INDIAN INSTITUTE OF TECHNOLOGY
KHARAGPUR

Stamp / Signature of the Invigilator

EXAMINATION (Mid Semester)										SEMESTER (Spring)			
Roll Number										**Section**		**Name**	
Subject Number	C	S	1	1	0	0	1	**Subject Name**		*Programming and Data Structures*			
Department / Center of the Student											**Additional sheets**		

Instructions and Guidelines to Students Appearing in the Examination

1. Ensure that you have occupied the seat as per the examination schedule.
2. Ensure that you do not have a mobile phone or a similar gadget with you even in switched off mode. Note that loose papers, notes, books should not be in your possession, even if those are irrelevant to the paper you are writing.
3. Data book, codes or any other materials are allowed only under the instruction of the paper-setter.
4. Use of instrument box, pencil box and non-programmable calculator is allowed during the examination. However, exchange of these items is not permitted.
5. Additional sheets, graph papers and relevant tables will be provided on request.
6. Write on both sides of the answer script and do not tear off any page. Report to the invigilator if the answer script has torn page(s).
7. Show the admit card / identity card whenever asked for by the invigilator. It is your responsibility to ensure that your attendance is recorded by the invigilator.
8. You may leave the examination hall for wash room or for drinking water, but not before one hour after the commencement of the examination. Record your absence from the examination hall in the register provided. Smoking and consumption of any kind of beverages is not allowed inside the examination hall.
9. After the completion of the examination, do not leave the seat until the invigilator collects the answer script.
10. During the examination, either inside the examination hall or outside the examination hall, gathering information from any kind of sources or any such attempts, exchange or helping in exchange of information with others or any such attempts will be treated as adopting 'unfair means'. Do not adopt 'unfair means' and do not indulge in unseemly behavior as well.

Violation of any of the instructions may lead to disciplinary action.

Signature of the Student

To be filled in by the examiner

Question Number	1	2	3	4	5	6	7	8	9	10	Total
Marks Obtained											

Marks obtained (in words)	Signature of the Examiner	Signature of the Scrutineer

Instructions: Answer all seven questions. Total marks = 10 + 15 x 6 = 100. Time = 3hrs. Write your answer only in the space provided. Use any other space for rough work. The question paper has total 12 pages.

Rough Work

1. (i) The value of `a[2]` after executing the code below is:

3

```
int a[5] = {0}; for (i = 1; i < 5; i++) a[i] = a[i - 1] + i;
```

(ii) What does the following program print?

19

`# 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 = 4; b = &c; a = &b;`
	`printf("%d", f(c, b, a));`
	`return 0;`
	`}`

(iii) The code below dynamically allocates space for a 2-D array:

```
double **arr;
arr = (double **) malloc(10 * sizeof(double *));
for (i = 0; i < 10; i++)
arr[i] = (double *) malloc(5 * sizeof(double));
```

How many values are stored in this array, and how are they organized?

B

A. 15 values, organized as one array of 10 values and a separate array of 5 values.
B. 50 values, organized as an array with 10 rows and 5 columns.
C. 50 values, organized as an array with 5 rows and 10 columns.
D. 100 values, organized as an array with 10 rows and 10 columns.
E 216 values, and they are not organized at all.

(iv) Following stack operations are performed on an initially empty stack:
push(5); push(1); push(4); push(3); pop( ); push(2); pop( ); pop( ); push(8); push(7); pop( );

What is the value stored in *top of the stack* after the above operations?

8

(v) Following queue operations are performed on an initially empty queue:

enqueue(6); enqueue(12); enqueue(13); dequeue( ); dequeue( ); enqueue(19); enqueue(21); enqueue(22); dequeue( ); enqueue(20);

What is the value of the *queue front* after the above operations?

19

2. Many calendar systems e.g., Julian, Gregorian, Saka, Hijri etc are prevalent in different parts of the world. A calendar system can be represented by the number of months in a year, and an array of the number of days in each of these months. For example, in Gregorian calendar there are 12 months with 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31, days respectively. A structure to represent a calendar system can be defined as:

```
struct calendar { int months; int days[15];};
```

A structure to represent a date (without year) can be defined as: `struct date {int dd; int mm;};`

(i) Complete the function *IsValidDate()* which takes as input a date *dt* and a calendar system *cs*, and returns 1 if the date is valid according to the calendar system *cs*, or zero if otherwise. **[3]**

```
int IsValidDate(struct date dt, struct calendar cs){

return (dt.mm <= cs.months) && (dt.dd <= cs.days[dt.mm-1]); }
```

(ii) Complete the function *Convert( )* which takes as input a valid date *dt1* in calendar system *cs1* and returns it as a date *dt2* in calendar system *cs2*. Assume that the start of years coincide in both calendar systems. **[12]**

```
struct date Convert(struct date dt1, struct calendar cs1, struct
calendar cs2){

struct date dt2;

int m, day_from_year_start = 0;

for ( m = 0; m < (dt1.mm -1); m++){

     day_from_year_start = day_from_year_start + cs1.days[m];

}      /* count of days from beginning of year is same for cs1 and cs2 */

day_from_year_start = day_from_year_start + dt1.dd;

m = 0;

while(day_from_year_start > cs2.days[m]) {

day_from_year_start = day_from_year_start – cs2.days[m];

m++;

}

dt2.mm = m+1;

dt2.dd = day from year start;

return dt2; }
```

