SOLUTIONS - DON'T READ TIL YOU'VE TRIED TO SOLVE THE PROBLEMS YOURSELF!

UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE AND ENGINEERING

APS 105 — Computer Fundamentals
Final Examination
December 15, 2014
9:30 a.m. – 12:00 p.m.
(150 minutes)

Examiners: B. Li, J. Rose, H. Timorabadi, B. Wang

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 to write `#include` directives in your solutions. You may use any math function that you have learned, 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	15	Total
/2	/3	/3	/3	/4	/4	/4	/2	/8	/10	/11	/11	/11	/12	/12	/100

Question 1 [2 Marks]

A student in APS 105 wrote a C program, expecting that its output would be as follows:

```
The value of x is 36.
```

Instead, the following output was produced by the code:

```
The value of x is 30.
```

Here is the code:

```
1  int x = 50;
2  int xBig = 30;

3  while (x > xBig)
4    x -= 2;
5    xBig++;

6  printf("The value of x is %d.", x);
```

Please explain why the code does not behave as expected, and indicate the necessary correction(s) for the code to generate the expected output.

Solution: Only line 4 was executed inside the `while` loop.

```
while (x > xBig) {
  x -= 2;
  xBig ++;
}
```

Question 2 [3 Marks]

Write a single C statement that dynamically allocates an array of 25 real numbers (with `double` types) and assigns the address of the array to a pointer called `dptr`.

Solution:

```
double *dptr = (double *) malloc(25 * sizeof(double));
```

Question 3 [3 Marks]

The following function will separate a number into three parts: a sign (which is set to one of +, −, or blank), a whole number magnitude, and a fractional part. (Recall that the <math.h> library is available as described on the front page of the exam.) There are several mistakes in this code. State what the error is, and give the corrected code.

```
void separate(double number, char *signPtr,
              int *wholePtr, double *fracPtr) {

  double magnitude;

  if (number < 0)
    *signPtr = '-';
  else if (number == 0)
    *signPtr = ' ';
  else
    *signPtr = '+';

  magnitude = fabs(number);
  wholePtr = floor(magnitude);
  fracPtr = magnitude - wholePtr;
}
```

Solution:

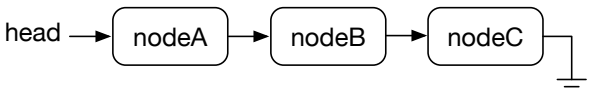
```
*wholePtr = floor(magnitude);
*fracPtr = magnitude - *wholePtr;
```

Question 4 [3 Marks]

Consider the following code:

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

Node *head;
```

There are three nodes in a linked list that uses this structure (labeled `nodeA`, `nodeB`, and `nodeC`), as shown in the following diagram:



Write a C code segment that deletes `nodeB`, which is no longer required by the program.
Solution:

```
Node *temp = head -> link;
head -> link = head -> link -> link;
free(temp);
```

Question 5 [4 Marks]

Will the following C program run successfully? If your answer is yes, show the output from running this program. If your answer is no, explain the reason.

```
#include <stdio.h>

int main(void) {
  int *x;
  int result;

  *x = 0;
  result = *x;
  printf("%d", result);
  return 0;
}
```

Solution: No, it will not run successfully, since the memory that the integer pointer `x` points to has not been allocated.

Question 6 [4 Marks]

An APS 105 student started to write a C program that reads a positive integer value from user input, and prints its digits in reverse order. Here are two examples:

Example 1:

```
Enter a positive integer: 723
327
```

Example 2:

```
Enter a positive integer: 54
45
```

Please complete his C program below to achieve this goal.

```
int main(void) {
  int number;
  printf("\nEnter a positive integer: ");
  scanf("%d", &number);
  do {
    // Please write your code here.
```

```
  } while (number > 0);

  printf("\n");
  return 0;
}
```

Solution:

```
do {
  printf(%d, number % 10);
  number /= 10;
} while (number > 0);
```

Question 7 [4 Marks]

Consider the following function which takes a parameter that is a *pointer to a pointer to an integer*, and as such is declared as `int **p;`.

