



Data Structure with C

Sub Code: 06CS34

Solved question papers – **Dec 08 / Jan 09 (New Scheme)**

Contributed by: Megha B.R, BMSCE, Bangalore

Reviewed by: Mr. Keshav Murthy, Asst. Professor, BIT
Bangalore

Please send us feedbacks about this solved paper at admin.tb3@enggresources.com

Copyright – tB3

Note: Publishing this paper in any form for commercial use without written permission from tB3 is strictly prohibited, failing which can lead to prosecution under copyright act of Govt. of India.

Part A

1.a Write the output for the following program

04

```
#include<stdio.h>
void main()
{
    int a=7,b=8,*p,*q;
    p=&a;
    q=&b;
    printf("\n%d",++a);
    printf("\n%d",++(*p));
    printf("\n%d",--*(q));
    printf("\n%d",--b);
}
```

soln: The output is:

```
8
9
7
6
```

b. What would be printed from the following block, explain.

08

```
Void main()
{
    int num[5]={3,4,6,2,1};
    int *p=num;
    int *q=num+2;
    int *r=snum[1]; (question mistake: should be taken as int *r=&num[1] )
    printf("\n%d%d",num[2],*(num+2));
    printf("\n%d%d",*p,*(p+1));
    printf("\n%d%d",*q,*(q+1));
    printf("\n%d%d",*r,*(r+1));
}
```

soln: Here, p, q and r are integer pointers. Ptr p points to the starting address of the array num i.e., to the first element of num. Ptr q points to the 3rd element in the array i.e., to element 6. Ptr r points to the 2nd element in the array i.e., 4. From this information we get the following output :

6 6 - num[2] and *(num+2) refer to the 3rd element in the array.
 3 4 - p points to 3. Thus (p+1) points to 4.

6 2 - q points to 6. Hence (q+1) points to 2.
4 6 - r points to 4. Thus (r+1) points to 6.

- c. What do you understand by Dynamic memory allocation? Explain any 3 functions that support dynamic memory allocation. 08

soln: Dynamic memory allocation is the process of allocating memory during execution time (run time). This allocation technique uses predefined functions to allocate and release memory for data during execution time. So if there is an unpredictable storage requirement, then the dynamic memory allocation technique is used.

Three functions that support dynamic memory allocation are:

1. malloc(size) -

This function allows the program to allocate memory explicitly as and when required and the exact amount needed during execution. This function allocates a block of memory, the size of which is the number of bytes specified in the parameter. The syntax is :

```
ptr = (datatype *) malloc (size);
```

On successful allocation, the function returns the address of first byte of allocated memory.

If specified size of memory is not available, the function returns NULL.

2. calloc(n,size)-

This function is used to allocate multiple blocks of memory. The number of blocks is determined by the first parameter n. The size of each block is equal to the number of bytes specified in the parameter ie., size. Thus the total number of bytes allocated is n*size and all bytes will be initialized to 0. The syntax is :

```
ptr = (datatype*)calloc(n,size);
```

Here, n is the number of blocks to be allocated.

size is the number of bytes in each block.

3. realloc(ptr,size)-

Sometimes, the allocated memory may not be sufficient and we may require additional memory space. Also sometimes, the allocated memory may be much larger and we want to reduce the size of allocated memory. In both situations, the size of allocated memory can be changed using realloc() and the process is called reallocation of memory.

```
Ptr = (data_type*)realloc(ptr,size);
```

Here, ptr is the pointer to a block of previously allocated memory.

Size is new size of the block.

- 2 a. Write a function newstrcpy and newstrcat that does the same job as strcat and strcpy without using library function. 06

Soln:

```
void newstrcpy(char str[], char newstr[])
```

```

{
    int i;
    for(i=0 ; str[i] != '\0' ; i++)
        newstr[i] = str[i];
    newstr[i] = '\0';
}

void newstrcat(char str1[], char str2[])
{
    int i, j;
    for(i=0 ; str1[i] != '\0' ; i++) ;
    for(j=0 ; str2[j] != '\0' ; j++)
        str1[i++] = str2[j] ;
    str1[i] = '\0';
}

```

b. Explain the following function with suitable examples.

