UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE AND ENGINEERING

FINAL EXAMINATION, APRIL 2025

DURATION: 2½ hours

ESC 190 H1S — Computer Algorithms and Data Structures

Calculator Type: None
Exam Type: B

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 10 questions on 22 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 and docstrings 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. Calling `free` is not required unless indicated otherwise.

In Python, you may not `import` any module except `math`, unless otherwise specified. In C programs, you may `include` any standard library (i.e., anything that would compile with `gcc` without installing extra libraries).

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:	_____/ 10
Q5:	_____/ 10
Q6:	_____/ 10
Q7:	_____/ 10
Q8:	_____/ 10
Q9:	_____/ 10
Q10:	_____/ 10
TOTAL:	_____/100

Good Luck!

Question 1. [10 MARKS]

Part (a) [8 MARKS]

In C, write a function that takes in an array of integers and its size, and returns 1 if there are two different elements in the array that sum to `target` and 0 otherwise.

```
int has_sum(int *arr, int n, int target)
{
    // your code here
```

Part (b) [2 MARKS]

What is the runtime complexity of the function you wrote in Part (a)? Justify your answer.

Question 2. [10 MARKS]

In C, write a function that takes in an array of integers and its size, and returns the number of elements in the array that are greater than the average of all the elements in the array. Assume n is at least 1.

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

Question 3. [10 MARKS]

Given a string that contains spaces and the words `"zero"`, `"one"`, `"two"`, and `"three"` in some order, spelling out a number, compute that number. For example,

`read_out("one two one zero")` should return 1210

`read_out("zero one")` should return 1.

Assume the string only contains words as specified above, separated by spaces.

```
int read_out(char *digits)
{
```

Question 4. [10 MARKS]

Part (a) [2 MARKS]

Complete the missing parts of these functions so that the value of `a` would become 42.

```
void change_a(        )  // fill in parameter
{
                         // fill in line
}
int main()
{
    int a;
                         // call change_a

    return 0;
}
```

Part (b) [2 MARKS]

Complete the missing parts of these function so that the value of `arr[0]` would become 42.

```
void change_arr(        )  // fill in parameter
{


                          // fill in line
}
int main()
{
    int arr[10];
                         // call change_arr


    return 0;
}
```

Part (c) [6 MARKS]

Consider the following structure:

```
typedef struct student{
    char *name;
    int age;
} student;
```

Write a function that creates a block of memory where you can store `N` students, for an arbitrary positive number `N`. Then, write a `main` function to use the function you wrote to create a block of 3 students. Then, write a function that would take in an element of the block of students, and capitalize the first letter in the student's name. The implementation details are left up to you, except that your code must work as specified. For this question, you do not need to `free` any `malloc`-ed space.

Question 5. [10 MARKS]

Consider the following structure for a singly linked list in C:

```
typedef struct node {
    int value;
    struct node* next;
} node;
```

Part (a) [5 MARKS]

Write a function `int sum_iterative(node* head)` that computes the sum of all values in the linked list whose first node is `head` using a loop. Your function should return 0 if the list is empty.

```
int sum_iterative(node *head)
{
```

Part (b) [5 MARKS]

Write a function `int sum_recursive(node* head)` that computes the sum of all values in the linked list whose first node is `head` using a recursive approach. Your function should return 0 if the list is empty. You must not use loops of any kind (no `for`, `while`, or `do-while` loops).

```
int sum_recursive(node *head)
{
```

Question 6. [10 MARKS]

In C, write a function `int is_palindrome(char *str)` that takes in a C string and returns 1 if the string is a palindrome and 0 otherwise. A palindrome is a string that reads the same forward and backward (e.g., "racecar", "madam", or "a").

You must use recursion and you may write and call your own helper functions, but you must not use loops of any kind (no `for`, `while`, or `do-while` loops) anywhere and you must not call `string.h` functions.

