ESC 101: Fundamentals of Computing	**End Semester Examination**	Date: 25 – 04 - 2019

Name	**ANSWER KEY**					**B**
Roll No.		Dept.		Section		

Total **120** Marks

Instructions:

1. This question paper contains a total of **20** printed side of paper.
2. **Write your roll number and section on every sheet of this booklet.**
3. Write final answers neatly with a blue/black pen in the given boxes.
4. Answers written outside the box will **not** be graded.

Q 1.1 Fill in the blanks. **(1 + 1 = 2 marks)**

a) The time complexity of merge sort for linked list is **O(n log n)** in big O notation.

b) Worst case time complexity of searching in binary search tree is O(log n) **False** (True/False).

Q 1.2 Fill in the blanks to complete the following code.

a) Function Description: Given a 2-dimentional **m** x **n** array mat having all its rows sorted in ascending order from left to right and all its columns sorted in ascending order from top to bottom, findKey checks whether key exists in the array or not. Returns 1 if key exists and 0 otherwise. You can assume that the 2D array was dynamically created in the main() function from an int** and that is passed to the function.

(1 + 1 + 1 + 1 + 1 + 1 = 6 marks)

```
int findKey(int** mat, int m, int n, int key) {

   int x = 0 ;

   int y = n - 1 ;

   While( x<m && y>=0 ) {

       if(mat[x][y] < key) {

           ++x ;

       }
       else if(mat[x][y] > key) {

           --y ;

       }
       else {
           return 1;
       }
   }
   return 0 ;
}
```

b) Function Description: Given a 2-dimentional **m** x **10** array **mat** having all its rows sorted in ascending order from left to right, **mwayMerge** merges the **m** rows of **mat** in one go by reading the rows in a loop into one sorted array in ascending order and returns a pointer to it. **(1 + 1 + 1 + 1 + 1 = 5 marks)**

```
#define INF 100000007
int* mwayMerge(int mat[][10], int m) {
   int ps[m];
   for(int i=0; i<m; ++i) {
       ps[i] = 0 ;
   }

   int* merged = (int *)malloc( sizeof(int)*m*10 );

   for(int i=0; i<m*10 ; ++i) {
       int p = 0;
       int minval = INF;
       for(int k=0; k<m; ++k) {
           if( ps[m] >= 10 ) {
               continue;
           }
           if( mat[k][ps[k]] < minval ) {
               minval = mat[k][ps[k]] ;
               p = k ;
           }
       }
       merged[i] = minval;
       ps[p]++ ;
   }

   return merged;
}
```

Roll No.		Section		Page No.	3

Q 1.3 What is the output of the following program for the given inputs (on the right) (3 + 4 = 7 marks)

```
#include<stdio.h>

int main() {
   int arr[50][50];
   int n;
   scanf("%d", &n);
   for(int i=0; i<n; ++i) {
       for(int j=0; j<n; ++j) {
           scanf("%d", &arr[i][j]);
       }
   }

   int x=-1, y=-1;
   int l=0;
   for(int i=0; i<n-1; ++i) {
       for(int j=0; j<n-1; ++j) {
           if(arr[i][j]==1) {
               for(int p=i+1; p<n; ++p) {
                   for(int q=j+1; q<n; ++q) {
                       if(arr[p][q] == 1 && p-i + q-j > l) {
                           x=i;
                           y=j;
                           l=p-i+q-j;
                       }
                   }
               }
           }
       }
   }
   printf("%d %d %d\n", x, y, l);
}
```

Input :
5
0 0 0 0 1
0 1 0 0 0
0 0 1 0 1
0 0 0 0 0
0 0 0 0 1

Output (Write below)
---
1 1 6

Input :
7
0 0 0 0 0 0 1
0 0 0 1 0 1 0
1 0 0 0 0 0 0
0 0 1 0 0 1 0
0 1 0 0 0 0 0
0 0 0 0 1 0 0
0 0 0 0 0 0 0

