Q.2 a. If you are using C language to implement the heterogeneous linked list, what pointer type will you use?

Answer: The heterogeneous linked list contains different data types in its nodes and we need a link pointer to connect them. It is not possible to use ordinary pointers for this. So we go for void pointer. Void pointer is capable of storing pointer to any type as it is a generic pointer type.

b. What is the data structures used to perform recursion?

Answer: Stack. Because of its LIFO (Last In First Out) property it remembers its 'caller' so knows whom to return when the function has to return. Recursion makes use of system stack for storing the return addresses of the function calls. Every recursive function has its equivalent iterative (non-recursive) function. Even when such equivalent iterative procedures are written, explicit stack is to be used.

c. Write a recursive function to find factorial of a given number.

Answer: Page 128 of Text Book 1

Q.3 a. What is difference between structure and union? Write a program that will read and print the data for 10 students using the following structure: struct student {char name [30]; float mark;}

Answer: Page 141 to 142 of Text Book 1

b. Define file. Explain the major operations performed on sequential file.

Answer: Page 147 of Text Book 1

Q.4 a. Write an algorithm in C to implement bubble sort technique.

Answer: Page 194 of Text Book 1

b. Explain the procedure for Binary Search technique with the help of an example.

Answer: Page 213 to 215 of Text Book 1

Q.5 a. Write the difference between a stack and a queue. Write a program in C to implement queue using linked list.

Answer: Page 252 to 253 of Text Book 1

b. Translate the following infix expression into its equivalent postfix expression using stack:
A*(B+D)/E-F*(G+H/K)

Answer:
A*(B+D)/E-F*(G+H/K) = A*[BD+]/E-F*(G+[HK/]) = [ABD+*]E-F*[GHK/+] =[ABD+*E/]-[FGHK/+*] = ABD+*E/FGHK/+*-

Q.6 a. Write a procedure to reverse a singly linked list.

Answer:
A procedure to reverse a singly linked list:

```
reverse(struct node **st)
{
struct node *p, *q, *r;
p = *st;
q = NULL;
while(p != NULL)
{
r =q;
q = p;
p = p _ link;
q _ link = r;
}
*st = q;
}
```

b. Write a function which inserts a newly created node after a specified node.

Answer: Page 281 of Text Book 1

Q7 a. Define Circular Linked List. Write a program for creating and printing the elements of a Circular Linked List.

Answer: Page 315 to 16-17 of Text Book 1

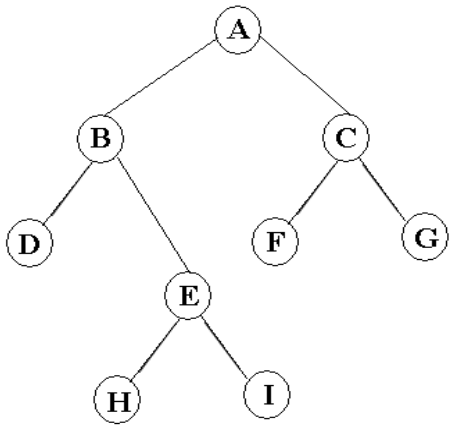
b. What are the disadvantages of single linked list? How these problems are solved by doubly linked list?

Answer: Page 330 to 331 of text Book 1

Q8. a. Define Binary tree. How binary trees are represented in memory?

Answer: Page 349 to 353 of text Book 1

b. Write the Preorder, Inorder and Postorder traversal for the following tree:



Answer: Page 355 of text Book 1

Q9 a. Define the following with the help of example:

(i) Adjacency Matrix of a graph
(ii) Adjacency List of a graph
(iii) In-degree and out-degree of a graph
(iv) Directed Acyclic Graph (DAG)

Answer: Page 390,392,393 to 420 of text Book 1

b. Write a program that implements depth first search algorithm

Answer:

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
#define TRUE 1
#define FALSE 0
#define MAX 8
struct node
{
int data ;
struct node *next ;
} ;
int visited[MAX] ;
void dfs ( int, struct node ** ) ;
struct node * getnode_write ( int ) ;
void del ( struct node * ) ;
void main( )
{
struct node *arr[MAX] ;
struct node *v1, *v2, *v3, *v4 ;
```

```
int i ;
clrscr( ) ;
v1 = getnode_write ( 2 ) ;
arr[0] = v1 ;
v1 -> next = v2 = getnode_write ( 3 ) ;
v2 -> next = NULL ;
v1 = getnode_write ( 1 ) ;
arr[1] = v1 ;
v1 -> next = v2 = getnode_write ( 4 ) ;
v2 -> next = v3 = getnode_write ( 5 ) ;
v3 -> next = NULL ;
v1 = getnode_write ( 1 ) ;
arr[2] = v1 ;
v1 -> next = v2 = getnode_write ( 6 ) ;
v2 -> next = v3 = getnode_write ( 7 ) ;
v3 -> next = NULL ;
v1 = getnode_write ( 2 ) ;
arr[3] = v1 ;
v1 -> next = v2 = getnode_write ( 8 ) ;
v2 -> next = NULL ;
v1 = getnode_write ( 2 ) ;
arr[4] = v1 ;
v1 -> next = v2 = getnode_write ( 8 ) ;
v2 -> next = NULL ;
v1 = getnode_write ( 3 ) ;
arr[5] = v1 ;
v1 -> next = v2 = getnode_write ( 8 ) ;
v2 -> next = NULL ;
v1 = getnode_write ( 3 ) ;
arr[6] = v1 ;
v1 -> next = v2 = getnode_write ( 8 ) ;
v2 -> next = NULL ;
v1 = getnode_write ( 4 ) ;
arr[7] = v1 ;
v1 -> next = v2 = getnode_write ( 5 ) ;
v2 -> next = v3 = getnode_write ( 6 ) ;
v3 -> next = v4 = getnode_write ( 7 ) ;
v4 -> next = NULL ;
dfs ( 1, arr ) ;
for ( i = 0 ; i < MAX ; i++ )
del ( arr[i] ) ;
getch( ) ;
}
void dfs ( int v, struct node **p )
{
struct node *q ;
```

```
visited[v - 1] = TRUE ;
printf ( "%d\t", v ) ;
q = * ( p + v - 1 ) ;
while ( q != NULL )
{
if ( visited[q -> data - 1] == FALSE )
dfs ( q -> data, p ) ;
else
q = q -> next ;
}
}
struct node * getnode_write ( int val )
{
struct node *newnode ;
newnode = ( struct node * ) malloc ( sizeof ( struct node ) ) ;
newnode -> data = val ;
return newnode ;
}
void del ( struct node *n )
{
struct node *temp ;
while ( n != NULL )
{
temp = n -> next ;
free ( n ) ;
n = temp ;
}
}
```

Text Book

C & Data Structuress , P.S. Deshpande and O.G. Kakde , Dream tech press, 2007