INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

	Stamp / Signature of the Invigilator

EXAMINATION (End 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. Other spaces can be used for rough works.

1. Answer the following questions.

   (a) Write down the outputs for the following code. **[2]**

```
#include <stdio.h>
int main(){
    int a = -100, b = 50;
    if(a > 0 && b < 0)
        a++;
    else if(a < 0 && b < 0)
        a--;
    else if(a < 0 && b > 0)
        b--;
    else
        b--;
    printf("%d\n",a + b);
    return 0;
}
```

   (b) What will be the output? **[5]**

```
#include <stdio.h>
#include <string.h>
int main(){
        char str[20] = "INDIAN_INSTITUTE";

        str[6] = '\0';
        strcat(str,"_TECHNOLOGY");

        printf("%s\n",str);
        return 0;
}
```

(c) What will be the output? **[3]**

```
#include<stdio.h>
int main(){
    int i = 0;
    char c = 'A';
    while(i < 2){
        i++;
        switch (c){
            case 'A':
            printf("%c", c);
            break;
            break;
        }
    }
    printf(" KGP\n");
    return 0;
}
```

2. Assume a shape that is composed of 4 triangles, but as a whole the shape constructs a square. Below are two examples of squares of length 5 and 6 (defined by variable `boardsize`), where the upper and the lower triangles are composed of symbol $ and the left and the right triangles are composed of symbol #.

Length 5

```
$$$$$
#$$$#
##$##
#$$$#
$$$$$
```

Length 6

```
$$$$$$
#$$$$#
##$$##
##$$##
#$$$$#
$$$$$$
```

A C program will print the above square. Fill up the incomplete portions of the C program. **[10x1=10]**

```
#include <stdio.h>
#define boardsize 6
int main(){
  int row, col;
  int diagA, diagB;
  for ( row=0; row<boardsize; row++ ){
    for ( col=0; col<boardsize; col++ ){
      /* Column numbers for the two diagonals */

      diagA = _____________________________;

      diagB = _____________________________;
      if ( diagA <= diagB ){

        if ( __________________ && ________________ )

          printf("_____________________");
        else

          printf("_____________________");
      }else{

        if (  ______________________ && ____________________   )

          printf("________________________");
        else

          printf("________________________");
      }
    }
    printf("\n");
  }
  printf("\n");
  return 0;
}
```

3. The following C program finds out the multiplication of two 2-D matrices using recursion. Complete the missing parts of the code. **[0.5+0.5+0.5+0.5+1x8=10]**

```
#include <stdio.h>
void multiply(int, int, int [][10], int, int, int [][10], int [][10]);
void display(int, int, int[][10]);

/*The two matrices are of dimension m1xn1 and m2xn2*/
int main(){
    int a[10][10], b[10][10], c[10][10] = {0};
    int m1, n1, m2, n2, i, j, k;

    printf("Enter rows and columns for Matrix A respectively: ");
    scanf("%d%d", &m1, &n1);
    printf("Enter rows and columns for Matrix B respectively: ");
    scanf("%d%d", &m2, &n2);
    if (n1 != m2)
        printf("Matrix multiplication not possible.\n");
    else{
        printf("Enter elements in Matrix A:\n");

        for (i = 0; i < ____________; i++)

            for (j = 0; j < ______________; j++)
                scanf("%d", &a[i][j]);
        printf("\nEnter elements in Matrix B:\n");

        for (i = 0; i < ________________; i++)

            for (j = 0; j < _______________; j++)
                scanf("%d", &b[i][j]);
        multiply(m1, n1, a, m2, n2, b, c);
    }
    printf("On matrix multiplication of A and B the result is:\n");
    display(m1, n2, c);
}
```

```
void multiply (int m1, int n1, int a[10][10], int m2,
                        int n2, int b[10][10], int c[10][10]){
    static int i = 0, j = 0, k = 0;
    if (i >= m1){
        return;
    }
    else if (i < m1){
        if (j < n2){
            if (k < n1){

                c[i][j] += _____________ * ___________________;
                k++;
                multiply(m1, n1, a, m2, n2, b, c);
            }

            k = _______________;

            j = ________________;
            multiply(m1, n1, a, m2, n2, b, c);
        }

        j = _________________;

        i = _________________;
        multiply(m1, n1, a, m2, n2, b, c);
    }
}

void display(int m1, int n2, ______________________){
    int i, j;
    for (i = 0; i < m1; i++){
        for (j = 0; j < n2; j++){

            printf("%d  ", _____________________);
        }
        printf("\n");
    }
}
```