Output (Write below)
---
2 0 7

Q 2.1 Write True or False against the following statements. **(1 x 4 = 4 marks)**

a)	On runnning command `"./a.out winter has come"` (without quotes) on command line, `"winter"` can be accessed using `argv[0]`.	**FALSE**
b)	In C, `atoi("23bc2")` returns `232`.	**FALSE**
c)	Standard Output i.e. `stdout` is buffered and standard error i.e. `stderr` is unbuffered. This is the reason why ``` ``fprintf(stdout, "2"); fprintf(stderr, "3");`` ``` (without backticks) prints `"32"`.	**TRUE**
d)	Choosing mode `"r+"` while opening a file using `fopen`, file opens for read and update. Also file is created if it doesn't exist.	**FALSE**

Q 2.2 We have discussed reading multiple files and writing content of both in a single file in class. **(16 marks)**

Given below is an incomplete program which reads 2 files, say **input1.txt** and **input2.txt** respectively and writes in a new file, say **output.txt.**

Following instructions are followed while writing in **output.txt :**

- Each line of input1.txt has a single integer. Each line of input2.txt contains a single character. Both files have same number of lines.

- Program reads one line from input2.txt (let that character be c) and corresponding line from input1.txt (let that integer be n). Then it writes the character ***c n*** times in output.txt followed by newline character.

- Above is continued for each line until input2.txt sees the character '*'. In that case program starts overwriting from the start of the output file. Note that instruction 2 has not to be followed for this line and corresponding integer in input1.txt i.e. integer corresponding to '*' is ignored. From next line, it follows instruction 2.

- Names of files are taken as command line arguments in the order :
<output_file> <input_file_1> <input_file_2>

Example:

Content of input1.txt:
2
3
4
1
2

Content of input2.txt:
a
b
c
*
d

Content of output.txt:
dd
bbb
cccc

Fill in the blanks in the following program so that it works according to the instruction given above.

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

int main (__1__, __2__) {

   FILE * fwrite; /* file to write to */
   FILE * fread[2] ; /* files to read from */

   if(__3__) {
       printf ("Incorrect number of arguments.\n");
       return 0;
   }

   fwrite = fopen (__4__, __5__);
   fread[0] = fopen (__6__, __7__);
   fread[1] = fopen (__8__, __9__);

   char ch;
   int n;
          /* still data to read . */
   while (!feof(fread[0]) && !feof(fread[1])){
       // read an integer from line in first file
       // read a character from line in second file
       fscanf (__10__);
       fscanf (__11__);

       if(ch == '*'){
           __12__;
           __13__;
       }

       char *str = (char*) malloc((n+1)*sizeof(char));
       for(int i=0;i<n;i++){
           str[i] = __14__;
       }
       // print the required output in output file
       __15__;
   }
   fclose (fread[0]); fclose (fread[1]); // cleanup
   fclose (fwrite);                      // close the
   return 0;                             // files
}
```

1 : int argc

2 : char ** argv

3 : argc != 4

4 : argv [1]

5 : "w"

6 : argv [2]

7 : "r"

8 : argv [3]

9 : "r"

10: fread[0] , "%d", &n

11: fread[1] , "%c\n", &ch

12: fseek(fwrite, 0, SEEK_SET);

13: continue

14: ch

15: fprintf (fwrite , "%s\n", str);

Roll No.		Section		Page No.	6

Q 3.1 Write True or False against the following statements. **(1 x 4 = 4 marks)**

a)	Return statements implicitly typecast variables to match function return types.	**TRUE**
b)	A function declaration must not have the types of the formal parameters.	**FALSE**
c)	A function with return type float must have atleast one return statement.	**FALSE**
d)	Dynamically allocated memory within a function is not freed when the function ends.	**TRUE**

Q 3.2 Write the output of the following program. **(1 + 1 + 1 = 3 marks)**

a)

```
#include <stdio.h>
int g(int x, int y) {
    if (x == y) {
    return x;
    } else if (x > y) {
    return g(y, x-y);
    } else {
    return g(x, y-x);
    }
    return -3;
}

int main() {
    int a, b;
    scanf("%d %d", &a, &b);
    printf("%d", g(b, a));
    return 0;
}
```

