INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

Stamp / Signature of the Invigilator

EXAMINATION (Mid Semester)	**SEMESTER (Autumn)**

Roll Number										**Section**		**Name**	

Subject Number	C	S	1	0	0	0	1	**Subject Name**	*Programming and Data Structures*

Department / Center of the Student		**Additional sheets**	

Important Instructions and Guidelines for Students

1. You must occupy your seat as per the Examination Schedule/Sitting Plan.
2. Do not keep mobile phones or any similar electronic gadgets with you even in the switched off mode.
3. Loose papers, class notes, books or any such materials must not be in your possession, even if they are irrelevant to the subject you are taking examination.
4. Data book, codes, graph papers, relevant standard tables/charts or any other materials are allowed only when instructed by the paper-setter.
5. Use of instrument box, pencil box and non-programmable calculator is allowed during the examination. However, exchange of these items or any other papers (including question papers) is not permitted.
6. Write on both sides of the answer script and do not tear off any page. **Use last page(s) of the answer script for rough work.** Report to the invigilator if the answer script has torn or distorted page(s).
7. It is your responsibility to ensure that you have signed the Attendance Sheet. Keep your Admit Card/Identity Card on the desk for checking by the invigilator.
8. You may leave the examination hall for wash room or for drinking water for a very short period. Record your absence from the Examination Hall in the register provided. Smoking and the consumption of any kind of beverages are strictly prohibited inside the Examination Hall.
9. Do not leave the Examination Hall without submitting your answer script to the invigilator. **In any case, you are not allowed to take away the answer script with you.** After the completion of the examination, do not leave the seat until the invigilators collect all the answer scripts.
10. During the examination, either inside or outside the Examination Hall, gathering information from any kind of sources or exchanging information with others or any such attempt will be treated as **'unfair means'**. Do not adopt unfair means and do not indulge in unseemly behavior.

Violation of any of the above instructions may lead to severe punishment.

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**

Write the answers in the boxes or in the blank spaces only. You can use the designated spaces for rough works. This question has 18 pages.

1. Write down the output for the following programs. If you think that there will be a compilation error or a runtime error, mention that as the output.

   (a) What will be the output of the following program? **[2]**

```
#include<stdio.h>

int main(){
    int a=10;
    a+=a*a%2;
    a%=2;
    if((a=0)) printf("Correct");
    else printf("Wrong");
    return 0;
}
```

   Wrong

   (b) Find out the output of the following program. **[2]**

```
#include <stdio.h>

int main(){
    int i, j;
    for (i = 0; i <= 2; i++) {
        for (j = 0; j <= 2;) {
            printf("%d ",j);
            break;
        }
        printf("%d ",i);
    }
    return 0;
}
```

   0 0 0 1 0 2

   (c) What would be the output of the following code? **[2]**

```
#include <stdio.h>

int main() {
    int j=(printf("TEN="),10);
    printf("%d",j);
    return 0;
}
```

   TEN=10

   (d) What would be the output of the following code if the input is 219? **[2]**

```
#include<stdio.h>

int main()
{
    int num,out;
    for(scanf("%d",&num),out=0;num!=0;
                    out=out*10+(num%10),num=num/10);
    printf("%d",out);
    return 0;
}
```

   912

2. Complete the following program such that it prints the following pattern. **[2+2+1+1+2+2+2 = 12]**

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

```
#include <stdio.h>

/* Define a macro to find out the minimum of two numbers*/

#define minimum(a, b) ((a) < (b) ? (a) : (b))

int main() {
    int row, col, min;
    int n=5;

    /* len denotes the number of rows or columns
     * in terms of n */

    int len = n * 2 - 1;
    for(row = 0; row < len; row++) {
        for(col = 0; col < len; col++) {
            /* min is the element of a cell that will be
             * printed; find out the value of min in
             * terms of len, row and col */

            min = minimum(row, col);

            min = minimum(min, len - row - 1);

            min = minimum(min, len - col - 1);

            printf("%d ", min);
        }
        printf("\n");
    }
    return 0;
}
```

3. Complete the following programs by filling up the missing codes.

   (a) Given a 1-D array, the following program counts the number of even-spaced inversions in the array. We call a pair `(i, j)` as an even-spaced inversion when both the following conditions satisfy: (i) `(i, j)` is an inversion, which means `arr[i] > arr[j]` where `i < j`, and (ii) the difference between `i` and `j` is even. Fill up the missing code below to complete the program. **[1x5 = 5]**

```
#include<stdio.h>
int main(){
  int arr[] = {10,9,8,7,6,5,4,3,2,1};
  /*Stores the count of even-spaced inversions*/

  int even_inv_cnt = 0;
  int i,j;
  /* Iterating over all pairs of elements*/
  for(i=0;i<9;i++){

      for(j= i+1; j<10; j++){
      /*Checking the two mentioned conditions
       *of even-spaced inversions*/

          if(( arr[i] > arr[j] ) && ( j-i)%2==0 )){
              ++even_inv_cnt;
          }
      }
  }
  printf("Number of even-spaced inversions:

                      %d",even_inv_cnt);
  return 0;
}
```

