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

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

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

Instructor: Simpson
Section: L0301

---

*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: _____/ 3

2: _____/ 3

3: _____/ 3

4: _____/ 9

5: _____/ 2

6: _____/ 5

TOTAL: _____/25

Question 1. [3 MARKS]

Assume you have a terminal open, and the current working directory contains a C file called `nums.c`.

Part (a) [1 MARK]

Write a command to compile `nums.c` into an executable called `nums`, including debug symbols and the flag to display all warning messages.

Part (b) [1 MARK]

Write a command that invokes `nums` and pipes the output to the program `sort`.

Part (c) [1 MARK]

The following command is used to set the permissions of `file.txt`:

```
$ chmod 643 file.txt
```

Check the boxes to indicate the file permissions of `file.txt` after the command above has been executed.

user:	☐ read	☐ write	☐ execute
group:	☐ read	☐ write	☐ execute
other:	☐ read	☐ write	☐ execute

Question 2. [3 MARKS]

Consider the following makefile:

```
FLAGS = -Wall -g -std=gnu99
DEPENDENCIES = queue.h

all: help_centre print_queue

help_centre: help_centre.o queue.o
        gcc ${FLAGS} -o $@ $^

print_queue: print_queue.o queue.o
        gcc ${FLAGS} -o $@ $^

%.o: %.c ${DEPENDENCIES}
        gcc ${FLAGS} -c $<
```

Suppose that the only files in the current working directory are the source files, the header file, and the Makefile. Write the actions that are executed (in the order that they are executed), when we run:
`make help_centre`

Makefile variables:

`$@` target
`$^` all prerequisites
`$?` all out of date prerequisites
`$<` first prerequisite

Question 3. [3 MARKS]

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

```
void func(int a, int *b, int *c) {
    int d = a;
    int *ptr = &a;
    *ptr -= 2;

    b = malloc(sizeof(int));
    *b = a;
    *c = d;
}

int main() {
    int x = 6;
    int *y = NULL;
    int ret;
    func(x, y, &ret);

    printf("x: %d\n", x);
    if (y == NULL)
        printf("y is NULL\n");
    else {
        printf("y: %d\n", *y);
        free(y);
    }
    printf("ret: %d\n", ret);

    return 0;
}
```

Answer:

Question 4. [9 MARKS]

Consider the code and memory diagram below.

Part (a) [8 MARKS]

Fill in the memory diagram to show the current state of the program exactly before the return statement on **line 7** is executed. If there are uninitialized blocks of memory at that point in the program, write their values as `???`.

```
1  char *truncate(char *s, int new_size) {
2      if (new_size >= strlen(s))
3          return 0;
4
5      s[new_size] = '\0';
6
7      return &s[new_size + 1];
8  }
9
10 int main(void) {
11     int len = strlen("CSC209H1-S1");
12     char *str = malloc(len + 1);
13     strcpy(str, "CSC209H1-S1");
14
15     char *ptr = strchr(str, '-');
16     char *chars = truncate(str, ptr - str);
17     printf("%s, %s\n", str, chars);
18     free(str);
19     return 0;
20 }
21
```

Section	Address	Value	Label
Read-only	0x100		
	0x104		
	0x108		
	0x10c		
	0x110		
	0x114		
	0x118		
	0x11c		
	⋮	⋮	
Heap	0x23c		
	0x240		
	0x244		
	0x248		
	0x24c		
	0x250		
	0x254		
	0x258		
	⋮	⋮	
Stack	0x448		
	0x44c		
	0x450		
	0x454		
	0x458		
	0x45c		
	0x460		
	0x464		
	0x468		
	0x46c		
	0x470		
	0x474		
	0x478		

Part (b) [1 MARK]

How many bytes are freed by the `free` statement on line 18? _____

Question 5. [2 MARKS]

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

```
struct Airplane {
    char *destination;
    int passengers;
};

void update_information(struct Airplane *a_ptr) {
    a_ptr->destination[2] = 'C';
    a_ptr->passengers += 50;
}

int main(void) {
    struct Airplane airplane;
    airplane.destination = malloc(strlen("YYZ") + 1);
    strcpy(airplane.destination, "YYZ");
    airplane.passengers = 100;

    struct Airplane *airplane2_ptr = malloc(sizeof(struct Airplane));
    airplane2_ptr->destination = airplane.destination;
    airplane2_ptr->passengers = airplane.passengers;
    update_information(airplane2_ptr);

    printf("(%s, %d)\n", airplane.destination, airplane.passengers);
    printf("(%s, %d)\n", airplane2_ptr->destination, airplane2_ptr->passengers);

    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;
};
```

Implement a function that iterates over the nodes of a linked list starting at the specified `head` and modifies the `name` of every node in the list by adding `n` additional copies of the name. For example, if a name is originally *Marcia* and `n` is 2, the name becomes *MarciaMarciaMarcia*.

Write your code so that it does not have a memory leak.

```
void repeat_name(struct node *head, int n) {

}
```

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