DATA STRUCTURES USING C

Assignment Set-II PAPER-II

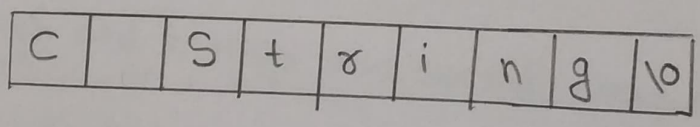
1. Write notes on various String Manipulation Function, Pointers and Structures of C with Suitable examples.

In C programming a string is a seallence of characters terminated will a null character \0.

for example

```
Char c[]="cString";
```

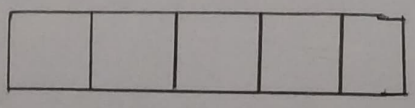
When the compiler encounters a seallence of characters enclosed in the double quation marks it appends a null character \0 at the end by default.



Memory diagram

Declaring a string

```
Char S[5];
```

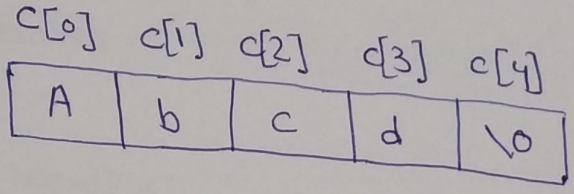


S[0] S[1] S[2] S[3] S[4]

String Declaration in C, Here we have declared a string of 5 characters.

Initializing a String

```
char c[] = "abcd";
char c[50] = "abcd";
char c[] = {'a', 'b', 'c', 'd', '\0'};
char c[5] = {'a', 'b', 'c', 'd', '\0'};
```



String Initialization in C

Assigning value to Strings

Arrays and strings are second-class citizens in C; they do not support the assignment operator once it is declared. for example

```
char c[100];
c = "C programming";   // Error! array type is not assignable
```

Note:- Use the strcpy() function to copy the string insted.

Read a string from the user

You can use the scanf() function to read a string. The scanf() function reads the sequence of characters until it encounters whitespace (space newline, tab etc)

example

```
#include <stdio.h>
int main()
{
Char name[20];
Printf("Enter name:");
Scanf("%s", name);
Printf("your name is %s.", name);
return 0;
}
```

output

Enter name: Faisal
your name is Faisal.

FUNCTIONS

Function are essentially just group of statements that are to be executed as a unit in a given order and that can be referenced by a unique name. The only way

to execute these statements is by invoking them or calling them using the functions name.

Traditional program design methodology typically involves a top-down or structured approach to developing software solution. The main task is first divided into a number simpler sub-tasks if these sub-tasks are still too complex they are subdivided further into simpler sub-tasks, and so on until the sub-tasks become simple enough to be programmed easily.

Function are highest level of the building blocks given to us in C and correspond to the sub-tasks or logical units referred to above. The identification of function in program design is an important step and will in general be a continuous process subject to modification as more becomes know about the programming

Problem in progress.

Syntax:
```
return_type function_name(parameter_list)
{
  body of function;
}
```

The above is termed the function definition

For example: An example Function program

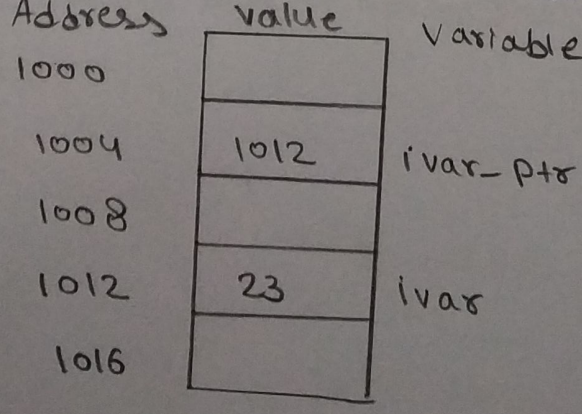
```
#include <stdio.h>      /* Standard I/O function Prototype*/
void Fun1(void);        /* Prototype*/
Void main(void)
{
   Fun1();              /* function call*/
}
Void Fun1()             /* function definition */
{
 Printf("inside the Function\n");
}
```

Pointers

Pointer are one of the most important mechanisms in C. They are the means by which we implement call by reference function, they are closely related to arrays and strings in C, they are fundamental to utilize C's dynamic memory allocation features, and they can lead to faster and more efficient code when used correctly.

