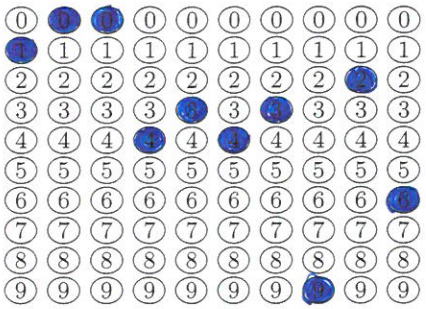
CSC 209H5 S 2019 Midterm Test
Duration — 120 minutes
Aids allowed: none

Student Number: 

UTORid: topiwal3

Last Name: Topiwala **First Name:** Mital

Instructors: Alaca & Vrbik

---

*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 12 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.

No error checking is required except where specifically requested.

You do not need to provide include statements for your programs.

If you use any space for rough work, indicate clearly what you want marked.

1: 1 / 7
2: 2 / 6
3: 6 / 6
4: 6 / 8
5: 8 / 7

TOTAL: 23 /34

Question 1. [7 MARKS]

Part (a) [3 MARKS] 1

Below is a sample script of your terminal:

```
dovahkiin@dh2020pc02:~/209_repo$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   t05/bitmap.c
    modified:   t05/bitmap_printer.c

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    t05/nirnroot.bmp

no changes added to commit (use "git add" and/or "git commit -a")
dovahkiin@dh2020pc02:~/209_repo$ cd a2
dovahkiin@dh2020pc02:~/209_repo/a2$
```

Write the commands that you would type (**after** typing the commands shown above) to commit your changes to all your `.c` files in your `t05` directory and update your remote repository on MarkUs, **without** adding any additional files to your repository. For full marks:

- You should **not** change directories.
- Your command(s) should be as short as possible, by taking advantage of **globbing** and **relative file paths**.

```
git add ../t05/bitmap.c ../t05/bitmap_printer.c
git commit -m 'updating .c files in t05';
git push
```

✗ need up to parent

No globbing

+1 add/push

Part (b) [2 MARKS] 0

You have written and compiled a program called `en2fr`, which takes a single command-line argument: the name of an input file containing English-language text. Your program translates the text into French and outputs the translated text to `stdout`. ↳ using text files

The UNIX utility `wc`, when used with no command-line arguments, reads input from `stdin` and prints out the newline, word, and byte counts to `stdout`. The `-w` argument can be used to print only the word count.

Write a single-line command that reads the English-language file `test.txt` and outputs the word count after it has been translated to French. For now, don't worry about saving the translated text into a file. Assume that both the `test.txt` file and the `en2fr` executable file are located in your current directory.

$ wc -w | ./en2fr (test.txt)

pipe wrong
wrong syntax

Part (c) [2 MARKS] 0

The problem with the command we asked you to write above is that the French-translated text doesn't actually get saved into a file—it is simply "consumed" by the `wc` command. It would be nicer if we could save our translated text into a file **and** get the word count, all in a single-line command. Thankfully, the UNIX utility `tee` will help us do just that. When used as follows, `tee` reads from `stdin` and copies the stream to `stdout` **and** to the file `OUTPUT`:

```
tee OUTPUT
```

Improve your previous one-line command so that it reads the English-language file `test.txt`, saves the translated text to `test_fr.txt`, and outputs the word count of the French translation.

$ tee test_fr.txt | (./en2fr (test.txt) > test_fr.txt) (wc-w ./en2fr (test.txt))

0

Question 2. [6 MARKS]

Déjà vu?

