MTH 307/417/515 Final Exam Solutions

1. Write the output for the following programs. Explain the reasoning behind your answer.

   (a)
   ```
   #include<stdio.h>
   int main()
   {
      int n;
      for(n = 7; n != 0; n--)
        printf("n = %d", n--);
      getchar();
      return 0;
   }
   ```

   Answer. This yields the output: `n = 7n = 5n = 3n = 1n...`

   Reasoning: The loop in the program is an infinite loop, as the test condition is never satisfied. The is because the two `n--` will reduce the value of `n` by 2 after each iteration of the loop.

   (b)
   ```
   int main()
   {
      int i = 0;
      for(i = 0; i < 10; i++)
      {
        switch(i)
   ```

```
    {
       case 0: i += 5;
       case 1: i += 2;
       case 5: i += 5;
       default: i += -1; break;
    }
    printf("%d  ", i);
  }
  getchar();
  return 0;
}
```

Answer. This yields the output: 11

Reasoning: During the first iteration of the loop, `i = 0`., so the `i` is incremented by 5 under the `switch` case - `case 0`. As there is no `break` after this case and `i = 5`, under the case - `case 5:`, `i` is again incremented by 5, so `i = 10`. Finally, the value of `i` is incremented by 1 inside the for loop, before the next iteration, so `i = 11` before the second iteration. However, as the loop condition is not satisfied, the loop is terminated, and the value is printed.

(c)
```
int main()
{
   char str[]= "mth307\nfinalexam";
   char *ptr1, *ptr2;
```

```
   ptr1 = &str[4];
   ptr2 = str - 5;
   printf("%c", ++*str - --*ptr1 + *ptr2 + 2);
   printf("%s", str);
   getchar();
   return 0;
}
```

Answer. This yields the output:

```
Anth3/7
finalexam
```

Reasoning: The important thing to note here is that a string is just a one-dimensional array of characters, and hence it follows the same rules of operator precedence and pointers as an one-dimensional integer array (see Sections 1.6, 1.8 and 2.1 and in the lesson plan).

(d)
```
int func(int n, int *fg)
{
   int t, f;
   if(n <= 1)
   {
     *fg = 1;
      return 1;
```

```
   }
   t = func(n-1, fg);
   f = t + *fg;
   *fg = t;
   return f;
}
int main( )
{
  int x = 15;
  printf ( "%d\n", func(5, &x));
  getchar();
  return 0;
}
```

Answer. This gives the output: `8`.

Reasoning: This because the function `func(k, & x)` inside the `printf` statement in `main` prints the $(k+2)^{th}$ term of the Fibonacci sequence: `0, 1, 1, 2, 3, 5, 8, 13, ...` This primarily uses the basic properties of pointers (detailed in 2.1.1 of the lesson plan) and the fact that the function `func` calls itself recursively (as was the case with question 1(c) of the midterm.)

2. Write `C` programs for the following.

   (a) Finding the rank of a 3×3 matrix.

Answer.

```
#include <stdio.h>
void swap(int mat[3][3], int row1, int row2, int col)
{
    for (int i = 0; i < col; i++)
    {
        int temp = mat[row1][i];
        mat[row1][i] = mat[row2][i];
        mat[row2][i] = temp;
    }
}

int rank(int mat[3][3])
{
    int rank = 3, row, col, flag = 0, i;
    for (row = 0; row < rank; row++)
    {
        if (mat[row][row])
        {
           for (col = 0; col < 3; col++)
           {
               if (col != row)
               {
```

```
                double mult = (double)mat[col][row] /mat[row][row];
                for (int i = 0; i < rank; i++)
                    mat[col][i] -= mult * mat[row][i];
            }
        }
    }
    else
    {
        flag = 1;
        for (i = row + 1; i < 3;  i++)
        {
            if (mat[i][row])
            {
                swap(mat, row, i, rank);
                flag = 0;
                break ;
            }
        }
        if (flag == 1)
        {
            rank--;
            for (int i = 0; i < 3; i ++)
                mat[i][row] = mat[i][rank];
        }
```

```
            row--;
        }
    }
    return rank;
}
void main()
{
    int i, j, mat[3][3];
    for(i = 0; i < 3; i++)
    {
        for(j = 0; j < 3; j++)
        {
                printf("\n Enter enter (%d, %d)-th entry of matrix : ",i+1,j+1);
                scanf("%d", &mat[i][j]);
         }
}
printf("\n Rank of the matrix entered is : %d", rank(mat));
}
```

(b) Implementing the quick sort algorithm on an unsorted array of integers, and then using the binary search algorithm to search for a given element in the sorted array.

Answer.

```
#include<stdio.h>
#define N 10
void swap(int* a, int* b)
{
    int t = *a;
    *a = *b;
    *b = t;
}
int partition (int arr[], int low, int high)
{
    int i, j, pivot;
    pivot = arr[high];
    i = (low - 1);
    for (j = low; j <= high - 1; j++)
    {
        if (arr[j] <= pivot)
        {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}
```

```
void quickSort(int arr[], int low, int high)
{
    if (low < high)
    {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
int binarySearch(int arr[], int l, int r, int x)
{
   if (r >= l)
   {
        int mid = l + (r - l)/2;
        if (arr[mid] == x)
            return mid;
        if (arr[mid] > x)
            return binarySearch(arr, l, mid-1, x);
        return binarySearch(arr, mid+1, r, x);
   }
   return -1;
}
void printArray(int arr[])
{
```

```
    int i;
    for (i = 0; i < N; i++)
        printf("%d ", arr[i]);
    printf("\n");
}
void main()
{
    int arr[N], x, i, res;
    for(i = 0; i < N; i++)
    {
        printf("\n Enter element %d of the array : ",i+1);
        scanf("%d", &arr[i]);
     }
    printf("\n Enter search element : ");
    scanf("%d",&x);
    quickSort(arr, 0, N - 1);
    printf("\n The sorted array is: \n");
    printArray(arr);
    res = binarySearch(arr, 0, N-1, x);
    if (res == -1)
    {
         printf("\n Element is not present in array");
    }
    else printf("\n The search element is present at position %d", res+1);
```

```
}
```

Note that the solutions to 2 (a) and (b) above are based on the examples available at: `https://www.geeksforgeeks.org/`. The solutions use recursion, but it is quite straighforward to write non-recursive programs for these questions.

(c) Implementing a binary tree (not necessarily strict) using the pre-order traversal for display.

Solution. The following is just a simple modification of the Example 18 (in Section 3.28) of the lesson plan.

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

struct node
{
    int key;
    struct node *left, *right;
};

struct node *newNode(int item)
{
    struct node *temp =  (struct node *)malloc(sizeof(struct node));
    temp->key = item;
```

```
    temp->left = temp->right = NULL;
    return temp;
}
void preorder(struct node *root)
{
    if (root != NULL)
    {
        printf("%d \n", root->key);
        preorder(root->left);
        preorder(root->right);
    }
}
struct node* insert(struct node* node, int key)
{
    if (node == NULL) return newNode(key);
    if (key < node->key)
        node->left  = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);
    return node;
}
int main()
{
    /* Let us create following BST
```

```
            50
          /     \
         30      70
        /  \    /  \
      20   40  60   80 */
    struct node *root = NULL;
    root = insert(root, 50);
    insert(root, 30);
    insert(root, 20);
    insert(root, 40);
    insert(root, 70);
    insert(root, 60);
    insert(root, 80);

    // print inorder traversal of the BST
    preorder(root);
    return 0;
}
```