A pointer is a variable that is used to store a memory address. Most commonly the address is the location of another variable in memory.

If one variable holds the address of another then it is said to point to the second variable



7

In the above illustration ivar is a variable of type int with a value 23 and stored at memory location 1012. ivar_ptr is a variable of type pointer to int which has a value of 1012 and is stored at memory location 1004. Thus ivar_ptr is said to point to the variable ivar and allows us to refer indirectly to it in memory.

example

```
#include <stdio.h>
int main() {
  int num = 10;
  int *ptr;
  ptr = &num;
  printf("value of num: %d\n", *ptr);
  *ptr = 20;
  printf("updated value of num: %d\n", num);
  return 0;
}
```

2. Discuss in detail about Arrays with matrix Multiplication Program.

An Array is a collection of data storage location, each having the same data type and the same name.

An Array can be visualized as a row in a table, whose each successive block can be thought of as memory bytes containing one element

Each storage location in an array is called an array element.

Arrays may be represented in Row-major form or Column-major form. In Row-major form all the elements of the first row are printed, then the elements of second row and so on up to the last row. In Column-major form, all the elements of the first column are printed, then the elements of the second column and so on up to the last column.

P.T.O

Matrix Multiplication program.

```
#include <stdio.h>
#define ROWS_A 3
#define ROWS_B 2
#define COLS_A 2
#define COLS_B 3

void matrix_multiply(int A[ROW_A][COLS_A],
     int B[ROWS_B][COLS_B],
     int c[ROWS_A][COLS_B]) {
     int i,j,k;
  for(i=0; i< ROWS_A; i++)
     {
     for(j=0; j<COLS_B; j++)
     {
      c[i][j]=0;
     for(k=0; k<COLS_A; k++){
        c[i][j]+= A[i][k] * B[k][j];
    }
   }
  }
  }
void display_matrix(int rows, int cols, int matrix[rows][cols])
    {
     int i,j;
```

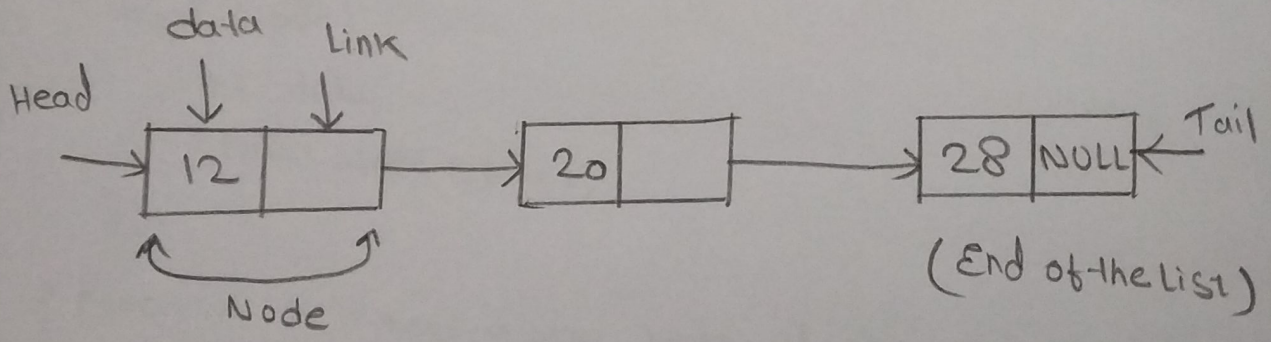
```
for (i= 0; i<rows; i++) {
    for (j =0; j<cols; j++) {
    Printf ("%d\t", matrix[i][j]); }
    Printf("\n");
  }
}
  int main() {
  int A[ROWS_A][COLS_A] = {{1,2},
                           {3,4},
                           {5, 6}};
int B[ROWS_B][COLS_B] = {{7,8,9},
                         {10, 11, 12}};
int c[ROWS_A][COLS_B];
matrix_multiply(A,B,C);
Printf("matrix A:\n");
display_matrix(ROWS_A, COLS_A, A);
printf("\n matrix B: \n");
display_matrix(ROWS_B, COLS_B, B);
Printf("\n Resultant matrix C(A*B): \n");
display_matrix(ROWS_A, COLS_B, C);
return 0;
}
```

3. Write notes on Linked Lists, Insertion operation on Linked Lists and their Applications

Linked lists can be defined as collection of objects called nodes that are randomly stored in the memory

A node contains two field i-e data stored at that particular address and the pointer which contains the address of the next node in the memory.

