WARNING!!

Our exam was exceptionally hard and tricky with a lot of scope for committing mistakes and errors. Don't get scared. You probably won't get such a hard paper. Try it out atleast once for the sake of practice.

Name: Section: Rollno:

ESC101: Fundamentals of Computing (End Semester Exam)

20 November, 2015 (4-7pm)

Total Number of Pages: 23 Total Points 120

Question	Points	Score
1	25	
2	12	
3	26	
4	16	
5	18	
6	23	
Total:	120	

Instructions

1. Read these instructions carefully.
2. Write you name, section and roll number on all the pages of the answer book, **including the ROUGH pages**.
3. Using pens (blue/black ink) and not pencils. Do not use red pens for answering.
4. Even if no answers are written, the answer book has to be returned back with name and roll number written.
5. Sign the attendance sheet.
6. Do not exchange question books or change the seat after obtaining question paper.
7. Write the answers cleanly in the space provided. Space is given for rough work in the answer book.
8. For your convenience, the longer questions are divided into several parts. Do remember to go through all the questions.

I PLEDGE MY HONOUR THAT DURING THE EXAMINATION I HAVE NEITHER GIVEN NOR RECEIVED ASSISTANCE.

.....

Signature

Question 1. Write the output of the following programs. If you think there is any error in the program, please explicitly mention the reason and type of error : compile error, runtime error and compiler dependent (i.e. non-portable). If there is an infinite loop, please indicate so.

(a) (6 points) :

```
#include <stdio.h>

int a, b = -1;
void prtFun ();

int main () {
    static int a = 1;
    prtFun();
    a += 1;
    prtFun();
    printf ( " %d %d \n" , a, b) ;
}

void prtFun () {
    static int a = 2;
    int b = 1;
    a += ++b;
    printf (" %d %d \n" , a, b);
}
```

Output: **1point for each. DEDUCT 1 for each extraneous output.**

```
4 2
6 2
2 -1
```

(b) (3 points) :

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

int main() {
   char p[20];
   char *s = "desserts";
   int length = strlen(s);
   int i;
   for (i = 0; i < length; i++)
      p[i] = s[length - i];
   printf("such %s\n",p);
}
```

Output: **no partials.**

```
such
```

(c) (3 points) :

```
#include<stdio.h>

int f(int *a, int n) {
  if(n <= 0) return 0;
  else if(*a % 2 == 0) return *a + f(a+1, n-1);
  else return *a - f(a+1, n-1);
}

int main() {
  int a[] = {12, 7, 13, 4, 11, 6};
  printf("%d", f(a, 6));
  return 0;
}
```

Output: **no partials.**

```
15
```

(d) (6 points) :

```
# include <stdio.h>
int main() {
   int i;
   for(i=0; i<=25; i++) {
     switch(i) {
       case  0: i += 4;
       case  5: i += 1;
       case  7: i += 4;
       case 15: i += 1;
       default: i += 4;
     }
     printf("%d ", i);
   }
}
```

Output: **2points for each. DEDUCT 2 for each extraneous output.**

```
14 20 25
```

(e) (4 points) :

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

void foo(char * a) {
   a = a + 6;
}

int main() {
   char a[] = "hello world";
   foo(a);
   printf("%s\n", a);

   free(a+3);
   a[2] = '\0';
   printf("%s\n", a);

   return 0;
}
```

Output: **2points each for "hello world" and "runtime error" (or equivalents).**

```
stdout: hello world
stderr: Runtime Error: free(): invalid pointer....
explanation: attempt to free a non-heap object.
```

(f) (3 points) :

```
#include <stdio.h>

int foo(int n, int sum) {
   int k = 0, j = 0;
   if(!(n < 0))
      if (n == 0) return 0;
   else  k = n % 10;
   j = n / 10;
   sum = sum + k;
   printf ("%d,", k);
   return foo (j, sum);
}

int main () {
   int a = 42, sum = 0;
   a = foo (a, sum);
   printf ("%d\n", sum);
}
```

Output: **1point for each. DEDUCT 1 for each extraneous output.**

```
2,4,0
```

Question 2. (12 points) In this question, we want to find a local minimum (LM) in a matrix. A *local minimum* is defined as an element A[i][j], which is less than or equal to its top, down, left and right neighbours. If the element is a corner/edge element, we only check the available neighbours.

