excerpt from strchr man page:

```
The strchr() function locates the first occurrence of c (converted to a char) in the
string pointed to by s.  The terminating null character is considered to be part of
the string; therefore if c is ‘\0’, the functions locate the terminating ‘\0’.
```

Excerpt from strcat man page:

```
The  strcat() function appends the src string to the dest string, overwriting the terminating null
byte ('\0') at the end of dest, and then adds a terminating null byte.
```

Useful Unix programs: cat, cut, wc, grep, sort, head, tail, echo, set, uniq, chmod

Makefile variables: `$@` target, `$^` all prerequisites, `$?` all out of date prereqs,`$<` first prereq

Print your name in this box.

End of Examination