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 16, 2013
2:00 p.m. – 4:30 p.m.
(150 minutes)

Solutions

Examiners: J. Anderson, B. Korst, 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. You may use any math function that you have learned, as necessary.

The examination has 21 pages, including this one.

Full Name: _____

Student Number: _____ UTORID: _____

MARKS

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

Question 1 [2 Marks]

Consider a C-language identifier `values` that points to a dynamically allocated array of N `double`-type variables. Write a *single* C statement to deallocate the memory consumed by the array. You may assume that `#include <stdlib.h>` appears at the top of the C file.

Solution:

```
free(values);
```

Question 2 [2 Marks]

Write a *single* C statement that declares and initializes an array, named `list`, of 2000 `int` type variables. The first three elements of the array should be initialized to the values 3,2,1 and the remaining elements of the array should be initialized to 0.

Solution:

```
int list[2000] = {3,2,1};
```

Question 3 [2 Marks]

Consider a `double`-type variable `X` that has already been declared and initialized. Write a *single* C statement that rounds `X` to the nearest thousandth.

Solution:

```
X = ((int)(X * 1000 + 0.5))/1000.0;
```

OR

```
X = rint(X*1000)/1000.0;
```

Question 4 [4 Marks]

The C function below is supposed to implement the Bubble Sort algorithm on the array `list`, which is an `int` type array with `n` elements. However, the function has errors in it. For each error that you find:

(a) Identify what the error is by copying the line that is in error and stating what is wrong is wrong with it.

(b) Give the corrected line of code

```
void bubbleSort(int list[], int n)
{
  int i, j;
  for (j = n - 1; j >= 2; j--)
  {

    for (i = 0; i < j; i++)
    {
      if (list[i] > list[i+1])
      {
        int temp = list[i+1];
        list[i] = list[i+1];
        list[i+1] = temp;
      }
    }
  }
}
```

Solution:

```
Two corrections:
In the outer loop, j should continue until >= 1 (instead of >= 2)
int temp = list[i];
```

Question 5 [4 Marks]

Consider the recursive C function *factorial* below. Give the printed output of the function that is produced if the function is called with an argument of 4.

```
int factorial(int n)
{
  printf("ENTER: %d\n", n);

  int ret;
  if (n == 0 || n == 1)
    ret = 1;
  else
    ret = n*factorial(n-1);

  printf("EXIT: %d\n", n);
  return ret;
}
```

Solution:

```
ENTER: 4
ENTER: 3
ENTER: 2
ENTER: 1
EXIT: 1
EXIT: 2
EXIT: 3
EXIT: 4
```

Question 6 [10 Marks]

You are to write a *complete* C function called `scoreSquare`, which has four parameters as shown in the prototype below. The first parameter, `board` is a two-dimensional array of characters, whose size is given by the second parameter, `dimension` - that is, the array has `dimension` rows and `dimension` columns. The array represents the board state of a Connect6 game, similar to Labs 6 and 7 in this course.

The values of the elements in the `board` array are `'U'` for unoccupied, `'B'` if the square is occupied with a black stone, and `'W'` if the square is occupied with a white stone.

The final two parameters of the function, `row` and `col` indicate a specific square in the board. Your function must determine a 'score' for that square, as follows: If the square is unoccupied, the function should return -1. Otherwise, the function should return the number of squares *immediately* adjacent to the square having the *same* colour stone as the square at `row, col`. To be clear, *immediately* adjacent means the neighbouring squares that are of distance 1 away from the square, vertically or horizontally. Thus, your function should only consider the north, south, east, west adjacent squares, and *not* the diagonally adjacent squares. This means that the highest possible score of an occupied square is 4.

Your program should never access the board array beyond its dimensions.

This is the function prototype:

```
int scoreSquare(char **board, int dimension, int row, int col);
```

For example, in the following board with `dimension` 4, the function should return -1 for `0,0`; it should return 1 for `1,3`; and it should return 2 for `1,2`.

```
UBUU
UBWW
UUWU
UUUU
```

Solution:

```
int scoreSquare(char **board, int dimension, int row, int col)
{
  if (board[row][col] = 'U')
    return -1;

  char colour = board[row][col];
  int count = 0;
  if (row + 1 < dimension)
    if (board[row+1][col] == colour)
      count++;
  if (col + 1 < dimension)
```

```
    if (board[row][col+1] == colour)
      count++;
  if (row > 0)
    if (board[row-1][col] == colour)
      count++;
  if (col > 0)
    if (board[row][col-1] == colour)
      count++;
  return count;
}
```

Question 7 [12 Marks]

Write a C function named `combineSort` that has three `int` type array parameters, `A`, `B` and `C`, where `A` and `B` are inputs, and `C` is an output of the function. When the function is called, `A` and `B` will each have 100 elements that are already sorted in ascending order (from smallest to largest). Your function must fill in the elements of array `C` to contain *all* of the elements of `A` and `B`, *and* the array `C` must be sorted into ascending order. Thus, the size of array `C` is 200 elements.

