UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE AND ENGINEERING

APS 105 — Computer Fundamentals
Final Examination
April 28, 2023 6:30 p.m. – 9:00 p.m. (150 minutes)

Examiners: P. Anderson, S. Emara, B. Korst, K. Shakiba

First name (please write as legibly as possible within the boxes)

Last name

UTorID

This is a "closed book" examination; no aids are permitted. No calculators or other electronic devices are allowed.

All questions are to be answered on the examination paper.

You must use the C language to answer programming questions. You are not required write `#include` directives in your solutions.

The examination has 26 pages, including this one.

Question 1 [2 Marks]

Write a single C statement that generates a random character between `'A'` and `'Z'` **(inclusive)**, and assigns it to a `char` variable named `randomCharacter` that you declare in the same statement.

Question 2 [2 Marks]

In a single statement, declare and initialize an array of 500 integers called `numbers` where the first 4 numbers are `1, 2, 3, 4` and the remaining 496 numbers are all `0`.

Question 3 [4 Marks]

Given the following defined data structure, write a single C statement terminated by **one** ;, that declares two variables named `plant1` and `plant2` of the type of the following data structure. In the same statement, initialize `plant1`'s name and height with `"Cactus"` and `5.3` respectively, and `plant2`'s name and height with `"Rose"` and `10.2`, respectively.

```
typedef struct plant {
  char name[30];
  double height;
} Plant;
```

Question 4 [6 Marks]

You are on a team to write code to help a client who runs an online store. **Note:** you will not be writing code to solve this question! Here is the situation needing a programmed solution:

A user enters a name (first and last). The program reads information from a database on the disk and scans it for the name that was entered, and takes the accompanying information for that name (ID, address, phone, email address) and reads a sales information database for orders in the last year from that ID. These are all to be displayed for the user.

The question for you to answer: In one phrase or sentence each describe 6 non-overlapping tasks that could be functions in your final code, or that might be divided into other functions. (These could share function calls in the final implementation.)

Task #	Description
1	
2	
3	
4	
5	
6	

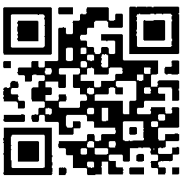
Question 5 [6 Marks]

The following code has a serious mistake in getNames function.

```
typedef struct student {
  char* lastName;
} Student;

void getNames(char* lastName, Student students[]) {


  for (int i = 0; i < 4; i++) {


    printf("Enter last name: ");


    scanf("%s", lastName);


    students[i].lastName = lastName;
  }
}

int main() {
  Student students[4];
  char lastName[1024];
  getNames(lastName, students);
  for (int i = 0; i < 4; i++) {
    printf("students[%d].lastName = %s\n", i, students[i].lastName);
  }
  return 0;
}
```

It produces the following output. The input is shown in bold.

Enter last name: **Handa**
Enter last name: **Brown**
Enter last name: **Li**
Enter last name: **Jamil**
students[0].lastName = Jamil
students[1].lastName = Jamil
students[2].lastName = Jamil
students[3].lastName = Jamil



Revise `getNames` function (by crossing out parts and writing the revision above) such that the output is as the desired below. You can assume the maximum string size is 1024.

Enter last name: **Handa**
Enter last name: **Brown**
Enter last name: **Li**
Enter last name: **Jamil**
students[0].lastName = Handa
students[1].lastName = Brown
students[2].lastName = Li
students[3].lastName = Jamil

Question 6 [6 Marks]

In the quicksort sorting algorithm, one of the key components is the function `partition` that divides the list between "high" and "low" into two parts – a part less that the pivot and a part greater than the pivot. It is part of an algorithm to sort a list low to high. The code for this is below.

Revise this code (by crossing out parts and writing the revision above) such that the revised `partition` function can be used in an algorithm to sort a list from highest value to lowest, **i.e. in descending order not ascending order**.

