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. You can use the designated spaces for rough works. This question has 28 pages including the space for rough works.

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? **[1.5]**

```
#include <stdio.h>

#define ROW 3
#define COL 3

int main() {
    int matrix[ROW][COL] = {{7,3,4},{4,5,6},{6,7,8}};

    for(int i=0; i<ROW; i++) {
        for(int j=0; j<COL; j++) {
            if(i>j){
                matrix[i][j+1]++;
                printf("%d ", matrix[i-1][j]+matrix[j][i-1]);
            }
            else if(i<j){
                matrix[i+1][j]--;
                printf("%d ", matrix[i][j-1]+matrix[j-1][i]);
            }
            else
                printf("%d ", matrix[i/2][j/2]
                                  +matrix[(i+1)/2][(j+1)/2]);
        }
        printf("\n");
    }
    return 0;
}
```

```
4   4   7

4   7   10

7   10  10
```

What will be the type of the output matrix ? **[0.5]**

```
Symmetric Matrix
```

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

```
#include <stdio.h>

struct student{
    char *name;
};

struct student stud1;

struct student stud(void){
    stud1.name = "alice";
    printf("%s\t", stud1.name);
    stud1.name = "bob";
    return stud1;
}

int main(){
    struct student stud2 = stud();
    printf("%s\t", stud2.name);
    stud1.name = "jane";
    stud2.name = "rolf";
    printf("%s\t", stud1.name);
    printf("%s\t", stud2.name);
}
```

```
alice    bob     jane    rolf
```

(c) Consider the following 32-bit IEEE-754 single precision floating point number: 11000001 10101000 01100110 01100110. What is the decimal equivalent of this number? **[2]**

(a) -21.05
(b) -21.725325
(c) -20.625
(d) -22.5275

```
-21.05
```

(d) For the given C program, if x is given as 45.625, what will be the values of y and a? **[1+1=2]**

```
#include <stdio.h>
int main (){
   double x, y;
   long int a;
   scanf("%lf", &x);
   y = x - (int)x;
   a = (int)x;
   if (y >= 0.5) ++a;
   return (0);
}
```

```
y=0.625
```

```
a=46
```

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

```
#include<stdio.h>

int A(int m, int n){
    if (!m) return (n + 1);
    if (!n) return A(m - 1, 1);
    return A(m - 1, A(m, n - 1));
}

int main(){
    printf("A(1,2) = %d", A(1,2));
    return 0;
}
```

```
A(1,2) = 4
```

Space for Rough Works

2. Consider the following C program to check whether a given string ‘data’ is palindrome or not. We implement a stack to accomplish the task. Stack is a data structure with two principal operations, (a) push, which inserts an element to the stack, and (b) pop, which removes the most recently added element (top element) from the stack. The push and pop operations occur only at one end of the stack, referred to as the top of the stack. Complete the following program, which checks the palindrome property of string ‘data’.

[1+1+1+1+1+1+1+0.5+1+0.5+1=10]

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

/*Structure defining stack data structure*/
struct stack{
     int *a;
     int top;
     int maxSize;
};

 /*Initialize stack*/
void initstack(struct stack *p, int maxSize){
    p->maxSize=maxSize;

    p->a=(int*)malloc(maxSize*sizeof(int));
    p->top=-1;
}

/*Implement push*/
void push(struct stack *p, int item){

    if (p->top == p->maxSize - 1) {
        printf("Stack is full\n");
        return ;
    }
    p->top++;

    p->a[ p->top] = item;
}

/* Removes top element from stack */
 int pop(struct stack *p){
    int num;

    if (p->top ==  -1 ) {
        printf("Stack is empty\n");
        return -1;
    }
    num = p->a[p->top];

    p->top--;
    return num;
}
```

```
void chk_palindrome(char inputString[], int length, struct stack p){
    for(int i = 0; i < length; i++){

        if( pop(&p) != inputString[i] ) {
            printf("Not a Palindrome String\n");
            return 0;
        }
    }

    printf("Palindrome String\n");
    return 0;
}

int main() {
    char *data, c;
    int i, length, m;
    struct stack p;
    printf("Enter the maximum length of the string\n");
    scanf("%d",&m);

    data= (char*)malloc((m+1)*sizeof(char));
    printf("Enter a string\n");
    scanf("%s",data);

    length = strlen(data) ;
    printf("Length of the string=%d\n",length);

    initstack( &p,length);