Note: You may assume that the input string:

- Is null-terminated
- Contains only lowercase letters
- Has a length of at least 1

```
int is_palindrome(char *str)
{
```

Question 7. [10 MARKS]

Suppose you want for a list of integers to work as follows:

```
IntList *L1;
create_list(&L1, 3)
int elems[] = {4, 5, 6};
set_elems(L1, elems, 3); // L1 now stores the three
                         // elements [4, 5, 6]
set_elem(L1, 2, 42)      // The elem at index 2 in L1 becomes
                         // 42, so that L1 is now [4, 5, 42]
```

Part (a) [5 MARKS]

Write the code necessary to enable this. In particular, write the definition of `IntList` and write `set_elems` and `set_elem`. In the `main` function, show how you would use the code to create three different lists.

Part (b) [5 MARKS]

Write a function that takes in an array of pointers to `IntList` and sorts them in lexicographic/alphabetic order. That is, sort by the first element, in case of ties in the first element sort by the second element, etc.

You may use `qsort`, but are not required to do so. Note that the reference sheet contains information on `qsort`.

Question 8. [10 MARKS]

You are given N days to study one of Programming, Praxis, and Calc. For every day, you are given the amount of knowledge points that you would gain if you study Programming, Praxis, Calc, and Linear Algera. The amount of knowledge (an integer, which could be positive or negative) gained from studying the above subjects on day i is given in the lists:

```
prog[i]
prax[i]
calc[i]
linalg[i]
```

On each day, you must study at most one subject, or none at all.

You have a rule that you cannot study math two days in a row. That means your schedule for 6 days could be `[prog, prog, prax, calc, prax, linalg]`

but the three-day schedules `[linalg, calc, prax]` or `[calc, calc, prax]` are not allowed.

Your goal is to take in the lists `prog, prax, calc, linalg`, and to output the studying schedule such that the sum of knowledge points gained is as large as possible.

Write a function that takes in the lists `prog, prax, calc, linalg` and prints the schedule for days 0, 1, 2, ..., n-1, where n is the length of `prog` (which is equal to the lengths of the other lists). A sample output might be

`["prog", "prog", "prax", None, "calc"]`. Output `None` if on a particular day the best strategy is to not study at all.

Knowledge points could be negative.

Suppose we have the following knowledge points for 2 days:

$$\begin{aligned} \texttt{prog} &= [2, 3] \\ \texttt{prax} &= [1, 4] \\ \texttt{calc} &= [5, 1] \\ \texttt{linalg} &= [3, 6] \end{aligned}$$

The optimal study schedule would be:

Day 0: `calc` (5 knowledge points)
Day 1: `prax` (4 knowledge points)

Total knowledge points: $5 + 4 = 9$

Note that although `linalg` would give 6 points on day 1 (more than `prax`), it cannot be selected because we studied `calc` (a math subject) on day 0, and the constraint prohibits studying math subjects on consecutive days.

For the above example, the function `best_study_schedule(prog, prax, calc, linalg)` should output:

```
["calc", "prax"]
```

(Write your solution on the next page.)

```
def best_study_schedule(prog, prax, calc, linalg):
```

Question 9. [10 MARKS]

You are given a maze represented as a 2D grid, where:

- Each cell contains either a positive integer representing the penalty for stepping on that cell, or the character `'X'`, which represents a blocked cell that cannot be traversed. Penalties are always positive.
- Your goal is to find the path from the start position to the end position with the minimum total penalty.
- You can only move to adjacent cells horizontally or vertically (left, right, up, or down), not diagonally.

An implementation of Dijkstra's algorithm that you must use (but need to modify and complete) is given on the next page. Note that this implementation is expecting a `Node` class with a `get_neighbors()` method/function and an `id` attribute/variable (i.e., the name of the node. For example, you can use the coordinates as the name). You will need to create an appropriate `Node` class and any other supporting code needed to use this implementation.

