UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE AND ENGINEERING

APS 105 — Computer Fundamentals
Final Examination
December 21, 2015
9:30 a.m. – 12:00 p.m.
(150 minutes)

Examiners: J. Anderson, B. Li, J. Rose

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 18 pages, including this one.

First Name: ______________________ Last Name: ______________________

Student Number: ____

MARKS

1	2	3	4	5	6	7	8	9	10	11	12	13	14	Total
/2	/2	/2	/3	/4	/3	/3	/2	/8	/10	/10	/12	/11	/10	/82

Question 1 [2 Marks]

Rewrite the following code using a **for** loop rather than a while loop:

```
int i = 1;
while (i < 10) {
    printf("%d\n",i);
    i++
}
```

Solution:

```
for (int i = 1; i < 10; i++)
    printf("%d\n",i);
```

Question 2 [2 Marks]

Write *one or more* C statements that use `malloc` to dynamically allocate an array of 1000 elements of type `double`. The allocated array should be called `list`.

Solution:

```
double *list;
list = (double *)malloc(1000*sizeof(double));
```

Question 3 [2 Marks]

Consider the following C-language statement, in which the variable x has been previously declared as an integer:

```
    x = ((rand() % 50) + 1) * 2;
```

What is the largest value that any execution of this statement would ever place into x?

Solution:
100.

Question 4 [3 Marks]

Consider the following C-language program:

```
int main(void) {

double A[4] = {2.2, 2.2, 3.0, 4.0};
double s = 100.0;

for (int i = 3; i >= 0; i--) s = s + A[i];
printf("%5.2lf\n",s);
return (EXIT_SUCCESS);
}
```

(a) What is the exact output of this program?

Solution:
111.40

(b) Name one style rule, taught in the labs in this class, that this program violates.

Solution:
Poor choice of variable names OR lack on indentation.

Question 5 [4 Marks]

The following C function is intended to convert any lower-case characters in a string `str` into upper case. Identify and fix **four** bugs in the function. State the line number and re-write the line below.

```
1: void toUpper(char *str)
2: {
3:   char *p;
4:   p = str;
5:   while (p != ’\0’) {
6:     if (*p >= ’a’ || *p <= ’z’)
7:       *p = ’A’ + *p - a;
8:     (*p)++;
9:   }
10: }
```

Solution:

```
5: while (*p != ’\0’)
6: change || to &&
7: change -a to -’a’
8: change to p++
```

Question 6 [3 Marks]

Consider the following code, where `head` points to the head of a linked list:

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

Node *head;
```

Complete the following C function that returns a pointer to the node that precedes (comes before) `searchNode`, if it is found in the linked list. If `searchNode` is not found or the linked list is empty, the function should return `NULL`. Read the function carefully: you may need to add code in several locations.

```
Node *predecessor(Node *head, Node *searchNode) {
  Node *current;
  current = head;

  if (head == searchNode)
    return NULL;

  // COMPLETE THE NEXT LINE:
  while (                                    ) {
    if (current -> link == searchNode)
      return current;

   // WRITE CODE HERE:


  }

  return NULL;
}
```

Solution:

```
Node *predecessor(Node *head, Node *searchNode) {
  Node *current;
  current = head;

  if (head == searchNode)
    return NULL;

  while (current != NULL) {
    if (current -> link == searchNode) {
      return current;

    // write your code HERE:
    current = current -> link;
  }

  return NULL;
}
```

Question 7 [3 Marks]

Will the following C program execute successfully without encountering an error that will stop the program? If your answer is yes, show the output from running this program. If your answer is no, explain the reason.

```
#include <stdio.h>

void func(int *p, int *q) {
  int *t;
  t = p;
  *p = *q;
  *q = *t;
}

int main(void) {
  int i = 0, j = 1;
  int *p, *q;
  p = &i;
  q = &j;
  func(p, q);

  printf("%d, %d.\n", *p, *q);
  return 0;
}
```

Solution: Yes, it will run successfully, and the output is `1, 1.`

Question 8 [2 Marks]

There have been two plenary lectures since the midterm in this course, and there is one question from each of these lectures below. You must answer both questions, and the answer should be one or two sentences.

Plenary Lecture 6 — An Electrical Engineer's Journey into Software (Professor Vaughn Betz)
Professor Betz described several examples of computer simulations, all of which were based on the simulation of a specific thing/phenomenon. What was that thing? That is, what was being simulated? In addition, give one example of what those simulations were used to design.

Solution:
First answer: Electro-magnetic radiation, or wireless signals. Second Antennas for low-frequency radio for US submarines or MRIs machines' radiation/protection.

Plenary Lecture 2 — Software at Google Scale (Danyao Wang)
What does the "20% time" at Google mean?

Solution: At Google, each employee can take 20% of their work week's time to work on a project of their own choosing that they believe could become useful.

Question 9 [8 Marks]

In mathematics, the number of ways of choosing a set k items from a larger set of n items is denoted by $\binom{n}{k}$, defined as follows:

$$\binom{n}{k} = \frac{n!}{k! \cdot (n-k)!} \tag{1}$$

Write a *complete* `C` function, for which the prototype is given below, that computes and returns $\binom{n}{k}$. You may not use *any* functions from `math.h` in your solution.

Reminder: $n!$ (factorial) is valid for non-negative integers n, where $n! = n \cdot (n-1) \cdot (n-2) \ldots \cdot 2 \cdot 1$, and $0! = 1$.

Note: In your solution, you must write and then use a function to compute factorial.

Solution:

