UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE AND ENGINEERING

APS 105 — Computer Fundamentals
Final Examination
April 24, 2019

Examiners: P. Anderson, M. Badr, B. Li, A. Poraria

Exam Type D: This is a "closed book" examination; no aids are permitted.

Calculator Type 4: No calculators or other electronic devices are allowed.

All questions are to be answered on the examination paper. If the space provided for a question is insufficient, you may use the last page to complete your answer. If you use the last page, please direct the marker to that page and indicate clearly on that page which question(s) you are answering there.

You must use the C programming language to answer programming questions. You are not required write `#include` directives in your solutions. The code provided to you may not have `#include` directives either. Except those excluded by specific questions, you may use functions from the `math` library as necessary.

The examination has 21 pages, including this one. If you use the extra page(s) note this on the original question page.

First Name: _____ Last Name: _____

Your Student Number: _____

MARKS

1-3	4-5	6	7	8	9	10	11	12	13	14	Total
/12	/8	/5	/6	/8	/8	/9	/10	/8	/13	/13	/100

This page is blank and may be used for extra work — be sure to refer to it in the question area to ensure that it is marked.

Question 1 [4 Marks]

In a single C statement create a data structure called `AnimalSizes` having two elements, a string `name` and a size `length`. This same statement should create an array of this type called `snakes` with two entries.

Initialize each entry such that the first index has name `"Anaconda"` and length 3.7, while the second index has the name `"Python"` and length 2.4. You may do this in the same statement or using additional statements.

Write your solution in the box below.

Question 2 [4 Marks]

Write a single C statement that declares a boolean type variable named `inputIsBefore` that assigns `true` to `inputIsBefore` if and only if the string `inputArtist` (of type `char *`) comes before the string `artist` in lexicographical order. You may assume that `string.h` and `stdbool.h` have been #include'd in the code.

Question 3 [4 Marks]

The following array of integers is the result of the first round of partitioning, used in the Quicksort algorithm to sort the array in ascending order. Identify the possible array element or elements that could have been used as the pivot in the first partitioning round. Justify your answer; guessing an answer with no justifications will result in a mark of zero.

```
{15, 6, 45, 60, 32, 71, 102, 81}
```

Question 4 [4 Marks]

Consider the following code segment, assuming that `jVar` is an `int` array:

```
int i;

for (i = 0; i < MAX; i++) {
  if (jVar[i] < 3) {
    printf("X\n");
  }
}
```

(a) Rewrite this code segment as a `while(...) {...}` loop;

(b) Rewrite this code segment as a `do {...} while (...)` loop.

Note that each of your solution code must perform exactly as the code provided.

Question 5 [4 Marks]

We have a number of TAs who have carefully marked a large number of final exams and now must sort them alphabetically.

[PLEASE NOTE that this question is not worth many marks, so answer with a phrase! Do not spend time elaborating.]

(a) Here are some sorting methods you know about. Which ones would work well, and which not well to allow the TAs to most quickly sort the exams? Why or why not would the particular method work well or not?

Method	*OK?*	*Reason*
Selection sort		
Bubble sort		
Insertion sort		
Quicksort		

(b) What may be a better sorting method than any of the methods above under these circumstances?

Question 6 [5 Marks]

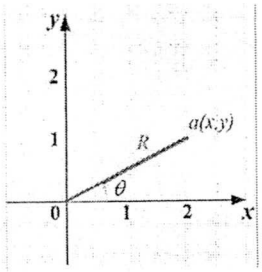
Consider the following code:

```
int main(void) {
    int i = 17;
    int myArray[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    double j = 5.4;

    for (int i = 4; i > 2; i--) {
      myArray[1] = myArray[1] + 1;
    }

    for (int bb = 7; bb < 10; bb++) {
      myArray[bb] = myArray[bb] - 1;
      bb++;
    }

    for (int k = 2; k < 5; k += 3) {
        myArray[k] = 4.73;
    }

    double x = (int) ((int) j + (3.78 + 9));

    // here
}
```

At the location marked `here`, what are the values of variables (or array elements) in the table below? Please show all non-zero decimal places; if the variable cannot be used, enter `unavailable`.