- i) fseek()
- ii) rewind()
- iii) ftell()

06

Soln: i) fseek() - The function fseek() is used to set the file pointer at the specified position. The file pointer can be moved backwards or forward any number of bytes. The syntax is:

fseek(fp,offset,start_point);

where,

- fp is a file pointer
- offset can take positive, negative or zero value and specifies the number of bytes to be moved from the location specified by start_point
- start_point can take one of the values as shown below :

CONSTANT	VALUE	POSITION OF THE FILE
SEEK_SET	0	Beginning of file
SEEK_CURRENT	1	Current position
SEEK_END	2	End of file

- ii) rewind() - The function rewind() is used to set the file pointer to the beginning of the file. The syntax is :

rewind(fp);

where fp is the file pointer

- iii) ftell() - The function ftell() gives the current position of the file pointer from the beginning of the file. So it always gives the number of bytes from the beginning of the file relative to zero. The syntax is:

ftell(fp);

where fp is the file pointer.

c. List the differences between union and structures. Write a structure student with id, name and marks1, marks2, marks3. Write functions read_data() to read 5 students data print_data() to display the student details.

08

Soln: STRUCTURE

UNION

1. The keyword struct is used to define a structure.	1. The keyword union is used to define a union.
2. The size of the structure is greater or equal to the sum of the sizes of its members.	2. The size of the union is equal to the size of the largest member.
3. Each member within a structure is assigned a unique storage area.	3. Memory allocated is shared by individual members of the union.
4. Individual members can be accessed at a time.	4. Only one member can be accessed at a time.
5. Altering the value of a member will not affect other members of the structure.	5. Altering the value of any of the member will alter other member values.

The required structure is :

```
struct student
{
    int id;
    char name[20];
    int marks1, marks2, marks3;
};
```

The required functions are :

```
void read_data()
{
    struct student s[5];
    int i;
    printf("Enter student id, name and marks in 3 subjects\n");
    for(i=0;i<5;i++)
    {
        printf("Student %d\n",i+1);
```

```

scanf("%d%s%d%d%d", &s[i].id, s[i].name, &s[i].marks1,
&s[i].marks2, &s[i].marks3);
}
}

void print_data(struct student s[])
{
int i;
printf("The student details are :\n");
printf("ID NAME MARKS1 MARKS2 MARKS3\n");
for(i=0;i<5;i++)
printf("%d %s %d %d %d\n",s[i].id, s[i].name, s[i].marks1,
s[i].marks2, s[i].marks3);
}

```

3.a. Transfer each of the following infix expression to its postfix form.

i) $(A+B)*(C\&(D-E)+F)-G$ (assuming $\&$ is \wedge (exponent))

$= T1*(C\&(D-E)+F)-G$	$T1 = A+B = AB+$
$= T1*(C\&T2+F)-G$	$T2 = D-E = DE-$
$= T1*(T3+F)-G$	$T3 = C\&T2 = C T2 \&$
$= T1*T4-G$	$T4 = T3+F = T3 F +$
$= T5-G$	$T5 = T1*T4 = T1 T4 *$

Converting into postfix by repeated substitution

$$\begin{aligned}
&= T5 G - \\
&= T1 T4 * G - \\
&= AB + T3 F + * G - \\
&= AB + C T2 \&F + * G - \\
&= AB + C D E - \& F + * G -
\end{aligned}$$

ii) $(A+B)*(C-D)\&E*F$ (Assuming $\&$ is \wedge (exponent))

$= T1 * (C-D) \& E * F$	$T1 = A+B = AB+$
$= T1 * T2 \& E * F$	$T2 = C-D = CD-$
$= T1 * T3 * F$	$T3 = T2 \& E = T2 E \&$
$= T4 * F$	$T4 = T1 * T3 = T1 T3 *$

Converting into postfix by repeated substitution

$$\begin{aligned}
&= T_4 F * \\
&= T_1 T_3 * F * \\
&= A B + T_2 E \& * F * \\
&= A B + C D - E \& * F *
\end{aligned}$$

iii) $A + (((B - C) * (D - E) + F) / G) \& (H - J)$ (Assuming $\&$ is exponent)

$$\begin{aligned}
&= A + ((T_1 * (D - E) + F) / G) \& (H - J) & T_1 = BC - \\
&= A + ((T_1 * T_2 + F) / G) \& (H - J) & T_2 = DE - \\
&= A + ((T_3 + F) / G) \& (H - J) & T_3 = T_1 T_2 * \\
&= A + (T_4 / G) \& (H - J) & T_4 = T_3 F + \\
&= A + T_5 \& (H - J) & T_5 = T_4 G / \\
&= A + T_5 \& T_6 & T_6 = HJ - \\
&= A + T_7 & T_7 = T_5 T_6 \&
\end{aligned}$$

Converting into postfix by repeated substitution

$$\begin{aligned}
&= A T_7 + \\
&= A T_5 T_6 \& + \\
&= A T_4 G / H J - \& + \\
&= A T_3 F + G / H J - \& + \\
&= A T_1 T_2 * F + G / H J - \& + \\
&= A B C - D E - * F + G / H J - \& +
\end{aligned}$$

b. Show the detailed contents of stack for a given postfix expression $623 + - 382 / + * 2 \& 3 +$ and evaluate the expression.