The last node of the containg pointer to the null.



Uses of Linked List

The list is not required to be contiguously present in the memory. The node can reside anywhere in the memory and linked together to make a list. This achieves optimized utilization of space.

List size is limited to the memory size and doesn't need to be declared in advance.

Empty node cannot be present in the linked list.

We can store values of primitive types or objects in the singly linked list.

Insertion operation on Linked.

```
#include <Stdio.h>
#include <Stdlib.h>

// Define the structure for a node
Struct Node{
    int data;
    Struct Node *next;
};

// Function to insert a new node at the beginning of the list
void insertAtBeginning(Struct Node** head_ref, int new_data) {
// Allocate memory for the new node
Struct Node * new_node = (Struct Node*) malloc(sizeof(Struct Node));
```

```
// Assign data to the new node
    new_node->data = new_data;
// Set the next of the new node to the current head
    new_node->next = *head_ref;
// Move the head to point to the new node
    *head_ref = new_node;
}
// Function to print the linked list
    void PrintList(Struct Node*node) {
    While (node != NULL) {
     Printf("%d", Node->data);
       node = node->next;
    }
    Printf("\n");
}
// main function to test the insertion operation
    int main() {
    // Initialize an empty linked list
     Struct Node* head = NULL;
// Insert some elements at the beginning of the list
```

```
insertAtBeginning(&head, 3);
insertAtBeginning(&head, 5);
insertAtBeginning(&head, 7);

// Print the linked list
printf("Linked list after insertion: ");
printList(head);

return 0;
}
```

Applications of Linked Lists

(i) Implementation of stacks and queues

(ii) Implementation of graph: Adjacency list representation of graph is most popular which is uses linked list to store adjacent vertices.

(iii) Dynamic memory allocation: we use linked list of free blocks.

(iv) maintainning directory of names

(v) performing arithmetic operation on long integers

(vi) manipulation of polynomials by storing constants in the node of linked list.

4. Discuss about Binary Trees, Various Binary Tree Representation and Binary Tree Traversals with examples.

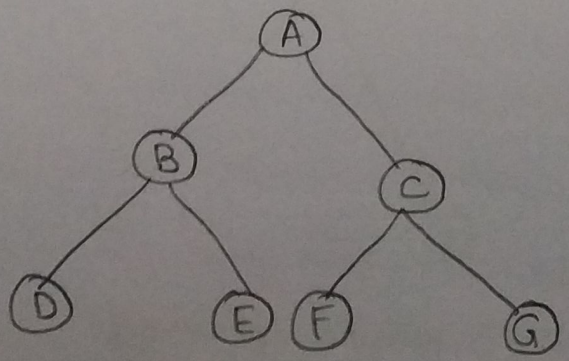
Binary Trees

In a normal tree, every node can have any number of children. A binary tree is a special type of tree data structure in which every node can have a maximum of 2 childred. one is known as as a left child and the other is know as right child

> A tree in which every node can have maximum of two children is called Binary tree.

In a binary tree, every node can have either 0 or 1 child or 2 childrsen but not more than 2 children.

