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

UTORID: 1,0,0,3,3,9,4,9,7,1
negmaree

Last Name: NEGM **First Name:** Areej

Instructor: Reid
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 5 questions on 8 pages (including this one). *When you receive the signal to start, please make sure that your copy is complete.*	# 1: 2 / 4
	# 2: 2.5 / 5
Comments are not required.	# 3: 1 / 6
No error checking is required.	# 4: 1 / 3
You do not need to provide the include statements for your programs.	# 5: 3 / 7
If you use any space for rough work, indicate clearly what you want marked.	TOTAL: 9.5 /25

Question 1. [4 MARKS]

These questions use the following struct:

```
struct rec {
    char *leader;
    int seats;
    struct rec *next;
};
```

Part (a) [1 MARK] Check the box that best explains the output of this program.

```
void set_record(struct rec *r, char *name, int seats) {
    r = malloc(sizeof(struct rec));
    r->leader = name;
    r->seats = seats;
}

int main() {
    struct rec party;
    set_record(&party, "Justin Trudeau", 155);
    printf("%s %d\n", party.leader, party.seats);
}
```

party->leader, party->seats

- ☐ Prints `Justin Trudeau 155`
- ☐ Prints empty string and `155` because the `leader` field is not initialized
- ☐ `Justin Trudeau` and garbage because `seats` is not initialized
- ☒ Unknown because `party` is not initialized

Part (b) [2 MARKS] Fill in the types so that the following statements are correct: (Assume appropriate memory has been allocated for all variables.)

```
struct rec party;

int * x = &party.seats;

char * y = &party.leader[0];

char * z = party.leader[2];

struct rec a = *party.next;
```

-1

Part (c) [1 MARK] Check the box that best describes the error in this function.

```
void freelist(struct rec *head) {
    while(head != NULL) {
        free(head);
        head = head->next;
    }
}
```

- [x] segmentation fault
- [ ] memory leak
- [ ] dangling pointer
- [ ] None of the above. There is nothing wrong with this code.

Question 2. [5 MARKS] 2.5

Part (a) [4 MARKS]

Suppose we have a directory that contains the following files:

```
Makefile    customer.o  item.o      library.h
customer.c  item.c      library.c   library.o
```

The Makefile contains the following:

```
library : library.o item.o customer.o
        gcc -Wall -g -std=gnu99 -o library library.o item.o customer.o

%.o : %.c library.h
        gcc -Wall -g -std=gnu99 -c $<
```

$< first prereq

How many times is gcc called if we type `make library`? For each of the options below, circle "possible" or "not possible". For the case or cases where it is possible, explain under what circumstances it will occur.

0 times	possible (not possible) ✓	what's this?
1 time	(possible) ✓ not possible	can execute the rule for library if non of the dependency files have not been modified
2 times	(possible) ✓ not possible	The %.o rule can execute Then library rule X
4 times	possible (not possible) X	X
5 times	possible (not possible) ✓	

-1.5

Part (b) [1 MARK] -0.5

Check the statements are true about the following rule.

```
all : simpletest mytest
```

-0.5

- [x] The rule will only be executed if `simpletest` and `mytest` are newer than `all`
- [ ] The rule has no actions
- [ ] The rule has no prerequisites
- [x] The rule will always evaluate the `simpletest` and `mytest` rules

Question 3. [6 MARKS]

For assignment 1 we could have dynamically allocated the two-dimensional matrix as illustrated in the following code.

Fill in the memory diagram to show the current state of the program exactly before the return statement on **line 13** is executed. If there are uninitialized blocks of memory at that point in the program, write their values as ???.

```
1  int **create_matrix(int rows, int cols) {
2
3      int **matrix = malloc(rows * sizeof(int *));
4
5      for(int i = 0; i < rows; i++) {
6          matrix[i] = malloc(cols * sizeof(int));
7
8          for(int j = 0; j < cols; j++) {
9              if(i == j) {
10                 matrix[i][j] = 1;
11             } else {
12                 matrix[i][j] = 0;
13             }
14         }
15     }
16     return matrix;
17 }
18
19 int main() {
20     int d = 2;
21     int **m = create_matrix(d, d);
22     printf("%d %d\n", m[0][0], m[0][1]);
23     return 0;
24 }
```

Section	Address	Value	Label
Read-only	0x100	0	i
	0x104		
	0x108	1	j
	0x10c		
	0x110		
	⋮	⋮	
Heap	0x23c	0x454	rows
	0x240		
	0x244		
	0x248		
	0x24c	0x458	cols
	0x250		
	0x254		
	0x258	0x100	matrix
	0x25c		
	0x260	0x104	
	0x264		
	⋮	⋮	
Stack	0x454	2	d
	0x458	2	
	0x45c	1	m
	0x460	0	
	0x464		
	0x468		
	0x46c		
	0x470	0x25c	matrix
	0x474		i
	0x478		j
	0x47c		rows
	0x480		cols

Question 4. [3 MARKS]

Consider the following program that illustrates how to use the `get_point` function. Assume no errors occur, opening the files is successful, and the files have the correct format.

The file "`points.b`" contains an array of `struct point` written to the file in binary using `fwrite`.

```
struct point {
  int x;
  int y;
};

int main(){
    FILE *fp1 = fopen("points.b", "rb");
    struct point *p1 = get_point(fp1, 2);
    printf("%d %d\n", p1->x, p1->y);

    return 0;
}
```

Complete the function below that returns a pointer to a `struct point` that contains the **nth point** in the binary file. The first `struct point` in the file would be stored at the beginning of the file. Assume the file is large enough to contain the nth point.

```
struct point *get_point(FILE *fp, int n) {
  int n;
FILE *fp;
  struct point header;
fread(&header, sizeof(struct, fp);
fseek(points.b, 24, SEEK_CURR);
while(fread(&points.b, size, fp) != n) {
    if(points.b = n) {
return ?
```

need malloc -1

wrong use of fread -0.5

wrong use of fseek -0.5

3

Question 5. [7 MARKS]

The function `inject` will return a string containing `str` but with every occurrence of `c` replaced with `substr`. If `c` does not occur in `str`, then a copy of `str` is returned.

For example, if `inject` is called as `inject("abcabc", 'a', "def")`, then it will return `"defbcdefbc"`

You must allocate exactly the right amount of space to store the new string. You may make use of the function `count_chars()` defined below that returns the number of occurrences of `c` in `str`: (Do not write `count_chars()`.)

```
int count_chars(char *str, char c);

char *inject(char *str, char c, char *substr) {
```

```
Char new_str; -1 (just one character)
int num = count_chars(str, c);
for(int i = 0; i < strlen(str); i++) {
    if (num != 0) {
        if (strcmp(str[i], c) == 0) {   (== 0 -> comparing chars with strcmp)
            // allocate space for substr
            // then set str[i] = substr
            new_str[i] = strcpy(str[i], substr)
            new_str = strcpy(new_str, str[i]);
    }
}
else {
    new_str = malloc(num * sizeof(char *substr)); -2 not quite, need to allocate first
}
new_str = strcpy(newStr, str);

return new_str;
}
```


11

Ugh :(

Sorry I tried

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)
       //set whence to SEEK_SET to seek from beginning of file
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream)
char *index(const char *s, int c)
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 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