Data Structure with C

Sub Code: 06CS34

Solved question papers – **June / July08 (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

DATA STRUCTURES WITH C
JUNE-JULY 2008

MEGHA.B.R
BMSCE
BANGALORE

PART A

1.a.What is a pointer variable? Can we have multiple pointer to a variable? Explain Lvalue and Rvalue expression. 06

soln: A variable which holds the address of another variable is called a pointer variable. In other words, pointer variable is a variable which holds pointer value (i.e., address of a variable.)

```
ex: int *p;    /*declaration of a pointer variable */
    int a=5;
     p = &a;
```

Yes, we can have multiple pointers to a variable.

```
Ex: int *p , *q , *r;
    int a=10;
     p = &a;
      q = &a;
      r = &a;
```

Here, p ,q, r all point to a single variable 'a'.

An object that occurs on the left hand side of the assignment operator is called Lvalue. In other words, Lvalue is defined as an expression or storage location to which a value can be assigned.
Ex: variables such as a,num,ch and an array element such as a[0], a[i], a[i][j] etc.

Rvalue refers to the expression on right hand side of the assignment operator. The R in Rvalue indicates read the value of the expression or read the value of the variable.
Ex: 5, a+2, a[2]+4, a++, --y etc.

b. Give atleast 2 differences between :

i. Static memory allocation and dynamic memory allocation.
ii. Malloc() and calloc(). 04

soln:

Static memory allocation	Dynamic memory allocation
1.Memory is allocated during	1. Memory is allocated during execution

compilation time.	time.
2. Used only when the data size is fixed and known in advance before processing.	2. Used only for unpredictable memory requirement.

malloc()	**calloc()**
1.The syntax of malloc() is: ptr = (data_type*)malloc(size); The required number of bytes to be allocated is specified as argument i.e., size in bytes.	1.The syntax of calloc() is: ptr = (data_type*)calloc(n,size); takes 2 arguments – **n** number of blocks to be allocated and **size** is number of bytes to be allocated for each block.
2. Allocated space will not be initialized.	2. Each byte of allocated space is initialized to zero.

c. i) Write a C program using pass by reference method to swap 2 characters.
ii) Give any 2 advantages and disadvantages of using pointers. 10

soln: i)

```
#include<stdio.h>

void swap(char *a, char *b)
{
        char c;

        c = *a;
        *a = *b;
        *b = c;
}

void main()
{
        char m,n;

        printf("Enter 2 characters\n");
        scanf("%c%c",&m,&n);

        printf("Before swapping\n");
        printf("m = %c, n = %c\n",m,n);

        swap(&m,&n);

        printf("After swapping\n");
```

```
        printf("m = %c, n = %c\n",m,n);
}
```

ii) Advantages of using pointers:

* More than one value can be returned using pointer concept(pass by reference).
* Data accesing is much faster when compared to arrays.

Disadvantages of using pointers;

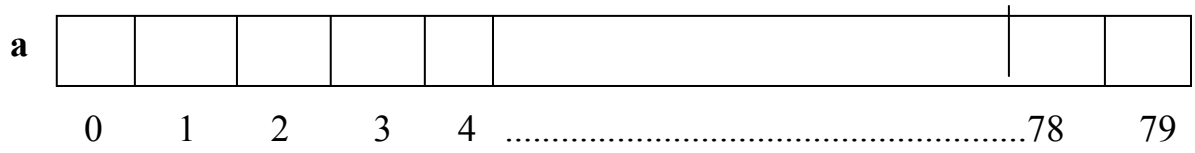
- Un-initialized pointers or pointers containing invalid addresses can cause system crash.
- They are confusing and difficult to understand in the beginning and if they are misused, the result is not predictable.

2.a. How is a string stored in memory? Is there any difference between string and charcter array? Write a C program to copy one string to another, using pointers and without using library functions. 06

soln: A string variable has to be declared before it is used in any of the functions. Consider the following declaration:

```
char a[80];
```

Using this declaration, the compiler allocates 80 memory locations for the variable **a** ranging from 0 to 79 as shown below:



Since the size of character is 1 byte, each character in the array occupies 1 byte. The array thus declared can hold maximum of 80 characters including NULL character.

Basically a string is an array of characters(character array). However a string should be terminated with the '\0'(NULL) character.There is no such rule for character array.

```
#include<stdio.h>
void my_strcpy(char *dest, char *src)
{
        /*Copy the string */
        while(*src != '\0')
                *dest++ = *src++;
```

```
        /* Attach null character at end */
        *dest = '\0';
}
void main()
{
        char src[20] , dest[20];

        printf("Enter some text\n");
        scanf("%s",src);

        my_strcpy(dest,src);

        printf("The copied string = %s\n",dest);
}
```

b. How does a structure differ from an union? Mention any 2 uses of structure. What is a bit field? Why are bit fields used with structures? 07

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.