Important Note: Your function *must* make use of the fact that `A` and `B` are already sorted (i.e. you may *not* simply use a known sorting algorithm after copying the contents of `A` and `B` into `C`. A grade of zero will be assigned in this case.)

The prototype of the function is as follows:

```
void combineSort(int A[100], int B[100], int C[200]);
```

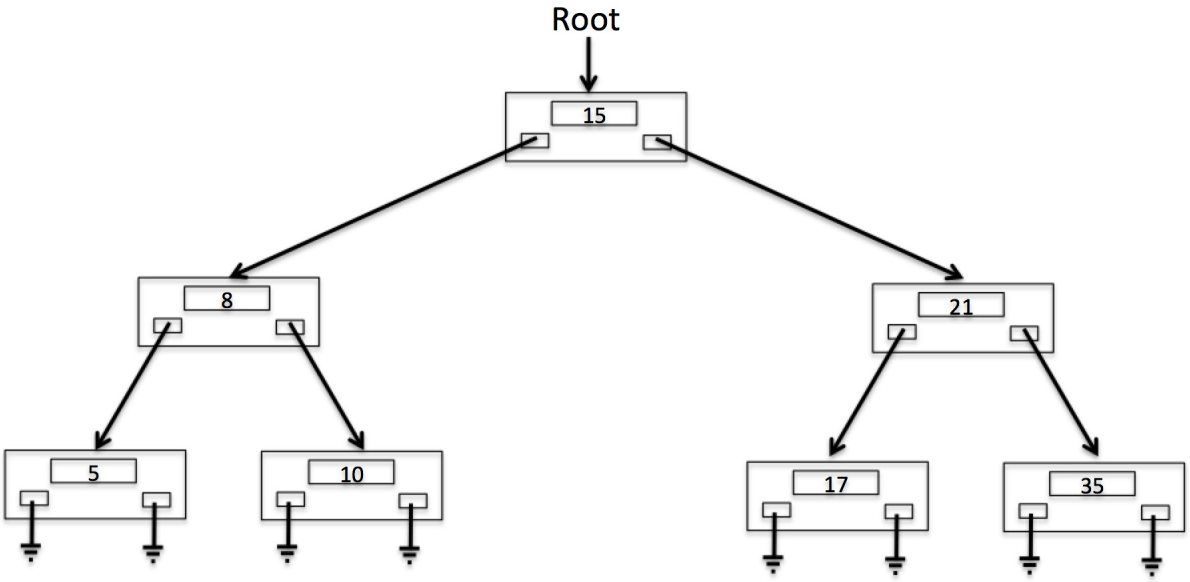
Solution:

```
void combineSort (int A[100], int B[100], int C[200]) {


bool ASearch;   // tells us which array is being used currently

int Aindex = 0, Bindex = 0, Cindex = 0;

while (Aindex < 100 || Bindex < 100) {

if ((Aindex < 100) && (Bindex < 100))
    if (A[Aindex] < B[Bindex]) ASearch = true;
    else ASearch = false;

if (ASearch) {
    C[Cindex] = A[Aindex];
    Aindex++;
    if (Aindex >= 100) ASearch = false;
}
else {
    C[Cindex] = B[Bindex];
    Bindex++;
    if (Bindex >= 100) ASearch = true;
}

        Cindex++;

}
```

```
}
```

Question 8 [10 Marks]

A binary tree is a data structure, similar to a linked list, that is illustrated below:



Each 'node' in the binary tree contains a single integer number, and two links to other nodes. The type and structure of a node is defined as follows:

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

Node *Root;
```

The pointer `Root` points to the top node of the binary tree. The above tree is structured in a special and useful way: the integer value of every node 'underneath' the left link of any node is **less** than the value of the node itself. Similarly, the value of every node 'underneath' the right link of a node is **greater than or equal** to the value of the node itself. You can see this is true, for example, by looking at the `Root` of the tree, the node with the value 15. The nodes 'underneath' the left side pointer (with values 10, 5 and 8) are all less than 15, the value of the `Root` node. In this question, you will make use of this property. There are two parts to this question, continued on the next page.

(a) Write a C function (the prototype of which is given below), which when called with a pointer to the root of a binary tree (i.e. `findLargest(Root);`) returns the largest `value` in the tree. Your function must not actually test (i.e. use an if statement to compare) the value of any of the nodes in the tree as this is not necessary, given the tree structure as described above. You can assume that the root pointer parameter, `startNode`, is not NULL (i.e. that there is at least one node in the tree). Your solution must use recursion.