Here, we use a recursive algorithm to search a local minimum in a given matrix. The input matrix entries may *not* be sorted in any way. Complete the blanks to get the working C code. Please note that you can assume that the matrix has all distinct and positive integer elements.

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

int** arr;
int rows, cols;

int CheckIfLM(int rIndex, int cIndex) {
  int value = arr[rIndex][cIndex];

  if(cIndex >= 1 && arr[rIndex][________________] < value)
        return 0;

  if(cIndex <= ________________ && arr[rIndex][cIndex+1] < value)
        return 0;

  return 1;
}

int FindMinInColumn(int col) {
  int i;
  int minValue = arr[0][col];
  int minIndex = 0;
  for(i = 1; i < rows; i++)
        if(arr[i][col] < ________________) {
                minValue = arr[i][col];
                minIndex = i;
        }

  return minIndex;

}

//recursive function
int FindLM(int startCol, int endCol){
        int midCol = (startCol + endCol)/2;
        int minRowIndex = FindMinInColumn(midCol);

        //base case
        if( ____________________________________ )
                return arr[minRowIndex][midCol];

        int left, right;

        if(midCol >= 1)
                left = arr[minRowIndex][midCol-1];
        else
```

```
47                  return FindLM(_____________, _____________);
48
49          if(midCol <= cols-2)
50                  right = arr[minRowIndex][midCol+1];
51          else
52                  return FindLM(_____________, _____________);
53
54          if(left < right)
55                  return FindLM(startCol,midCol-1);
56          else
57                  return FindLM(midCol+1,endCol);
58
59  }
60
61  int main() {
62     scanf("%d %d",&rows,&cols);
63     arr = (int**) malloc( rows * _____________ );
64     int i, j;
65     for(i = 0; i < rows; i++)
66           arr[i] = (int*) malloc(cols * sizeof(int));
67
68     for(i = 0; i < rows; i++)
69       for(j = 0; j < cols; j++)
70                  scanf("%d",&arr[i][j]);
71
72     int min = FindLM(_____________, _____________);
73
74     printf("A local minima is: %d\n",min);
75
76     return 0;
77  }
```

Solution:

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

int** arr;
int rows, cols;

int CheckIfLM(int rIndex, int cIndex) {
  int value = arr[rIndex][cIndex];

  //1 point
  if(cIndex >= 1 && arr[rIndex][cIndex-1] < value)
        return 0;

  //1 point
  if(cIndex <= (cols-2) && arr[rIndex][cIndex+1] < value)
        return 0;

  return 1;
}

int FindMinInColumn(int col) {
  int i;
  int minValue = arr[0][col];
  int minIndex = 0;
  for(i = 1; i < rows; i++)
        //1 point
        if(arr[i][col] < minValue) {
                minValue = arr[i][col];
                minIndex = i;
        }

  return minIndex;

}

int FindLM(int startCol, int endCol){
        int midCol = (startCol + endCol)/2;
        int minRowIndex = FindMinInColumn(midCol);

        //2 points
        if( CheckIfLM(minRowIndex,midCol) )
                return arr[minRowIndex][midCol];

        int left, right;

        if(midCol >= 1)
                left = arr[minRowIndex][midCol-1];
        else
                //2 x 1 point
```

```
                return FindLM(midCol+1,endCol);

        if(midCol <= cols-2)
                right = arr[minRowIndex][midCol+1];
        else
                //2 x 1 point
                return FindLM(startCol,midCol-1);

        if(left < right)
                return FindLM(startCol,midCol-1);
        else
                return FindLM(midCol+1,endCol);

}

int main() {
   scanf("%d %d",&rows,&cols);
   //1 point
   arr = (int**) malloc( rows * sizeof(int*) );
   int i, j;
   for(i = 0; i < rows; i++)
         arr[i] = (int*) malloc(cols * sizeof(int));

   for(i = 0; i < rows; i++)
     for(j = 0; j < cols; j++)
                scanf("%d",&arr[i][j]);

   //2 x 1 point
   int min = FindLM(0,cols-1);

   printf("A local minima is: %d\n",min);

   return 0;
}
```

Question 3. (26 points) Consider the $n \times n$ chessboard (with the n^2 squares colored alternately black and white). In chess, a *queen* can move as far as she pleases, horizontally, vertically, or diagonally. In this problem you are required to place n queens on the chessboard so that none of them can hit any other in one move.