Consider the following program:

```
#define ARRAY_SIZE 10
#define TERMINATOR -1
#include <stdio.h>
#include <stdlib.h>

// Returns the number of integers currently stored in our "array list" of integers,
// excluding the TERMINATOR element that signifies the end of the list.
int array_len(int *a) {
    int i;
    for(i = 0; i < ARRAY_SIZE && a[i] != TERMINATOR; i++);
    return i;
}

void jumbleArrays(int* a1, int* a2) {
    int a1_len = array_len(a1);
    a2 = a1;
    a2[a1_len] = 42;
    if(a1_len <= ARRAY_SIZE) // Avoid buffer overflow
        a2[a1_len+1] = TERMINATOR;

    a1 = malloc(sizeof(int)*ARRAY_SIZE);
    a1[0] = 41;
    a1[1] = TERMINATOR;
}

int main() {
    int a1[ARRAY_SIZE];
    int a2[ARRAY_SIZE];

    a1[0] = 10;
    a1[1] = 4;
    a1[2] = TERMINATOR;

    a2[0] = 7;
    a2[1] = 9;
    a2[2] = 11;
    a2[3] = TERMINATOR;

    jumbleArrays(a1, a2);

   // Continued on next page...
```

```
    printf("Contents of a1: ");
    for(int i = 0; i < ARRAY_SIZE && a1[i] != TERMINATOR; i++)
        printf("%d ", a1[i]);
    printf("\n");

    printf("Contents of a2: ");
    for(int i = 0; i < ARRAY_SIZE && a2[i] != TERMINATOR; i++)
        printf("%d ", a2[i]);
    printf("\n");

    return 0;
}
```

Part (a) [4 MARKS]

Print the **exact output** generated by running the above program:

Contents of a1: 41

Contents of a2: 10 4 42

Part (b) [1 MARK]

You might have noticed that there is a mistake in the following code:

```
if(a1_len <= ARRAY_SIZE) // Avoid buffer overflow
    a2[a1_len+1] = TERMINATOR;
```

Fix the mistake, and write the corrected code below:

```
if (a1_len < ARRAY_SIZE)
    a2[a1_len+1];
```

Part (c) [1 MARK]

What kind of error do you risk encountering if you don't fix the code above?

You could get a segmentation fault, because if a1_len == ARRAY_SIZE which is 10, a1_len + 1 is 11, and we have only allocated space for 10 integers, not 11. Doing so would cause the user to index to an adress that has not been allocated for that array. => error!

6

Question 3. [6 MARKS]

Files and I/O

Part (a) [2 MARKS] 2

use fget

Write code to read an input sequence from standard input into a string. Assume that the **input sequence** (bolded for emphasis) should be no more than 50 characters. Declare any necessary variable(s), and write safe code.

Hint: If you decide to use `scanf`, you must include a **field width** in your format specifier to limit the length of the input sequence to be read (otherwise it is not safe).

```
char s[51] = NULL;
char * ret = fgets(s, 50, stdin);
if (ret == NULL) {
    fprintf(stderr, "error occured when reading sequence");
}
s[strlen(s)] = '\0';
```

Part (b) [4 MARKS] 4

Write code to open the binary file named "`data.bin`" from the current working directory. Then, read a series of 100 integers from the file which are stored consecutively starting from offset 512. Do not assume the size of any data types.

```
FILE * data = fopen("data.bin", "rb");
fseek(data, 512, SEEK_SET);
int * re = malloc(sizeof(int)*100);
int ret = fread(re, sizeof(int), 100, data);
//Now re should contain a series of 100 integers
```

6

Question 4. [8 MARKS]

Memory Model

Part (a) [3 MARKS]

3

Write a main function that declares 3 strings. The first named `first` should be set to the value `"January"`, and be stored on the stack frame for `main`. `second` should be a string literal with the value `"February"`. `third` should have value `"March"` and be on the heap.

```
void main() {
    char [] first = "January";
    char * second = "February";
    char [] third = malloc(sizeof(char) * 6);
    third [0] = 'M';
    third [1] = 'a';
    third [2] = 'r';
    third [3] = 'c';
    third [4] = 'h';
    third [5] = '\0'; // add null terminator
}
```

Part (b) [3 MARKS]

