Indian Institute of Technology, Kharagpur

Department of Computer Science and Engineering

Class Test 2 Solution, Spring 2017-18

Programming and Data Structure (CS 11001 / CS 10001)

---

1. (5 points) In the following program we implement four functions which takes a matrix as an argument and performs the following functions:

a) Compute sum of left diagonal elements (a “left diagonal” of a square matrix A[n][n] is the set of elements lying on the diagonal line connecting the elements A[0][0] and A[n-1][n-1])

b) Compute sum of right diagonal elements (a “right diagonal” of a square matrix A[n][n] is the set of elements lying on the diagonal line connecting the elements A[0][n-1] and A[n-1][0])

c) Print the lower diagonal matrix (the lower diagonal matrix of a square matrix A[n][n] is the set of elements lying strictly ‘below’ its left diagonal)

d) Print the upper diagonal matrix (the upper diagonal matrix of a square matrix A[n][n] is the set of elements lying strictly ‘above’ its left diagonal)

Fill in the blanks to complete the program. One blank can have AT MOST ONE statement.

Marks: (0.25+0.5+0.25) + (0.25+0.5+0.25) + (0.25+0.5+0.25) + (0.25+0.5+0.25) + (0.25+0.25+0.25+0.25)= 5

```
#include<stdio.h>

/* computes the sum of left diagonal elements */
int sumLeftDiagElements(int mat[][4], int rows, int cols) {
   int i, j, sum = 0;
   for ( i = 0; i < rows; i++ ) {
      for ( j = 0; j < cols; j++ ) {
         if (i==j) {
            sum += mat[i][j];
         }
      }
   }
   return sum;
}

/* computes the sum of right diagonal elements */
int sumRightDiagElements(int mat[][4], int rows, int cols) {
   int i, j, sum = 0;
   for ( i = 0; i < rows; i++ ) {
      for ( j = 0; j < cols; j++ ) {
         if ((i + j) == (rows - 1)) {
            sum += mat[i][j];
         }
      }
   }
   return sum;
}
```

```
/* Print the lower diagonal elements */
void lowerDiagMatrix(int mat[][4], int rows, int cols) {
   int i, j;
   for ( i = 0; i < rows; i++ ) {
      for ( j = 0; j < cols; j++ ) {
         if (i>j) {
            printf("%d ",mat[i][j]);
         }
      }
      printf("\n");
   }
}

/* Print the upper diagonal elements */
void upperDiagMatrix(int mat[][4], int rows, int cols) {
   int i, j;
   for ( i = 0; i < rows; i++ ) {
      for ( j = 0; j < cols; j++ ) {
         if (i<j) {
            printf("%d ",mat[i][j]);
         }
         else {
            printf("  ");
         }
      }
      printf("\n");
   }
}

int main() {
   int mat[4][4] = { { 2, 3, 8, 4 },
                     { 5, 1, 7, 3 },
                     { 9, 2, 6, 8 },
                     { 1, 4, 5, 7 } };
   int left_diag_sum = sumLeftDiagElements(mat, 4, 4);
   printf("Sum of Left Diagonal elements: %d\n", left_diag_sum);
   int right_diag_sum = sumRightDiagElements(mat, 4, 4);
   printf("Sum of Right Diagonal elements: %d\n", right_diag_sum);
   printf("Lower Diagonal Elements :-\n");
   lowerDiagMatrix(mat, 4, 4);
   printf("Upper Diagonal Elements :-\n");
   upperDiagMatrix(mat, 4, 4);
   return 0;
}
```