Fill in the blanks so that the following recursive C implementation works correctly. The input n is given in the command line. Output is the $n \times n$ boolean matrix; with 1 depicting the presence of a queen and 0 the absence.

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

int gridSize, **grid;

void printGrid() {
        int i, j;
        for(i=0;i<gridSize;i++) {
                for(j=0;j<gridSize;j++)
                        printf("%d ", grid[i][j]);
                printf("\n");
        }
}

// Checks if the point is a valid point on chessboard
int isValidPoint(int i, int j) {
        if(i>=0 && i<gridSize && _____________________________________________)
                return 1;
        return 0;
}

//checks if the point can be filled without clashing an older queen
int isValid(int row, int col) {
        int i, s;

        if(_________________!=0) return -1;

        // Check the points in the same row
        for(i=0;i<gridSize;i++)
                if(!(i==col || _________________==0)) return -1;

        // Check the points in the same column
        for(i=0;i<gridSize;i++)
                if(!(i==row || _________________==0)) return -1;

        for(s=1 ; s < _________________ ; s++) {

                // Check along major diagonals
                int i=row+s, j=col+s;
                if (isValidPoint(i,j) && grid[i][j]==1) return -1;

                i = _________________, j = _________________;
                if (isValidPoint(i,j) && grid[i][j]==1) return -1;


```

```
47                  // Check along minor diagonals
48
49                  i = _____, j = _____;
50                  if (isValidPoint(i,j) && grid[i][j]==1) return -1;
51
52                  i=row-s, j=col+s;
53                  if (isValidPoint(i,j) && grid[i][j]==1) return -1;
54          }
55          ____________________________________________
56  }
57
58  // Recursion with backtracking to find a valid configuration
59  int fillGrid(int N) {
60          int i, j;
61          //base case
62          if(N==0) {
63                  printGrid();
64                  return 0;
65          }
66          for(i=0;i<gridSize;i++)
67                  for(j=0;j<gridSize;j++)
68                    if(isValid(i, j) == 1) {
69                          grid[i][j] = 1;
70                          if(fillGrid(_____) == -1)
71
72                            ____________________________________________
73                          else
74                            return 0; //fillgrid(N) succeeds
75                    }
76          return -1; //fillgrid(N) fails
77  }
78
79  int main(  ____________________________________  ) {
80    int i, j;
81    if(argc!=2) {
82          printf("Invalid number of arguments!\n");
83          return -1;
84    }
85    gridSize = atoi(argv[1]);
86
87    grid = ____________________________________
88    for (i=0; i<gridSize; i++)
89
90          grid[i] = ____________________________________
91
92    for(i=0;i<gridSize;i++)
93          for(j=0;j<gridSize;j++) grid[i][j]=0;
94
95    fillGrid(_____);
96    return 0;
97  }
```

Solution:

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

int gridSize, **grid;

void printGrid() {
        int i, j;
        for(i=0;i<gridSize;i++) {
                for(j=0;j<gridSize;j++)
                        printf("%d ", grid[i][j]);
                printf("\n");
        }
}

// Checks if the point is a valid point on chessboard
int isValidPoint(int i, int j) {
        //2points (do not give partial marks)
        if(i>=0 && i<gridSize && j>=0 && j<gridSize)
                return 1;
        return 0;
}

// checks if the point can be filled without a clash
int isValid(int row, int col) {
        int i, s;

        //1 point
        if(grid[row][col]!=0) return -1;

        // Check the points in the same row
        for(i=0;i<gridSize;i++)
                //1 point
                if(!(i==col || grid[row][i]==0)) return -1;

        // Check the points in the same column
        for(i=0;i<gridSize;i++)
                //1 point
                if(!(i==row || grid[i][col]==0)) return -1;

        //1 point
        for(s=1 ; s < gridSize ; s++) {

                // Check along major diagonals
                int i=row+s, j=col+s;
                if (isValidPoint(i,j) && grid[i][j]==1) return -1;

                //2 x 1point
                i = row-s, j = col-s;
```

```
                if (isValidPoint(i,j) && grid[i][j]==1) return -1;


                // Check along minor diagonals
                //2 x 1point
                i = row+s, j = col-s;
                if (isValidPoint(i,j) && grid[i][j]==1) return -1;

                i=row-s, j=col+s;
                if (isValidPoint(i,j) && grid[i][j]==1) return -1;
        }
        return 1;
}

// Recursion with backtracking to find a valid configuration
int fillGrid(int N) {
        int i, j;
        if(N==0) {
                printGrid();
                return 0;
        }
        for(i=0;i<gridSize;i++)
                for(j=0;j<gridSize;j++)
                  if(isValid(i, j) == 1) {
                        grid[i][j] = 1;
                        //1 point
                        if(fillGrid(N-1) == -1)
                          //1 point
                          grid[i][j]=0;
                        else
                          return 0; //fillgrid(N) succeeds
                  }
        return -1; //fillgrid(N) fails
}

//4 x 1point
int main(  int argc, char** argv  ) {
  int i, j;
  if(argc!=2) {
        printf("Invalid number of arguments!\n");
        return -1;
  }
  gridSize = atoi(argv[1]);

  //4 x 1point
  grid = (int **) malloc( gridSize * sizeof(int *) );
  for (i=0; i<gridSize; i++)
        //4 x 1point
        grid[i] = (int *) malloc( gridSize * sizeof(int) );

  for(i=0;i<gridSize;i++)
        for(j=0;j<gridSize;j++) grid[i][j]=0;
```

```
102
103     //1 point
104     fillGrid(gridSize);
105     return 0;
106 }
```