example

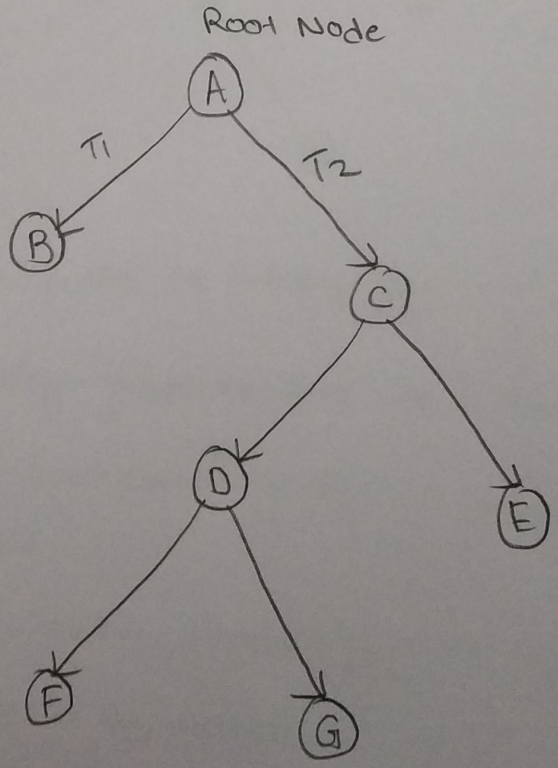


There are different types of binary tree and thyare

Strictly Binary Tree

In a binary tree, every node can have a maximum of two children. But in strictly binary tree, every node should have exactly two children or none. That means every internal node must have exactly two children. A strictly Binary Tree can be defined as follow.

> A binary tree in which every node has either two or zero number of children is called Strictly Binary Tree



Strictly Binary Tree

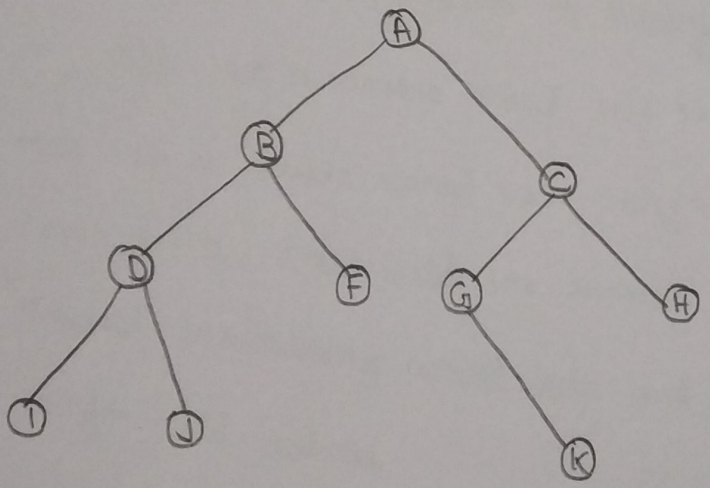
Binary Tree Representations

A Binary tree data structures is represented using two methods. Those methods are as follow

Array Representation

Linked List Representation

Consider the following binary tree



Array Representation of Binary Tree

In ~~one~~ array representation of a binary tree, we use one-dimensional array (1-D Array) to represent a binary tree.

Consider the above example of a binary tree and it is represented as follows

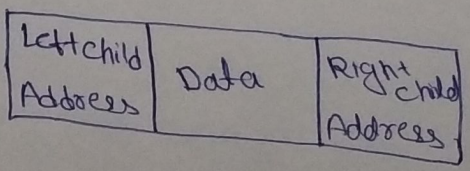
A	B	C	D	F	G	H	I	J	-	-	-	K	-	-	-

To represent a binary tree of depth 'n' using Array representation, we need one dimensional array with a maximum size of $2n+1$.