Part (a) [5 MARKS]

Write a function `find_shortest_path(maze, start, end)` that:

1. Takes a 2D list `maze` where each cell contains a positive integer penalty or 'X' for a blocked cell
2. Takes the coordinates of the `start` and `end` positions as tuples `(row, col)`
3. Prints the minimum total penalty to reach the end position and the path as a list of coordinates
   You must use the provided Dijkstra's algorithm implementation (with any necessary supporting code) to solve this problem. You'll need to modify and/or extend the implementation to keep track of the path.

Sample input/output are given below.

```
maze = [
    [2, 3,  1],
    [6, 'X',2],
    [2, 1,  3]
]
start = (0, 0)  # Top-left corner
end = (2, 2)    # Bottom-right corner
```

Expected output:

```
Minimum penalty: 11
Path: [(0,0), (0,1), (0,2), (1,2), (2,2)]
```

You can modify the Dijstra code by crossing out lines and writing your own. You likely need to modify the signature. For your convenience, the code below is also available on the reference sheet.

```
import heapq
def dijkstra(start_node):
    # Dictionary to store the shortest distance to each node
    distances = {start_node.id: 0}


    # Elements are tuples of (distance, node)
    priority_queue = [(0, start_node)]

    # Set of visited nodes
    visited = set()

    while priority_queue:


        current_distance, current_node = heapq.heappop(priority_queue)


        if current_node.id in visited:
            continue


        visited.add(current_node.id)


        for neighbor, weight in current_node.get_neighbors():

            if neighbor.id in visited:
                continue


            distance = current_distance + weight


            if neighbor.id not in distances or distance < distances[neighbor.id]:



                distances[neighbor.id] = distance



                heapq.heappush(priority_queue, (distance, neighbor))




    return distances
```

SPACE FOR SOLUTIONS

Part (b) [5 MARKS]

Modify the Dijkstra's algorithm (copied below) to implement the A* algorithm instead. Your modified algorithm should still work with the same `Node` class implementation from part (a).

The function `star` must print out the path it found and its cost.

```
def astar(start_node):
    # Dictionary to store the shortest distance to each node
    distances = {start_node.id: 0}

    # Elements are tuples of (distance, node)
    priority_queue = [(0, start_node)]

    # Set of visited nodes
    visited = set()

    while priority_queue:


        current_distance, current_node = heapq.heappop(priority_queue)


        if current_node.id in visited:
            continue


        visited.add(current_node.id)


        for neighbor, weight in current_node.get_neighbors():

            if neighbor.id in visited:
                continue


            distance = current_distance + weight


            if neighbor.id not in distances or distance < distances[neighbor.id]:



                distances[neighbor.id] = distance



                heapq.heappush(priority_queue, (distance, neighbor))



    return distances
```

SPACE FOR SOLUTIONS

Question 10. [10 MARKS]

Suppose a training set of images with corresponding labels (`"Paul"`, `"Ringo"`, `"George"`, `"John"`) is given. Each image is a 64×64 grayscale photo represented as a list of lists, where each pixel value ranges from 0 (darkest) to 255 (brightest).

For illustration purposes, here are some examples of what entries in the dataset might look like, simplified to 3×3 pixel images:

```
# Example of what entries in the dataset might look like
example_training_entries = [
    ([[45, 190, 55],
      [180, 54, 185],
      [50, 210, 48]], "John"),

    ([[210, 45, 205],
      [58, 195, 62],
      [198, 52, 202]], "Paul"),

    ([[190, 51, 182],
      [58, 188, 60],
      [55, 190, 58]], "George"),

    ([[62, 205, 59],
      [185, 60, 192],
      [65, 201, 63]], "Ringo"),

    ([[175, 48, 170],
      [63, 192, 65],
      [50, 195, 53]], "George"),

    # ...
]
```

Write a function `nearest_neighbor(training_data, new_image)` that implements the 1-nearest neighbor algorithm to classify a new image. Make any reasonable design choices. The function takes in a 64x64 image and returns a name. For this question, you may import `math` only.

```
def nearest_neighbour(training_data, new_image):
```

SPACE FOR SOLUTIONS

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.