Soln:

String = $623 + - 382 / + * 2 \& 3 +$

sym=string[i]	OP2=s[top--]	OP1=s[top--]	RES	s[++top]	STACK S
6				6	6
2				2	2 6
3				3	3 2 6
+	3	2	2+3=5	5	5 6
-	5	6	6-5=1	1	1

3				3	3 1
8				8	8 3 1
2				2	2 8 3 1
/	2	8	8/2=4	4	4 3 1
+	4	3	3+4=7	7	7 1
*	7	1	1*7=7	7	7
2				2	2 7
&(exponent)	2	7	7&2=49	49	49
3				3	3 49
+	3	49	49+3=51	51	51

The required result = 51

- c. Write a C function to check whether a string is palindrome or not using stack.

06

Soln:

```
int is_palindrome( char str[] )
{
    int i, top = -1;
    char s[30];        /* Used for stack operations */
    char stk_item;     /* item deleted from the stack */

    /* push all the characters of given string */
    for(i=0;i<strlen(str);i++)
        s[++top] = str[i];

    /* Check whether the string is palindrome or not */
    for(i=0; i<strlen(str); i++)
```

```

    {
        stk_item = s[top--];
        if(str[i] != stk_item) return 0; /*input not equal to stack symbol*/
    }
    return 1;          /* The string is palindrome */
}

```

4.a. What is recursion? Write a recursive function for Binary search. 06

Soln: Recursion is a technique for defining a problem in terms of one or more versions of the same problem. A function, which contains a call to itself or a call to another function, which eventually causes the first function to be called, is known as a recursive function.

```

/* recursive function for binary search */

int bin_search(int a[],int low,int high,int key)
{
    int mid;
    if(low>high) return -1; /* unsuccessful search */
    mid = (low+high)/2;
    if(a[mid]==key)
        return (mid+1);
    if( key<a[mid] )
        return bin_search(a,low,mid-1,key);
    else
        return bin_search(a,mid+1,high,key); }

```

b. What is priority queue? Explain about different types of priority queues. 05

soln:

- The priority queue is a special type of data structure in which items can be inserted or deleted based on the priority.
- Always an element with highest priority is processed before processing any of the lower priority elements.
- If the elements in the queue are of same priority, then the element, which is inserted first into the queue, is processed.
- Priority queues are used in job scheduling algorithms in the design of operating system where the jobs with highest priorities have to be processed first.

The priority queues are classified into 2 groups -

- i) Ascending priority queue : In an ascending priority queue, elements can be inserted in any order. But, while deleting an element from the queue, only the smallest element is removed first.
- ii) Descending priority queue : In descending priority queue, the elements can be

inserted in any order. But, while deleting an element from the queue, only the largest element is deleted first.

c. Write a C program to Simulate the working of circular queue of integers using array. Provide the following operations: i) Insert ii) Delete iii) Display.