Note: You may have been introduced to the `partition` function as `quickSortHelper` in your lectures.

```
int partition(int list[], int low, int high) {

   int pivot = low, left = low + 1, right = high;

   while (true) {

      while (left <= right && list[left] <= list[pivot])

         left++;

      while (left <= right && list[right] > list[pivot])

         right--;

      printf("left = %d, right = %d\n", left, right);

      if (left < right)

         swap(list, left, right);

      else {

         swap(list, pivot, right);

         return right;

      }
   }
}
```

Question 7 [8 Marks]

The following code intends to find a node data equal to key and return a pointer to the node preceding (before) it. Complete the condition of the while loop in the box shown below such that current is pointing to the preceding node, and no segmentation fault is caused. You will notice from the code that LinkedList and Node data structures were the same ones used in class.

```
Node* prevSearch(LinkedList* list, int key) {
  Node* current = list->head;
  if (list->head == NULL || list->head->data == key) {
    return NULL;
  }

  // write condition of while loop here

  while(                                                    ){
    current = current->next;
  }
  if (current->next == NULL) {
    return NULL;
  }
  return current;
}
```

Question 8 [8 Marks]

A program that simulates a circuit with impedances in series makes use of a linear linked list. Each impedance in the circuit is a node in the linked list represented by the data structure shown below. The simulation program aims at calculating the current flowing through the circuit.

```
struct Node{
    double resistance;
    double reactance;
    double voltage;
    struct Node *next;
};
```

Write a function to calculate the total magnitude of the impedance in the circuit. Since all of them are in series, the magnitudes of each of them are added. In this function you will write, you are to use the function `calculateMagnitude` to calculate the magnitude of one impedance, shown below

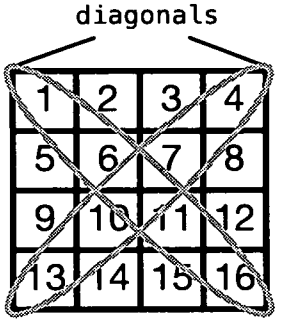
```
double calculateMagnitude (double resistance, double reactance){
    return sqrt(pow(resistance, 2) + pow(reactance, 2));
}
```

The prototype of your function is shown below, where `ptr` is a pointer to the first impedance/node or head of the linked list and `n` is the number of nodes.

```
double equivalentMagnitude (struct Node *ptr, int n){
```

Question 9 [10 Marks]

Write a C function called `diagonalSum` that returns the sum of both the diagonal elements of a square `n`×`n` array named `square`. Diagonal elements are circled in the following figure. **Note** if `n` is odd, the central element should be counted twice in your solution.




```
int diagonalsSum(int n, int square[][n]) {
```

Question 10 [12 Marks]

You are to complete the function below that will sort an array of integers `inArray[]` that are within the range 0 and 599. It will do this by:

1. creating a second integer array `countArray[]` of size 600, initially set to zero.
2. for each value in the input array, incrementing the element of the `countArray` array with the index equal to that value.

   For example, if it encounters a 15 in the first array it will add one to element `countArray[15]`. At the end, each element of `countArray[]` will indicate the number of each value. For example, `countArray[15]` will indicate how many 15s are in `inArray[]`.
3. go through `countArray[]` in order to populate a third array, `outArray[]`, that has the elements of `inArray[]` in order.

The function will return the number of elements in `outArray[]`.

You can assume that `outArray[]` is large enough to contain all the elements of `inArray[]` (this value is passed in `sizeOfInArray`. Your solution must use the methodology described. There is an additional page to use for your answer.

```
int sortArray(int inArray[],int outArray[], int sizeOfInArray){

    int countArray[600];
```

Please continue to write your solutions to the previous question on this page if needed.

Question 11 [10 Marks]

Write C code for a *recursive* function named `printReverse` that reads a sequence of positive integers and outputs the sequence in reverse order. **The input sequence ends when the user enters the number 0.**

Assume the user always gives a valid input. Your `printReverse` function must be recursive, otherwise the solution will not receive any marks.

You may **not** declare any arrays in your solution.

Below is a sample output from an execution of the program:

```
Enter num: 10 <enter>
Enter num: 5 <enter>
Enter num: 2 <enter>
Enter num: 1 <enter>
Enter num: 0 <enter>

Reversed sequence: 0 1 2 5 10
```

Note: no newline character is printed after the last number in the reversed sequence.