```
void pointerStuff(int **p) {
  // Your two lines of code would go here
  return;
}
```

Now, assume that this function is called from the following C program:

```
int main(void) {
  int *q;
  int myNumbers[10];

  q = &(myNumbers[3]);

  pointerStuff(&q);
}
```

Give two lines of C Code in the function `pointerStuff` that would cause the following to happen after the execution of the function:

(a) Set the value of the array element `myNumbers[3]` in the `main` program to be 100.

(b) Change the value of the pointer `q` in the `main` function to point to the element `myNumbers[4]`.

Solution:

```
**p = 100;    (or *(*p) = 100;)
(*p)++;    (or (*p) = (*p) + 1; or *p = *p + 1;)
(but *p++ is not correct.)
```

Question 8 [2 Marks]

There have been four plenary lectures since the midterm in this course (not including the one on debugging). There is one question from each of those lectures below. You should give a written answer for **two of these four questions.** The answer should be one or two sentences. If you answer more than two questions, you must indicate which two you wish to be graded.

Plenary Lecture 7 – Programming Games. What is the role of the software that is responsible for the *physics* of a game? Give an example of a specific thing that the physics software would control.
Solution: Physics software models the real world to make sure that the game world displayed is physically correct with respect to the laws of physics. Examples: cloth, foot planting, collision detection, fluid simulation (anything reasonable here).

Plenary Lecture 8 – Software Startup Companies. What product did the company *Nanoport* provide? Hint: it related to magnets and USB connectivity.

Solution: A device that allows magnetic connection between two pieces of hardware that magnetically joins devices physically and electrically.

Plenary Lecture 9 – Machine Learning and Big Data. What was one of the applications of machine learning that Kobo makes use of? Describe it briefly.

Solution: Possible answers: 1) Differentiating book content - safe from 'dangerous' (unacceptable word content). 2) Make book recommendations, based on previous choices and other people's choices, and their likliehood of buying. 3) Find best layout of a web page, based on many people's buying habits.

Plenary Lecture 10 – Computer Vision. What did Professor David Fleet observe about the human visual processing in his talk on Computer Vision?

Solution: That the human brain does a great deal of processing of the incoming visual images a human sees.

Question 9 [8 Marks]

Write a C function called `newtonSquareRoot`, the prototype of which is given below, that returns the square root of a real number, computed using Newton's method. To compute the square root of a real number `x` using Newton's method, we first take a guess of the square root `z`, initialized to `x/2`, and then improve the guess by computing:

$$z_{\text{new}} = z_{\text{old}} - \frac{z_{\text{old}}^2 - x}{2z_{\text{old}}}$$

We perform this computation iteratively, until the gap between the improved guess and the previous one is less than or equal to 10^{-3}.

Solution:

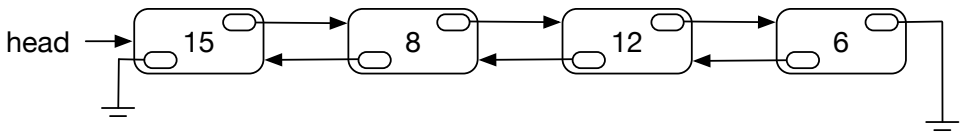
```
double newtonSquareRoot(double x) {
  double z = x / 2, newZ;
  while (fabs(z - (newZ = z - (z * z - x)/(2 * z))) > 1e-3)
    z = newZ;

  return z;
}
```

Alternative solution:

```
double newtonSquareRoot(double x) {
  double z = x / 2, newZ;
  do {
    newZ = z - (z * z - x)/(2 * z);
    if (fabs(z - newZ) <= 1e-3)
      return z;
    z = newZ;
  } while (true);
}
```

Question 10 [10 Marks]

A *doubly-linked list* is a linked list-like data structure that has pointers that point in *two* directions: to both the next item and the previous item in the list, as illustrated below:



The doubly-linked list shown above uses a structure that has three fields, as described in the C code below:

