Assignment -1

What is data structure? What are the various types of data structure? Explain.

Data structure is method of organizing and storing data in computer's memory to perform operations on data.

The various operations performed are data retrieval, data storage and manipulation of data. Data structures are divided into 2 main types is

1) Linear data structures
2) Non-Linear data structures

In linear structures all the elements are stored in a continuous manner.

Examples of linear structures are:

1) Array: A collection of elements that are identified using its index value
2) Linked Lists: All the elements are linked together by using pointers
3) Stacks: It follows last in First out (LIFO). Special
4) Queue operations are push, pop.

4) Queue: Follows First In, First out (FIFO). Special operations are enqueue and dequeue.

- In Non-Linear data structures the elements involve structural relationships between each other. Examples are:

1) Trees: Heirarchical method of storing data which has main node and multiple sub nodes.

2) Graphs: It is a collection of vertices and edges that connect pairs of nodes.

2 What is a structure? How is it different from arrays? Explain different types structure declaration with example.

Ans: A structure is a collection of data items, where each item is identified as to its type and name.

- It can hold data of multiple data types and not a single data type.
- It is different from arrays as arrays are collection of data of same type.
- Elements of arrays are accessed

using array indexing and members of structure are accessed using dot operator (.) and field name.

- The types of structure declaration are:

1) Along with structure definition

```
struct str1 {
        int i;
        char c;
} var;
```

We can add values to it in main part by using var

Ex: var.i = 1;
    var.c = 'a';

2) Outside structure definition

```
struct str1 {
        int i;
        char c;
};

void main() {

    struct str1 var;

}
```

We can add values to it by

using ~~a variable~~ var and assign and retrieve values.

Ex: var i = 1 ;
var c = 'c' ;

3) Using typedef keyword:

```
typedef struct str1 {
        int i;
        char c;
    } ter structure1;

void main()
{
    structure1 var;
    var.i = 1;
    var.c = 'a';
}
```

3 Define pointers. how to declare and initialize pointers, explain with examples

Ans
- Pointers are variables that stores memory address of another variable.
- Pointers provide a way to work directly with memory and help in dynamic memory allocation

Syntax for declaring a pointer is
datatype *ptr;
where i) datatype represents the type of data pointing to.
ii) ptr represents the name of pointer

Ex
```
int *ptr;
char *ptr;
void *ptr;
```

A void pointer can be declared but it has to be type casted to make it to point to a variable
A pointer is initialized by using an & operator

```
int var = 10;
int *ptr;
ptr = &var;
```

4 Explain dynamic memory allocation in detail

Ans
- Dynamic memory allocates memory for variables or data structures during runtime of program.
- Memory can be dynamically allocated using malloc(), calloc(), realloc(), free() which is present in stdlib.h header file

i) malloc() - It allocates single large block of memory with specific size

It returns a pointer of type void.
Syntax: ptr = (cast-type*) malloc(byte-size);
Ex: ptr = (int*) malloc(100*sizeof(int));
If the int size is 4, 400 bytes of memory is allocated
The pointer holds address of first byte in allocated memory.
· If space is insufficient it returns NULL.

2) calloc() -> · It dynamically allocates specified no of blocks of memory of specified type
· It initializes each block with default value 0.
Syntax: ptr = (cast-type*) calloc(n, elem size);
where n -> ~~size of~~ no. of elements
elem-size -> size of each element
Ex: ptr = (float*) calloc(25, sizeof(float));
The statement allocates contagious space for 25 elements each with size of float.

· If space insufficient it returns NULL pointer

3) realloc() -> · It helps in resizing of allocated memory.
· It increases or decreases size

Syntax: ptr = realloc(ptr, newSize);

Ex:
```
int *ptr;
int *iptr;
ptr = (int*) malloc(3*sizeof(int));
iptr = realloc(ptr, 4*sizeof(int));
```

4) free() - It is used to free or deallocate dynamically allocated memory and helps in reducing memory wastage

Syntax: free(ptr);

Ex: free(ptr);

5 What is a sparse matrix? Show with suitable example sparse matrix representation stating as triplets

Ans: A sparse matrix is a matrix of m x n order in which the number of 0 elements is greater than non-zero elements

Ex: $A = \begin{bmatrix} 0 & 0 & 0 & 5 \\ 3 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$

In this matrix, the No. of 0 elements are 9 and Non-zero elements are 3

· Representation of sparse matrix is done with the help of tuples

This is because if we use 2D array, lot of space will be wasted since no of 0 entries is large.