4. A *Stack* is a linear data structure of fixed size, which follows a *Last In First Out* (LIFO) order. The following three basic operations are performed over a stack:
   Push: Adds an item in the stack. If the stack is full, then it is said to be an Overflow condition.
   Pop: Removes an item from the stack. The items are popped in the reversed order in which they are pushed, that means the item that is inserted (pushed) last is popped first. If the stack is empty, then it is said to be an Underflow condition.
   Top: Returns top element of stack.
   isEmpty: Returns true if stack is empty, else false.
   Given a stack, sort it using recursion. Use of any loop constructs like while, for, etc is not allowed.
   Input: 2, -5, 1, 12, 20
   Output: 20, 12, 2, 1, -5 **[10x1=10]**

```
#include <stdio.h>
#include <stdlib.h>
struct stack{
    int data;

    ____________________________ *next;
};

void initStack(struct stack **s){
    *s = NULL;
}

/* Check if the stack is empty*/
int isEmpty(struct stack *s){
    if (s == NULL)
        return 1;
    return 0;
}

/*Push an item to the stack*/
void push(struct stack **s, int x){
    struct stack *p = (struct stack *)malloc(sizeof(*p));
    if (p == NULL){
        printf("Memory allocation failed.\n");
        return;
    }
    p->data = x;
    p->next = *s;
    *s = p;
}

/*Pop an item out of stack */
int pop(struct stack **s){
    int x;
    struct stack *temp;
    x = (*s)->data;
    temp = *s;
    (*s) = (*s)->next;
    free(temp);
    return x;
}

/* Return top item of the stack*/
int top(struct stack *s){
    return (s->data);
}

/* Recursive function to insert an item in the sorted way.  The function
  * first checks if the stack is empty or the newly inserted item is greater
  * than top, i.e, more than all existing. In that case, the new element is
  * inserted. If the top element is greater, then it is removed and other
   * items recur. Once done, the top is put back. */
```

```
void sortedInsert(_____________________, int x){

    if ( ___________________ || ______________________ ){

        push(_________________);
        return;
    }

    int temp = pop(s);

    sortedInsert(________________________);

    push(__________________________);
}

/* Sort the stack items. The function removes the top element
  * from a non-empty stack and sorts the remaining stack items
  * in recursion. Following this it uses the sortedInsert function
  * to insert the top element at the correct position.*/
void sortStack(struct stack **s){
    if (!isEmpty(*s)){

        int x = pop(________________________);

        ________________________________ ;

        ________________________________ ;
    }
}

/*Print the stack contents*/
void printStack(struct stack *s){
    while (s){
        printf("%d ", s->data);
        s = s->next;
    }
    printf("\n");
}

int main() {
    struct stack *top;
    initStack(&top);
    push(&top, 2); push(&top, -5); push(&top, 1);
    push(&top, 12); push(&top, 20);
    printf("Stack elements before sorting:\n");
    printStack(top);
    sortStack(&top);
    printf("\n\nStack elements after sorting:\n");
    printStack(top);
    return 0;
}
```

5. Write a program which reads a number of elements from the input file “input.txt”, sort the elements using insertion sort and stores the sorted list of elements in “output.txt” file. Complete the following program. **[0.5+1+1+1+1+0.5+1+1+1+1+1=10]**

```
#include <stdio.h>
#include<stdlib.h>
int inArray(int* arr, char* filename);
void itemSort(int arr[], int n);
void outArray(int* arr, int n, char* filename);

int main(){
    char outfilename[]="output.txt";
    char infilename[]="input.txt";
    int *item;
    int n;

    _______________=inArray(item, infilename);
    itemSort(item, n);
    outArray(item, n, outfilename);
}

int inArray(int* arr, char* filename)
{
    int count=0, i=0, var;
    FILE *fp;
    fp=fopen(filename, "r+");

    while(fscanf(_____________________)!=EOF)
        count++;
    fclose(fp);

    arr=(int*)malloc(______________________);

    while(fscanf(__________________)!=EOF)
        i++;
    fclose(fp);

    return(____________________________);
}


void outArray(int* arr, int n, char* filename)
{
    int i;
    FILE *fp;

    fp=fopen(filename, __________________);
    for (i=0; i < n; i++)

    fprintf(____________________________);
    fprintf(fp, "\n");
    fclose(fp);
}

void itemSort(int* arr, int n){
    int i, key, j;
    for (i = 1; i < n; i++){
        key = arr[i];
        j = i-1;

        while (j >= 0 && _______________){

            arr[___________] = arr[__________];
            j = j-1;
        }

        arr[j+1] = _________________;
    }
}
```

6. Consider two sorted linked lists of decreasing order. Write a C program to merge the two lists and produce a sorted list of decreasing order. For example, if the two input lists are {12, 6, 2} and {32, 11, 4}, then the output list will be {32, 12, 11, 6 ,4 ,2}. **[20x1=20]**

```
#include<stdio.h>
#include<stdlib.h>
struct _node{
    int data;
    struct _node* next;
};
typedef struct _node node;

/* function to take the node from the front of the
   source list and move it to the front of the dest list.
 */
void movefront(______________ destRef, _______________ sourceRef){
    node* newnode = *sourceRef;

    *sourceRef = _____________________;
    newnode->next = *destRef;

    *destRef = ____________________;
}

/* Takes two lists sorted in increasing order, and merge
   their nodes together to make one sorted list
 */
node* mergelist(node* a, node* b){
    node temp;
    node* tail = &temp;
    temp.next = NULL;

    while (___________________){
        if (a == NULL){
            tail->next = ______________;
            break;
        }
        else if (b == NULL){

            tail->next = ________________;
            break;
        }

        if (a->data <= b->data)

            movefront(______________, ____________);
        else

            movefront(______________, ____________);

        tail = ___________________________;
    }

    return(___________________);
}

/* Function to insert a node at the beginging of the
   linked list */

void insertbegin(_________________ head_ref, int new_data){

    node* new_node =

        (_______________) malloc(sizeof(_________________));

    new_node->data  = new_data;

    new_node->next = _____________________;

    ______________________ = new_node;
}

/* Function to print nodes in a given linked list */
void displaylist(node *node){

    while (________________________){
        printf("%d ", node->data);

        node = ____________________________;
    }
}

int main(){
    node* res = NULL;
    node* a = NULL;
    node* b = NULL;
    insertbegin(&a, 66); insertbegin(&a, 34);
    insertbegin(&a, 29); insertbegin(&a, 19);
    insertbegin(&b, 88); insertbegin(&b, 57);
    insertbegin(&b, 46);
    res = mergelist(a, b);
    printf("Merged Linked List is: \n");
    displaylist(res);
    return 0;
}
```

Space for Rough Works