---

2. (5 points) A coder wants to improvise the sorting procedure over a set of integer elements by coding the bubble and selection sort together. The idea is that -- (i) (s)he will pairwise compare two elements from the beginning of the array to its end and thereby push the maximum element at the end of the array (as done in bubble-sort), and (ii) while doing so, (s)he maintains the location of minimum element in the array and later exchange the minimum element with the initial element of the array (as done in selection-sort). In doing so (at every iteation), (s)he will place the maximum and minimum elements at both ends of the array and hence (s)he has to perform the above two iterative tasks [(i) and (ii)] over the array size being reduced by two elements (one in both ends of the array) everytime. Having such an insight, can you complete the following code snippets to realize this? **Marks: 1x5 = 5**

```
#include<stdio.h>

void swap(int *x, int *y)
{
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
}

int main()
{
    // given set of integer elements to sort
    int a[] = {3,1,0,7,5,4,9,2,6,8};
    // calculate number of elements in array a[]
    int n = sizeof(a)/sizeof(int);
    int i, j, minIndex;
    for( i = 0 ; i < n-1-i ; i++ ) { //start outer-loop
        /* selecting the index for the minimum element */
        minIndex = i;
        for( j = i+1 ; j < n-i ; j++ ) { // start inner-loop-1
            if( a[j] < a[minIndex] ) {
                minIndex = j ;
            }
        } // end inner-loop-1
        /* pushing maximum element at the end of array
           by pairwise compare and swap */
        for( j = i+1 ; j < n-i ; j++ ) { // start inner-loop-2
            if ( a[j-1] > a[j] ) {
                swap( &a[j-1] , &a[j] );
                /* updating minimum index if corresponding
                   elements get exchanged */
                if( minIndex == j ) {
                    minIndex = j-1 ;
                }
            }
        } // end inner-loop-2
        /* exchanging the minimum element with the
           initial element of array */
        swap( &a[i] , &a[minIndex] );
    } // end outer-loop
    return 0;
}
```

Note:
1st Blank: Instead of n-i, **10-i** is also okay
3rd Blank: Instead of > as comperator, **>=** comparator is also okay

---

3. (2 points) What will be the output of the following function? **Marks : 2**

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

int main ()
{
      char A[10] = "14FB61R23";
      int i, sum = 0;
      for (i=0; A[i]; ++i) {
          if (A[i] < '0') continue;
          if (A[i] > '9') continue;
          sum += A[i] -'0';
      }
      printf("sum = %d", sum);
}
```

Answer: 17

---

4. (3 points) The following function takes two null-terminated strings S, T as parameters. The function returns 0 if the two strings are equal, 1 if the first string (S) is a proper prefix of the other (T), 2 if the second string is a proper prefix of the other, -1 otherwise. Fill it the blanks. EAch blank can have atmost one statement. **Marks: 3 x 1 = 3**

```
int compstr ( char *S, char *T )
{
      int i = 0;
      while (1) {
             if( (S[i] == '\0') && (T[i] == '\0') ) return 0;
             if (S[i] == '\0') return 1;
             if ( T[i] == '\0' ) return 2;
             if ( S[i] != T[i] ) return -1;
             ++i;
        }
}
```

---

5. (2 points) Let the function g be defined as follows:

```
int g ( int n )
{
      if (n < 7)
            return n;
      return g(n/2);
}
```

What is the value returned by the call g(346)? **Marks: 2**

Answer: 5

---

6. (2 points) What value does the call h(5) return, where h is defined as follows? **Marks: 2**

```
int h ( int n )
{
      if (n == 0)
            return 1;
      if (n%2 == 0)
            return 2*h(n-1);
      else
            return 3*h(n-1);
}
```

Answer: 108

---

7. (1 point) Please fill in the blank in the following program for calculating the power of a number using recursion. Each blank can have at most one statement. **Mark: 1**

```
#include<stdio.h>
int power(int base, int exp);
int main()
{
    int base, exp;

    printf("Enter base: ");
    scanf("%d", &base);
    printf("Enter exponent: ");
    scanf("%d", &exp);

    printf("%d to the power of %d is %d", base, exp,
                                          power(base, exp));
    return 0;
}
```

```
int power(int base, int exp)
{
    if(exp==0) // base condition
    {
        return 1;
    }

    else
    {
        return   base*power(base, exp-1) ;
    }
}
```