    /* Push all characters of input String to Stack */
    for(i = 0; i < length; i++){

        push( &p, data[i]);
    }

    chk_palindrome( data,length,p );
    return 0;
}
```

3. Consider that the air traffic control of an airport maintains an array of flight plans for the in-air flights. Every flight plan contains the departure time of the flight, the arrival time of the flight and an unique flight ID. The time is expressed in hours and minutes of the day. Assume that the hours is represented in 24-hour format, and the flights where both the departure and the arrival are on the same day are considered. The following C program defines the structures to store the flight plans and then finds out the flight ID with maximum flight time. The flight time is the difference between the arrival time and the departure time.

   Complete the missing parts of the code to find out the flight with maximum flight time. Assume that in a single day, there can be maximum 10 flights that depart from the airport. **[0.5x20 = 10]**

```
#include<stdio.h>

/*structure to store time information*/

typedef struct{
    int hour;
    int mins;
} time;

/*structure to store flight plans*/

typedef struct{
    int flightid;

    time depart;//Departure time

    time arrive;//Arrival time
} flightplan;

/*structure to store the flight time
*for every flight*/

typedef struct{
    int flightid;
    int ftime;
} flighttime;

/* Function to take input from
the users*/

int getdata(flightplan _fp[]){
    int n,i;
    printf("\nHow many data:");
    scanf("%d",&n);

    for(i=0; i<n; i++){
        printf("\nFlight ID:");

        scanf("%d",&_fp[i].flightid);
        printf("\nDeparture Hour:");
        scanf("%d",&_fp[i].depart.hour);
        printf("\nDeparture Mins:");

        scanf("%d",&_fp[i].depart.mins);
        printf("\nArrival Hour:");

        scanf("%d",&_fp[i].arrive.hour);
        printf("\nArrival Mins:");

        scanf("%d",&_fp[i].arrive.mins);
    }
    return n;
}

/* Function to calculate flight time for all the flight plans*/

void calculate_time(flightplan _fp[], flighttime _ft[], int n){
    int i;

    for(i=0; i<n; i++){
        _ft[i].flightid = _fp[i].flightid;

        /* Convert differences in hours and mins to total
         * minutes between the departure time and the
         * arrival time */

        _ft[i].ftime = (_fp[i].arrive.hour - _fp[i].depart.hour)*60

                        + (_fp[i].arrive.mins - _fp[i].depart.mins);
    }
}


/* Function to find the flight ID with maximum flight time */

int maxflight (flighttime _ft[], int n){
    int maxid=-1, maxtime=-1;
    int i;

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

        if(_ft[i].ftime > maxtime){

            maxtime = _ft[i].ftime;

            maxid = _ft[i].flightid;
        }
    }

    return maxid;
}
```

```
int main() {
    flightplan fp[10];
    flighttime ft[10];
    int n;

    n = getdata(fp);

    calculate_time(fp, ft, n);

