UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE AND ENGINEERING

APS 105 — Computer Fundamentals
Final Examination
April 24, 2018
6:30 p.m. – 9:00 p.m.
(150 minutes)

Examiners: B. Li, B. Korst, H. Shokrollah-Timorabadi and M. Stumm

Exam Type A: 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. Except those excluded by specific questions, you may use functions from the `math` library as necessary.

The examination has 17 pages, including this one.

First Name: Bill Last Name: Hu

Student Number: key

MARKS

1-2	3-4	5	6	7	8	9	10	11	12	13	14	15	16	Total
/8	/8	/4	/4	/4	/4	/6	/6	/6	/10	/10	/10	/10	/10	/100

Question 1 [4 Marks]

Assume you have a function declared as:

```
void specialSort(SpecialType a[], int arraySize);
```

This function is able to sort an array of elements of type `SpecialType`. It takes two parameters: `a`, a pointer to the array to be sorted, and `arraySize`, the number of elements in the array.

Further, assume you have an array `specialArray` that was declared as follows:

```
SpecialType specialArray[1000];
```

Assume that all 1000 elements in this array have been initialized with randomly generated data. Write a single C statement that calls the function `specialSort()` so that it sorts `specialArray`, the array with 1,000 elements.

specialSort (specialArray, 1000);

Question 2 [4 Marks]

Write a single C statement that generates a random even number in the range of `[-150, 150]` (inclusive), and uses it to declare and initialize an `int`-type variable `randomChoice`.

int randomChoice = (rand()%151)*2 − 150;

rand()%151 ∈ [0, 150]

=> (rand()%151)*2 ∈ [0, 300] even number

=> (rand()%151)*2 − 100 ∈ [-150, 150] even number

Question 3 [4 Marks]

Consider the following declarations:

```
typedef struct name {
    char *firstname;
    char *lastname;
} Name;

typedef struct employee {
    int SIN;
    int employeeNumber;
    Name *emplName;
} Empl;

Empl employees[1000];
```

Assume that all 1000 elements in the `employees` array have been initialized and *none* of the pointers are `NULL`. Write a single C statement that declares a character variable `c` and assigns it the first character of the last name of the second employee in the `employees` array.

```
char c = ((employees[1].emplName)->lastname)[0];
```

Question 4 [4 Marks]

Write a single C statement that declares a variable called `intPtrArray`, initialized to point to an array of 10 integer pointers that is dynamically allocated.

```
int **intPtrArray = malloc (sizeof(int*) * 10);
```

Question 5 [4 Marks]

Complete the following C program, designed to search for an int-type item, called key, in a linked list, pointed to by head.

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

Node *search(Node *head, int key) {
    Node *current = root;

    // insert your code in the line below between the parentheses

    while ( current != NULL && current->data != key      ) {

        current = current -> link;
    }
    return current;
}
```

Question 6 [4 Marks]

Without using any functions in the standard C library (including all string-related functions), write a C function `stringLength()` that takes a string `str` as its only parameter, and returns the number of characters in the string. If `str` has a value of `NULL`, the function should return 0.

```
int stringLength(char *str) {
    if (str == NULL)                    // Special Case: No existing string
        return 0;
    int count = 0;
    while (*(str++) != '\0')            // For every non-null character, increment count
        count++;
    return count;
}
```

Question 7 [4 Marks]

Evaluate the following relational expressions by circling the right answer.

`'\0' == 0`	false	(true)	'\0' has ASCII code of 0
`int x = 10 % 8;` `(x > 0) && (x % 2 == 0) && !false`	false	(true)	x = 2
`'c' - 3 == 'a'`	(false)	true	'c' - 'a' == 2
`int w = rand() % 75 * 2 - 99;` `(w < -99) || (w > 49)`	(false)	true	$0 \le$ rand() % 75 ≤ 74 $\therefore -99 \le w \le 74 \times 2 - 99 = 49$ $\therefore w < -99 ==$ false $w > 49 ==$ false

Question 8 [4 Marks]

What does the following program output?