```
int findLargest (Node *startNode);
```

Solution:

```
int findLargest (Node *startNode) {

if (startNode->rightLink == NULL) return startNode->Value;
else return (FindLargest(startNode->rightLink);

        }
```

(b) Write a C function (the prototype of which is given below), which prints the values in the entire tree in sorted order, from *highest to lowest*. You may not use any sorting algorithm; rather, you must make use of the structure of the tree as described above. Your solution must use recursion.

```
void printTreeDescending (Node *startNode);
```

You can assume that the `startNode` pointer parameter is not NULL (i.e. that there is at least one node in the tree). **Important Note:** again, your solution must make use of the structure of the tree, and may not employ comparison if statements on the values given in the tree. To print the entire tree, the function would be called in the following way:

```
printTreeDescending (Root);
```

where `Root` is a pointer to the top node in the tree, as illustrated. The output of your function, should be the value of each node on a different line. The output if called on the tree illustrated above, is as follows:

```
35
21
17
15
10
8
5
```

Solution:

```
void printTreeDescending (Node *startNode) {


if (startNode->RightLink == NULL)
printf("%d\n", startNode->value);

else {
     PrintTreeDescending(startNode->RightLink);
     printf("%d\n", startNode->value);
}


if (startNode->LeftLink != NULL)
     PrintTreeDescending(startNode->LeftLink);
}
```

Question 9 [10 Marks]

You are to write a C function called `printSentence` that prints out a sentence consisting of words selected from an array of strings. The words to be printed are specified by a linked list for which each element in the list contains the *index* of the array element with the word to be printed. A pointer to the head of the linked list is passed in as a parameter called `Sentence`. The 2D array that contains the words, called `wordList`, is also passed into the function as a parameter. The linked list type and structure is defined in the following way:

```
typedef struct wordsDS {
int wordIndex;
struct wordsDS *link;
} wordNode;
```

The prototype of the function you must write is as follows:

```
void printSentence(wordNode *Sentence, char *wordList[]);
```

The main function (for example) could call the function `printSentence` with a pointer to the head of a linked list of structures of type wordNode, and also with the 2D array of words `wordList`. The function will print out, in sequence, the words in `wordList` at the `wordIndex` indicated in each node in the linked list. The list is terminated by a NULL link.

For example, if the list of `wordIndex` values in the linked list were as follows:

```
3->4->10->8->6->7->NULL
```

And if the `wordList` argument were as follows:

```
char wordList[13][20] = { "now", "is", "the", "time", "for", "all", "good",
 "people", "to", "come", "aid", "of", "country" };
```

The output of the program should be (note that there are spaces between the words):

```
time for aid to good people
```

Solution:

```
void PrintSentence (WordNode *Sentence) {

WordNode *CurrentNode;

CurrentNode = Sentence;

while (CurrentNode != NULL) {

printf("%s ", WordList[CurrentNode->WordIndex]);
```

```
CurrentNode = CurrentNode->Link;

}

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

You may use this page to continue answering the previous question.

Question 10 [10 Marks]

Write a function called `revStr` that makes use of recursion to reverse the characters of a string. The reversal should happen two elements at a time, from the ends to the centre of the string. The function prototype is given below. Parameter `str` is the string to be reversed and parameter `len` is the length of the string. For example, if the function is called with the string `"Hello"`, it must reverse the characters so that the string contains `"olleH"`.

```
void revStr(char* str, int len);
```

Solution:

```
void revStr (char *str, int len)
{
  char tmp;
  if (len < 2) return;

  len = len -1;

  tmp = *str;
  *str = str[len];
  str[len] = tmp;

  revStr (str+1, len -1);
}
```

Notes: if just printed the characters without swapping -3 marks if swapping is from middle to outside -1 mark in case of iterative solution, max 3 marks for swapping

Question 11 [10 Marks]

Finish writing the *complete* C program below which uses command-line arguments. The function should examine each command-line argument and test whether it is one of the strings in the `engProg` array given, and then print a message accordingly. For example, if the program is compiled to a binary executable called `myProg` and executed as follows:

```
./myProg ECE MSE CHEM INDY
```

The program should print the following output:

```
ECE is in the list.
MSE is not in the list.
CHEM is in the list.
INDY is not in the list.
```

You may use any string functions you wish.