Two uses of a stucture:

- Structures are used to represent more complex data structures.
- An be used to pass arguments so as to minimize the number of function arguments.
- Extensively used in applications involving database management.

Bit field is defined as a special type of structure or union element that allows the user to access individual bits.

Bit fields can provide greater control over the structure's storage allocation and allow tighter packing of information in memory. Bit-fields are useful for the following reasons:

- Data can be densely packed using bit-fields.
- Some devices will ransmit encoded status information in one or more bits within a byte. Using this bitwise operators, one can see the flags that are set or reset and based on this information, necessary action can be taken by the programmer.
- Ertain encrypted routines sometimes access the bits within a byte.
- Even though bit manipulations can be performed using bitwise operators, using bit-fields the programs will be more structured and we can easily read and understand.

Syntax of bit-field:

```
struct tag
{
        data_type identifier1 : bit_length;
        data_type identifier2 : bit_length;
        .....
        .....
        data_type identifier3 : bit_length;
};
```

where

- data_type is of type int or unsigned int.
- Identifier1, identifier2 etc., are the names of the bit-fields.
- bit_length is the number of bits used for the identifiers.

```
Ex: typedef struct
      {
        char name[30];
        unsigned sex        : 1;
        unsigned age        : 5;
        unsigned rollno     : 7;
        unsigned branch     : 2;
}STUDENT;

STUDENT s[100];
```

c. What is a file pointer? Explain with syntax fopen(), fread() and fwrite() functions.

07

soln: A file pointer fp is a pointer to a structure of type FILE.
FILE *fp;

- fopen() - The file should be opened before reading a file or before writing into a file. The syntax to open a file for either read/write operation is shown below:

```
#include<stdio.h>
FILE *fp;
.....
.....
fp = fopen(char *filename , char *mode);
```

where,
fp is a file pointer of type FILE.
Filename holds the name of the file to be opened.
Mode infoms the library function about the purpose of opening a file.
Mode = r,w,a,r+,w+,a+,rb,wb ....

Return values: The function may return the following-
file pointer of type FILE if successful
NULL if unsuccessful.

Ex: fp = fopen("t.c","rb");

- **fread()** - This function is used to read a block of data from a given file. The prototype of fread() is shown below:

```
int fread(void *ptr, int size , int n , FILE *fp);
```

The parameters passed are:

- fp is a file pointer of an opened file from where the data has to be read.
- Ptr : The data read from the file should be stored in the memory. For this purpose, it is required to allocate the sufficient memory and address of the first byte is stored in ptr.
- N is the number of items to be read from the file.
- Size is the length of each item in bytes.

Ex:
```
#include<stdio.h>
void main()
{
  int a[10];
  ........
  .........
  fread(a,sizeof(int),5,fp);
  .........
```

```
.....
}
```

- **fwrite()** - This function is used to write a block of data into a given file. The prototype of fwrite() is shown below:

```
int fwrite(void *ptr, int size, int n, FILE *fp);
```

where,

- fp is a file pointer of an opened file into which the data has to be written.
- Ptr : ptr points to the buffer containing the data to be written into the file.
- N is the number of bytes to be written into the file.
- Size is the length of each item in bytes.

Ex:
```
#include<stdio.h>
void main()
{
  int a[8] = { 10,20,30,40,50,60,70,80};
  ........
  .........
  fwrite(a,sizeof(int),5,fp);
  .........
  ..........
}
```

3.a. How do you define a data structure? How is stack a data structure? Give a C program to construct a stack of integers and perform all the necessary operations on it. 10

soln: A data structure is a method of storing data in a computer so that it can be used efficiently. The data structures mainly deal with :

- The study of how data is organised in the memory.
- How efficiently can data be stored in the memory.
- How efficiently data can be retrieved and manipulated.
- The possible ways in which different data items are logically related.

A stack is a special type of a data structure where elements are inserted from one end and elements are deleted from the same end. This position from where elements are inserted and from where elements are deleted is called top of stack. Using this approach, the last element inserted is the first element to be deleted out, and hence stack is also called Last In First Out(LIFO) data structure.

The various operations that can be performed on stacks are :

- push – Inserting an element on top of stack

- pop – Deleting an element from the top of stack
- display – Display contents of stack

Since a stack satisfies all the requirements mentioned in the definition of a data structure, stack as a type of data structure.

