UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE AND ENGINEERING

MIDTERM EXAMINATION, FEBRUARY 2025

DURATION: 1 hour and 50 minutes

ESC 190 H1F — Computer Algorithms and Data Structures

Calculator Type: None
Aids allowed: reference sheet distributed with the exam
Examiner(s): M. Guerzhoy

Student Number: _____

UTORid: _____

UofT email: _____@mail.utoronto.ca

Family Name(s): _____

Given Name(s): _____

---

*Do **not** turn this page until you have received the signal to start.*
In the meantime, please read the instructions below *carefully*.

This final examination paper consists of 6 questions on 12 pages (including this one), printed on both sides of the paper. *When you receive the signal to start, please make sure that your copy is complete, and fill in the identification section above.*

Answer each question directly on this paper, in the space provided. Use the pages at the end of the exam for extra space. If you use extra pages, indicate that you have done so in the space under the question.

Write up your solutions carefully! Comments are *not* required to receive full marks, except where explicitly indicated otherwise. However, they may help us mark your answers, and part marks *might* be given for partial solutions with comments clearly indicating what the missing parts should accomplish.

When you are asked to write code, *no* error checking is required: you may assume that all user input and argument values are valid, except where explicitly indicated otherwise. Unless specifically requested, you do not need to worry about memory leaks.

Use the C programming language. You may *not* `include` any library that would work on ECF, unless otherwise specified.

You **must** write your student number on every odd-numbered page of the exam (except empty pages). **Failure to so will result in a 3-point penalty.**

MARKING GUIDE

Q1:	_____/10
Q2:	_____/10
Q3:	_____/10
Q4:	_____/20
Q5:	_____/10
Q6:	_____/20
TOTAL:	_____/80

Question 1. [10 MARKS]

Write a function that takes in an address of the first element of a block of ints, and sets all the elements at even indices to 0. For example, if the input block is $\{5, 6, 7, 8\}$, it should be changed to $\{0, 6, 0, 8\}$. The function signature should be `void set_even_to_zero(int *block, int size)`.

```
// Function prototype
void set_even_to_zero(int *block, int size);

int main()
{
    // Example test case
    int array[] = {5, 6, 7, 8};
    int size = 4;

    printf("Before: ");
    for (int i = 0; i < size; i++) {
        printf("%d ", array[i]);
    }

    set_even_to_zero(array, size);

    printf("\nAfter: ");
    for (int i = 0; i < size; i++) {
        printf("%d ", array[i]);
    }

    return 0;
}

void set_even_to_zero(int *block, int size)
{









}
```

Question 2. [10 MARKS]

Given `sz` integers stored in a block starting with `arr`, return the median of the array `arr`. Assume that `sz` is odd. The median is the number such that at least $(sz + 1)/2$ elements are smaller or equal to it and at least $(sz + 1)/2$ elements are greater or equal to it.

```
#include <stdlib.h>

int get_median(int *arr, int sz)
{









}
```

Question 3. [10 MARKS]

A string is formatted as follows: it contains lower-case characters as well as zero or two of the upper-case letters 'X'. Your task is to find the number of characters not enclosed within the pair of 'X's, if they are there. If the X's are not in the string, all the characters are considered to be outside the 'X's.

For example `betweenX("1X2345X78")` should return 3, because $[1, 7, 8]$ are outside the two X's. For example `betweenX("11")` should return 2, because $[1, 1]$ are outside of the two X's.

```
#include <string.h>

int betweenX(char *s)
{









}
```

Question 4. [20 MARKS]

Part (a) [10 MARKS]

Complete this code, by completing the definition and the call to `f` so that the value of `a[1]` becomes 42.

```
void f(             )
{
    a[1] = 42;
}

int main()
{
    int arr[] = {1, 2, 3};
    f(              ); // a[1] should become 42

}
```

Part (b) [10 MARKS]

Write code that would make the string `s` contain the repetitions of 0123456789, 1,000 times. (So the string would be "012345678901234567890123456789012345678901234567890123…").

Hint: think about what you need to do with memory there.

```
#include <stdlib.h>
#include <string.h>

void make_str(             )
{









}

int main()
{
    char *s;
    make_str(       );
    printf("%s\n", s); // prints 012345678901234567890123456789012345678901234567890123......
}
```

Question 5. [10 MARKS]

Consider the following C structures representing a linked list:

```
typedef struct node {
    int value;
    struct node *next;
} node;
typedef struct LL{
    node *head;
    int sz;
} LL;
```

Write a function `prepend` that takes a linked list `LL` and an integer value, and adds a new node with the given value at the beginning of the linked list. The function should update the list's size accordingly and handle all edge cases appropriately.

If the value the integer 1 is prepended to the list $5 \to 6 \to 7$, the resulting list should be $1 \to 5 \to 6 \to 7$.

```
// Function prototype
void prepend(LL *list, int value);
int main()
{
    // Example test case
    LL list;
    list.head = NULL;
    list.sz = 0;

    prepend(&list, 5);  // List becomes: 5
    prepend(&list, 10); // List becomes: 10 -> 5
    prepend(&list, 15); // List becomes: 15 -> 10 -> 5

    // Print the list to verify
    node *current = list.head;
    while (current != NULL) {
        printf("%d ", current->value);
        current = current->next;
    }

    return 0;
}
```

(Implementation on next page)

```
// typedef struct node {
//     int value;
//     struct node *next;
// } node;
// typedef struct LL{
//     node *head;
//     int sz;
// } LL;

void prepend(LL *list, int value)
{









}
```

Question 6. [20 MARKS]

You are given a list of filenames, where each file corresponds to a particular class. Each file contains multiple rows, with each row consisting of a 10-digit student number followed by a space and then a mark between 0 and 100.

For example, a file might look like this:

```
1234567890 85
5678901234 92
9012345678 78
1234567890 90
```

A student number will appear once in the file if the student has taken the class, and will not appear otherwise.

Write a C program that reads all the files, calculates the average mark for each student (for just the classes the student took), and outputs the student number of the student with the highest average mark. You can assume there is only one such student.

You can assume that there are no more than 300 unique student numbers in the files all together.

```
// Function prototype
char* find_highest_average_student(char** files, int num_files);

int main() {
    char* files[] = {"ESC180.txt", "CIV102.txt", "ESC194.txt"};
    char* highest_student = find_highest_average_student(files, 3);
    printf("Student with highest average: %s\n", highest_student);

    // Don't forget to free any allocated memory.
    free(highest_student);  // this one is malloc'd
    return 0;
}
```

Assume that the following code will print out every line in file `i`:

```
void print_file_contents(char **files, int i) {
    FILE *file;
    char line[256]; // Buffer to store each line
    file = fopen(files[i], "r");
    while (fgets(line, sizeof(line), file) != NULL) {
        printf("%s", line); // Print the line
    }
    fclose(file);
}
```

```
char* find_highest_average_student(char** files, int num_files)
{

}
```

Use the space on this "blank" page for scratch work, or for any solution that did not fit elsewhere.
Clearly label each such solution with the appropriate question and part number.

Use the space on this "blank" page for scratch work, or for any solution that did not fit elsewhere.
Clearly label each such solution with the appropriate question and part number.