09

```
soln: /* circular queue */

#include<stdio.h>
#include<conio.h>
#include<process.h>
#define queue_size 5

void insert_rear(int item, int q[], int *r, int *count)
{
    if(*count == queue_size)
    {
        printf("Overflow of queue\n");
        return;
    }
    *r = ( *r + 1 ) % queue_size;
    q[*r] = item;
    *count += 1;
}

void delete_front(int q[],int *f, int *count)
{
    if(*count == 0)
    {
        printf("Underflow of queue\n");
        return;
    }
    printf("The deleted item is %d\n", q[*f]);
    q[*f] = NULL;
    *f = ( *f + 1 )%queue_size;
    *count -= 1;
    if(*count == 0)
    f=0,r=-1;
}

void display(int q[], int f, int count)
{
    int i,f1;
```

```

if(count==0)
{
    printf("Queue is empty\n");
    return;
}
printf("Contents of queue is:\n");
f1=f;
for(i=1;i<=count;i++)
{
    printf("%d\n",q[f1]);
    f1 = (f1+1)%queue_size;
}
}

void main()
{
    int ch,item,f,r,count,q[20];
    clrscr();
    f=0;
    r=-1;
    count=0;
    while(1)
    {
        printf("1:Insert 2:Delete 3:Display 4:Exit\n");
        printf("Enter your choice\n");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1: printf("Enter the item to be inserted\n");
                    scanf("%d",&item);
                    insert_rear(item,q,&r,&count);
                    break;

            case 2: delete_front(q,&f,&count);
                    break;

            case 3: display(q,f,count);
                    break;

            default: exit(0);
        }
    }
}

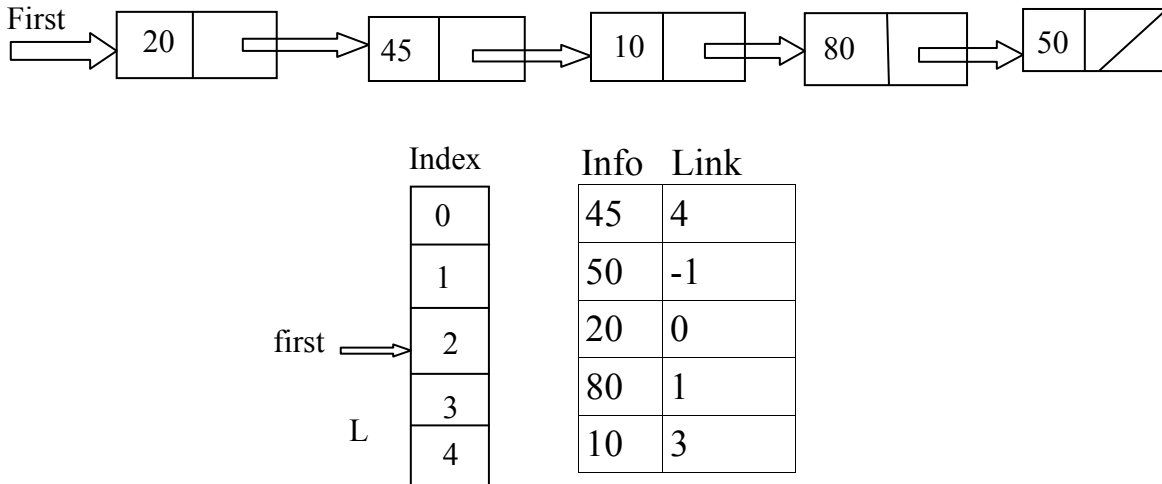
```

PART B

5.a. Explain how the linked list can be represented using arrays.

04

soln: An array is considered to be a collection of nodes. Each node has 2 fields, info and link. The figure below shows how a linked list is represented as an array -



Here the variable first is used as an index variable from which different nodes can be accessed. In the figure the initial value of index variable first is 2. The first item 20 in this location can be accessed using L[2].info. The index of the next item is stored in L[2].link which in this case is 0. Using this index value 0, the next item L[0].info i.e., 45 can be obtained. Proceeding this way all the items 20,45,10,80 and 50 can be accessed.

In general, given an index i, L[i].info gives the actual data and L[i].link gives the index of the next item. If L[i].link has the value -1, the end of list is reached.

An availability list L, which is an array of free nodes can have the following structure :

```
typedef int NODE;
```

```
struct node  
{  
    int info;  
    NODE link;  
};
```

where info field contains the actual information and link field contains the index of the next node.

b. Write a C function to merge two ordered linked list.

06

soln:

```

/* Function to merge two lists */
NODE concat(NODE first , Node second)
{
    NODE cur; /*Holds the address of the last node of first list */

    if(first == NULL)
        return second;

    if(second == NULL)
        return first;

    /* Obtain address of the last node of first list */
    cur = first;
    while(cur->link != NULL)
        cur = cur->link;

    cur->link = second;
    return first;
}

```

- c. Write a C program to perform the operation on stack using singly linked list.

10

soln:

```

#include<stdio.h>
#include<conio.h>
#include<process.h>

struct node
{
    int info;
    struct node *link;
};
typedef struct node* NODE;

NODE getnode()
{
    NODE x;
    x = (NODE)malloc(sizeof(struct node));
    if(x == NULL)
    {
        printf("Out of memory\n");
        exit(0);
    }
    return x;
}

```

```

void freenode(NODE x)
{
    free(x);
}

```

```

NODE insert_front(int item, NODE first)
{
    NODE temp;
    temp = getnode();
    temp->info = item;
    temp->link = first;
    return temp;
}

```

```

void display(NODE first)
{
    NODE temp;
    if(first == NULL)
    {
        printf("List is empty\n");
        return;
    }
    printf("The contents of the stack are\n");
    temp=first;
    while(temp!=NULL)
    {
        printf("%d ",temp->info);
        temp = temp->link;
    }
    printf("\n");
}

