DEC 2014

Q1 a

What is recursion. WAP to find sum of n natural numbers using recursion (5)

Recursion is a phenomenon in which a function calls itself. A function which calls itself is called recursive function. Eg.factorial function can call itself recursively to find factorial of integer n

Program to find sum of n natural numbers

```
#include <stdio.h>
int sum(int n)
{
if (n==1)
return 1;
else
return n + sum(n-1);
}

void main()
{
int n , s;
printf("Enter value of n ");
scanf("%d" , &n);
  S = sum(n);
printf("sum = %d\n" , s);
}
```

Q1 b

What is a multiway search tree. Explain with example (5)

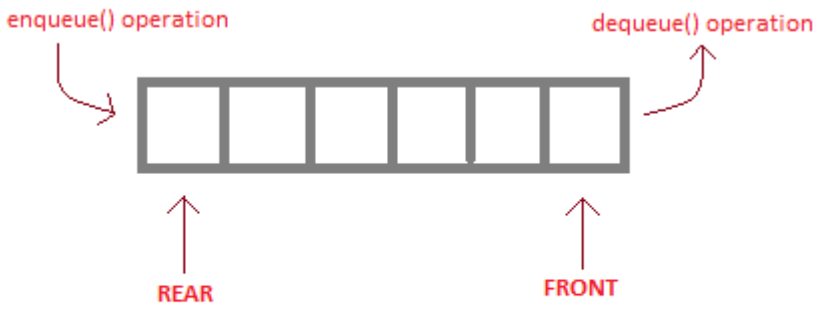
Refer Note Book

Q1 c

Give ADT for queue. Give any 2 applications of queue. (5)

Queue is also an abstract data type or a linear data structure, in which the elements are inserted from one end called **REAR**(also called tail), and the deletion of existing element takes place from the other end called as **FRONT**(also called head). This makes queue as FIFO data structure, which means that element inserted first will also be removed first.

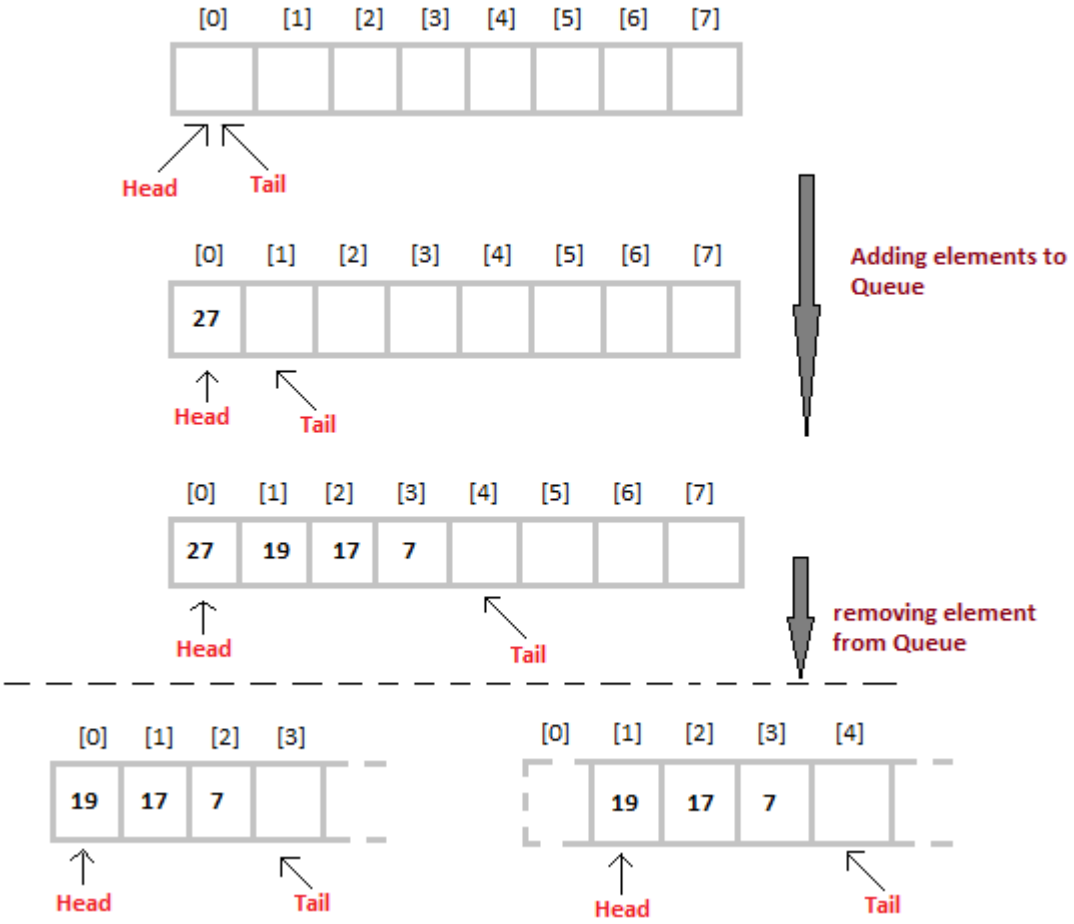
The process to add an element into queue is called **Enqueue** and the process of removal of an element from queue is called **Dequeue**.