```
Input     18 24
Output    6
-------------------------------------
Input     89 144
Output    1
-------------------------------------
Input     150 3
Output    3
```

b) In this part, assume that arrays are initialised with 0 in all entries. **(1 + 1 + 2 = 4 marks)**

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

int f(int* x, int* y, int n) {
     int a[10][10] = {};
     for (int i = 1; i < n; i++) {
     for (int j = 1; j < n; j++) {
          if (i == 1) {
               a[i][j] = a[i][j-1] ? 1 : (x[i] == y[j]);
          } else if (j == 1) {
               a[i][j] = a[i-1][j] ? 1 : (x[i] == y[j]);
          } else if (x[i] == y[j]) {
               a[i][j] = 1+a[i-1][j-1];
          } else {
               a[i][j] = a[i-1][j] > a[i][j-1] ? a[i-1][j] : a[i][j-1];
          }
     }
     }
     return a[n-1][n-1];
}
int main() {
     int n;
     scanf("%d", &n);
     int* a = (int *) malloc(sizeof(int) * n);
     int* b = (int *) malloc(sizeof(int) * n);
     for (int i = 0; i < n; i++) {
     scanf("%d %d", (a + i), (b + i));
     }
     printf("%d", f(a, b, n));
     return 0;
}
```

```
Input:
3
1 2 3 4 5 6

Output:
------------------
0
```

```
Input:
4
1 4 2 1 3 2 4 3

Output:
------------------
2
```

```
Input:
6
1 1 4 5 2 6 3 2 3 3 4 3

Output:
--------------------------
3
```

Q 3.3 Fill in the blanks to complete the program according to the instructions given below.

a) We want to write a recursive program to compute the binomial coefficients.
Note that (n choose k) = ((n - 1) choose (k)) + ((n - 1) choose (k – 1)).

(1 + 0.5 + 1 + 0.5 + 1 + 0.5 + 1 + 0.5 = 6 marks)

Using this directly to compute the coefficients will result in a lot of calls to compute various other binomial coefficients, which we want to avoid by saving intermediate computations.
We want to do this by using the array **p** to store (**i** choose **j**) for all **i**, **j**.

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

long binom(int n, int k, long** p) {

     if ((n == k) || (k == 0)) {

     p[n][k] = 1;

     return 1;

     }

     int x, y;

     p[n-1][k-1] ? (x = p[n-1][k-1]) : (x = p[n-1][k-1] = binom(n-1, k-1, p));

     p[n-1][k] ? (y = p[n-1][k]) : (y = p[n-1][k] = binom(n-1, k, p));


     p[n][k] = x+y;
     return x+y;
}

int main() {
     int n, k;
     scanf("%d %d", &n, &k);

     long** p = (long **)malloc(sizeof(long*) * (n+1));

     for (int i = 0; i < n+1; i++) {

     p[i] = (long *)calloc(k+1, sizeof(long*));
     }
     printf("%ld", binom(n, k, p));
     return 0;
}
```

b) We want to write a program that takes inputs two strings of length atmost 9, and a positive integer n which is also less than 9. **(3 marks)**

The program should then concatenate the first n letters of the second string to the first string, and then print the result.

There is a string library function that does exactly this, but instead of calling it directly, we want to call a function that calls this.

Complete the code.

```
#include <stdio.h>

#include <string.h>

char* g(char* , char*, int);

int main() {

    char x[10];

    char y[10];

    int n;

    scanf("%s %s %d", x, y, &n);

    printf("%s", g(x, y, n));

    return 0;

}

char* g(char* x, char* y, int n) {

    return strncat(x, y, n);

}
```

Roll No.		Section		Page No.	10

Q 4.1 Write <u>True</u> or <u>False</u> against the following statements. **(1 x 4 = 4 marks)**

a)	`int i = 5;`
	`while(1) {`
	`  if (i-- < 0) break;`
	`}`
	
	The value of i after the above piece of code is executed will be **-1**.
		FALSE

b)	`// Piece 1`
	`if (a == 3)`
	`  b = 5;`
	`else`
	`  b = 4;`
	
	`// Piece 2`
	`b = (a == 3) ? 4 : 5;`
	
	The above two pieces of code are equivalent
		FALSE

c)	`int a = 2, b;`
	
	`switch(b = (a == 2)) {`
	`  case 1:`
	`  printf("5");`
	`  case 2:`
	`  printf("4");`
	`  case 3:`
	`  printf("3");`
	`  case 4:`
	`  printf("2");`
	`  case 5:`
	`  printf("1");`
	`  default:`
	`  break;`
	`  }`
	`}`
	The output of the above program is **5**.
		FALSE

d)	`int i = 1, j = 1;`
	`for (--i && j--; j < 4; j++) {`
	`  printf("B");`
	`}`
	
	The output of the above code is equal to BBB;
		TRUE

Q 4.2 Convert the given for loop to a while loop. **(3 marks)**

