UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE AND ENGINEERING

MIDTERM EXAMINATION, FEBRUARY 2026

DURATION: 1 hour and 50 minutes

ESC 190 H1S — Computer Algorithms and Data Structures

Calculator Type: None
Exam Type: A

Aids allowed: Aid sheet distributed with the computer
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 midterm examination paper consists of 8 questions on 16 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.*

Write in the C programming language.

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. Note, though, that only code will actually be graded.

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. Calling `free` is not required unless indicated otherwise.

You may `include` any standard library (i.e., anything that would compile with `gcc` without installing extra libraries), unless otherwise specified.

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

MARKING GUIDE

Q1:	_____/	5
Q2:	_____/	15
Q3:	_____/	15
Q4:	_____/	15
Q5:	_____/	10
Q6:	_____/	15
Q7:	_____/	10
Q8:	_____/	15
TOTAL:	_____/	100

Good Luck!

Question 1. [5 MARKS]

Write a function

```
int sum(int *arr, int n)
```

that returns the sum of the `n` elements of `arr`.

For example, if `arr` contains `{3, 1, 4, 1, 5}`, then `sum(arr, 5)` should return `14`.

```
int sum(int *arr, int n)
{
```

Question 2. [15 MARKS]

Write a function

```
int is_palindrome(const char *s)
```

that returns `1` if the string `s` is a palindrome, and `0` otherwise. A string is a palindrome if it reads the same forwards and backwards. You may *not* use any `string.h` functions.

For example:

```
is_palindrome("racecar")  == 1
is_palindrome("hello")    == 0
is_palindrome("abba")     == 1
is_palindrome("a")        == 1
is_palindrome("")         == 1
```

```
int is_palindrome(const char *s)
{
```

Question 3. [15 MARKS]

Part (a) [5 MARKS]

Write a function `void rot3(int *a, int *b, int *c)` that rotates the values pointed to by its arguments: the value of `*a` goes to `*b`, `*b` goes to `*c`, and `*c` goes to `*a`.

For example, if before the call `*a == 1`, `*b == 2`, `*c == 3`, then after `rot3(&a, &b, &c)`, we should have `a == 3`, `b == 1`, `c == 2`.

```
void rot3(int *a, int *b, int *c)
{
```

Part (b) [5 MARKS]

Write a `main` function that creates an array of three `int`s, calls `rot3` to rotate the values in the array, and then prints the three values using `printf`.

```
int main()
{
```

Part (c) [5 MARKS]

Write a function `void my_strcpy(char *dest, const char *src)` that copies the string `src` (including the null terminator) into `dest`. You may *not* use any library functions (for examples, you may not `#include string.h`).

```
void my_strcpy(char *dest, const char *src)
{
```

Question 4. [15 MARKS]

Consider the following structure:

```
typedef struct student{
    char name[200];
    int grade;
} student;
```

Part (a) [6 MARKS]

Write a function

```
student **create_class(const char **names, int *grades, int n)
```

that uses `malloc` to allocate a block of `n` `student` pointers, then for each one allocates a `student` and fills in the `name` and `grade` fields. The function should return the block of pointers.

```
student **create_class(const char **names, int *grades, int n)
{
```

Part (b) [4 MARKS]

Write a function

```
void destroy_class(student **class, int n)
```

that frees all the memory allocated by `create_class`.

```
void destroy_class(student **class, int n)
{
```

Part (c) [5 MARKS]

Write a function

```
void print_above(student **class, int n, int threshold)
```

that prints the name of every student whose grade is strictly above `threshold`.

```
void print_above(student **class, int n, int threshold)
{
```

Question 5. [10 MARKS]

For each snippet below, indicate whether it will cause a potential segmentation fault (memory error) or undefined behaviour. If you say it causes a segmentation fault or undefined behaviour, explain why in one sentence. Assume `struct student` from Q4.

Part (a) [2 MARKS]

```
char *s = "hello";
s[0] = 'H';
```

☐ No problem ☐ Segfault or undefined behaviour

Part (b) [2 MARKS]

```
int *arr = (int *)malloc(3 * sizeof(int));
arr[3] = 10;
```

☐ No problem ☐ Segfault or undefined behaviour

Part (c) [2 MARKS]

```
student *s = (student *)malloc(sizeof(student));
free(s);
printf("%d\n", s->grade);
```

☐ No problem ☐ Segfault or undefined behaviour

Part (d) [2 MARKS]

```
char *s = (char *)malloc(5);
strcpy(s, "hello");
```

☐ No problem ☐ Segfault or undefined behaviour

Part (e) [2 MARKS]

```
student **block = (student **)malloc(3 * sizeof(student *));
block[0]->grade = 90;
```

☐ No problem ☐ Segfault or undefined behaviour

Question 6. [15 MARKS]

Scientific notation has the format:

```
[digits].[digits]e[sign][digits]
```

where `[sign]` is `'+'` or `'-'`.

For example:

```
"3.14e+0"   ->  3.14
"2.5e+3"    ->  2500.0
"1.0e-2"    ->  0.01
"9.81e+0"   ->  9.81
"6.67e-11"  ->  0.0000000000667
```

You may assume the input is always valid and always has exactly one digit before the decimal point, and always contains `'e'`.

Write a function

```
double my_sci_to_double(const char *s)
```

that converts the string to a `double`. Do *not* use `atof` or `strtod`/`{strtok}`.

Hint: Parse the part before `'e'` as a `double`, parse the exponent after `'e'` as an integer, then multiply by 10^{exponent}.

```
double my_sci_to_double(const char *s)
{
```

Question 7. [10 MARKS]

Write a *recursive* version of `strncmp`:

```
int my_strncmp(const char *s1, const char *s2, int n)
```

The function compares at most `n` characters of `s1` and `s2`. It returns a negative value if `s1` comes before `s2` lexicographically, 0 if they are equal (up to `n` characters), and a positive value if `s1` comes after `s2`.

You must use recursion and you may *not* use loops of any kind (no `for`, `while`, or `do-while`). You may *not* use any `string.h` functions.

```
int my_strncmp(const char *s1, const char *s2, int n)
{
```

Question 8. [15 MARKS]

You are given a dictionary file where each line contains exactly one word, and the words are sorted in alphabetical order. For example, the file `dict.txt` might contain:

```
apple
banana
cherry
grape
mango
orange
```

Part (a) [8 MARKS]

Write a function

```
char **read_dict(const char *filename, int *p_n)
```

that reads all words from the dictionary file into a dynamically allocated array of strings. The function should store the number of words in `*p_n` and return the array. You may assume each word is at most 100 characters long.

```
char **read_dict(const char *filename, int *p_n)
{
```

Part (b) [7 MARKS]

Write a function

```
int lookup(char **dict, int n, const char *word)
```

that returns 1 if `word` is found in the sorted dictionary array `dict` (of size `n`), and 0 otherwise. Your function *must* run in $O(\log n)$ time.

```
int lookup(char **dict, int n, const char *word)
{
```

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.