∴ We use <row, col, value> where

row - Total no of rows

col - Total no. of columns

value - ~~Total no. of non-zero values~~

Ex: $A = \begin{bmatrix} 0 & 0 & 5 & 0 \\ 1 & 2 & 0 & 0 \\ 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5 \end{bmatrix}$

~~Consider~~ Consider this sparse matrix.

```
typedef struct {
    int row;
    int col;
    int val;
} * term;
term a[50];
```

When we consider 'a' which is an array which consists of structures, the representation will be

a[0]	4	4	5
a[1]	0	2	5
a[2]	1	0	1
a[3]	1	1	2

a[4]	2	0	4
a[5]	3	3	5

a[49]

The 0th index represent the total no. of rows, columns, Non-zero values followed by the position of non-zero value from indexing 1

6 Express the given sparse matrix as triplets and find its transpose and also write transpose algorithm to transpose a sparse matrix

$$\begin{bmatrix} 15 & 0 & 0 & 22 & 0 & -15 \\ 0 & 11 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & -6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 91 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 28 & 0 & 0 & 0 \end{bmatrix}$$

Algorithm:

1) Declare matrices a and b where a is original sparse matrix & b is transpose sparse matrix
2) Declare variables n which stores no. of non-zero values, i, j and currentb.
3) Assign ~~first~~ 0 indexing of row of b matrix to 0 indexing of column of a matrix.
4) Similarly for 0 b[0].column and b[0].value
5) Initiate 2 loops i & j where i is from 0 to a[0].col, j is from 1 to n
6) compare a[j].col to i and if it is true perform the transpose and at ~~or~~ end increment currentb by 1

How do you represent a polynomial using an array of structures and also write a func to ADD 2 polynomials

A polynomial can be represented in following manner:

```
typedef struct {
        float coef;
        int expon;
        } polynomial;

The function to add 2 polynomials
is
polynomial terms[50];
void padd (int starta, int finisha,
           int startb, int finishb,
           int * startd, int * finishd)
{
    float coefficient;
    *startd = avail;
    while (starta <= finisha &&
           startb <= finishb)
    {
        switch (compare (terms[starta].
                expon, terms[startb].expon)
        {
            case -1:
                attach (terms[startb].
                        coef, terms[startb].
                        expon);
                startb++;
            case 0:
                coefficient = terms[starta].
                        coef + terms[startb].
                        coef
                attach (coefficient, terms[
                        starta].expon)
```

```
        starta++;
        startb++;
        break;
    case -1:
        attach(terms[starta].coef,
               terms[startb].expon);
        starta;
        break;
}

for ( ; starta <= finisha; starta++)
    attach(terms[starta].coef,
           terms[starta].expon);
for ( ; startb <= finishb; startb++)
    attach(terms[startb].coef,
           terms[startb].expon);
*finishd = avail - 1;

void attach(float coeff, int expon)
{
    if (avail >= 50)
    {
        printf("too many terms");
        exit(1);
    }
    terms[avail].coef = coef;
    terms[avail++].expon = expon;
}
```

8 Write the 'knuth morris pratt' algorithm for pattern matching and apply the same to search the pattern 'abcdabcy' in the text: "abceabcdabcxabcdabcdabcy".