```
for (char c; (c = getchar()) != '\0';)
{
    printf("%d", c - '0');
}
```

```
char c;
while ((c = getchar()) != '\0') {
    printf("%d", c - '0');
}
```

Q 4.3 Write the output of the following program for the given inputs. **(2 x 3 = 6 marks)**

```
#include <stdio.h>
short arr[100];
void func(int n)
{
    arr[0] = arr[1] = 0;
    for (int p = 2; p <= n; p++) arr[p] = 1;

    for (int p=2; p <= n; p++) {
    if (arr[p]) {
        for (int i = 2 * p; i <= n; i += p)
            arr[i] = 0;
    }
    }
}

int main()
{
    int n;
    scanf("%d", &n);

    func(n);

    for (int p = 2; p <= n; p++) {
    if (arr[p])
        printf("%d ", p);
    }
    printf("\n");
    return 0;
}
```

Input: 7
Output: 2 3 5 7

Input: 16
Output: 2 3 5 7 11 13

Input: 23
Output: 2 3 5 7 11 13 17 19 23

Q 4.4 Count Sort (Descending) **(1 + 1 + 2 + 2 = 6 marks)**

Count sort is an algorithm for sorting a collection of non-negative integers where the maximum possible value of the integers is known beforehand. It operates by counting the number of objects that have each distinct key value, and using arithmetic on those counts to determine the positions of each key value in the output sequence.
Since we're only sorting strings, thus our max range is 255(0-255). Fill in the blanks in the code below so that the code successfully performs an *descending* count sort.

```
#include <stdio.h>

#define RANGE 255

void countSort(char arr[])
{
     int count[RANGE + 1], i;
     for (i = 0; i <= RANGE; i++)
     count[i] = 0;

   for(i = 0; arr[i] != '\0'; i++)
     count[arr[i]]++;

     int c = 0;
     for (i = 1; i <= RANGE; i++) {

     for (int j = 0; j < count[i]; j++) {

           arr[c++] = arr[c] = i;

     }
     }
}
```

Q 5.1 Multiple choice, single correct. **(2 x 5 = 10 marks)**

a) What is the output of the following program:

1) 30 ☐
2) 80 ☒
3) Compiler error ☐
4) Runtime error ☐

```
#include <stdio.h>
void fun(int *p) {
 int q = 30;
 p = &q;
}
int main() {
 int r = 80;
 int *p = &r;
 fun(p);
 printf("%d", *p);
 return 0;
}
```

b) Assume the sizes of an integer and a pointer are 4 and 8 bytes respectively. What is the output of the following program ?

1) 400 ☐
2) 4 ☐
3) 1600 ☒
4) 8 ☐

```
#include <stdio.h>
#define R 10
#define C 40
int main(){
  int (*p)[R][C];
  printf("%d",  sizeof(*p));
  getchar();
  return 0;
}
```

c) What is the output of the following program.

1) HavingFun ☒
2) avingFun ☐
3) Garbage Value ☐
4) Compiler Error ☐

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

void fun(char** str_ref)
{
   str_ref++;
}

int main()
{
   char *str = (void *)malloc(100*sizeof(char));
   strcpy(str, "HavingFun");
   fun(&str);
   puts(str);
   free(str);
   return 0;
}
```

d) What is the output of the following program ?

1) Dangling Pointer ☐
2) Memory Leak ■
3) Compiler Error:
   free can't be applied on NULL pointer ☐
4) The program may crash as free() is
   called for NULL pointer. ☐

```
#include<stdio.h>
#include<stdlib.h>
int main() {
     int *x = (int* )malloc(sizeof(int));
     x = NULL;
     free(x);
     return 0;
}
```

e) What is the output of the following program ?