enqueue( ) is the operation for adding an element into Queue.

dequeue( ) is the operation for removing an element from Queue .

Queue can be implemented using an Array or Linked List. The easiest way of implementing a queue is by using an Array. Initially the **head**(FRONT) and the **tail**(REAR) of the queue points at the first index of the array (starting the index of array from 0). As we add elements to the queue, the tail keeps on moving ahead, always pointing to the position where the next element will be inserted, while the head remains at the first index.



Applications of queue

Queue, as the name suggests is used whenever we need to have any group of objects in an order in which the first one coming in, also gets out first while the others wait for their turn, like in the following scenarios :

1. Serving requests on a single shared resource, like a printer, CPU task scheduling etc.
2. In real life, Call Center phone systems will use Queues, to hold people calling them in an order, until a service representative is free.
3. Handling of interrupts in real-time systems. The interrupts are handled in the same order as they arrive, First come first served.

Q1 d

Compare & contrast between quick sort and radix sort in terms advantages and disadvantages (5)

Radix-sort is not comparison based, hence can be faster than quick sort. Where as quick sort is comparison based sorting technique.

Radix sort does not compare elements of array but it compares digits at units , tens , hundredth place and so on. The sorting is done by arranging numbers on the basis of individual digits.

Radix sort can be used only if all numbers are positive integers and have equal number of digits. But there is no such limitation in quick sort.

Quick sort used divide and conquer method to sort the array. It partitions the array into left and right parts and then sort each partition independently.

Show an example of Radix sort and quick sort with help of diagram

Q2 a

WAP to implement priority queue. (8)

Q2 b

What different file types. What are various file handling operations of C ? (7)

A file represents a sequence of bytes on the disk where a group of related data is stored. File is created for permanent storage of data. It is a readymade structure. In C language, we use a structure pointer of file type to declare a file. FILE *fp;

Opening a File or Creating a File The fopen() function is used to create a new file or to open an existing file.

General Syntax : *fp = FILE *fopen(const char *filename, const char *mode);

Here filename is the name of the file to be opened and mode specifies the purpose of opening the file. Mode can be of following types, *fp is the FILE pointer (FILE *fp), which will hold the reference to the opened (or created) file.

mode	description
r	opens a text file in reading mode
w	opens or create a text file in writing mode.
a	opens a text file in append mode

Closing a File The fclose() function is used to close an already opened file.

fgetc() is a function to read single character from file.

fputc()is a function to write a single character to the file.