Variable	*Value*
`i`	
`j`	
`k`	
`x`	
`myArray[0]`	
`myArray[1]`	
`myArray[2]`	
`myArray[3]`	
`myArray[4]`	
`myArray[5]`	

Question 7 [6 Marks]



In various engineering applications, it is required to convert Polar to Cartesian coordinates. The polar coordinates (R, θ) and rectangular coordinates (x, y) are related as follows:

$x = R\cos(\theta)$ and $y = R\sin(\theta)$

Complete the code below, which defines the data structures and a function that takes polar coordinates of a point and return the rectangular coordinates. The angle is assumed to be in radians. Assuming the values passed to this function is in degrees, you need to convert the radian value using the constant D2R below.

```
#include <math.h>

const double D2R = 3.1415926535 / 180.0;

// rectangular coordinate structure
typedef struct rectV {




}                       ;

// polar coordinate structure
typedef struct polarC { // angle in radians




}                       ;

// polar to rectangular
RectCoor polToRec (PolarCoor polin) {







    return rv;
}
```

Question 8 [8 Marks]

As you might remember from the midterm: In a Pascal's Triangle, the first row, row #0, has a single element 1. Each succeeding row elements are the sum of the two elements just above (if there is only one number just above, then that number is duplicated). So the first 5 rows (numbering from zero) are:

```
        1
      1   1
    1   2   1
  1   3   3   1
1   4   6   4   1
```

Looking at the last row, row #4, we have sums: $0+1, 1+3, 3+3, 3+1, 1+0$ (getting the values from the row above) to give us $1, 4, 6, 4, 1$.
A way of calculating the row elements of a Pascals Triangle is that element r in a row (numbering elements and rows from 0) is

$$\frac{n!}{r!(n-r)!} \tag{1}$$

Where $n!$ is n factorial, *i.e.*, $n! = n \times (n-1) \times (n-2) \times \ldots \times 1$. (0! Is defined as 1.) Write a function that calculates the factorial then use it to find the elements of row #7 (counting from 0) of the Pascals Triangle in your main function. Use the following program outline:

```
int factorial(int n) { // returns n!
```

```
    return result;
}

int main(void) {
    int const RowDesired = 8; //counts from 1
    int pascalRow[8]; //hint: row #n has n+1 elements
```

Please continue your solution to Question on this page:

Question 9 [8 Marks]

Write a function, `countLetters`, that counts the number occurrences of each alphabetical letter found in a string. The function has two parameters: a string (i.e., `char *`) and an array of integers.

The string should not be modified and can be any length. You may assume that the string is null-terminated and all letters are lower case. The string may contain characters that are not part of the alphabet (e.g., 0, 1, !, &, etc). The integer array has a size of 26, one for each letter in the alphabet. The first index corresponds to the letter `'a'`, the second index to the letter `'b'`, and so on. You may assume that, initially, all 26 elements in the array have a value of zero.

You must abide by the following constraints. Failure to meet a constraint will result in a grade of zero for this question.

1. You cannot modify the characters inside the string.
2. You cannot create any other data structures (e.g., array, linked list, etc).
3. Your function must only access valid indices of the array.
4. You cannot call any functions in your implementation.

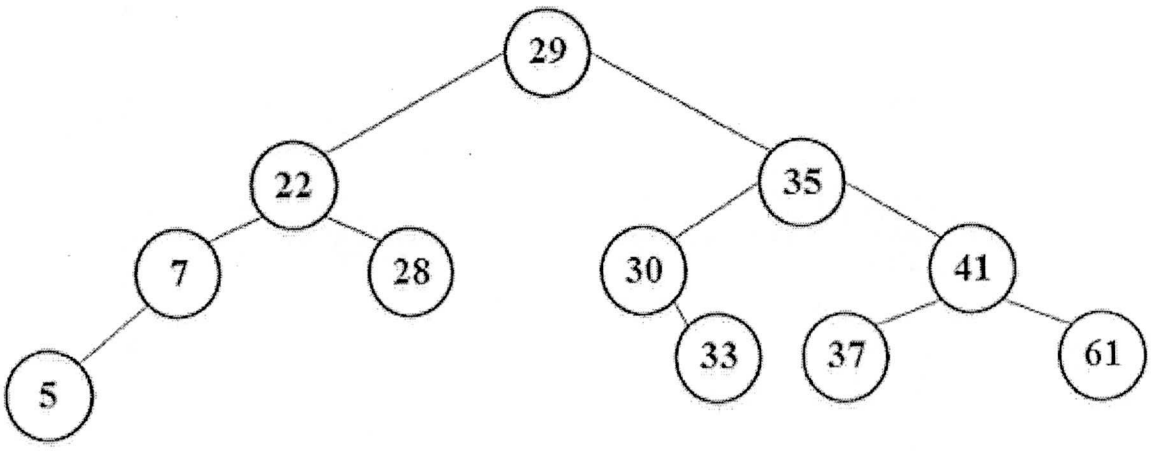
An example of one run of the program is below. You **only** need to implement the `countLetters` function. Assume a main function (that reads in a string, calls your function, and outputs the integer array) already exists.

```
Enter a sentence: hello, world
The frequency that each lower case letter appears is:
d: 1
e: 1
h: 1
l: 3
o: 2
r: 1
w: 1
```

```
int countLetters(char *s, int count[]) {




}
```