```

```

NODE delete_front(NODE first)
{
    NODE temp;
    if(first == NULL)
    {
        printf("List is empty. Cannot delete\n");
        return first;
    }
    temp = first;
    temp = temp->link;
    printf("Item deleted = %d\n",first->info);
    freenode(first);
}

```

```

        return temp;
    }

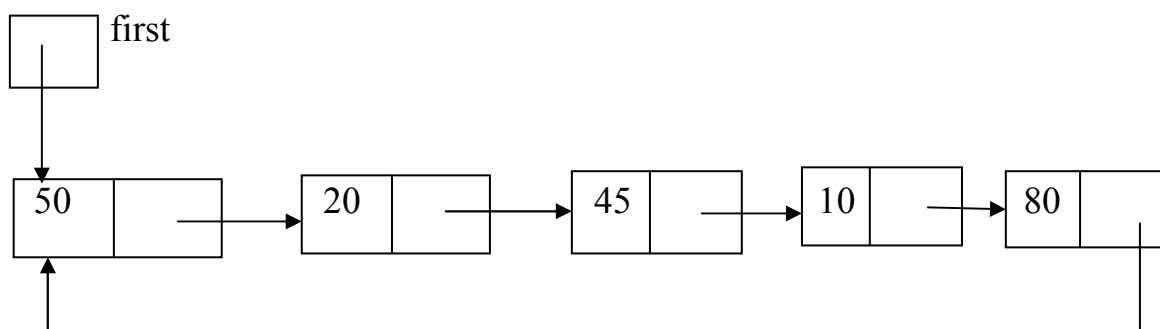
void main()
{
    NODE first = NULL;
    int choice, item;
    clrscr();
    while(1)
    {
        printf("1.Insert front 2.Delete front 3.Display 4.Exit\n");
        printf("Enter the choice\n");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1: printf("Enter the item to be inserted\n");
                    scanf("%d",&item);
                    first = insert_front(item,first);
                    break;
            case 2: first = delete_front(first);
                    break;
            case 3: display(first);
                    break;
            default: exit(0);
        }
    }
}

```

6.a. Explain the following : i)Circular list ii)Doubly linked list using suitable examples.

06

soln: i) Circular list – A circular list is a variation of ordinary linked list in which link field of the last node contains address of the first node. An example of a circular linked list is shown below:



Advantages of circular lists :

- Every node is accessible from a given node by traversing successively using the link field.
- To delete a node cur, the address of the first node is not necessary. Search for the predecessor of node cur can be initiated from cur itself.
- Certain operations on circular lists such as concatenation and splitting of lists etc. Will be more efficient.

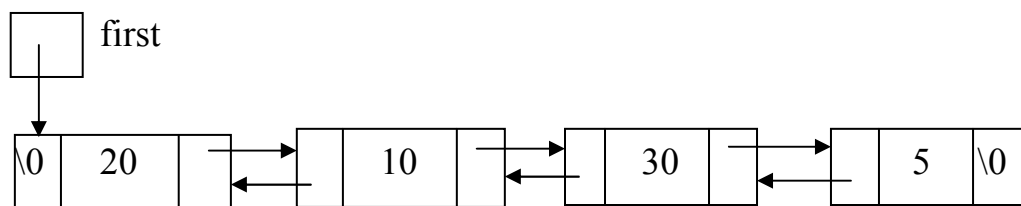
The disadvantage of these circular lists is that if proper care is not taken, it is possible that we may end up in an infinite loop unless we detect the end of the list.

ii) Doubly linked list - A doubly linked list is a linear collection of nodes where each node is divided into three parts :

- info – this is a field where the information has to be stored.
- Link – This is a pointer field which contains address of the left node or previous node in the list.
- Rlink – This is a pointer field which contains address of the right node or next node in the list.

Using such lists, it is possible to traverse the list in forward and backward directions. Such a list where each node has 2 links is also called a two way list.

A doubly linked list is pictorially represented as follows -



A circular doubly linked list is a variation of doubly linked list in which

- rlink of last node contains address of first node.
- Llink of first node contains address of last node.

b. Write a C routine to perform following operations using circular linked list :

- To place the elements of a list in increasing order
- To find the sum of integers and the number of elements in a list.