Write your `printReverse` function below.

```
void printReverse(){
```

Please continue to write your solutions to the previous question on this page if needed.

Question 12 [10 Marks]

Write a C function named `removeSpecialChars` that takes a string and removes any characters that are not lower case alphabets: not between `'a'` and `'z'` (inclusive). Assume `<string.h>` is included. For example, if we run the code below, the output should be `appleisafruit`.

```
int main(void) {
  char str[30] = "a&pp*(leisa(fr*$uit";
 printf("str with special characters: %s\n", removeSpecialChars(str));
  return 0;
}

  char* removeSpecialChars(char* str) {
```

Please continue to write your solutions to the previous question on this page if needed.

Question 13 [12 Marks]

Write a **recursive** function called `numOccurrences` which takes two strings as parameters. The function should return the number of occurrences of the second string within the first string. You may use any of the functions defined within the `string.h` library as discussed in class. Your function must adhere to the provided prototype, and it must be **recursive** or else you will lose all the marks for this question.

```
int numOccurrences(const char *str, const char *search){
```

Please continue to write your solutions to the previous question on this page if needed.

Question 14 [15 Marks]

The Node structure in a linked list has been defined as follows:

```
typedef struct node {
  int data;
  struct node *next;
} Node;
```

The LinkedList structure has also been defined to contain the head of a linked list:

```
typedef struct linkedList {
  Node *head;
} LinkedList;
```

A linked list list has nodes ordered in ascending order according to data; however, some **consecutive** nodes are missing. These consecutive nodes are in another ascending ordered linked list named sequence. For example, list = 1 –> 2 –> 5 and sequence = 3 –> 4. You can safely assume that no two nodes have the same data.

Write a C function called insertSequence, the prototype of which is given below, that inserts a linked list named sequence into in another linked list named list. sequence and list have the data in their nodes sorted in ascending order. sequence should be inserted in a way to maintain the ascending order of list.

Example 1

```
list = 1 -> 2 -> 12 -> 18
sequence = 7 -> 8 -> 9
```

list after calling insertSequence will be 1 -> 2 -> 7 -> 8 -> 9 -> 12 -> 18, as 7 -> 8 -> 9 was inserted between 2 and 12.

Example 2

```
list = 1 -> 2 -> 5 -> 7 -> 9
sequence = -1 -> 0
```

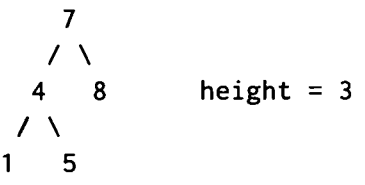
list after calling insertSequence will be -1 -> 0 -> 1 -> 2 -> 5 -> 7 -> 9, as -1 -> 0 was inserted in the front.

```
void insertSequence(LinkedList *list, LinkedList* sequence) {
```

Please continue to write your solutions to the previous question on this page if needed.

Question 15 [15 Marks]

The height of a binary tree is defined as the maximum number of edges from a leaf node to the root node of the tree. This is illustrated in the following example:



The height of the tree is 3 as there are 3 generations of nodes. A tree with just 1 node has a height of 1. As each node in a tree is a tree itself, we can calculate the height of each subtree separately. In the provided example, the height of the subtree starting at the (4) node is 2 as there are at most 2 generations of nodes between it and the leafs of the tree.

Write a **recursive** function that, given a tree, calculates the height of the **largest subtree whose first node contains an even number**. In the provided example, this height would be 2 as the largest subtree that begins with a node containing an even number is the (4) subtree and it has a height of 2. A tree with only odd numbers has a height of 0.

You have been provided a prototype for a `heightFromEven` function. You may define new functions if you wish; however, ultimately, this function must be implemented and return the height of the input tree (i.e. you must call any functions you wish to create from this function). The definitions of the binary tree and node structs have been provided as well the function's prototype. **You may not use any global variables.**

```
typedef struct node {
    int data;
    struct node *left, *right;
} Node;

typedef struct binarytree {
    Node *root;
} BinaryTree;

int height(BinaryTree *tree);
```

Please continue to write your solutions to the previous question on this page if needed.

This page has been left blank intentionally. You may use it for answers to any question in this examination.