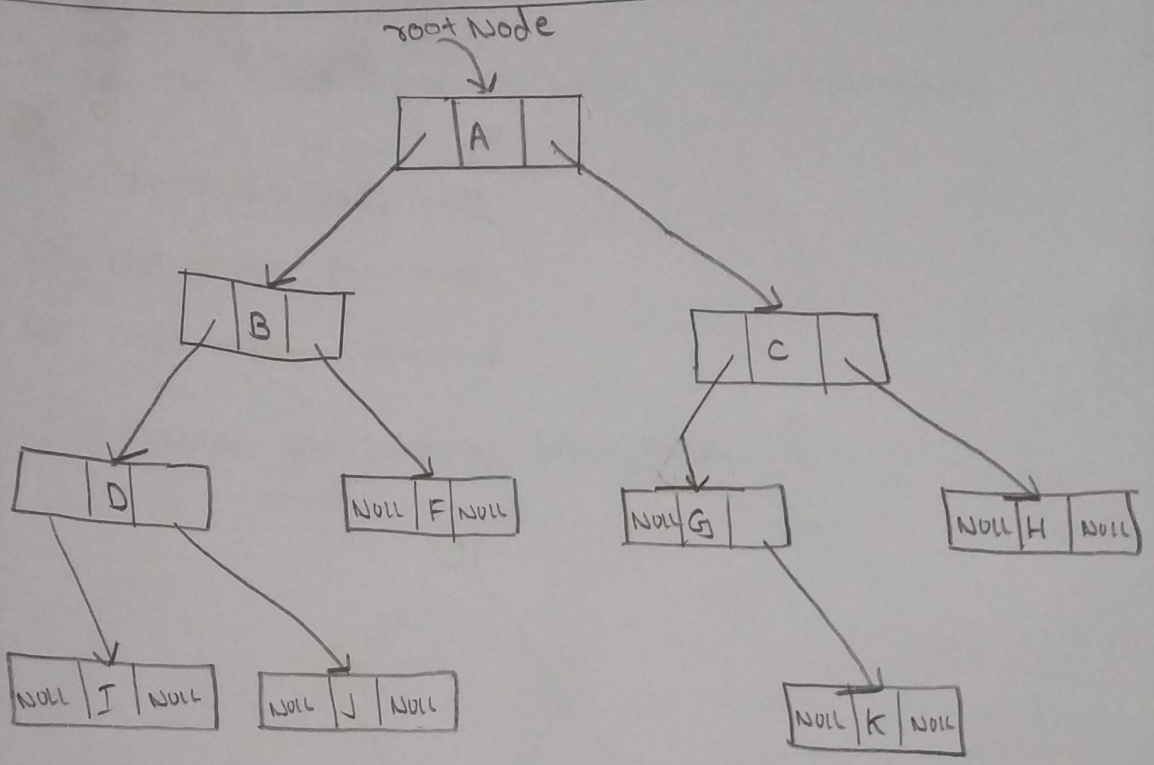
Linked List Representation of Binary Tree

We use a double linked list to represent a binary tree. In a double linked list, every node consists of three fields. First field for storing left child address, second for storing actual data and third for storing right child address

In this linked list representation, a node has the following structure



The above example of binary tree represented using Linked list representation is show as follows



Binary Tree Traversals

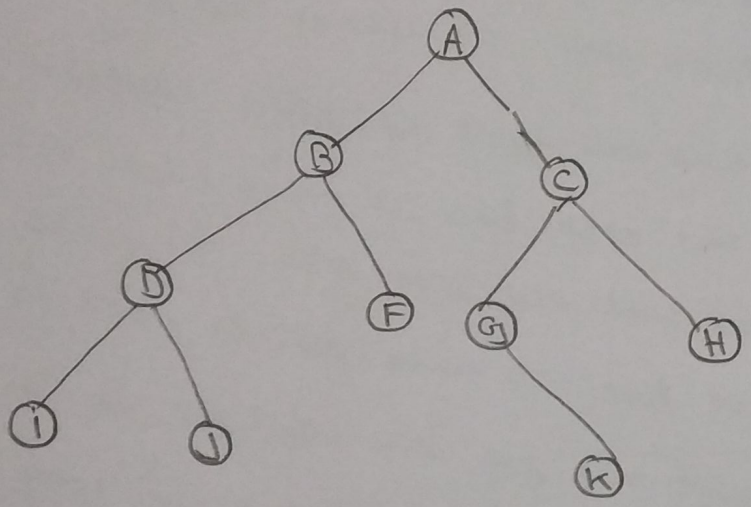
When we wanted to display a binary tree, we need to follow some order in which all the nodes of that binary tree must be displayed. In any binary tree, displaying order of nodes depends on the traversal method

> Displaying (or) visiting order of nodes in a binary tree is called as Binary Tree Traversal.

There are three types of binary tree traversals.

(1) In-order Traversal
(2) pre-order Traversal
(3) post-order Traversal

Consider the following binary tree..