```
typedef struct dNodeDS {
    int value;
    struct dNodeDS *left, *right;
} DNode;
```

You are to write a C function called `insertAtNode` that inserts a new `DNode` (pointed to by the variable `newNode`) into this list. It should be inserted *before* (*i.e.* on the left side of) a `DNode` pointed to by the variable `placeNode` that already exists in the list. The prototype of this function is as follows:

```
DNode *insertAtNode(DNode *head, DNode *placeNode, DNode *newNode);
```

You should make the following assumptions when writing this function:

- `placeNode` is not `NULL` (it points to an actual item in the doubly-linked list).
- `newNode` is not `NULL`, and its `value` field has been set, but its pointer fields have not been set.
- the function returns the value of the `head` pointer after the insertion has taken place.
- the structure described above is declared globally, outside of the main function. However, the head pointer itself is not declared globally, and must be accessed by passing it to this function as a parameter, as shown in the prototype.

Solution:

```
DNode *insertAtNode(DNode *head, DNode *placeNode, DNode *newNode) {
  if (placeNode -> left == NULL) {
    newNode -> left = NULL;
    newNode -> right = placeNode;
    placeNode -> left = newNode;
    return newNode;
  }
  else {
    newNode -> right = placeNode;
    newNode -> left = placeNode -> left;
    placeNode -> left -> right = newNode;
    placeNode -> left = newNode;
    return head;
  }
}
```

Question 11 [11 Marks]

The *Sieve of Eratosthenes* is an ancient algorithm for finding prime numbers. To use this algorithm to find all prime numbers less than a given integer, say 100, we start by making a list of consecutive integers less than 100. We first take $p = 2$, the smallest prime number, and print it. We then eliminate all multiples of p less than 100 in the list, $(2p, 3p, 4p, \ldots)$, from the list, since they are multiples of p and are therefore not prime numbers. After eliminating the multiples of p, we find the first number after p that has not yet been eliminated, as it must be the next prime number. We assign this new prime number to p, print it, and eliminate its multiples from the list, and so on. We repeat this procedure until p^2 is greater than or equal to 100. The numbers that remain in the list are prime numbers, and we finish by printing them out.

Write a complete C program that uses the *Sieve of Eratosthenes* algorithm to print all prime numbers less than 100. Your implementation must not use the `%` operator. The output of your program should be:

```
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
```

Hint: Use an array of size 100 to keep track of whether an integer has been eliminated or not.

Solution:

```
int main(void)
{
    int sieve[100] = {0}, i, p = 2;

    while (p < sqrt(100))
    {
        printf("%d ", p);

        // eliminate multiples of p
        i = 2;

        while (i * p < 100)
        {
            sieve[i * p] = 1;
            i ++;
        }

        p++;
        while (sieve[p] == 1)
            p++;
    }

    for (; p < 100; p++)
        if (sieve[p] == 0)
            printf("%d ", p);
```

```
    printf("\n");
}
```

Question 12 [11 Marks]

Consider the following type definition for a node in a binary tree, which is organized (as described in class) to be sorted in ascending order:

```
typedef struct treeNode {
  int value;
  struct treeNode *left, *right;
} TreeNode;
```

Write a recursive function called `countBetween`, the prototype of which is given below, that returns the total number of nodes in the binary tree that have a value between `minimum` and `maximum`, inclusive. Your implementation must be *recursive*.

```
int countBetween(TreeNode *root, int mininum, int maximum);
```

Solution:

```
int countBetween(TreeNode * root, int mininum, int maximum) {
  if (root == NULL)
    return 0;

  if (root -> value < minimum)
    return countBetween(root -> right, minimum, maximum);

  if (root -> value > maximum)
    return countBetween(root -> left, minimum, maximum);

  return (1 + countBetween(root -> left, minimum, maximum)
     + countBetween(root -> right, minimum, maximum));
}
```

Question 13 [11 Marks]

Write a C function called `coinSort` that sorts an array containing coins. The array contains the integer values of the coins, which are *only* 1 cent, 5 cents, 10 cents, and 25 cents. The function sorts all the coins in ascending order.

As an example, an array of size 10 is shown below:

```
25 25 1 10 5 1 10 25 5 1
```

The sorted array after calling the function is:

```
1 1 1 5 5 10 10 25 25 25
```

Your implementation **must not** compare any elements of the array with each other, and should not duplicate the array. You will have to use an alternative method to sort the array.

The prototype of the function is the following, where parameter `size` gives the number of elements in the array.