    printf("\nFlight ID for maximum flight time is:%d",

                                maxflight(ft,n));
    return 0;
}
```

Space for Rough Works

4. Answer the following questions.

   (a) The following C program computes the common factors for a given set of numbers. Fill up the incomplete portions of the given C code. **[0.5x10 = 5]**

   Example:

   Input:

   Number of values = 4

   Value-1 = 32

   Value-2 = 64

   Value-3 = 160

   Value-4 = 128

   Output:

   Common factors of Value-1, Value-2, Value-3 and Value-4 are 1,2,4,8,16,32.

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

int *q;
int *common_factor(int *p,int n){
  int j=0,min=*p,ct;
  for(int i=0;i<n;i++)

   if(*(p+i)< min)
      min = *(p+i);

   q=(int*)malloc(min*sizeof(int));
   if(q==NULL){
      printf("\nMemory not available\n");
      exit(1);
   }
   /* Loop to compute common factors till the
       minimum of given set of numbers */

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

      for(int k=0;k<n;k++){

        if(*(p+k)%i==0)

         ct++;
      }
      if(ct==n){

         *(q+j)=i;

          j++;
      }
    }
    *(q+j)=-1;
    return q;
 }
```

```
int main(){
  int a[50],n,*r;
  printf("\nEnter the value of n(no. of elements)\n");

  scanf("%d",&n);
  printf("\nEnter the n elements\n");
  for(int j=0;j<n;j++)
    scanf("%d",&a[j]);

  r=common_factor(a,n);

  printf("\nThe common factors of given no.'s are:\n");

  for(int i=0;*(r+i)>0;i++)

    printf("%d ",*(r+i));
  printf("\n");

  free(q);

  return 0;
}
```

Space for Rough Works

(b) The following C program is used to enter student records and sort the records by the name of the student. Fill up the incomplete portions of the given C code. **[0.5x10 = 5]**

Example:

Number records to enter = 3

Original student records (before sorting)

Roll	Name	Age
1	Tomy	17
2	Nithin	22
3	John	19

Sorted records based on student name

Roll	Name	Age
3	John	19
2	Nithin	22
1	Tomy	17

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

struct record{
    int roll,age;
    char name[20];
};

struct record *namesorting(struct record *s,int n){
    int i=0,a,b,j=0;
    char c[20];
    for(i=0;i<n;i++){
        for(j=i;j<n;j++){

            if(strcmp((s+i)->name,(s+j)->name)>0){

                a=(s+i)->age;
                (s+i)->age=(s+j)->age;

                (s+j)->age=a;

                b=(s+i)->roll;
                (s+i)->roll=(s+j)->roll;
                (s+j)->roll=b;

                strcpy(c,(s+i)->name);

                strcpy((s+i)->name,(s+j)->name);
                strcpy((s+j)->name,c);
            }
        }
    }
    return(s);
}
```

```
int main(){
    int n,i=0;
    struct record *s;
    printf("Enter the no of records: \n");
    scanf("%d",&n);
    s=(struct record *)malloc(n*sizeof(struct record));
    while(i<n){
        printf("Enter the data of the students %d:
                    roll name age\n",i+1);

        scanf("%d %s %d",&((s+i)->roll),(s+i)->name,

                          &((s+i)->age));
        i++;
    }
    printf("\n Original student records (before sorting) \n");
    for(i=0;i<n;i++){
        printf("\nStudent [%d] = ",i+1);
        printf("%d %s %d",(s+i)->roll,(s+i)->name,(s+i)->age);
    }

    s=namesorting(s,n);

    printf("\nSorted records based on student name\n");
    for(i=0;i<n;i++){
        printf("\nStudent [%d] =  ",i+1);
        printf("%d %s %d",(s+i)->roll,(s+i)->name,(s+i)->age);
    }
    printf("\n");
}
```

Space for Rough Works

5. Answer the following questions.

(a) The following C program is used to compute the first, second and third largest numbers from the given set of positive integers entered through keyboard. Fill up the incomplete portions of the given C code. **[0.5x10 = 5]**

Example:
Input = 123, 529, 99, 47, 872
Output :
Largest = 872
Second largest = 529
Third largest = 123

```
#include<stdio.h>

int main(){
    int N, first=0, second=0, third = 0, n, i;
    printf("Enter the number of values: \n");

    scanf("%d",&N);
    for(i=0;i<N;i++){
        printf("Enter element %d\t", i);

        scanf("%d",&n);
        if(n >= first){

            third = second;

            second = first;

            first = n;
        }

        else if(n >= second){

            third = second;

            second = n;
        }

        else if(n >= third)

            third = n;
    }
    printf("\nFirst largest = %d\n Second largest = %d\n
            Third largest = %d\n", first, second, third);
    return(0);
}
```

Space for Rough Works

(b) The following C program is used to enter the numbers through keyboard and pass them into a file. Later the contents of the file are read out, and the program computes the sum and average of the numbers. Finally, the computed sum and average are appended to the existing contents of the file, and then display the file contents. Complete the missing parts of the code. **[0.5x10 = 5]**

Example:

```
Enter how many numbers you want to enter into file: 3
Enter the numbers one after another:
10
15
5
File contents are:
10.000000
15.000000
5.000000
File contents after Sum and Avg computation:
10.000000
15.000000
5.000000
30.000000
10.000000
```

```
#include<stdio.h>

int main(){
    char filename[] = "foo.in";
    FILE *ifp,*ofp;
    ofp = fopen(filename,"w");
    int i,N;
    float avg = 0.0, no, sum=0.0, temp;

    printf("\nEnter how many numbers you want to
            enter into file: \n");
    scanf("%d",&N);
    printf("\nEnter the numbers one after another.\n");
    for(i=0;i<N;++i){
       printf("\n Enter the Number %d :",i+1);

       scanf("%f", &no);

       fprintf(ofp, "%f\n",

                    no);
    }

    fclose(ofp);

    printf("\n File Contents are \n");
    ifp =fopen("foo.in","r");

    while(fscanf(ifp,"%f",&no)

            != EOF ){
```

```
      printf("\n%f",no);
      sum += no;
   }
   avg = sum/N;

   ofp = fopen(filename, "a");

   // append  sum value to the EOF
   fprintf(ofp,"%f\n",sum);
   // append  Avg value to the EOF
   fprintf(ofp,"%f\n",avg);
   fclose(ofp);

   printf("\n After SUM and AVG computation,
               File Contents are : \n");

   ifp =fopen("foo.in", "r");

   while(fscanf(ifp,"%f",&temp)!= EOF){

      printf("\n%f",temp);
   }
   fclose(ifp);
}
```

Space for Rough Works

6. Answer the following questions.

(a) Consider the following code.

```
#include <stdio.h>
int main()
{
    int i, sum = 0,count=0;

    printf("Numbers: ");
    for(i = 1; !((i+1)%5==0 || (i+7)%6 == 1 ||
                    (i-27)%7==2); i = i*(i+3)/2, count++){
        printf("%d ", i%17);
        sum += i/12345678;
    }
    printf("\n");
    printf("Count = %d\n",count);
    printf("Sum = %d\n", sum);
    return 0;
}
```

Does the loop terminate (yes/no)? **[0.5]**

```
Yes
```

What will be the output of the above code? **[2+0.5+1=3.5]**

```
Numbers: 1 2 5 3 9 3 9

/*The set of numbers getting printed when the instruction
printf("%d ", i%17); is executed */
```

```
Count = 7
```

```
Sum = 29
```

(b) Consider a program that sorts an array `arr[0...n-1]` in the wave form, i.e. `arr[0] >= arr[1] <= arr[2] >= arr[3] <= arr[4] >= arr[5] ....` Complete the following program to sort an array in wave form. **[0.5x7=3.5]**

```
#include<stdio.h>

// A utility method to swap two numbers

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

void sortInWave(int arr[], int n){
    // Traverse all even elements
    for (int i = 0; i < n; i+=2) {
        // If current even element is smaller than previous

        if (i>0 && arr[i-1] > arr[i])
            swap(&arr[i], &arr[i-1]);

        // If current even element is smaller than next

        if (i<n-1 && arr[i] < arr[i+1])
            swap(&arr[i], &arr[i + 1]);
    }
}

// Driver program to test above function
int main(){
    int arr[] = {10, 90, 49, 2, 1, 5, 23};

    int n = sizeof(arr)/sizeof(arr[0]);

    sortInWave(arr, n);

    for (int i=0; i<n; i++)
          for (int i=0; i<n; i++)
             printf("%d ", arr[i]);
          return 0;
}
```

Considering `n` number of inputs, what will be the complexity of the algorithm? **[0.5]**

O(n)

(c) Consider the following C program that checks whether a matrix is symmetric or not. Complete the missing parts of the code. **[0.25x8=2]**