Following program shows how to read contents of a text file and display all the contents on the screen.

```
#include <stdio.h>
void main()
{
File *fp;
charch;
fp = fopen(“abc.txt” , “r”);

while (1)
{
ch = fgetc(fp);
if ( ch == EOF)
break;
else
```

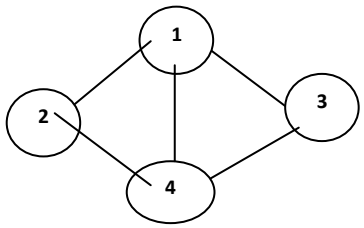
```
printf("%c" , ch);
}

fclose(fp);
}
```

Q2 c

Explain different techniques to represent graph in computer memory (5)

Graph is a data structure which is collection of vertices and edges. Each edge connects two vertices of a graph.

Example graph is shown below



This Graph named G can be defined as follows

G = (V , E)

V = {1,2,3,4}

E = {(1,2) , (1,3) , (1,4) , (2,4) , (3,4)}

Representation of graph G in computer memory

a) Adjacency matrix method

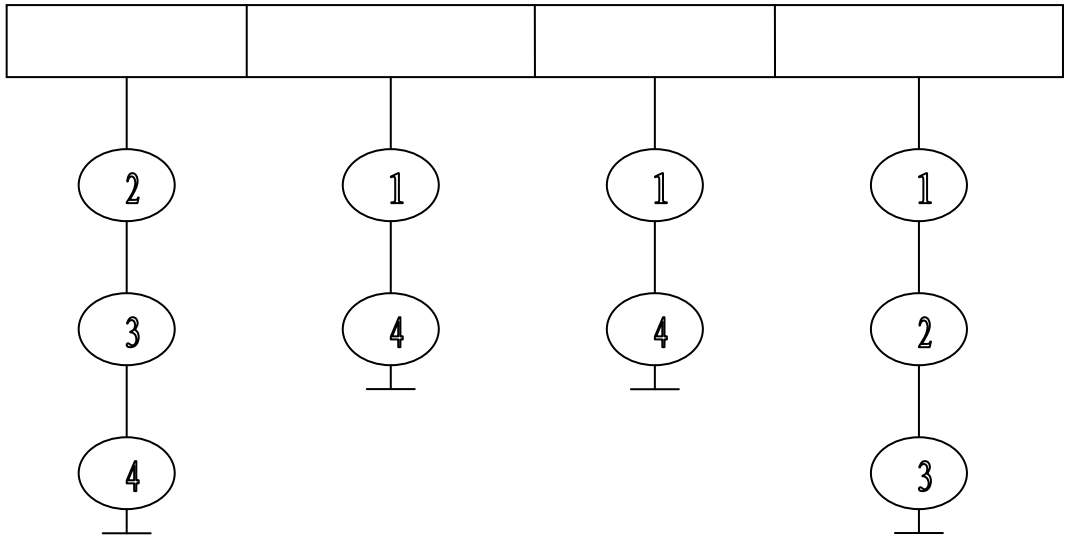
In this method a matrix (2 dimensional array) is declared of size V x V (V = number of vertices of the graph). The entry (i , j) is set to 1 if there is edge from vertex i to vertex j. Otherwise entry (i , j) is set to zero. The adjacency matrix G for the above graph is shown below.

Matrix G

	1	2	3	4
1	0	1	1	1
2	1	0	0	1
3	1	0	0	1
4	1	1	1	0

b) Adjacency List method

In this method a link list is used to store all adjacent vertices of a particular vertex. For a V vertex graph there will be V different link list. adjacency list for the above graph is shown below



Q3 a

Sort the following numbers using heap sort (10)

67 12 89 26 38 45 22 79 53 9 61

As solved in MAY 2015 paper

Q3 b

WAP to create singly link list. The program must have following operations

1. insert a node at beginning
2. insert a node at the end.
3. insert a node at specific position
4. delete a node at a specific position
5. display the list