Question 4. (16 points) Consider the following C code and write the output in the clearly marked space.

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

struct node {
    int val;
    struct node *next; };

struct node* create_node(int v) {
    struct node * new_node = (struct node *) malloc( sizeof(struct node) );
    new_node->val = v;
    return new_node;
}

struct node *myfunction(struct node *head, int k) {
    struct node* current = head;
    struct node* next = NULL;
    struct node* prev = NULL;
    int count = 0;

    while (current && count < k) {
       next  = current->next;
       current->next = prev;
       prev = current;
       current = next;
       count++;
    }
    if(next) head->next = myfunction(next, k);

    return prev;
}

void print_list(struct node *ll) {
    if(!ll) return;
    print_list(ll->next);
    printf("%d ",ll->val);
    return;
}

int main() {
    int n,k;
    struct node *head = NULL;
    struct node *prev, *temp;

    scanf("%d %d",&n,&k);

    while(n--) {
        temp = create_node(n);
        prev = head;
        head = temp;
        head->next = prev;
    }
```

```
52      head = myfunction(head,k);
53      print_list(head);
54      return 0;
55
56  }
```

Fill the Output box for the following four independent Inputs.

1 point for each correct integer. If more integers are printed than expected, then DEDUCT 1 point for each extra integer.

a) 5 2

```
1  4 2 3 0 1
```

b) 5 3

```
1  3 4 0 1 2
```

c) 3 1

```
1  2 1 0
```

d) 3 3

```
1  0 1 2
```

Question 5. (18 points) Complete the following C program which does the following: Given a word, it prints whether the word has a **palindromic anagram** or not. A *palindrome* is a word which reads the same backwards or forwards. *Anagram* is a word formed by rearranging the letters of another. In other words, the program is required to determine if the characters of an input word could be rearranged to form a palindrome.

Assume that the word contains only lowercase characters and also that the word has at most 100 characters.

Example: The string "aac" has a palindromic anagram (namely "aca"), but the string "abc" has no palindromic anagram.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define _____ 101
5
6  #define _____ 26
7
8  //returns 1 iff input is a palindromic anagram
9  int isAnagramPalindrome(char input[]) {
10         int * charCount;
11         charCount = (int *) _____(ALPHABET_SIZE _____);
12         int i, oddCount, evenCount;
13
14         oddCount = evenCount = _____;
15
16         for(i=0 ; input[i]!= _____ ; i++)
17
18                         charCount[ _____ -'a'] += _____;
19
20         for(i=0;i<ALPHABET_SIZE;i++)
21
22                 if(charCount[i]% _____ ==0) _____;
23
24                 else _____;
25
26         return oddCount > _____ ? _____ : 1;
27 }
28
29
30 int main() {
31         char input[SIZE];
32
33         scanf("%s", _____ );
34
35         if( _____ )
36                 printf("String %s has a palindromic anagram\n", input);
37
38         else
39                 printf("String %s does not have a palindromic anagram\n",
                       input);
40
41 }
```