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

int main(int argc, char *argv[])
{
    char *engProg[]={"ECE", "MIE", "CHEM", "CIV"};
```

Solution:

```
int main(int argc, char *argv[])
{
     char *engProg[]={"ECE", "MIE", "CHEM", "CIV"};

     int i,j;

     for(i=1; i<argc; i++){
          for(j=0; j<4;j++){
               if (strcmp(argv[i], engProg[j])==0){
                    printf("%s is in the list\n", engProg[j]);
                    break;
               }
          }
          if (j==4)
                printf("%s is not in the list\n",argv[i]);
     }
     return (0);
}
```

Question 12 [12 Marks]

Write a *complete* C program to allocate, initialize, print and de-allocate a three-dimensional array of `int` type variables, according to the specifications below. The sizes of the three array dimensions, x, y and z, are 3, 2 and 4, respectively. **Hint:** one way to view this task is to allocate a 3-element array of pointers, each of which points to a 2D array with 2 rows and 4 columns. Your program must use `malloc` and `free` to allocate and release memory.

(a) The array elements should be initialized according to the following function:

$f(x, y, z) = 5x + 6y + 7z$

Which means your initialization code will look like this:

```
myArray[x][y][z] = 5 * x + 6 * y + 7 * z;
```

(b) Printing the array

After allocation and initialization, your program should print three 2 x 4 matrices (each corresponding to a different value for the first dimension), with a new line (`\n`) in between them. The output is as follows:

```
0 7 14 21
6 13 20 27

5 12 19 26
11 18 25 32

10 17 24 31
16 23 30 37
```

(c) De-allocating memory

After the matrices are printed, your program must free all memory used.

Solution:

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

void main()
{
    // 3D Array indices
    int x, y, z;

    //  Allocate memory
    int ***myArray = malloc(3 * sizeof(int **));

    for(x = 0; x < 3; x++)
    {
        myArray[x] = malloc(2 * sizeof(int *));
        for(y = 0; y < 2; y++)
        {
            myArray[x][y]=malloc(4 * sizeof(int));
        }
    }
    //  Access and print array elements
    for(x = 0; x < 3; x++)
    {
        printf("\n");
        for(y = 0; y < 2; y++)
        {
            printf("\n");
            for(z = 0; z < 4; z++)
            {
                myArray[x][y][z] = 5*x+6*y+7*z;
                printf("%d ", myArray[x][y][z]);
            }
        }
    }
    //  Deallocate 3D array
    for(x = 0; x < 3; x++)
    {
        for (y=0; y<2;y++)
            free (myArray[x][y]);
        free(myArray[x]);
    }
    free (myArray);
    return 0;
}
```

You may use this page to continue your answer to the previous question.

Question 13 [12 Marks]

Humans normally use decimal (base 10) numbers, whereas computers use binary (base 2). A convenient way to represent a long binary number is using hexadecimal (base 16). In hexademical numbers, the digits 0 through 9 have the usual meaning, and the digits A, B, C, D, E, F are used to represent the decimal (base 10) numbers 10, 11, 12, 13, 14, 15, respectively. For example, the hexadecimal number A5 is the decimal number 165, because A represents 10, and we therefore have $10 \cdot 16^1 + 5 \cdot 16^0 = 165$.

For this question, you must write a C function called `computeDecimal` that converts a hexadecimal string (parameter `hex`) into the corresponding decimal integer and returns the computed decimal value. The prototype is given below. Your function may assume the string argument only contains the characters `'0'` ... `'9'` and `'A'` ... `'F'`. For example, if the function is called with the string `"2C"`, the function should return 44 because $2 \cdot 16^1 + 12 = 44$. Your function should work with strings of digits of *any* length.

```
int computeDecimal(char *hex);
```

Solution:

```
int computeDecimal(char *hex)
{
  int d = strlen(hex);
  int i;
  int count = 1;
  int total = 0;
  for (i = d-1; i >= 0; i--)
  {
    int digit;
    if ((hex[i] >= '0') && (hex[i] <= '9'))
      digit = hex[i] - '0';
    else
      digit = 10 + hex[i] - 'A';
    total += digit*count;
    count *= 16;
  }
  return total;
}
```

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