Question 10 [9 Marks]

Consider the binary search tree below:



The lowest common ancestor, LCA, of two nodes p and q is either one of the two nodes having the other as a descendant, or the lowest node in the tree that has both p and q as descendants.
For example, in the binary search tree above:

- The LCA of nodes 5 and 28 is 22.
- The LCA of nodes 7 and 30 is 29.
- The LCA of nodes 35 and 61 is 35.
- The LCA of nodes 30 and 41 is 35.

Complete the recursive helper function, with the prototype given on the next page, which takes the tree root and two values of node data, and returns the LCA (a node) of the given nodes.

Assume that the two `int` values provided always exist in the tree.

Assume that the binary search tree is designed as follows:

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

typedef struct bstree {
    Node *root;
} BSTree;
```

The main function, `lca()`, uses an auxiliary helper function to perform the task; this helper function is called inside the `lca` function:

```
Node * lca(BSTree *tree, int na, int nb) {

      return lcaHelper(tree->root, na, nb);
}


Node * lcaHelper(Node * p, int na, int nb) {
```

Question 11 [10 Marks]

In general, the bubble sort algorithm can be explained in two steps.

1. For each pair of adjacent elements: if they are out-of-order, then swap.
2. Repeat the first step until no swaps are done.

Write a function, `bubbleSortLinkedList`, that sorts a linked list using the bubble sort algorithm. The function has one parameter: a pointer to a LinkedList (assume that this pointer is not NULL). The function will modify the linked list **in-place** so that the values are in ascending order (i.e., 1 comes before 2). The definitions for a `LinkedList` and `Node` are shown below.

You must abide by the following constraints. Failure to meet a constraint will result in a grade of zero for this question.

1. Your function must not modify the `next` pointer of any node.
2. You cannot create any other data structures (e.g., an array, another linked list, etc.).
3. Your function cannot call any other functions.
4. Your function must not cause a segmentation fault.

Definitions of Node, LinkedList:

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

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

Implement the function on the next page. Do any drawing here:

```
void bubbleSortLinkedList(LinkedList *list) {




}
```

Question 12 [8 Marks]

Complete a recursive helper function and a main calling function to scan an existing binary search tree and create a new tree with only the even values from the first tree. You can use the functions developed in class, including:

```
void initBSTree(BSTree *tree);
bool insert(BSTree *tree, int value);
```

You may assume that the following data structure types have been defined:

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

typedef struct bstree {
    Node *root;
} BSTree;


Node *treeWithEvens(BSTree *inputTree) {

}
```

```
bool evensHelper(Node *current, BSTree *evenTree) {
// helper function: check this node and connected nodes using a recursive call,
// placing even values in the new tree



}
```

Question 13 [13 Marks]

Complete the definition of a C function `permutate`, whose prototype is shown below, that prints all permutations of the string `str`. For example, if the string passed to this function is defined as:

```
char str[] = "abc";
```

A call to `permutate(str)` will print:

```
abc
acb
bac
bca
cba
cab
```

You **must** use recursion to solve this problem[1].

Hint: You may wish to define a helper function, and you may also find the `swap()` function defined below useful:

```
void swap(char *a, char *b) {
    char temp = *a;
    *a = *b;
    *b = temp;
}

void permutate(char *str) {
```

[1] In fact, a non-recursive implementation would be more challenging.

–continue your work here —

Question 14 [13 Marks]

You are given an array of strings, `char *inStrings[]`. Complete the function `char *alignText(char inStrings[], int lineLength)` where the input is a "period terminated" array of strings and the output a pointer to a new string with newlines inserted so that no output line will exceed `lineLength`.

As an example, if the array holds:

```
"This", " is", " my", " example", " for", " the", " exam", " question", "."
```

and the line length given is 15, the output string should be:

```
"This is my\n example for the\n exam question.\n"
```

Note the "." terminator in the input strings. It will always appear exactly this way. You can assume the input array of strings always has this terminator. Do not worry about handling words longer than `lineLength`.

You should allocate room for the new string that is large enough to accommodate the string. (Small over-allocations of space are allowed if it makes it easier for you). You may use the functions from string.h

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

char *alignText(char *inStrings[], int lineLength){
    char *outString; //assemble the string here
    int charcount = 0; //counts number of characters
```

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