```
int correct(int a) {
  int b;

  if (a == 0) {
    b = 0;
  } else {
    b = a % 2 + 10 * correct(a / 2);
  }

  return b;
}

int main(void) {
  int number;

  number = correct(121);
  printf("The correct value for 121 is: %d\n", correct(121));
  return 0;
}
```

The correct value for 121 is: 1111001

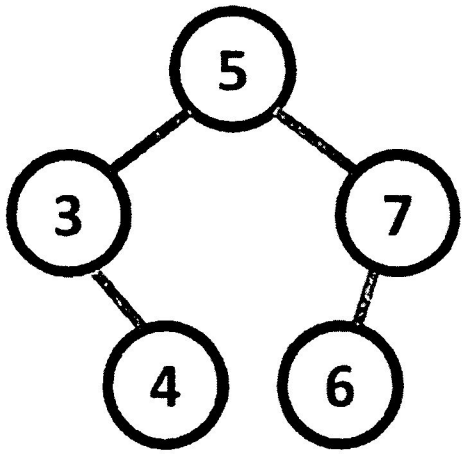
/*

a	b
121	1111 00×10+1 = 1111001
60	1111 0×10+0 = 111100
30	1111×10+0 = 11110
15	111×10+1 = 111
7	11×10+1 = 111
3	1×10+1 = 11
1	0×10+1 = 1
0	0

*/

Question 9 [6 Marks]

Consider the following binary search tree:



This tree may have been created by inserting the elements in the following order: 5, 3, 7, 4, 6.

Or it may have been created by inserting the elements in the following order: 5, 7, 3, 6, 4.

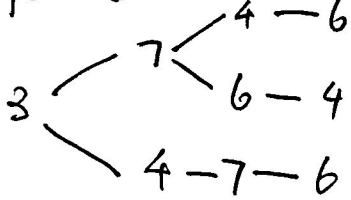
But a different tree would have been created by inserting the elements in the following order: 5, 4, 6, 7, 3.

How many different ways can the elements {3, 4, 5, 6, 7} be inserted into a binary tree so that the same tree is created as in the figure above?

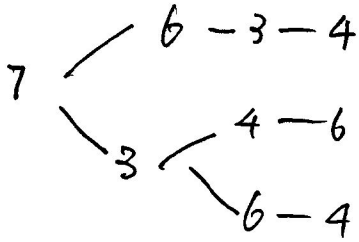
Must insert ⑤ before ③ or ⑦

Must insert ④ before ③, ⑦ before ⑥

If ③ is the second element, then



If ⑦ is the second element, then



∴ 6 different ways.

{5, 3, 7, 4, 6} {5, 3, 7, 6, 4} {5, 3, 4, 7, 6}

{5, 7, 3, 4, 6} {5, 7, 3, 6, 4} {5, 7, 6, 3, 4}

Question 10 [6 Marks]

Identify and correct all compile-time errors you find in the C program below. Compile-time errors are **errors** — not **warnings** — that the compiler will report when compiling the program. Each line may or may not contain compile-time errors, and there may be more than one error per line.

```
1  #include <stdio.h>
2  #include <stdlib.h>

3  typedef struct node {
4      int data;
5      struct node *left, right;
6  } Node;

7  Node *insert(Node *root, int item) {
8      if (root == NULL) {
9          return newNode(item);
10     }

11     if (item <= (*root).data)
12         insert(root -> left, item)
13     return root;
14 }

14 int main(void) {
15     int list[] = {15, 3, 2};
16     Node *root = NULL;