```
void coinSort(int array[], int size);
```

Hint: Develop your sorting algorithm by considering the fact that the array has only integer values of 1, 5, 10, and 25.
Solution:

```
Solution 1:

void coinSort(int array[], int size){
        // Count the coins
int counter[4]={0, 0, 0, 0};

        for(int i=0; i < size; i++){
            int coinValue = array[i];

            if(coinValue == 1)
                counter[0]++;
            else if(coinValue == 5)
                counter[1]++;
            else if(coinValue == 10)
                counter[2]++;
            else
                counter[3]++;
        }
        // Overwrite the original array
        int index = 0;
        for (int j = 0; j < 4; j++) {
            for(int i = 0; i < counter[j]; i++){
                if(j == 0)
                    array[index] = 1;
                else if(j == 1)
                    array[index] = 5;
                else if(j == 2)
                    array[index] = 10;
                else if(j == 3)
                    array[index] = 25;
```

```
                index++;
            }
        }
}
```

Solution 2:

```
void coinSort(int array[], int size) {
        // Count the coins
        int counter[4]={0, 0, 0, 0};

        for(int i=0; i < size; i++){
            int coinValue = array[i];

            if(coinValue == 1)
                counter[0]++;
            else if(coinValue == 5)
                counter[1]++;
            else if(coinValue == 10)
                counter[2]++;
            else
                counter[3]++;
        }
// Solution 2
int index = 0;
        for (int i = 0; i < counter[0]; i++) {
            array[index] = 1;
            index++;
        }
        for (int i = 0; i < counter[1]; i++) {
            array[index] = 5;
            index++;
        }
        for (int i = 0; i < counter[2]; i++) {
            array[index] = 10;
            index++;
        }
        for (int i = 0; i < counter[3]; i++) {
            array[index] = 25;
            index++;
        }
}
```

Question 14 [12 Marks]

Write a complete C program that reads in a person's full name in the following form:

`<first name> <second name> ... <last name>`

The goal of the program is to print out that name in the following form:

`<last name>, <first initial>. <second initial>. ...`

The program should prompt the user for one person's full name at a time, ending when the name `ZZZ` is entered. You can assume that the maximum length of the full name is 50 characters. An example of the program input and output is given below:

```
Enter a name (ZZZ to stop): James Earl Jones
Your name is: Jones, J. E.
Enter a name (ZZZ to stop): Beyonce
Your name is: Beyonce
Enter a name (ZZZ to stop): Pillsbury D. Boy
Your name is: Boy, P. D.
Enter a name (ZZZ to stop): ZZZ
Goodbye
```

You may also assume that the following `safegets()` function is available for you to use.

```
void safegets(char s[], int size) {
  int i = 0, maxIndex = size - 1;
  char c;

  while (i < maxIndex && (c = getchar()) != '\n') {
    s[i] = c;
    i = i + 1;
  }
  s[i] = '\0';
}
```

Solution:

```
#define MAX_LENGTH 50
int main(void) {
  char name[MAX_LENGTH + 1];

  char *lastNamePtr = NULL, *firstNamePtr = NULL;

  printf("Enter a name (ZZZ to stop): ");
  safegets(name, MAX_LENGTH + 1);

  if (name[strlen(name) - 1] == '\n')
    name[strlen(name)-1] = '\0';

  while (strcmp(name, "ZZZ") != 0) {
```

```
    firstNamePtr = name;
    lastNamePtr = name + strlen(name) - 1;