10

Soln: /* increasing order */

```

void inc_selection_sort(NODE first)
{
    NODE cur , temp;
    int small;

    for( cur = first ; cur->link != first ; cur = cur->link)
    {
        small = cur->info;
        temp = cur;
        for(x = cur->link ; x != first ; x = x->link)
        {
            if( x->info < small)
            {
                small = x->info;
                temp = x;
            }
        }
        temp->info = cur->info;
        cur->info = small;
    }
}

```

ii) /* To find the sum of integers and the number of elements in the circular list*/

```

void req_fn(NODE first)
{
    NODE cur;
    int sum, count;

    if(first == NULL)
    {
        printf(“The list is empty.\n”);
        return;
    }

    sum = first->info;
    count = 1;

    cur = first->link;

    while( cur != first )
    {
        sum += cur->info;
        count++;
    }
}

```

```

    cur = cur->link;
}

printf("The sum of integers = %d\n", sum);
printf("The number of elements in the list = %d\n", count);
}

```

c. What are the advantages and disadvantages of representing group of items as an array versus a linear linked list ? 04

soln: Advantages of representing group of items as an array versus a linear linked list

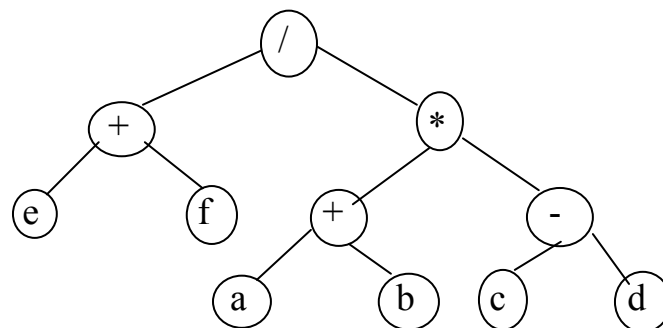
- A node in a linked list may have 2 or more fields and naturally occupies more storage space than the corresponding element in the array.
- Data accessing is very fast and is done by specifying the array name along with the index. Time taken to access any element in the array is the same. But in a linked list, the data at the beginning can be accessed faster. However, to access any other data, the list has to be traversed and hence requires more time.

Disadvantages of representing group of items as an array versus a linear linked list -

- A fixed amount of storage is allocated – The compiler allocates only specified number of memory locations for arrays. If more memory is allocated but using only a small amount of memory results in wastage of memory space. If less memory is allocated when we insert more items, there is a possibility of overflow.
- Items are stored contiguously – Since memory is allocated for an array continuously, if a new element has to be inserted in between, then all the remaining elements have to be shifted by one position.

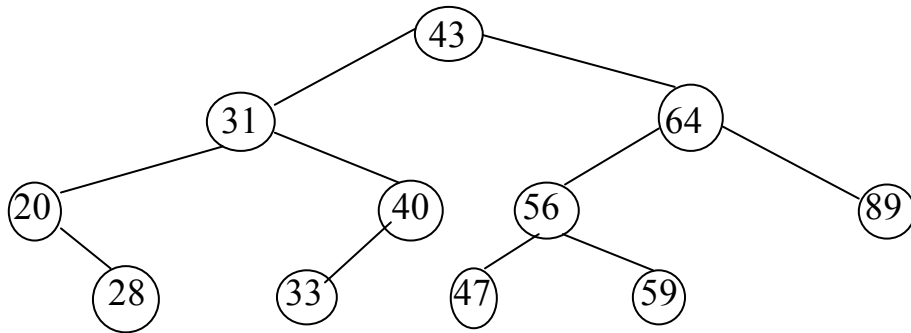
7.a. Write an expression tree for the following postfix expression. $ab+cd-*ef+ /$ 06

soln:



Note: You can also convert to infix and write the tree

b. Write inorder, preorder and postorder traversals for the following tree. 06



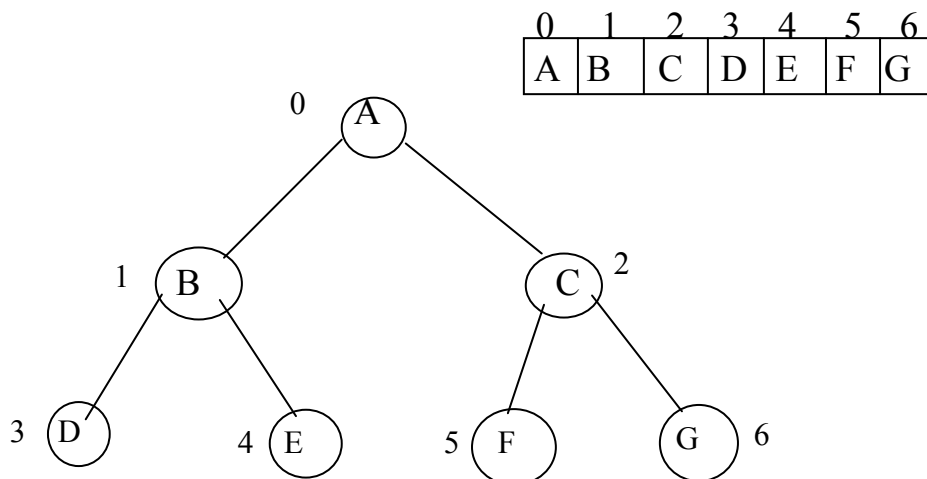
soln: INORDER = left root right
20 28 31 33 40 43 47 56 59 64 89