1) 21 ☐
2) 22 ■
3) 24 ☐
4) 25 ☐

```
#include <stdio.h>
int f(int x, int *py, int **ppz){
  int y, z;
  **ppz += 1;
  z  = **ppz;
  *py += 2;
  y = *py;
  x += 3;
  return x + y + z;
}
int main(){
  int c, *b, **a;
  c = 5;
  b = &c;
  a = &b;
  printf( "%d", f(c,b,a));
  return 0;
}
```

Q 5.2 What is the output of the following program
(1 x 4 = 4 marks)

```
20 8 8 8
```

```
#include <stdio.h>
#include <stdlib.h>
int main() {
     int arri[] = {1, 2 ,3, 4, 5};
     int *ptri = arri;

     char* arrc = (char*)malloc(5*sizeof(char));
     arrc[0] = 101;
     arrc[1] = 102;
     arrc[2] = 103;
     arrc[3] = 104;
     arrc[4] = 105;
     char *ptrc = arrc;

     printf("%lu ", sizeof(arri));
     printf("%lu ", sizeof(ptri));
     printf("%lu ", sizeof(arrc));
     printf("%lu ", sizeof(ptrc));
     return 0;
}
```

Q 5.3 A linked list is said to contain a cycle if any node is visited more than once while traversing the list. The function **has_cycle** returns true if the passed linked list has a cycle, otherwise it returns false. You are tasked to fill in the blanks in the **has_cycle** function. **(1 x 3 = 3 marks)**

```
typedef struct node {
    int data;
    struct node* next;
} SinglyLinkedListNode;

int has_cycle(SinglyLinkedListNode* head) {
   if(head==NULL)
    return 0;
   SinglyLinkedListNode* rabbit = head;
   SinglyLinkedListNode* turtle = head;

   while(rabbit!=NULL && rabbit->next!=NULL){

       rabbit = rabbit->next->next;

       turtle = turtle->next;
       if(rabbit == turtle)
           return 1;
   }
   return 0;
}
```

Q 5.4 For the following questions, select the correct option. **(0.5 x 6 = 3 marks)**

a) Local variables are stored in an area called _____

1) Permanent storage area ☐
2) Stack ■
3) Heap ☐
4) Free memory ☐

b) Queue data structure works on the principle of _____

1) First In Last Out (FILO) ☐
2) Last In First Out (LIFO) ☐
3) Last In Last Out (LILO) ☐
4) First In First Out (FIFO) ■

c) Choose the statement which is incorrect with respect to dynamic memory allocation.

1) Execution of the program is faster than that of static memory allocation ■
2) Memory is allocated in a less structured area of memory, known as heap ☐
3) Allocated memory can be changed during the run time of the program based on the requirement of the program ☐
4) Used for unpredictable memory requirements ☐

d) Which of the following header files must necessarily be included to use dynamic memory allocation functions?

1) dos.h ☐
2) stdlib.h ■
3) memory.h ☐
4) stdio.h ☐

e) The type of linked list in which the node does not contain any pointer or reference to the previous node:

1) Circular doubly linked list ☐
2) Singly linked list ■
3) Doubly linked list ☐
4) Circularly singly linked list ☐

f) The advantage of using linked lists over arrays is that _____

1) The size of a linked list is fixed ☐
2) Linked list can be used to store a collection of homogenous and heterogeneous data types ☐
3) Insertion and deletion of an element can be done at any position in a linked list ■
4) Linked list is an example of linear data structure ☐

Q 6.1 Write True or False against the following statements. **(1 x 2 = 2 marks)**

a)	Given the pointer to the middle element of a singly linked list; you can not reach the head of the list.	**TRUE**
b)	A binary tree will never have equal number of nodes in the left and right subtree of the root	**FALSE**