2

Consider the code fragment below:

```
int a[10]; //on stack
int *i; // i is on stack
i = malloc(sizeof(int)*10);
    // this memory is on heap
```

List each block of memory that is allocated by the code fragment above, specifying both the **size** (in the form `N*sizeof(...)`) and **location** (i.e. stack, heap, or program data).

size	location	
2*sizeof(int*)	stack	← X and i pointers
10*sizeof(int)	stack	← memory allocated that a points to
10*sizeof(int)	heap	← memory allocated that i points to

Part (c) [1 MARK]

1

Arrays and pointers are the same thing: **TRUE** or (**FALSE**) (Circle the correct answer)

8

Question 5. [7 MARKS]

Consider the following `StudentNode` struct intended for building a linked list of students.

```
struct StudentNode {
    char *name;                 // The name of the student.
    int num;                    // The student number.
    struct StudentNode *next;   // Next node in the list (NULL if this is the last node).
};
```

Part (a) [7 MARKS]

7

Complete the function below that frees node `n` (if it exists) in the linked list; node `0` is the first node in the list. The `**p` parameter is a pointer to the head pointer of the linked list (the head pointer points to the first node of the list). Assume that both the node itself and the name inside were dynamically allocated.

Hint: Your function should handle three special cases in the following order: (1) The list is empty (i.e. head pointer is NULL). (2) `n = 0` (freeing the head node). (3) All other cases. Remember to fix any broken links.

```
void free_node(struct StudentNode **p, int n) {
    if (p == NULL) {
        return ; //i.e do nothing
    }
    if (n ==0) {
        free ((*p)-> name);
        struct studentNode * ns = (*p)->next;
        free (*p);
        *p = ns;
        return ;
    }
    int count = 0;
    struct studentNode **curr = p;
    while ((*curr)->next != NULL) {
        if (count == (n-1)) {
            struct studentNode * replace_next = ((*curr)->next)-> next;
            free (((*curr)->next)->name);
            free ((*curr)->next);
            *curr->next = replace_next;
            return 0;

        }else{
            *curr = (*curr)->next;
            count++;
        }
    }//if the while loop completes without returning then n > # of nodes
    //so i choose to do nothing, but we can emit an error message
}
```

Part (b) [2 MARKS]

Consider the code below, which uses the same `StudentNode` struct as the first part of this question.

```
void updateStudent(struct StudentNode s) {
  s.num = s.num + 1;
  strcat(s.name, " v2"); // Using unsafe function for simplicity; don't try this at home!
}

int main() {
  struct StudentNode s1;
  s1.num = 12345;
  s1.name = malloc(256*sizeof(char));
  strcpy(s1.name, "Bob");
  updateStudent(s1);
  printf("Student name: %s\n", s1.name);
  printf("Student number: %d\n", s1.num);
}
```

Print the exact output of the `main` function below.

Student name: Bob
Student number: 12345

[Use the space below for rough work. This page will not be marked unless you clearly indicate the part of your work that you want us to mark.]

[Use the space below for rough work. This page will not be marked unless you clearly indicate the part of your work that you want us to mark.]

C function prototypes:

```
pid_t fork(void);
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);
int lstat(const char *restrict path, struct stat *restrict buf);
void *malloc(size_t size);
void perror(const char *s)
int scanf(const char *restrict format, ...);
size_t strlen(const char *s)
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)
pid_t wait(int *stat_loc);
```

Excerpt from the fseek man page:

If whence is set to SEEK_SET, SEEK_CUR, or SEEK_END, the offset is relative to the start of the file, the current position indicator, or end-of-file, respectively.

Excerpt from the wait man page:

WIFEXITED(status)
: True if the process terminated normally by a call to _exit(2) or exit(3).

WEXITSTATUS(status)
: If WIFEXITED(status) is true, evaluates to the low-order 8 bits of the argument passed to _exit(2) or exit(3) by the child.

Print your name in this box.

Mital Topiwala

END OF EXAMINATION