```
#include<stdio.h>

int main(){
  int m, n, c, d, matrix[10][10], transpose[10][10];

  printf("Enter the number of rows and columns of matrix\n");
  scanf("%d%d", &m, &n);
  printf("Enter elements of the matrix\n");

  for (c = 0; c < m; c++)
    for (d = 0; d < n; d++)
      scanf("%d", &matrix[c][d]);

  for (c = 0; c < m; c++)
    for (d = 0; d < n; d++)

      transpose[d][c] = matrix[c][d];

  if (m == n){
    /* check if order is same */
    for (c = 0; c < m; c++){
      for (d = 0; d < m; d++){

        if (matrix[c][d] != transpose[c][d])
          break;
      }
      if (d != m)
        break;
    }
    if (c == m)

      printf("The matrix is symmetric\n");
    else

      printf("The matrix is not symmetric\n");
  }
  else
    printf("The matrix is not symmetric.\n");

  return 0;
}
```

Space for Rough Works

7. Answer the following questions.

(a) What will be the output of the following code.

[0.5+2+1.5=4.5]

```
#include <stdio.h>

int main(){
    int i,j; //loop counters

    int arr[]={100, 200, 31, 13, 97, 10, 20, 11};
    int len = sizeof(arr)/sizeof(arr[0]);
    printf("Len = %d\n",len);
    printf("Flag = ");
    for(i=0; i<len; i++){
           int flag=0;
           for (j=i;j<len;j++){
                   if ((arr[i]+arr[j])%55==0)
                            flag = 1;
            }
            printf("%d  ", flag);
            if(flag){
                   for(j=i; j<len; j++){
                           arr[j] = arr[j+1];
                   }
                   //decrease loop counter by 1,
                    //to check shifted element
                    i--;
                    //decrease the length
                    len--;
            }
    }

    printf("\n");
    printf("MODIFIED ARRAY ");
    for(i=0; i<len; i++)
           printf("%d ",arr[i]);

    printf("\n");

    return 0;
}
```

```
Len = 8
```

```
Flag = 1  1  0  1  0  0  0  0
```

```
MODIFIED ARRAY  31 97 10 20 11
```

(b) You are given a 2D array (`n x m`), where `n` is the number of rows and `m` is the number of columns and indexed from [0,0]. You start from the top leftmost corner of the matrix indexed at [0,0]. The following program prints the given 2D array in an diagonal-wise matrix traversal .

Ex : Given the following 5x3 matrix,

```
1   2   3
4   5   6
7   8   9
10  11  12
13  14  15
```

The diagonal-wise matrix traversal will yield – 1 4 2 3 5 7 10 8 6 9 11 13 14 12 15

The diaganoal traversal can be concieved as one comprising of four different types of movements: (a). Diagonally up or (b). Diagonally down, (c).One row down (when one hits the first or last column) or (d). one column right (when one hits the first or last row). The program follows the algorithm. Complete the program by filling up the missing codes.

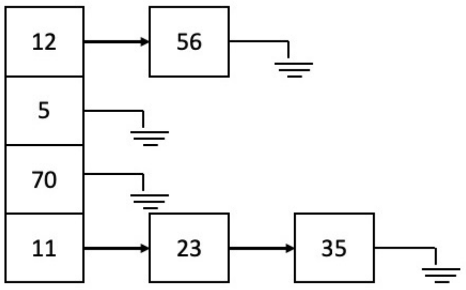
[0.5x12 = 6]