    // locate last name in the string
    while (lastNamePtr > firstNamePtr && *lastNamePtr != ' ')
      lastNamePtr--;

    if (lastNamePtr == firstNamePtr) {
      // print last name (if no first name entered)
      printf("Your name is %s", lastNamePtr);
    }
    else {
      printf("Your name is %s,", lastNamePtr + 1);
      printf(" %c.", *firstNamePtr);
      firstNamePtr++;

      while (firstNamePtr < lastNamePtr) {
        if (*firstNamePtr == ' ') // locate next given name
          printf(" %c.", *(firstNamePtr + 1));
        firstNamePtr++;
      }
    }

    printf("\n");
    printf("Enter a name (ZZZ to stop): ");
    safegets(name, MAX_LENGTH + 1);

    if (name[strlen(name) - 1] == '\n')
      name[strlen(name) - 1] = '\0';

  } // end of while loop

  printf("Goodbye");
  return 0;
}
```

Question 15 [12 Marks]

Consider the C program given, in part, below. It declares an integer array of dimensions `SIZE` × `SIZE`, where `SIZE` is a defined constant as shown. The first part of the main program calls the `fillArray` function which sets each element of the array to the value `0`, `1`, or `2`. The program then asks the user to provide a value, `n`, which must be an odd integer. (You can assume that the user does enter an odd number, and that `n` $\geq$ `1` and `n` $\leq$ `SIZE`.) The goal of the program is to find all patterns of *crosses* of size `n` in the array of all `1` or `2`. For example, in the array shown below the code, there is a cross of size `n = 3` of `1` centred at `row = 2`, `column = 4`, a cross of size `n = 3`

of 2 centred at row = 5, col = 8, and another one centred at row = 6, column = 2.

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

#define SIZE 10

int main(void) {

    int array[SIZE][SIZE];
    int n;

    fillArray(array);

    printf("Enter Cross Size to search (must be odd): ");
    scanf("%d", &n);

    // Your main program code would go here

    return 0;
}
```

Example array:

```
Row 0: 0000010000
Row 1: 0000110000
Row 2: 0001110000
Row 3: 0000100000
Row 4: 0000000020
Row 5: 0020000222
Row 6: 0222000020
Row 7: 0020000000
Row 8: 0000000000
Row 9: 0000000000
```

The output of program is as follows, if the array above was the input, and n = 3:

```
Found Cross of Size 3 of 1 at (2,4)
Found Cross of Size 3 of 2 at (5,8)
Found Cross of Size 3 of 2 at (6,2)
```

Your answer should consist of the code that goes inside the main program above, and additional functions. You answer must make use of at least one function. You do not need to rewrite the C code provided to you in the space below.

Solution:

Main function:

```
int main(void) {
  int array[SIZE][SIZE];
  int n;

  fillArray(array);

  printf("Enter Cross Size to search (must be odd): ");
  scanf("%d",&n);

  for (int row = 0; row < SIZE; row++)
    for (int col = 0; col < SIZE; col++) {

      if (findCross(array, n, 1, row, col))
        printf("Found Cross of Size %d of %d at (%d,%d)\n", n, 1, row, col);

      if (findCross(array, n, 2, row, col))
        printf("Found Cross of Size %d of %d at (%d,%d)\n", n, 2, row, col);
    }

  return 0;
}
```

FINDCROSS FUNCTION

```
bool findCross(int array[SIZE][SIZE], int n, int searchNum, int row, int col) {

    /* look across one column */
    bool found = true;
    int start, end;

    start = row - n/2;
    end = row + n/2;

    // first check if size precludes the result
    if (start  >= 0 && end < SIZE) {

      // now look and see if the right number in a row
      for (int k = start; k  <= end && found; k++)
        if (array[k][col] != searchNum)
          found = false;
    }
    else
      found = false;
```

```
    // look across row
    start = col - n/2;
    end = col + n/2;

    // first check if size precludes the result
    if (start >= 0 && end < SIZE) {

      // now look and see if the right number in a column
      for (int k = start; k <= end && found; k++)
        if (array[row][k] != searchNum)
          found = false;
    }
    else
      found = false;

    return found;
}
```

This page has been left blank intentionally. You may use it for answers to any questions.