3. An element of a 2D array is a saddle point if it is the "maximum" in its column and the "minimum" in its row. Assume all array elements as having distinct values. For example the element with index *(1, 1)* is a saddle point for the matrix: 1 2 3
7 5 6
8 4 9

(i) Complete the function below to check if the element *(r,c)* is a saddle point for the matrix *M* of size *n x n*. **[5]**

```
int isSaddlePt(int M[100][100], int n, int r, int c){
int i, j, flag = 1;
for(i = 0; i < n; i++)
     if (M[i][c] >  M[r][c]) {flag = 0; break;}
for(j = 0; j < n; j++)
     if (M[r][j] <  M[r][c]) {flag = 0; break;}
return flag;
}
```

(ii) Complete the program fragment below to print all the saddle points of the matrix *M* having size *n x n*. Assume that value of *n,* and elements of the matrix *M,* are already available. **[10]**

```
int M[100][100], n, i = 0, j = 0, flag = 0;
for (i = 0; i < n; i++) {
     for (j = 0; j < n; j++) {
          if (isSaddlePt(M, n, i, j)){
               printf("M[%d][%d] is a saddle point.\n", i, j);
               flag = 1;  break;   /* there can be only one */
          }                        /* saddle point in a row */
     }
     if (flag == 1) {
          i++;          /* there can be only one */
          flag = 0;     /* saddle point in a column */
     }
}
```

4. A *user* of a social networking site like facebook can be represented by a simplified structure consisting of a unique *uid*, a *username*, *number of friends*, and a list of *uid* of users who are *friends* of the said user:
`struct user { int uid; char username[30]; int num_frnds; int frnds[100]};`

(i) Complete the function *isFriend(struct user u1, struct user u2),* that returns 1 if users *u1* and *u2* are friends. Assume that if *u1* is a friend of *u2*, *u2* is also a friend of *u1*. **[4]**

```
int isFriend(struct user u1, struct user u2) {
int i;
for (i = 0; i < u2.num frnds; i++){
     if (u1.uid == u2.frnds[i]) return 1;
}    /*check friend list of u2, for presence of u1 */
return 0;
}
```

(ii) A measure of closeness of two users is the number of mutual friends they have. Complete the function *mutualFriends(struct user u1, struct user u2)*, that returns the number of mutual friends of users *u1* and *u2*. **[8]**

```
int mutualFriends(struct user u1, struct user u2){
int i, j, mfcount = 0; /* mcount stores number of mutual friends */
for(i = 0; i < u1.num frnds ; i++){

     for(j = 0; j < u2.num frnds ; j++){

          if ( u1.frnds[i] == u2.frnds[j])(There is an error in this
     question. Other answers will also be awarded marks)

               mfcount ++;
     }
}
return mfcount;}
```

(iii) A measure of difference between two users is the sum total of number of users who are friend of *u1* but not of *u2* and the number of users who are friend of *u2* but not of *u1*. Assume that a user is not friend of herself/himself. Complete the function: **[3]**
difference (struct user u1, struct user u2), that returns the difference value between users *u1* and *u2*.

```
int difference(struct user u1, struct user u2){
int mfcount = mutualFriends(u1, u2); /* think of friend list as set */

return ( u1.num frnds + u2.num frnds – 2 * mfcount );
}
```

5. A robot lands on the surface of Mars and moves around to explore it. The robot's location can be represented by its (*x, y*) co-ordinates. The robot can move in one of the four directions in the co-ordinate system: left (L), right (R), up (U), and down (D) in a single move. Every single move changes position in the corresponding direction by 1 unit. In order to communicate its location to earth the robot transmits its initial location (*x, y*), and a character string representing its move sequence. For example, if the initial location of the robot is (1, 1) and the sequence of moves is "LURR", the new location is (2, 2). A 2-dimensional *location* is represented by the structure: `struct location {int x; int y;};`

(i) Complete the function *newLocation(),* which takes as input a *location* pointer *locp*, a single move (character L/R/U/D), and updates the location after executing the move. **[4]**

```
void newLocation( struct location *locp, char move){
switch(move){
     case 'L': locp->x--; break ;
     case 'R': locp->x++; break ;
     case 'D': locp->y--; break ;
     case 'U': locp->y++; break ;
}
}
```