Space for Rough Works

(b) The following program counts the number of unique numbers present in a 1-D array. For example, for an array = {1,2,3,4,5,4,3,2} as input, the output is 5. Fill up the missing codes.

[1x5 = 5]

```
#include<stdio.h>
int main(){
    int arr[] = {1,2,2,2,5,9,14,6,7,3,3,5};
    int unique_cnt=0,arr_size=12,i,j;

    for(i=0;i<arr_size; i++ ){
        if(arr[i] != -9999){

            unique_cnt++;
            printf("%d ",arr[i]);

            for(j=i+1 ;j<arr_size;j++){

                if( arr[j] == arr[i]){

                    arr[j] = -9999 ;
                }
            }
        }
    }
    printf("Count of unique numbers : %d", unique_cnt);
    return 0;
}
```

Space for Rough Works

(c) Given a 1-D array, the following program converts a k-chunked array, sorted in non-increasing order chunk-wise, into an array sorted completely in non-decreasing order. The length of the array will always be a multiple of k.

Ex : $A = \{6,5,5,9,6,3,1,5,7\}$ is a 3-chunked array, where $k = 3$, implying that chunks of 3 consecutive elements, starting from the first position of the array, is sorted in non-increasing order. $\{6,5,5\},\{9,6,3\},\{1,5,7\}$ are individually sorted in non-increasing order.

The desired output will be : $\{1,3,5,5,5,6,6,7,9\}$

Fill up the missing code below to complete the program.

[1x5 = 5]

```
#include<stdio.h>

void chunkedSortArray(int arr[], int size, int k){
    int i,sortIndex=0,min_val,min_part;
    // Stores the total number of partitions for the array

    int partitions = size/k;

    /*ptr_list stores present index of the smallest element
     *for each partition; part_count stores the number of
     *elements covered per partition; sorted_list stores the
     *final sorted list in non-decreasing order*/
    int ptr_list[100],  part_count[100], sorted_list[100];

    for(i=1;i<=partitions;i++){
        /*Assigning starting index for each partition,
         *consisting of the smallest element of
         *that partition*/
        ptr_list[i-1] = i*k-1;
        /* Initializing the number of elements
          *utilized per partition to 0*/
        part_count[i-1] = 0;
    }
    /* Repeat till sorted_list array is complete*/
    while(sortIndex < size){
        /*Storing the minimum value across all
          *partitions and the partition number*/
        min_val = 1000,min_part=-1;

        for(i=0;i<partitions;i++){
             /*Check to determine whether the
               *partition is not empty*/

            if(part_count[i] < k){
                /*Finding the minimum value
                  *across all the partitions*/
                if(arr[ptr_list[i]] < min_val){

                    min_val = arr[ptr_list[i]];
                    min_part = i;
                }
```

```
                    }
                }
                if(min_part != -1){
                /*Since the final array is sorted is in
                 *non-decreasing order, we add the
                 *smallest element in each iteration*/

                    sorted_list[sortIndex++] = min_val;
                    part_count[min_part] = part_count[min_part]+1;

                    ptr_list[min_part] = ptr_list[min_part]-1;
                }
            }
            printf("Sorted list :\n");
            for(i=0;i<size;i++){
                printf("%d,",sorted_list[i]);
            }
        }

        int main(){
            int arr[] = {10,6,3,2,9,5,5,3,5,2,1,1};
            int k=4, arr_size=12;
            chunkedSortArray(arr,arr_size,k);
            return 0;
        }
```

4. What would be the output of the following code? **[5]**

```
#include<stdio.h>

int main(){
    int arr[] = {12,14,21,9, 3, 1, 15};
    int d=3,arr_size=7,iterations=5,i=0,j,temp,k;

    while(iterations--){
        j= (i+d)%arr_size;
        temp= arr[i];
        arr[i] = arr[j];
        arr[j] = temp;

        d++;
        i=(i+1)%arr_size;
        printf("\n");
        for(k=0;k<arr_size;k++){
            printf("%d ", arr[k]);
        }
    }
    return 0;
}
```

```
9 14 21 12 3 1 15
9 1 21 12 3 14 15
21 1 9 12 3 14 15
21 1 12 9 3 14 15
21 1 12 9 3 14 15
```