```
#include<stdio.h>

int main(){
    int i,j,count=1,row_num=3, col_num=5;
    int edge = 0;
    /* Edge variable determines whether present
        cell position is at one of the edges*/
    int arr[row_num][col_num];
    int row_i,col_j,up;
    // Initialising the 2-D array
    for(i=0;i<row_num;i++){
        for(j=0;j<col_num;j++){
            arr[i][j]= count++;
        }
    }
    row_i = 0;
    col_j = 0;
    for(count = 0; count < (row_num*col_num); count++){
        printf("[%d,%d,",row_i,col_j);
        printf("%d]\n",arr[row_i][col_j]);
        //Write the four condition for edges

        if((row_i == 0) || (row_i == row_num -1) ||

                (col_j == 0) || (col_j == col_num -1))
            edge = (edge + 1)%2;
        if(edge){

            if(row_i == row_num -1){
                col_j++;
                up = 1;
                continue;
            }
            if(col_j ==0){
                row_i++;
                up = 1;
                continue;
            }
            if(col_j == col_num -1){

                row_i++;
                up = 0;
                continue;
            }

            if(row_i ==0){
                col_j ++;
                up = 0;
                continue;
            }
        }
        if (up){

            row_i --;

            col_j ++;
        }
        else{

            row_i ++;

            col_j --;
        }
    }
}
```

Space for Rough Works

8. A hash-table is a one-dimension array where every data item `d` is inserted at array index `i = d % maxindex`, such that `maxindex` is the maximum value of the array index. Here `d % maxindex` is a hash function used to determine the array index. For example, consider `maxindex=20`. Then a data item `143` will be stored in array index `143 % 20 = 3`. However, note that multiple data items can be mapped to the same array index. For example, both `143` and `163` map to the same array index `3`. Therefore, at every array index, we maintain a linked-list to store the data items which map to the same index.

   As an example, consider that we insert the following data items in a hash-table with `maxindex = 4`: 5, 11, 23, 12, 70, 56, 35. The resultant hash table looks as follows.



   The following C program implements a hash-table with `maxindex=4`. Complete the missing parts of the code. **[0.5x20 = 10]**

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

#define MAXINDEX 4

struct _node{
    int data;
    struct _node *next;
};

typedef struct _node node;


/* Function to insert a new data in the hash-table*/

void inserthash(node **hashhead, int data){
    int h;
    node *temp, *thead;

    h = data % MAXINDEX;
    printf("\nThe data is inserted at index: %d", h);

    if(*(hashhead+h) == NULL){

        *(hashhead+h) = (node*)malloc(sizeof(node));

        (*(hashhead+h))->data = data;

        (*(hashhead+h))->next = NULL;
    }
    else{

        temp = (node*)malloc(sizeof(node));
        temp->data = data;
        temp->next = NULL;

        thead = *(hashhead+h);

        while(thead->next != NULL){

                thead = thead->next;
        }

        thead->next = temp;
    }
}


/* Function to display the hash-table*/

void display(node **hashhead){
    node *thead;
    int i;

    printf("\nThe Hashtable is:\n");

    for(i=0; i<MAXINDEX; i++){
        printf("Index %d:\t",i);

        thead = *(hashhead+i);

        while(thead!=NULL){

            printf("%d\t",thead->data);

            thead = thead->next;
        }

        printf("\n");
    }
}


/* Main function -- get input data from the user until
 * the user enters a negative number. Construct a
 * hash-table dynamically with the input numbers.
 * Print the hash-table at the end. */

int main() {
    int d, i;
    node *head[MAXINDEX];
    for(i=0; i<MAXINDEX; i++){
        head[i]=NULL;
    }

    printf("\nEnter data (Enter negative number to stop)::");
    scanf("%d",&d);

    /* Insert data until a negative number is inserted */
    while(d > 0){
        inserthash(head,d);
        printf("\nEnter data (Enter negative number to stop)::");
        scanf("%d",&d);
    }

    /* Print the hash-table*/
    display(head);

    /* Free the used space */
    for(i=0; i<MAXINDEX; i++){
        if(head[i]!=NULL)

                free(head[i]);
    }
    return 0;
}
```