(ii) Suppose that the robot has lost communication and can be in any one of *n* locations. The space-ship radar now begins to search for it in a rectangular area. Complete the function *boundingRectangle( )*, which takes as input an array of *n* (< 10) locations and returns the smallest axis parallel rectangle which encloses all these locations. A rectangle structure is defined in terms of the co-ordinates of its bottom left and top right corners as: `struct rectangle { int xbl; int ybl; int xtr; int ytr;};` **[5]**

```
struct rectangle boundingRectangle(struct location locs[10], int n){
struct rectangle box ; int i;
box.xbl = locs[0].x; box.ybl = locs[0].y;
box.xtr = locs[0].x; box.ytr = locs[0].y;
for(i = 1 ; i < n; i++){
     if(locs[i].x  < box.xbl) box.xbl = locs[i].x;
     if(locs[i].x  > box.xtr) box.xtr = locs[i].x;
     if(locs[i].y  < box.ybl) box.ybl = locs[i].y;
     if(locs[i].y  > box.ytr) box.ytr = locs[i].y;
}
return box;}
```

(iii) The robot was exploring Mars and communicating its move sequence to Earth. However, in the process of interplanetary communication a single character in the movement string was lost and had to be replaced by '?' (e.g., "LU?R"), where '?' might be any of the four movements. Finally, after a total of *m (<100)* moves, the robot exhausted its battery and became immobile. Since the final location of the robot is uncertain because of the '?' in the movement string, the space-ship radar has to search for it within an axis parallel search rectangle. Complete the function *searchBox( )* below, which takes as input the initial location of the robot *loc*, a movement string *moves* (with a missing character replaced by '?'), total number of moves *m* (before getting immobile), and returns the smallest rectangle where the robot might be finally located. **[6]**

```
struct rectangle searchBox(struct location loc, char moves[100], int m){
int i; struct location ll[4]; /* stores possible final locations */
for(i = 0; i < m; i++){
if(moves[i] != '?') newLocation(&loc, moves[i]);
}
ll[0].x = loc.x-1; ll[0].y = loc.y;
ll[1].x = loc.x+1; ll[1].y = loc.y;
ll[2].x = loc.x; ll[2].y = loc.y-1;
ll[3].x = loc.x; ll[3].y = loc.y+1;
return boundingRectangle(ll,4); }
```

6. (i) Consider the stack data type: `struct stack{ int data[MAXSIZE]; int top;};`
Complete the following functions to (i) *create* an empty stack, (ii) check if a stack *is empty*, (iii) *push* into a non-full stack, and (iv) *pop* from a non-empty stack. We define an empty stack to have *top = -1*. **[5]**

```
struct stack create(){
struct stack s;
s.top = -1;
return s; }
```

```
int isempty(struct stack *s){
return (s->top == -1) ;
}
```

```
void push(struct stack *s, int x){
s->top ++ ;
s->data[s->top] = x;
}
```

```
int pop(struct stack *s){
int x = s->data[s->top];
s-> top --;
return x; }
```

(ii) A decimal number is converted to a binary number by continually divide-by-2 to give a result and a remainder of either a "1" or a "0" until the final result equals zero. For example, the binary equivalent of the decimal number 9 is 1001. Complete the program below which uses a stack to print the binary equivalent of the decimal number *n*. You can use the stack functions defined above. **[10]**

```
#include <stdio.h>
#define MAXSIZE 128
void main(){
struct stack s; int n;
scanf("%d", &n);
s = create();
while(n != 0){
     push(&s, n%2);    /* divide-by-2, and store remainder in stack */
     n = n/2;          /* continue with result of division */
}
printf("The binary number is: \n"); /*get binary number from stack */
while(!isempty(&s))  printf("%d",pop(&s));
}
```

7. (i) Complete the `SortedMerge()` function below that takes two non-empty lists, each of which is sorted in increasing order, and merges the them into one list which is in increasing order. `SortedMerge()` should return the new list. Assume that the elements of the lists are distinct and there are no common elements among the lists. For example if the first linked list `a` is 5->10->15 and the other linked list `b` is 2->3->20, then `SortedMerge()` should return a pointer to the head node of the merged list 2->3->5->10->15->20. The linked list node structure is defined as: `struct node {int data; struct node *next;}`. **[7]**

```
struct node *SortedMerge(struct node *a, struct node*b){

struct node *mergedList, *head;

if(a->data < b->data) mergedList = a;   else  mergedList = b;

head = mergedList;

while(a!= NULL && b != NULL){

        if(a->data < b->data){mergedList->next = a; a = a->next; }

        else{ mergedList->next = b ;  b = b->next;}

        mergedList = mergedList->next;

    }               /* if a list is exhausted before the other */

    if(a == NULL)  mergedList->next = b;

    else  mergedList->next = a;

    return head;

}
```

(ii) Now, complete the recursive function, `RecursiveSortedMerge()`, to merge two sorted lists. **[8]**

```
struct node* RecursiveSortedMerge(struct node* a, struct node* b){
struct node* result = NULL;

if (a==NULL) return(b);        /* base cases */
else if (b==NULL) return(a);

if (a->data < b->data) {       /* pick either a or b, and recur */
result = a;
result->next = RecursiveSortedMerge(a->next, b);
}
else {
result = b;
result->next = RecursiveSortedMerge(a, b->next);
}
return(result);
}
```

Rough Work

Rough Work