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

```
struct SLL
{
struct node *first , *last;
};

voidinsertbegin(struct SLL *p , int x)
{
struct node *newrec;
newrec = (struct node *) malloc (sizeof(struct node));
newrec->data = x;
newrec->next = p->first;
if (p->first == NULL)
p->first = p->last = newrec;
else
p->first = newrec;
}
voidinsertend(struct SLL *p , int x)
{
struct node *newrec;
newrec = (struct node *) malloc (sizeof(struct node));
newrec->data = x;
newrec->next = NULL;
p->last->next = newrec;
p->last = newrec;
}

voidinsertatpos(struct SLL *p , int x , intpos)
{
struct node *newrec, *a, *b;
inti;
newrec = (struct node *) malloc (sizeof(struct node));
```

```
newrec->data = x;
  a = b = p->first;
for (i = 1 ; i<= pos-1 ; i++)
  {
   b = a;
   a = a->next;
  }

b->next = newrec;
newrec->next= a;

}


voiddeleteatpos(struct SLL *p , intpos)
{
struct node *a,*b;
inti;
   a = b = p->first;
for (i = 1 ; i<= pos-1 ; i++)
  {
   b = a;
   a = a->next;
  }

b->next = a->next;

free(a);
}
```

```
void display(struct SLL *p)
{
struct node *a;
a = p->first;

while (a != NULL)
{
printf("%d " , a->data);
  a = a->next;
}
}

void main()
{
struct SLL d;
d.first = d.last = NULL;
insertbegin(&d, 10);
insertbegin(&d, 5);
insertbegin(&d, 2);
insertend(20);
insertatpos(&d, 7 , 3);
printf("after inserting \n\n");
display(&d);

deleteatpos(&d , 2);
printf("after deleting\n\n");
display(&d);
}
```

Q4 a

WAP to convert polish notation to reverse polish notation (10)

NOTE : Polish notation means Prefix expression and reverse polish notation means Postfix expression. So we have to write a program which converts prefix to postfix.

Q4 b

Consider following set of numbers (10)

18 25 16 36 8 29 45 12 32 19

Create binary search tree using these numbers. And display the numbers in non decreasing order.

Program for creating binary search tree

```
//program to create binary search tree
#include <stdio.h>
struct node
{
  int data;
  struct node *left, *right;
};

struct tree
{
  struct node *root;

};

  void insert(struct tree *p , int x)
  {
    struct node *newrec;
    newrec  = (struct node *)malloc(sizeof(struct node));
    newrec->data = x;
```

```
   newrec->left = newrec->right = NULL;
  if (p->root == NULL)
  p->root = newrec;
  else
  {
   struct node *a,*b;
   a = b = p->root;

   while (a != NULL)
   {
    b = a;
    if (newrec->data <= a->data)
     a = a->left;
    else
     a = a->right;
   }

    if (newrec->data <= b->data)
     b->left = newrec;
    else
     b->right = newrec;
  }
  }
  void inorder(struct node *p)
  {
   if (p != NULL)
   {
    inorder(p->left);
    printf("%d\n" , p->data);
    inorder(p->right);
   }
  }
void main()
{

   struct tree t;
   t.root = NULL;
   insert(&t , 100);
   insert(&t , 50);
   insert(&t , 200);
```

```
  insert(&t , 150);
  insert(&t , 60);
  insert(&t , 75);

  printf(“data in increasing order \n\n”);
  inorder(t.root);
}
```