Solution:

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

//1 point
#define SIZE 101
//1 point
#define ALPHABET_SIZE 26

//returns 1 iff input is a palindromic anagram
int isAnagramPalindrome(char input[]) {
        int * charCount;

        //2pts(calloc) + 1pt(comma) + 2pts(sizeof-int)
        charCount = (int *) calloc(ALPHABET_SIZE , sizeof(int) );
        int i, oddCount, evenCount;

        //1 point
        oddCount = evenCount = 0;

        //1 point (also for '\0')
        for(i=0 ; input[i]!= 0 ; i++)
                        //2 x 1 point
                        charCount[ input[i] -'a'] += 1;

        for(i=0;i<ALPHABET_SIZE;i++)

                //3 x 1point
                if(charCount[i]% 2 ==0) evenCount++;

                else oddCount++;

        //2 x 1point
        return oddCount > 1 ? 0 : 1;
}


int main() {
        char input[SIZE];

        //1 point
        scanf("%s", input );

        //1point
        if( isAnagramPalindrome(input) )
                printf("String %s has a palindromic anagram\n", input);

        else
                printf("String %s does not have a palindromic anagram\n
                    ", input);

```

```
50          return 0;
51
52  }
```

Question 6. Jim is practicing programming these days. She has not seriously coded for a long time and is likely to add a lot of bugs. Your task is to either give the correct statement, with minimal changes, or report OK. (Note: Changing a statement that is already correct, or changing a statement by an unnecessary amount, will be penalized.)

She wants to make a *tree* like data-structure which has *move-up* and *move-down* functions defined over it but has taken a rather complicated approach to do so. We would explain and debug this in parts. (**Warning: This is a long question.**)

1. She wants to define a tree which is a network of nodes such that each node has a value, a *left child* and a *right child*. Thus each node points to two other nodes. In case, both these pointers point to NULL, the node is said to be a *leaf* node. The structure of the node is :

```
1  struct node {
2          int value;
3          struct node * left;
4          struct node * right; };
```

2. She wants to define a *stack* of items (these items are themselves complicated structures). A stack is like a tower, things can be added on the top of it and removed from the top. The structure for stack is :

```
1  enum direction {left, right};
2  struct stackds {
3          enum direction d;
4          int value;
5          struct node * p;
6          struct stackds * next; };
```

3. She solves her problem by maintaining a *state* which has a stack and a tree. The structure of state is :

```
1  struct state {
2          struct stackds * head;
3          struct node * tree;
4  } State;
```

She defines her functions – move-up and move-down by playing over this stack and tree. This tree is the subtree from her current position and the stack stores the history of upward/downward movements that have been performed to reach this state. They can be reverted to climb up.

(a) (9 points) Using the above data-structures, she wants to write auxiliary functions for her stack. Debug the code for *push* and *pop* functions on the stack. Note that `tos` points to the top (head) of the stack. They should behave as commented.