```
int fact(int n)
{
int ret = 1;
for (; n > 0; n--)
     ret *= n;
    return ret;
}

int nChooseK(int n, int k) {
     return fact(n) / (fact(k) * fact(n-k);
}
```

Question 10 [10 Marks]

Consider the following `C` `struct` and `typedef` declarations:

```
struct studentRecord {
  char *name;
  double GPA;
};

typedef struct studentRecord Record;
```

Write a complete `C` function `bubbleSortRecords`, for which the prototype is given below, that uses the Bubble Sort algorithm to sort an array of `n` elements of type `Record`. The function should sort the elements in *ascending* order of `GPA`, and in the event of a tie, should break ties in *alphabetical* order of `name`, where `name` is a pointer to a string. You may use a function from the string library, `string.h`, in your answer. Hint: use the library function to compare two strings.

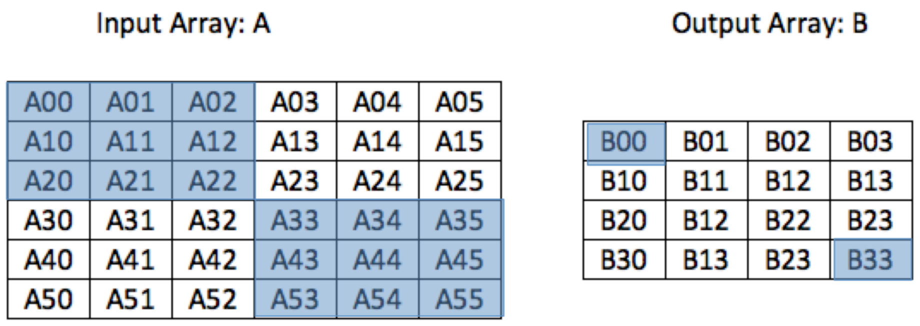
Solution:

```
void bubbleSortRecords(Record records[], int n) {
  int i,top;
  top = n-1;
  while (top > 0) {
    for (i = 0; i < top; i++) {
      if ((records[i].GPA > records[i+1].GPA) ||
  ((records[i].GPA == records[i+1].GPA) && (strcmp(records[i].name,records[i+1].nam
    Record temp = records[i];
    records[i] = records[i+1];
    records[i+1] = temp;
      }
    }
    top--;
  }
}
```

Question 11 [10 Marks]

Pictures that come from a smartphone camera can be represented as a two-dimensional array of numbers, where each number corresponds to the colour of each pixel in the picture. These cameras suffer from various effects that cause errors in the colour they measure (this variation is often called 'noise') which makes the picture have poor quality. One way to reduce that noise is to average each pixel with the eight pixels that surround it.

In this question, you will write a C-language function that takes in such an image as an array, **A**, of 100 by 100 of double-type numbers. It computes the 'averaged' array, **B**, which is a slightly smaller 98 by 98 array in which each element is the computed average of 9 pixels in the input array. The figure below illustrates the computation for a smaller example version of A (a 6x6 array) that would produce a 4x4 array B. (The reason that B is smaller than A by 2 is that, at the edges of A, there are not enough "surrounding" pixels to produce a full result for B).



Input Array: A

A00	A01	A02	A03	A04	A05
A10	A11	A12	A13	A14	A15
A20	A21	A22	A23	A24	A25
A30	A31	A32	A33	A34	A35
A40	A41	A42	A43	A44	A45
A50	A51	A52	A53	A54	A55

Output Array: B

B00	B01	B02	B03
B10	B11	B12	B13
B20	B12	B22	B23
B30	B13	B23	B33

The computation can also be described by these examples: the element B00 of the output array B is computed as the arithmetic average of the 9 elements of A that are shaded in the upper left corner of array A in the above figure. That is, B00 is the average the nine elements surrounding and including element A11 (i.e. A00, A01, A02, A10, A11, A12, A20, A21, A22). Similarly, B01 is the average of the 9 elements surrounding and including A12. B02 is the average of the 9 surrounding/including to A13. B10 is the average of the 9 surrounding/including A21. As a final example, B33 is the average of the 9 surrounding/including A44, the set of which is illustrated as the shaded section of the lower right hand set of values in the array A above.

You are to write a C function with a prototype as follows:

```
void averageImage(double A[100][100], double B[98][98]);
```

Where A is the input array, and B is the array that the function computes. A specific requirement is that you must use one or more loops to compute the average of the 9 elements of A. That is, you *must not* explicitly write out the sum of nine separate elements of A to compute each element of B.

Solution:

```
void averageImage(double A[100][100], double B[98][98]) {
    double sum;

    for (int i = 0; i < 98; i++) {
        for (int j = 0; j < 98; j++) {
            sum = 0.0;

        for (int k = 0; k < 3; k++) {
            for (int m = 0; m < 3; m++)
                sum = sum + A[i+k][j+m];