PREORDER = root left right
43 31 20 28 40 33 64 56 47 59 89

POSTORDER = left right root
28 20 33 40 31 47 59 56 89 64 43

- c. Explain array representation of binary tree and write a function to search a given element in a binary search tree using array representation. 08

soln: A tree can be represented using an array, called sequential representation, as shown below :



We can observe the following points -

- The nodes are numbered sequentially from 0.
- The node with position 0 is considered as the root node.
- Given the position of any node i , $2i+1$ gives the position of the left child and $2i+2$ gives the position of the right child.
- Given the position of any node i , the parent position is given by $(i-1)/2$. If i is

odd, it points to the left child otherwise, it points to the right child.

- In practice, one can initialize each location in the array to 0 indicating the node is not used. Non-zero value in the location indicates the presence of the node.

/* Function to search a given element in a binary search tree using array representation */

```
int search(int a[], int key)
{
    int i = 0;
    while(a[i] != 0)    /* Node is not vacant */
    {
        if (a[i] == key)
        {
            printf("Search successful\n");
            return i;
        }
        if(key > a[i])
            i = 2i+2;    /* right child */
        else
            i = 2i+1;    /* left child */
    }
    printf("Search unsuccessful\n");
    return -1;
}
```

- 8.a. Write a function to: i) Find the maximum element in the binary search tree.
ii) To search an element in the tree 08

```
soln: /* Maximum element in the BST */
NODE maximum( NODE root )
{
    NODE cur;

    if(root == NULL)
        return root;

    cur = root;
    while( cur->rlink != NULL )
    {
        cur = cur->rlink;
    }

    return cur;
}
```

```
/* To search an element in the tree */
```

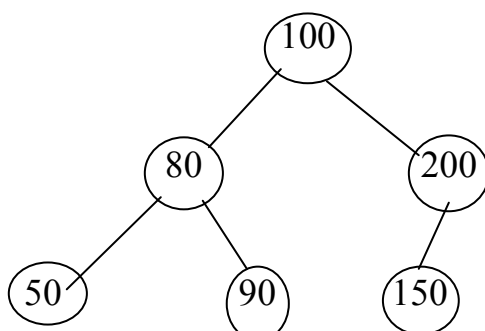
```
NODE iterative_search( int item , NODE root )  
{  
    if(root == NULL)  
        return root;  
  
    while(root != NULL)  
    {  
        if(item == root->info) break;  
  
        if(item < root->info)  
            root = root->llink;  
        else  
            root = root->rlink;  
    }  
  
    if(root == NULL)  
    {  
        printf("item not found\n");  
        return root;  
    }  
  
    printf("Key found\n");  
    return root;  
}
```

b. Explain the following : i) Binary search tree ii) Threaded binary tree iii) Strictly binary tree iv) almost complete binary tree.

08

soln : i) Binary search tree – A binary search tree is a binary tree in which for each node say x in the tree, elements in the left sub-tree are less than info(x) and elements in the right sub-tree are greater than or equal to info(x). Every node in the tree should satisfy this condition, if left sub-tree or right sub-tree exists.

Ex:

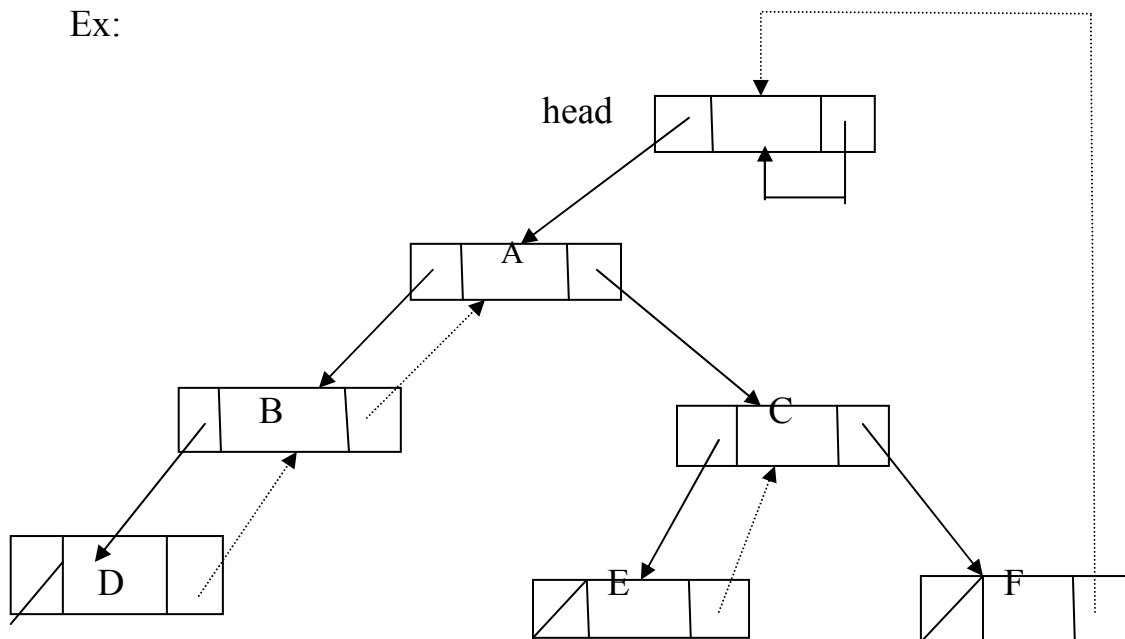


ii) Threaded binary tree – The link fields in a binary tree which contain NULL values can be replaced by address of some nodes in the tree which facilitate upward movement in the tree. These extra links which contain addresses of some nodes are called threads and the tree is termed as threaded binary tree.

There are three types of threaded binary trees -

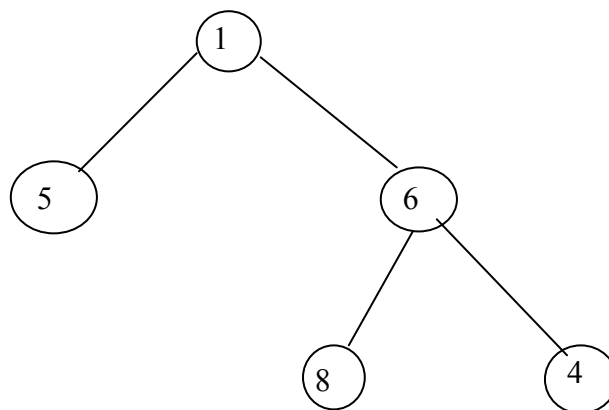
1. In-threaded binary tree.
2. Pos-threaded binary trees.
3. Pre-threaded binary trees.

Ex:



iii) Strictly binary tree – If the outdegree of every node in a tree is either 0 or 2, then the tree is said to be strictly binary i.e., each node can have maximum two children or no children.

Ex:

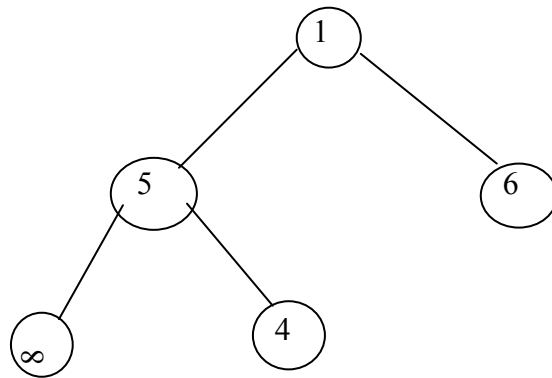


iv) Almost complete binary tree - A tree of depth d is an almost complete binary

tree if following two properties are satisfied -

1. If the tree is complete upto the level $d-1$, then the total number of nodes at the level $d-1$ should be 2^{d-1} .
2. The total number of nodes at level d may be equal to 2^d . If the total number of nodes at level d is less than 2^d , then the number of nodes at level $d-1$ should be 2^{d-1} and in level d the nodes should be present only from left to right.

Ex:



- c. Write a C routine to count the number of nodes in a binary search tree. 04

soln: /* Function to count the number of nodes in a tree */

```
void count_node( NODE root )
{
    if(root == NULL)
        return;

    count_node(root->llink);

    count++;

    count_node(root->rlink);
}
```

Here, the variable count is considered as a global variable and initialized to zero.