```
1  /* Insert data in the item format at top of stack */
2  struct stackds * push(int lr, int value, struct node * q, struct
       stackds * tos) {
3          struct stackds * tmp;
4          tmp -> d = lr;
5          tmp -> value = value;
6          tmp -> p = q;
7          tmp -> next = NULL;
8          if(tos == NULL) {
9                  tos = tmp;
10         }
11         else {
12                 tmp -> next = tos;
13                 tmp = tos;
14         }
15         return tos;
16 }
17
18 /* Return the top element after popping it out of the stack */
19 struct stackds * pop(struct stackds * node, struct stackds * tos) {
20
21         if(tos == NULL)
22                 return NULL;
23         *node.d = *tos.d;
24         node -> value = tos -> value;
25         node -> p = tos -> p;
26         node -> next = NULL;
27         if(tos -> next == NULL) {
28                 free(tos);
29                 tos = NULL;
30         }
31         else {
32                 struct stackds * p = tos;
33                 p -> next = tos;
34                 free(p);
35         }
36         return tos;
37 }
```

Suspicion in	Your response
line 3	**3 x 1point** tmp = (struct stackds *) malloc(sizeof(struct stackds))
line 4	**1 point** OK
line 13	**1 point** tos = tmp
line 23	**2 x 1point** (*node).d = (*tos).d ..OR.. node− >d = tos− >d
line 26	**1 point** OK
line 33	**1 point** tos = p− >next

(b) (6 points) Assume a correct implementation for push and pop for further analysis.

Jim is now in a state to define traversal functions over her weird implementation of tree. Move-down is a function which essentially moves down the tree, along left or right sub-tree, and simultaneously maintains the stack. (In particular, the stack-top should contain the `direction` taken and the sibling sub-tree *not* taken.) Debug the following code for her.

```
1  int move_down(enum direction dir) {
2          if(dir == left) {
3                  if(State.tree -> left == NULL) {
4                    struct node * left_subtree = State.tree -> left ;
5                    int current_value = State.tree -> value ;
6                    struct node * sibling_node = State.tree -> right ;
7                    State.head = push(left, current_value, State.head);
8                    State.tree = left_subtree ;
9                    return 1;
10                 }
11                 else
12                   return 0;
13         }
14         else {
15                 if(State.tree -> right)
16                   return 0;
17                 else {
18
19                   struct node  right_subtree = State.tree -> right ;
20                   int current_value = State.tree.value ;
21                   struct node * sibling_node = State.tree -> left ;
22                   State.head = push(current_value, right, sibling_node,
                         State.head);
23                   State.tree = right_subtree ;
24                   return 1;
```

```
25                      }
26              }
27              return 0;
28      }
```

Suspicion in	Your response
line 3	**1 point** State.tree − > left != NULL ..OR.. equivalents
line 7	**1 point** State.head = push(left , current_value, sibling_node , State.head)
line 15	**1 point** !(State.tree − > right) ..OR.. equivalents
line 19	**1 point** struct node* right_subtree = State.tree− >right
line 20	**1 point** int current_value = State.tree − > value
line 22	**1 point** State.head = push(right, current_value, sibling_node, State.head)

(c) (8 points) Jim now wants to write a function to move up the tree, this essentially involves getting the last changes from the stack and reverting it, simultaneously maintaining both the stack and the tree. Debug the code of move-up for her.

```
1  int move_up() {
2          if(State.head == NULL)
3                  return 0 ;
4          else {
5                  struct node * new_tree = (struct node *) malloc( sizeof
                       (struct node) ) ;
6                  struct stackds * temp;
7                  State.head = pop(State.head);
8                  new_tree -> value = temp -> value;
9                  if(temp -> d) {
10                         new_tree -> right = temp -> p;
11                         new_tree -> left = State.tree;
12                         State.tree = new_tree;
13                         free(temp);free(new_tree);
14                         return 1;
15                 }
16                 else {
17                         new_tree -> left = temp -> p;
18                         new_tree -> right = State -> tree;
19                         State -> tree = new_tree;
20                         free(temp);free(new_tree);
21                         return 1;
22                 }
23                 free(temp);
24                 return 0;
25         }
26 }
```

Suspicion in	**Your response**
`line 6`	**1point** struct stackds * temp = (struct stackds *) malloc(sizeof(struct stackds))
`line 7`	**1 point** State.head = pop(temp , State.head)
`line 9`	**2 points** temp − > d == left ..OR.. equivalents
`line 12`	**1 point** OK
`line 13 & 20`	**1 point** free(tmp); // no free for new tree
`line 18`	**1 point** new_tree − > right = State.tree
`line 19`	**1 point** State.tree = new_tree