17     for (int i = 0; i < 13; i ++) {
18         root = insert(root, list[i]);
19     }
19     return 0;
20 }
```

Please write your answer using the table on the next page. You will be penalized if the errors you have identified are not compile-time errors.

Line #8

Line #	Description of error	Correction
5	right ~~bos~~ is not a pointer to structure	Struct node *left, *right;
	Structure cannot contain itself	
#12	Missing semi-colon	insert(root->left, item);

Question 11 [6 Marks]

Write a C function called `preamble()` that takes two parameters: a string `str` and an `int`-type integer `n`. The function will then return a new string that is dynamically allocated, and that contains **at most** the first `n` characters in the string `str`. For example, if `str` is "Toronto", and `n` is 3, then the function will return "Tor" (the first three characters in "Toronto"). If `str` is "Toronto" and `n` is 8, then the function will return "Toronto". If `str` is `NULL`, the function will also return `NULL`.

```
char *preamble(char *str, int n) {
    int len = strlen(str); // Get the length of string (excluding '\0')
    int min = fmin(len, n); // Take the smaller number
    if (str == NULL)
        return NULL;
    char *str1 = malloc(sizeof(char) * (min + 1)); // +1 for '\0'
    for (int i=0; i < min; i++)
        str1[i] = str[i];
    // Alternatively, strncpy(str1, str, min);
    str1[min] = '\0'; // Set the null character
    return str1;
}
```

Question 12 [10 Marks]

The constant E is defined as a double constant of 2.718281828459045.

```
const double E = 2.718281828459045;
```

A first positive integer is called a **mirror** of a second one if they both contain two digits, and when the two digits in the first integer are flipped, the first integer becomes the second one. For example, 81 is a mirror of 18 (and vice versa).

Implement a function called `firstMirrorInE()` that returns the first two-digit number found in consecutive digits of E whose mirror have appeared earlier in the sequence of digits. You should only consider the first 16 digits of E — 2718281828459045. The function returns `0` if such a mirror pair does not exist in the first 16 consecutive digits of E. Your program must extract the digits from the constant variable E.

Hint: The `firstMirrorInE()` function should return 28, since its mirror, 82, has appeared earlier in the sequence of digits. Your function must not simply return 28 without doing any work. It is also incorrect to return 81, because even though its mirror, 18, appeared previously, 81 is not the first in the sequence that can be found.

Feel free to declare and implement additional functions when needed.

Please write your solution to the question here and continue on next page:

```
int firstMirrorInE(void) {
  const double E = 2.718281828459045;
  int digits[16]; double EE = E;
  for (int i=0; i<16; i++) {
      int digit = (int)EE; // Take the first digit by integer casting
      digits[i] = digit; // Store into array
      EE = (EE - digit) * 10; // Update EE
  }
  for (int i=0; i<15; i++) {
       int digit1 = digits[i];
       int digit2 = digits[i+1];
       for (int j=0; j<i; j++) // Loop all choices before the current number
           if (digits[j]==digit2 && digits[j+1]==digit1) //If match
                return digit1 * 10 + digit2;
  }
  return 0; //If not found
}
```

Question 13 [10 Marks]

The following C structure is used to define each node in a linked list:

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

Write a C function called `printDuplicates` that receives a pointer to the first node (*head*) of a linked list as a parameter. The function should find and print the duplicate integers in the linked list. For example, if the linked list contains the integers `6, 3, 3, 6, 7, 4`, then the `printDuplicates()` function should print:

```
6
3
```

Note: In your solution, you may assume that a given integer occurs at *most* twice in the linked list.

```
void printDuplicates(Node *head) {
    //Method #1: O(n²) Brute force
    Node *current;
    int val;
    while (head != NULL) {
        val = head->data; // Retrieves the target value for comparison
        current = head->link; // Start searching from the second element
        while (current != NULL) {
            if (current->data == val) {    //If match
                printf("%d\n", val);
                break; // No need to continue searching
            }
            current = current->link;
        }
        head = head->link;
    }
}
```

Please continue your solution to Question 13 on this page:

```
// Method 2 : O(n) keep a list of appeared numbers

void clearList (Node *head) {   // Frees all nodes in the list
    while (head != NULL) {
        Node *second = head->link;
        free (head)
        head = second ;              // Proceeds to free the next node
    }
}

bool search ( Node *head, int val ) {  // Search if a number has appeared
    while (head != NULL ) {
        if (head->data == val)   //If found, return true
            return true;
        head= head->link;
    }
    return false; // If not found. return false
}
void printDuplicates (Node *head) {
    Node *dupList = NULL; // Declare a linked list for storing duplicates
    while (head != NULL) {
       int val = head->data ;
       if (search (dupList, val))
            printf ("%d\n", val);   //If found, print the duplicate number
       else {
            Node *n = createNode (val);   // Create a new node
            n->link = dupList;    // Store the new number at front
            dupList= n;
       }
       head = head->link;
    }
    clearList (dupList);  // Must free!
}
```

Question 14 [10 Marks]

Consider the following function that returns the index of a char c in a string string (i.e., the position of the first c in the string), or returns -1 if c does not occur in string:

```
int findIndex(char *string, char c) {
    int n = 0;
    while (*string != c && *string != '\0') {
       string = string + 1;
       ++n;
    }
    if (*string == '\0')
       return -1;
    return n;
 }
```

Write a C function recursiveFindIndex(char *string, char c) that does not use any loops and yet behaves like the findIndex() function above. Your function may have additional parameters, but at the minimum must include the parameters string and c.

```
// main function: calls recursiveFindIndex(string, c, 0);
int recursiveFindIndex(char *string, char c, int index) {
    if (string[index] == '\0')    //Base Case #1: Reached end of string
        return -1;

    if (string[index] == c)       //Base Case #2: Found first occurrence of c
        return index;             //Terminate recursion

    return recursiveFindIndex(string, c, index+1); //Searches the next index
}
```

Question 15 [10 Marks]

Write a C function called `sortOddEven()` that rearranges the order of the elements in an integer array such that all odd numbers are to the left of all even numbers. The function has two parameters: a pointer to the integer array and an integer specifying the number of elements in the array. The odd numbers can be in any order, as long as they are all to the left of any even number, and the even numbers can be in any order, as long as they are all to the right of any odd number.

For example, if the elements of the array initially are:
1 4 6 5 9 3 8 2
then after `sortOddEven()` processes the array, the elements may become:
1 3 9 5 6 4 8 2

Note: In your solution, you may not declare or use another array.

```
void sortOddEven(int input[], int size) {
    int left = 0, right = size-1; // Set the left & right for partition
    while (true) {
        while (input[left] %2 == 1 && left <= right)
            left++;    //Find the leftmost even number
        while (input[right]%2==0 && right >= left)
            right++;   //Find the rightmost odd number
        if (left < right) {//If still not partitioned
            int temp = input[left];
            input[left] = input[right];
            input[right] = temp;
        } else {
            return; // Done
        }
    }

}
```

Question 16 [10 Marks]

The following C structure is used to define each node in a binary search tree:

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

Write a C function:

```
Node *secondLargestNode(Node *root);
```

that finds and returns a pointer to the node that contains the *second largest* value in the binary search tree. The parameter root is a pointer to the root node of a binary search tree. If the binary search tree is empty or has one node only, the function returns NULL. For example, if the function is called with root pointing to the binary search tree on page 7, it will return a pointer to the node that contains 6.

```
Node *secondLargestNode(Node *root) {
    if (root == NULL) || (root->left == NULL && root->right == NULL))
        return NULL;  // Two special cases : no node or only one node
    Node *curr = root , *parent = NULL ;
    while (curr->right != NULL) {
        parent = curr;
        curr = curr->right;   // Find the largest element in the tree
    }
    if (curr->left == NULL )   // If it is the leaf node
        return parent;         // Second largest element is the parent
    //If the node has a left subtree
    //The second largest element is the largest of this subtree
    curr = curr-> left;
    while (curr-> right != NULL)
        curr = curr-> right;
    return curr;
}
```

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