5. The following C program evaluates the value of *cos* function using the following series: $cos(x) = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n}}{(2n)!} = 1 - x^2/2! + x^4/4! - x^6/6! + ...$ As the expression contains infinite series, the value of the expression is evaluated using finite number of terms specified by the user. The program uses three functions to evaluate the *cos* function. Fill up the incomplete portions of the C program. **[0.5x20=10]**

```
#include<stdio.h>
double fact(double n){
    if (n == 0)
        return 1;
    else

        return (n*fact(n-1));
}

float power(double x, double n){
    if (n==0)
        return 1;
    if (n>0)

        return (x*power(x, n-1));
    if (n<0)

        return (power(x, n+1)/x);
}

double cosine(double x, double n){
    double p;
    if (n == 0)
        return 1;
    else

        p= (power(-1, n)*

            power(x, 2*n))/

                fact(2*n) +

                    cosine(x, n-1);
    return p;
}

int main(){
    double s,y,x,n;
    printf("Enter the number of terms to be considered: \n");
    scanf("%lf",&n);
    printf("Enter the value of x in degrees: \n");
    scanf("%lf",&x);

    y=((float)((int)x %

                        360)/180)*3.142;
```

```
        printf("y = %lf\n",y);
        s=cosine(y,n);

        printf("cos(%d) = %lf\n", (int)x,s);
        return(0);
    }
```

Space for Rough Works

6. Answer the following questions.

(a) Find out the output of the following program. **[4]**

```
#include<stdio.h>
int find(int number){
    if (number == 0)
        return 0;
    else
        return (number % 2 + 10 * find(number / 2));
}

int main(){
    int number = 639;
    printf("%d", find(number));
    return 0;
}
```

1001111111

(b) The following C program is used to determine whether the entered string is a palindrome or not, by using two C functions. One of the function is used to reverse the string and other one is used to compare the given string with reversed one. Fill up the incomplete portions of the C program. Note: Palindrome property of the string is that if the given string is same as it's reverse string (consider a single word with no blank space). Example-1: *madam*; Example-2: *amanaplanacanalpanama* **[0.5 x 12 = 6]**

```
#include <stdio.h>

int str_cmp (char arr1[], char arr2[]);
void reverse (char arr1[], char arr2[]);

int main (void)
{
    char str1[350], str2[350];
    printf ("Enter a string to check if it is a
                            palindrome or not.\n");

    scanf(" %s", str1);
    reverse (str1, str2);
    if (str_cmp(str1, str2)){
        printf ("The entered string is not palindrome.\n");
    }
    else{
        printf ("The entered string is a palindrome.\n");
    }
    return 0;
}

void reverse (char arr1[], char arr2[]){
    int i, j = 0;

    for (i = 0; arr1[i] != '\0'; i++);
    while (i--){
```

```
            arr2[j++] = arr1[i];
        }

        arr2[j] ='\0' ;
    }

    int str_cmp (char arr1[], char arr2[]){
        int i;
        int flag;

        for (i = 0; arr1[i] != '\0'

                            && arr2[i] != '\0'; i++){
            if (arr1[i] != arr2[i]){
                flag = 1;
                break;
            }
        }

        if (arr1[i] == '\0' && arr2[i] == '\0')
            flag = 0;
        else
            flag = 1;

        return flag;
    }
```

Space for Rough Works

7. Consider a polynomial with real (floating-point) coefficients: $f(X) = c_0X^{d_0} + c_1X^{d_1} + c_2X^{d_2} + \ldots + c_{t-1}X^{d_{t-1}}$ with integer degrees $0 \leq d_0 < d_1 < d_2 < \ldots < d_{t-1}$ and with coefficients c_i. We call each $c_iX^{d_i}$ as a nonzero term in $f(X)$.

   We store f as the sequence of coefficient, degree pairs, such as $(c_0;d_0);(c_1;d_1);(c_2;d_2);:::;(c_{t-1};d_{t-1})$ which is sorted with respect to the degrees (the second components in the pairs). For example, consider $f(x) = 4x^3 + 6x^2 + 7x + 9$. We would store f as the following sequence: $(4,3);(6,2);(7,1);(9,0)$

   To achieve this, We first define a term as follows:

```
typedef struct term
{
    int coeff;
    int expo;
} Term;
```

   A polynomial may be represented using array of structure "Term". Consider that maximum degree of the polynomial as 10. The following program performs addition and multiplication of two polynomials. This program does the following:

   (a) First invokes `readPoly` function to read both the polynomials. Then it invokes the `addPoly` function to add both the polynomials. Complete the `addPoly` function. **[1+1+1+1+0.5+0.5=5]**

   (b) Next, the program invokes `multPoly` function to multiply both the polynomials. Complete the `multPoly` function. **[1+1.5+1.5=4]**

   (c) Finally it displays the result by invoking `displayPoly`. Complete the `displayPoly` function. **[1]**

```
#include<stdio.h>

/* declare three arrays first_poly, second_poly, result_add,
* result_mult of type structure poly.
* each polynomial can have maximum of ten terms
* addition result of first_poly and second_poly is stored in
* result_add
multiplication result of first_poly and
* second_poly is stored in result_mult */

typedef struct term
{
    int coeff;
    int expo;
} Term;

Term first_poly[10], second_poly[10];
Term result_add[10], result_mult[100];

/* function prototypes */
int readPoly(Term [ ]);
int addPoly(Term [ ], Term [ ], int , int , Term [ ]);
int multPoly(Term [ ], Term [ ], int , int , Term [ ]);
void displayPoly( Term [ ], int terms);

int main(){
```

```
    int t1, t2, t3, t4;

    /* read the first polynomial */
    t1 = readPoly(first_poly);

    /* read and display second polynomial */
    t2 = readPoly(second_poly);

    /* add two polynomials and display resultant polynomial */
    t3 = addPoly(first_poly, second_poly, t1, t2, result_add);
    printf(" \n\n Resultant polynomial after addition : ");

    displayPoly(result_add, t3);
    printf("\n");

    t4=multPoly(first_poly, second_poly, t1, t2, result_mult);
    printf(" \n\n Resultant polynomial after multiplication : ");

    displayPoly(result_mult, t4);
    printf("\n");
    return 0;
}

int readPoly(Term poly_term[ ])
{
    int t1, i;
    printf("\n\n Enter the total number of terms in
                    the polynomial:");
    scanf("%d", &t1);

    printf("\n Enter the COEFFICIENT and EXPONENT in
                    DESCENDING ORDER\n");
    for(i=0;i<t1;i++)
    {
        printf("   Enter the Coefficient(%d): ", i+1);
        scanf("%d", &poly_term[i].coeff);
        printf("   Enter the exponent(%d): ", i+1);
        scanf("%d", &poly_term[i].expo);
    }
    return(t1);
}

int addPoly(Term first_poly[ ], Term second_poly[ ],
                            int t1, int t2, Term result_add[ ])
{
    int i, j, k;
    i=0; j=0; k=0;

    while(i<t1 && j<t2)
    {
```

```
            if(first_poly[i].expo==second_poly[j].expo)
            {

                result_add[k].coeff=first_poly[i].coeff +

                                                second_poly[j].coeff;
                result_add[k].expo=first_poly[i].expo;

                i++; j++; k++;
            }

            else if(first_poly[i].expo>second_poly[j].expo)
            {
                result_add[k].coeff=first_poly[i].coeff;
                result_add[k].expo=first_poly[i].expo;
                i++; k++;
            }
            else
            {
                result_add[k].coeff=second_poly[j].coeff;
                result_add[k].expo=second_poly[j].expo;

                j++; k++;
            }
        }
        /* for rest over terms of polynomial 1 */

        while(i<t1)
        {
            result_add[k].coeff=first_poly[i].coeff;
            result_add[k].expo=first_poly[i].expo;
            i++; k++;
        }
        /* for rest over terms of polynomial 2 */
        while(j<t2)
        {
            result_add[k].coeff=second_poly[j].coeff;
            result_add[k].expo=second_poly[j].expo;
            i++; k++;
        }
        return(k); /* k is number of terms in resultant polynomial*/
    }


    int multPoly(Term first_poly[ ], Term second_poly[ ],
                                int t1,int t2, Term result_mult[ ])
    {
        int i,j,k;
        i=0;j=0; k=0;
        for(i=0; i< t1; i++)
        {
```

```
            for(j=0; j< t2; j++)
            {

                result_mult[k].expo=first_poly[i].expo +
                                        second_poly[j].expo;


                result_mult[k].coeff=first_poly[i].coeff *
                                        second_poly[j].coeff;
                k++;
            }
        }
        return(k);  /* k is number of terms in resultant polynomial*/

    }

    void displayPoly(Term poly_term[ ],int term)
    {
        int k;
        for(k=0;k<term-1;k++)
        printf("%d(x^%d)+", poly_term[k].coeff, poly_term[k].expo);

        printf("%d(x^%d)", poly_term[term-1].coeff,
                             poly_term[term-1].expo);
    }
```

Space for Rough Works