In-order Traversal (leftchild - root - right child)

In In-order traversal, the root node is visited between the left child and right child. In this traversal, the left child node is visited first, then the root node is visited and later we go for visiting the right child node. This in-order traversal is applicable for every root node of all subtree in the tree.

In the above example of a binary tree, first we try to visit left child of root node 'A', but A's left child 'B' is a root node for left subtree. So we try to visit its (B's) left child 'D' and again D is a root for subtree with nodes D, I and J. So we try to visit its left child 'I' and it is leftmost child. So first we visit 'I' then go for its root node 'D' and later we visit D's right child 'J' with this we have completed the left part of node B. Then visit 'B' and next B's right child 'F' is visited. with this we have completed left Part of node A. Then visit root node 'A' with this we have completed left and root parts of node A. Then we go for the right part of the node A.

In right of A again there is a subtree with root C. So go for left child of C and again it is a subtree with root G. But G does not have left part so we visit 'G' and then visit G's right child K. With this we have completed the left part of node C. Then visit root node 'C' and

next visit C's right 'H' hich is rightmost child in the tree. So we That means here we have visited in the order of I-D-J-B-F-A-G-K-C-H using in order Traversal

In-order Traversal for above example binary tree is

I-D-J-B-F-A-G-K-C-H

Pre-order Traversal (root - leftChild - right child)

In pre order traversal, the root node is visited before the left child and right child nodes. This pre-order traversal is applicable for every root node of all sub tree in the tree. In the above example of binary tree, first we visit root node 'A' then visit its left child 'B' which is a root for D and F. So we visit B's left child 'D' and again D is a root for I and J. So we visit D's left child 'I' which is the leftmost child. So next we go for visiting D's right child 'J' with this we have completed root, left and right parts of node 'D' and root, left parts of node B. Next visit B's right child 'F' with

this we have completed root and left parts of node A So we go for A's right child 'C' which is a root node for G and H. After visiting C, we go for its left child 'G' which is a root for node K So next we visit left of G, but it doesn't have left child so we go for G's right child 'K' with this, we have completed node C's root and left parts. Next visit C's right child 'H' which is the rightmost child in the tree. So we stop the process.

That means here we visited in the order of A-B-D-I-J-F-C-G-K-H using pre order Traversal

Pre-order Traversal for above example binary tree is

A-B-D-I-J-F-C-G-K-H

Post-order Traversal (Left child-right child-root)

In post-order traversal, the root node is visited after left child and right child. In this traversal, left child node is visited first, then its right child and then it's root node. This is recursively performed until the right

right most node visited

Here we have visited in the order of I-J-D-F-B-K-G-H-C-A using Post-order Traversal.

Post-order Traversal for above example binary tree is

I-J-D-F-B-K-G-H-C-A.

5. Write notes on Searching, Sorting and Complete C Program to implement Quick Sort.

Searching

Searching is a process of finding a specific value in a list of existing value. In other words, searching is the process of locating given value position in a list of value.

Sorting

Arranging a list of given elements in an order (ascending or descending) is called as Sorting.

There are various sorting technique available. namely,

1. Bubble Sort
2. Selection Sort
3. Insertion Sort
4. Merge sort
5. Quick sort. e.t.c.

Program to implement Quick Sort.

```
#include<stdio.h>
void quicksort(int number[25], int first, int last){
int i, j, pivot, temp;
```

```
if(first<last){
pivot = first;
i=first;
j = last;
while(i<j){
While (number [i]<=number[pivot]&& i<last)
 i++;
While (number[j]>number[pivot]
 j--;
if(i<j){
temp =number [i];
 number[i] =number[j];
number [j] = temp;
}
}
temp = number [pivot];
number [pivot] =number [j];
number [j] = temp;
quicksort (number, first, j -1);
quicksort (number, j+1, last);
}
}
 int main(){
```

```
int i, count, number[25];
Printf("Enter some elements (max. -25):");
Scanf("%d", &count);
Printf ("Enter %d elements:", count);
for (i=0; i<count; i++)
Scanf("%d", &number[i]);
QuickSort (number, 0, count-1);
Printf("The sorted order is:");
for (i=0; i<count; i++)
Printf ("%d, number[i]);
return 0;
}
```