Space for Rough Works

9. A maze (table with collection of paths, typically from an source to a destination) is represented as a NxM binary matrix of cells where is each cell can be uniquely identified as (i, j). Consider a rat is initially positioned at cell (0, 0) i.e.. maze[0][0] and the rat wishes to eat food which is present at cell (N-1, M-1) in the maze. However, in the maze, only some of the cells are accessible; other cells have obstacles, hence cannot be accessed. The obstacle and accessible information is stored in the maze matrix where maze[i][j]=1 indicates that the cell (i,j) has an obstacle and maze[i][j]=0 indicates that the cell (i, j) can be accessed to reach the destination. The rat can move in four directions- up, down, left, and right (not diagonally) from one cell to next cell (provided that cell does not have any obstacle).

   Consider the example given below. The maze is a 5x5 matrix with a value of 1 representing that the cell has an obstacle and a value of 0 indicating that the cell is accessible and can be used as a passage by the rat. The path marked with the grey color indicates the path taken by the rat to go from the source (0,0) to destination (4,4).

0	1	0	1	1
0	0	0	0	0
1	0	1	0	1
0	0	1	0	0
1	0	0	1	0

   Write a program in C which decides if the rat, initially positioned at cell (0, 0), can reach the food, located at cell (N-1, M-1) by traversing only the accessible cells. Essentially, the task is to check if there exists any path from source cell (0,0) to destination cell (N-1, M-1) and print the path if it exists, otherwise print "No solution". This problem can be solved by a recursive function, which implements the following approach. First check if the current cell is the destination. If yes, visit the cell and return 1. Otherwise, check if the rat is currently inside the maze, the current cell does not have any obstacle, and this cell is previously not visited. If true, then try to move in the four direction recursively. If successful, return 1. Finally, if the rat is unable to move in any of the directions, return 0. Complete the following program. Make your solution general by using the variables such as N,M whenever necessary, instead of writing the answer for a particular case. Your solution should work even if we change the value of SIZE to 10.

[1+(1+1)+(1+1)+1+(0.5+0.5)+(0.5+0.5)+(0.5+0.5)+(0.5+0.5)=10]

```
#include <stdio.h>

#define SIZE 5

/*the maze matric storing obstacle and accessible information */
int maze[SIZE][SIZE] = {
    {0,1,0,1,1},
    {0,0,0,0,0},
    {1,0,1,0,1},
    {0,0,1,0,0},
    {1,0,0,1,0}
};

/*matrix to store the solution path from source to destination*/
int solution[SIZE][SIZE];
```

```
/*function to print the solution matrix*/
void printsolution()
{
    int i,j;
    for(i=0;i<SIZE;i++){
        for(j=0;j<SIZE;j++){

            printf("%d\t", solution[i][j]);
         }
        printf("\n\n");
    }
}

/*recursive function to find the path*/
int solvemaze(int r, int c){
    /*if destination is reached, maze problem is solved*/

    if((r==SIZE-1) && (c==SIZE-1)){
        solution[r][c] = 1;
        return 1;
    }
    /*Otherwise, checking if we can visit in this cell */
    if(r>=0 && c>=0 && r<SIZE && c<SIZE &&

            solution[r][c] == 0 && maze[r][c] == 0){
        /*if possible to visit then visit the cell*/

        solution[r][c] = 1;

        /*going different direction*/
        if(solvemaze(r+1, c))
            return 1;

        if(solvemaze(r, c+1))
            return 1;

        if(solvemaze(r-1, c))
            return 1;

        if(solvemaze(r, c-1))
            return 1;
        /*If not possible, mark it*/
        solution[r][c] = 0;
        return 0;
    }
    return 0;

}
```

```
int main(){
    /*making all elements of the solution matrix 0*/
    int i,j;
    for(i=0; i<SIZE; i++){
        for(j=0; j<SIZE; j++){
            solution[i][j] = 0;
        }
    }

    if (solvemaze(0,0))
        printsolution();
    else
        printf("No solution\n");
    return 0;
}
```

Space for Rough Works

Space for Rough Works