Q5 a

Discuss how memory allocation for sparse matrix can be optimised using linked list. Write a C program for the same (15)

```
#include<stdio.h>
#include<conio.h>
#include <stdlib.h>
struct node
{
 int data,row,col;
 struct node *nextrow,*nextcol;
};

struct list
{
 struct node *first,*last;
};

struct list row[10] , col[10];
struct node *newrec,*a;
void main()
{
 int A[10][10], m,n,i,j;
 clrscr();
 printf("\nEnter the order m x n of the sparse matrix\n");
```

```
scanf("%d%d",&m,&n);
printf("\nEnter the elements in the sparse matrix(mostly zeroes)\n");
for(i=0;i<m;i++)
{
 for(j=0;j<n;j++)
 {
  printf("\n%d row and %d column:   ",i,j);
  scanf("%d",&A[i][j]);
 }
}
printf("The given matrix is:\n");
for(i=0;i<m;i++)
{
 for(j=0;j<n;j++)
 {
  printf("%d ",A[i][j]);
 }
 printf("\n");
}

 /* intialize the arrays row and col to NULL */
 for (i = 0 ; i <= 9 ; i++)
 {
   row[i].first = col[i].first = NULL;
   row[i].last = row[i].last = NULL;
 }

/*creating linked list for the sparse matrix */

  for(i=0;i<m;i++)
 {
  for(j=0;j<n;j++)
  {
   if(A[i][j]!=0)
   {
```

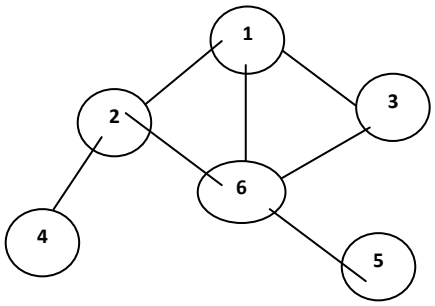
```
    newrec = (struct node *) malloc(sizeof(struct node));
    newrec->data = A[i][j];
    newrec->row = i;
    newrec->col = j;
    newrec->nextrow = newrec->nextcol = NULL;
    if (row[i].first == NULL)
    {
          row[i].first = row[i].last = newrec;
    }
    else
    {
         row[i].last->nextcol = newrec;
         row[i].last = newrec;
    }

    if (col[j].first == NULL)
    {
          col[j].first = col[j].last = newrec;
    }
    else
    {
         col[j].last->nextrow = newrec;
         col[j].last = newrec;
    }
   } /* if */
  } /*for j */
 } /* for i */

 /*now printing linked list in such a way that only non zero numbers will be
printed */
 for (i = 0 ; i <= m-1 ; i++)
 {
   if (row[i].first != NULL)
   {
     a = row[i].first;
```

```
    while (a != NULL)
    {
      printf("data = %d " , a->data);
      printf("row = %d " , a->row);
      printf("col = %d\n\n" , a->col);
      a = a->nextcol;
    }
   }
  }

}/*end main */
```

Q5 b

Write a function for DFS traversal of a graph. Explain its working with example. (5)

Depth first search traversal of a graph



DFS is a method to traverse a graph. basic principle of DFS can be explained using graph shown above.

a) Visit vertex 1

b) Check adjacent vertices of 1 which are 2 , 3 , 6. Take the least among them which is not yet visited. So vertex 2 is visited next.

c) Now check adjacent vertices of 2 which are 4, 6. Take the least adjacent vertex 4 which is not yet visited. So visit vertex 4.

d) Next check adjacent vertices of 4 which is only vertex 2. As vertex 2 is already visited it cannot be revisited. We call vertex 4 as dead end. In such case we return back to that vertex from where we came to vertex 4. So we return to vertex 2 and continue checking adjacent vertices.

Output due to DFS for the above graph is

1 2 4 6 3 5

Algorithm

Comments : Given a graph with V vertices and E edges. This graph is represented by adjacency matrix G. The algorithm uses an array named visited which has V elements. All elements of visited array are initially zero. The dfs algorithm is called from main program as dfs(1) , that is dfs starts processing vertex 1.

```
algorithm dfs(k : vertex)
{
        print vertex k
        visited[k] = 1    // mark vertex k as visited
```

```
        // check adjacent vertices of vertex k
        for  i  = 1  to V
        {
                if (G[k][i] != 0  and visited[i] != 1)
                call dfs(i)
        } //end of for
} //end of algorithm
```

Q6 a
Insert following data in AVL tree (10)
44 17 32 78 50 88 48 62 54

Q6 b
Write a C program for indexed sequential search. (10)

As discussed in MAY 2015 paper