Ans
```
#include <stdio.h>
#include <string.h>
int pmatch
void fail();
int pmatch();
int failure[100];
char string[100];
char pat[100];

int pmatch(char *string, char *pat)
{
    int i=0, j=0;
    int lens = strlen(string);
    int lenp = strlen(pat);
    while(i<lens && j<lenp){
        if(string[i]==pat[j])
        {
            i++; j++;}
        else if(j==0) i++;
        else j=failure[j-1]+1;
    }
    return((j==lenp)?(i-lenp):-1
}
```

```
void fail(char *pat)
{
    int n = strlen(pat);
    failure[0] = -1;
    for(j=1; j<n; j++)
    {
        i = failure[j-1];
        while((pat[j] != pat[i+1]) && (i >= 0))
            i = failure[i];
        if(pat[j] == pat[i+1])
            failure[j] = i+1;
        else failure[j] = -1;
    }
}
```

9 Write a c program to
a. compare strings
b. concatenate 2 strings

Ans a)
```
#include <stdio.h>
#include <string.h>
void main() {

    char s1[] = "abc";
    char s2[] = "def";
    int r = strcmp(s1, s2);
    if (r == 0)
    {
        printf("strings are equal");
    }
    else if (result < 0)
    {
      printf("string 1 is less
            than string 2");
    }
    else
    {
      printf("string 1 is greater
            than string 2");
    }
}
```

6) void strins (char *s, char *t, int i)

```
{
    char str1[50], *temp = str1;
    if (i < 0 && i > strlen(s))
    {
        printf ("Position out of
                 bounds");
        return;
    }
    if (! strlen(s))
        strcpy (s, t);
    else if (strlen(t))
    {
        strncpy (temp, s, i);
        strcat (temp, t);
        strcat (temp, (s+i));
        strcpy (s, temp);
    }
}
```

10 Define stack. Give the implementation of pop, push and display functions. Include check for empty and full condition

Ans · Stack is a data structure that follows last in, First out principle.
· It is an ordered list in which insertions and deletions are made at one end which is called top end.

Function to push an element onto stack (add element):

~~Void push(int top,~~

```
Void push()
{
    if (top < size-1)
    {
        printf("Enter element to be
                added: ");
        scanf("%d", &ele);
        s[++top] = ele;
    }
    else
    {
        printf("Stack overflow \n");
    }
}
```

Function to pop item from stack

```
int pop()
{
    if (top == -1)
    {
        printf("stack underflow\n");
        return -1;
    }
    else
    {
        return s[top--];
    }
}
```

Function to display elements of stack

```
Void display()
{
    if (top == -1)
    {
        printf("stack is empty\n");
    }
    else {
        printf("stack content:");
        for(i = top; i >= 0; i--)
        {
            printf("%d\n", s[i]);
        }
}
```

11 Write functions to convert infix expression to postfix and convert the following expression to postfix using stack:

A*(B+C*D)+E

Ans
```
#include <stdio.h>
char infix[100], postfix[100], k=0;
char s[50], top = -1;
void push (char elem)
{
    s[++top] = elem;
}
char pop()
{
    return (s[top--]);
}
int precedence(char elem)
{
    switch(elem)
    {
        case '(' : return 1;
        case '+':
        case '-' : return 2;
        case '*':
        case '/':
        case '%' : return 3;
        case '^' : return 4;
        default : return 0;
    }
}
```

```
void convert(char ch)
{
    switch(ch)
    {
        case '(':
            push(ch);
            break;
        case ')':
            while((ch=pop())!='(')
            postfix[k++]=ch;
            break;
        case '+':
        case '-':
        case '/':
        case '*':
        case '%':
        case '^':
            while(precedence(ch)<=
                  precedence(s[top]))
            postfix[k++]=pop();
            push(ch);
            break;
        default: postfix[k++]=ch;
    }
}
int main()
{
    printf("Enter infix expression:");
    gets(infix);
    push('\0');
```

```
for (int i=0; infix[i] != '\0'; i++)
  {
      convert (infix[i]);
  }
while (s[top] != '\0')
      postfix[k++] = pop();
printf("Postfix Expn: %s\n", postfix);
return 0;
}
```

For the expression: A*(B+C*D)+E

Input	stack	Postfix
A		
*		
(		
B		
+		
C		
*		
D		
)		
+		
E		