Q 6.2 **(1.5 + 1.5 + 2 = 5 marks)**

a)	**Consider the following two definitions of a node:**
	a) `struct Node {`
	`int data;`
	`struct Node next;`
	`};`
	b) `struct Node {`
	`int data;`
	`struct Node * next;`
	`};`
		Which of the two are valid definitions?
		1) a ☐
		2) b ■
		3) Both ☐
		4) None ☐

b)	**Consider the following definition in c programming language**
	`struct node{`
	`int val;`
	`struct node * next;`
	`};`
	`typedef struct node NODE;`
	`NODE *p;`
		Which of the following code is used to create new node?
		1) p=(NODE*)malloc(NODE); ☐
		2) p=(NODE)malloc(sizeof(NODE)); ☐
		3) p=(NODE*)malloc(sizeof(NODE)); ■
		4) p=(NODE*)malloc(sizeof(NODE*)); ☐

c)	**Consider the following code:**
	`struct node {`
	`int x;`
	`int y;`
	`};`
	`int main(){`
	`int arr1[5] = {31,22,3,42,65};`
	`struct node p1;`
	`p1.x = 11, p1.y = 25;`
	`int arr2[5];`
	`struct node p2;`
	`arr2 = arr1;  // called line x`
	`p2 = p1;  // called line y`
	`return 0`
	`}`
		Which of the lines x and y will cause an error?
		1) x ■
		2) y ☐
		3) Both ☐
		4) None ☐

Q 6.3 Write the output of the following programs. **(1 x 3 = 3 marks)**

a) Head points to the linked list

4 --> 2 --> 3 --> 5 --> 1 --> NULL

```
1 5 3 2 4
```

```
struct Node {
    int data;
    struct Node * next;
};

void fun1(struct Node* head)
{
    if(head == NULL)
    return;

    fun1(head->next);
    printf("%d ", head->data);
}
```

b) Consider the following lines of code:

Here, the stack function initialises an empty stack and push and pop functions are implemented correctly.

Print the final state of the stack with a space between each element and the first element being top of the stack.

```
4 45 34
```

```
stc = stack()
stc.push(34)
stc.push(78)
stc.pop()
stc.push(97)
stc.pop()
stc.push(45)
stc.push(12)
stc.pop()
stc.push(4)
```

c) Consider the following lines of code:

Here the queue function initialises an empty queue and enqueue and dequeue functions are implemented correctly.

Print the final state of the queue with a space between each element and the first element being start of the queue and last being the end.

```
1 9 18 4
```

```
stc = queue()
stc.enqueue(23)
stc.enqueue(17)
stc.dequeue()
stc.enqueue(1)
stc.enqueue(9)
stc.enqueue(18)
stc.dequeue()
stc.enqueue(4)
```

Q 6.4 Fill in the blanks. Read the instruction before each question.

a) The following function should print the middle element of a linked list. Input to the function is the start of the linked list. **(1 + 2 + 1 + 1 = 5 marks)**

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

void printMiddle(struct node *start)
{
   struct node *p2 = start;
   struct node *p1 = start;

   if (start!=NULL)
   {
       while (p2 != NULL && p2->next != NULL)
       {
           p2 = p2->next->next;
           p1 = p1->next;
       }
       printf("The middle element is %d", p1->val);
   }
}
```

b) Circular Linked List **(1 + 1 + 1.5 + 1.5 = 5 marks)**

Circular linked list is similar to singly linked list. In circular linked list, next pointer to last node points to head of the linked list i.e. the first node in linked list. Given pointers to front and rear node of the circular linked list you have to implement Insert and Delete function.

`void Insert(int val)` function add node n to rear side of linked list.

`int Delete()` function remove node from front of the linked list and return the value of removed node.

Assume there always exist a node before we call Delete.

```
struct node {
   int key;
   struct node* next;
};
struct node* front=NULL;
struct node* rear=NULL;
void Insert(int val)
{
   struct node* temp = (struct node*)malloc(sizeof(struct node));
   temp->key = val;

   if (front == NULL)

      front = temp;                    // when linked list is empty

   else
       rear->next = temp;


   rear = temp;

   rear->next = front;

          // complete the circular linked list by updating rear pointer
}

int Delete ()
{
   int val = front->key;

   if(front == rear){                  // only one element in linked list
       val = front->key;
       free(front);
       front = NULL;
       rear = NULL;
   }
   else{
       struct node* temp = front;

       rear->next = front->next;      // update rear pointer

       front = front->next;           // update front pointer

       free(temp);
   }
   return val;
}
```

END OF QUESTION PAPER