        }
        B[i][j] = sum/9.0;
      }
    }
}
```

Question 12 [12 Marks]

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

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

Assume that nodes in the linked list are maintained in order of their values, such that the value stored in each node is greater than or equal to the value in predecessor nodes. Write a C function:

`void simplify(Node *head)`

that deletes any duplicate items in the linked list. The parameter `head` is a pointer to the head node of a linked list. Note that the head node of the linked list will remain unchanged after the deletions are made. For example, if before calling `simplify`, the linked list contains:

```
13 13 15 15 17 17 17 19 22 25 25 28
```

then after calling the function, the list should contain:

```
13 15 17 19 22 25 28
```

Solution:

```
void simplify(Node *head) {
  Node *current;
  current = head;
  if (current == NULL)
    return;

  while (current -> link != NULL) {
    if (current -> value == current -> link -> value) {
      Node *nodeToRemove = current -> link;
      current -> link = current -> link -> link;
      free(nodeToRemove);
    }
    else
      current = current -> link;
  }
}
```

Question 13 [11 Marks]

Two strings are said to be anagrams of each other if they are of the same length, and contain the same characters in a (possibly) different order. For example, "elvis" is an anagram of "lives", and "the morse code" is an anagram of "here come dots". You are to write a C function isAnagram, with the prototype given below, that returns true if the two strings s1 and s2 are anagrams of each other; otherwise, it returns false. In your code, you may modify any of the characters in s1 and s2, and may use any function in the string.h library. You can also assume that both strings will **only** contain characters from the alphabet (upper case and lower case) and the space character. Also, note that the upper case A is NOT considered as the same character as lower case a.

Solution:

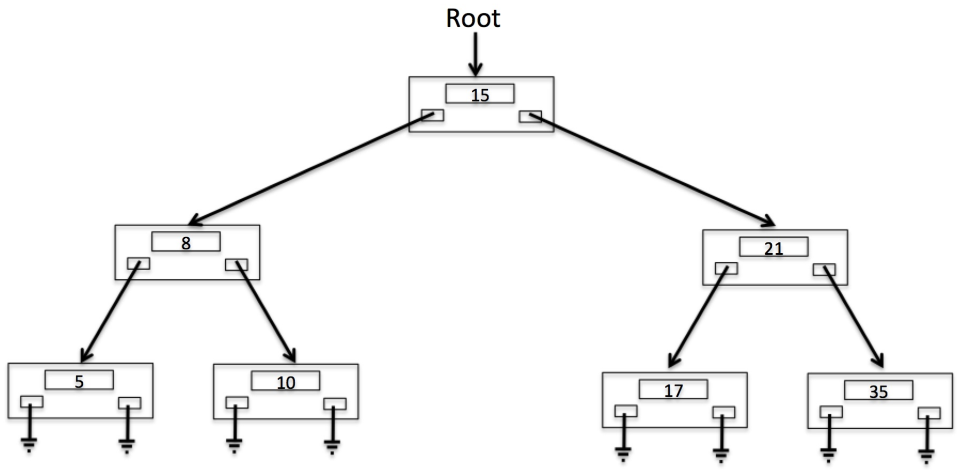
```
bool isAnagram(char *s1, char *s2) {
  if (strlen(s1) != strlen(s2))
    return false;

  for (int i = 0; i < strlen(s1); i++) {
    char *p = strchr(s2, s1[i]);

    if (p != NULL)
      // replace the character found from s2
      *p = '?';
    else
      return false;
  }
  return true;
}
```

Question 14 [10 Marks]

Recall the binary sorted tree data structure, described in class, an example of which is illustrated below:



Assume that each node in this tree is defined by the structure/type with an associated root pointer as follows:

```
typedef struct btree {
    int value;
    struct btree *leftLink;
    struct btree *rightLink;
    } treeNode;

treeNode *root;
```

Write a C function (the prototype of which is given below) which counts and returns the number of nodes in the entire tree that are *less than or equal* to the value of the **threshold** parameter.

An example of a call to this function is as follows: `countBelow(20,root);` If this call was made with root pointing to the tree illustrated above, the answer returned would be 5.

Note two additional requirements:

- Your solution *must* use recursion - solutions without recursion will receive 0 marks.
- Your solution *must* make use of the binary sorted tree structure to avoid visiting nodes unneccessarily.

```
int countBelow (int threshold, treeNode *startNode);
```

Solution:

```
int countBelow (int threshold, treeNode *startNode) {

        int meAndBelow = 0;
        if (startNode != NULL) {  // base case NULL, send back the 0
            // always look down left subtree, because if we got here, we must look
            meAndBelow += countBelow(threshold, startNode->leftLink);

            // check the current node to see if we have to count that

            if (startNode->value <= threshold) {
                    meAndBelow++;

                // if the current node is greater than or equal to the threshold
                // don’t have to look down right side, making use of BST structure!
                if (startNode->value < threshold)
                    meAndBelow += countBelow(threshold, startNode->rightLink);
            }
         }

    }
```