/* C program to construct a stack of integers */

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

void push( int item, int *top, int s[])
{
        /*check for overflow of stack */
        if(*top == stack_size-1)
        {
                printf("Stack overflow \n");
                return;
        }
        *top = *top+1;
        s[*top] = item;
}

int pop(int *top, int s[])
{
        int item_deleted;

        /*stack underflow */
        if(*top == -1) return 0;

        item_deleted = s[*top];
        *top -= 1;

        return item_deleted;
}

void display(int top, int s[])
{
        int i;

        /* is stack empty */
        if(top==-1)
        {
                printf("Stack is empty\n");
```

```
            return;
        }

        /*display contents of stack */
        printf("Contents of stack are\n");
        for(i=0 ; i<=top ; i++)
            printf("%d\n",s[i]);
}

void main()
{
        int top = -1;
        int s[10];
        int item;
        int item_deleted;
        int choice;
        clrscr();

        while(1)
        {
            printf("1: Push     2: Pop\n");
            printf("3: Display   4: Exit\n");
            printf("Enter your choice\n");
            scanf("%d",&choice);


            switch(choice)
            {
                case 1: printf("Enter the item to be inserted\n");
                        scanf("%d",&item);
                        push(item , &top , s);
                         break;

                case 2: item_deleted = pop(&top , s);
                        if(item_deleted == 0)
                            printf("Stack is empty\n");
                       else
                            printf("Item deleted = %d\n",item_deleted);
                       break;

                case 3: display(top,s);
                        break;

                default: exit(0);
            }
```

```
        }
}
```

b. Write an algorithm to convert a valid infix expression to a postfix expression. Also evaluate the following suffix expression for the values: A=1 B=2 C=3.

AB+C-BA+C$- 10

SOLN:

Symbols	Stack precedence function F	Input precedence funcyion G
+,-	2	1
*,/	4	3
$ or ^	5	6
operands	8	7
(	0	9
)	-	0
#	-1	

Algorithm to convert from infix to postfix:

- As long as the precedence value of the symbol on top of the stack is greater than the precedence value of the current input symbol, poop an item from the stack and place it in the postfix expression. The code for this statement can be of the form:

```
while(F(s[top]) > G(symbol))
{
postfix[j++] = s[top—];
                    }
```

- Once the condition in the while loop is failed, if the precedence of the symbol on top of the stack is not equal to the precedence value of the current input symbol, push the current symbol on to the stack. Otherwise, delete an item from the stack but do not place it in the postfix expression. The code for this statement can be of the form :

```
if(F(s[top]) != G(symbol))
s[++top] = symbol;
else
                    top--;
```

- These 2 steps have to be performed for each input symbol in the infix expression.

AB+C-BA+C$- A=1 B=2 C=3.
= 1 2 + 3 – 2 1 + 3 $ -

sym=string[i]	OP2=s[top--]	OP1=s[top--]	RES	s[++top]	STACK S
1				1	1
2				2	2 1
+	2	1	1+2 = 3	3	3
3				3	3 3
-	3	3	3-3 = 0	0	0
2				2	2 0
1				1	1 2 0
+	1	2	2+1 = 3	3	3 0
3				3	3 3 0
$	3	3	3^3 = 27	27	27 0
-	27	0	0-27 = -27	-27	-27

Required result = -27

4.a. Define recursion. Give atleast 3 differnces between iteration and recursion. 04

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.

ITERATION	RECURSION
1. Uses repitition structures such as for-loop, while-loop or do-while.	1. Uses selection structures such as if-statement, if-else or switch statement .
2. Iteration is terminated when the loop condition fails.	2. Recursion is terminated when base case is satisfied.
3. Iterative functions execute much faster and occupy less memory and can be designed easily.	3. It occupies more stack and most of the time is spent in pushing and popping. Thus recursion is expensive in terms of processor time and memory usage.

b. Write a C program using recursion to find the GCD of 2 numbers. 06

soln:

```
#include<stdio.h>

int gcd(int m , int n)
{
        if(n==0)      return m;
        if(m<n)       return gcd(n,m);
        return gcd(n , m % n);
}
void main()
{
        int m,n,res;

        printf("Enter the value of m and n\n");
        scanf("%d%d",&m,&n);

        res = gcd(m,n);

        printf("GCD(%d,%d) = %d\n",m,n,res);
}
```

c. What is the advantage of circular queue over ordinary queue? Mention any 2 applications of queues. Write an algorithm CQINSERT for static implementation of circular queue. 10

soln: Advantages of circular queue over ordinary queue:

- Rear insertion is denied in an ordinary queue even if room is available at the front end. Thus, even if free memory is available, we cannot access these memory locations. This is the major disadvantage of linear queues.
- In circular queue, the elements of a given queue can be stored efficiently in

an array so as to "wrap around" so that end of the queue is followed by the front of queue.

- This circular representation allows the entire array to store the elements without shifting any data within the queue.

Applications of queues:

- Queues are useful in time sharing systems where many user jobs will be waiting in the system queue for processing. These jobs may request the services of the CPU, main memory or external devices such as printer etc. All these jobs will be given a fixed time for processing and are allowed to use one after the other. This is the case of an ordinary queue where priority is the same for all the jobs and whichever job is submitted first, that job will be processed.
- Priority queues are used in designing CPU schedulers where jobs are dispatched to the CPU based on priority of the job.

Algorithm CQINSERT for static implementation of circular queue -

Step 1 : **Check for overflow :** This is achieved using the following statements-

```
if(count == queue_size)
{
        printf("Queue is full\n");
        return;
}
```

Step 2: **Insert item :** This is achieved by incrementing r by 1 and then inserting as shown below-

```
r = (r+1)%queue_size;
q[r] = item;
```

Step 3: **Update count :** As we insert an element, update count by 1. This is achieved using the following statement-

```
count++;
```

PART B

5.a. List out any two applications of linked list and any two advantages of doubly linked list over singly linked list. 04

soln: Applications of linked list:

- Arithmetic operations can be easily performed on long positive numbers.
- Manipulation and evaluation of polynomials is easy.
- Useful in symbol table construction(Compiler design).

Advantages of doubly linked list over singly linked list:

Singly linked list	Doubly linked list
1. Traversing is only in one direction.	1. Traversing can be done in both directions.
2. While deleting a node, its predecessor is required and can be found only after traversing from the beginning of list.	2. While deleting a node x, its predecessor can be obtained using llink of node x. No need to traverse the list.
3. programs will be lenghty and need more time to design.	3. Using circular linked list with header, efficient and small programs can be written and hence design is easier.

b. Write a C program to simulate an ordinary queue using a 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_rear(int item, NODE first)
{
    NODE temp,cur;

    temp = getnode();
    temp->info = item;
    temp->link = NULL;

    if(first == NULL) return temp;

    cur =first;
    while(cur->link != NULL)
        cur = cur->link;

    cur->link = temp;
    return first;
}

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 display(NODE first)
    {
        NODE temp;
        if(first == NULL)
        {
            printf("List is empty\n");
            return;
```

```
        }
        printf("The contents of the queue are\n");
        temp=first;
        while(temp!=NULL)
        {
                printf("%d ",temp->info);
                temp = temp->link;
        }
        printf("\n");
    }

void main()
    {
        NODE first = NULL;
        int choice, item;
        clrscr();
        while(1)
        {
                printf("1.Insert rear  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_rear(item,first);
                                break;
                        case 2: first = delete_front(first);
                                 break;
                        case 3: display(first);
                                  break;
                        default: exit(0);
                }
        }
    }
```

c. Give an algorithm to insert a node at a specified position for a given singly linked list. 06

soln:

Algorithm insert_pos

{

step 1: create a node and get the item from the user. Let the node be temp.

```
Temp = getnode;
info(temp) = item;
```

step 2: If the list is empty (first == NULL) and given position = 1, return temp;

```
first = temp;
```

step 3: if list is empty and pos != 1, then report invalid position.

Step 4: if pos = 1

```
link(temp) = first
first = temp;
```

step 5: When none of the above cases hold, find the appropriate position:

```
count = 1
prev = NULL
cur = first

while(cur != NULL and count != pos)
{
        prev = cur
        cur = link(cur)
        count = count+1
}
```

step 6: If count == pos,

```
link(prev) = temp
link(temp) = cur
else
report invalid position.
}
```

6.a. Write a C program to perform the following operations on a doubly linked list:

i) To create a list by adding each node at the front.
ii) To display all the elements in the reverse order. 10

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

struct node()
```

```
{
        int info;
        struct node *rlink;
        struct node *llink;
};
typedef struct node *NODE;

NODE create_front(NODE head)
{
        NODE temp,cur;
        int item;
        char ch = 'y';
        while(ch == 'y')
        {
                temp = (NODE)malloc(sizeof(struct node));
                printf("Enter the item\n");
                scanf("%d",&temp->info);

                cur = head->rlink; // obtain address of first node

                head->rlink = temp;
                temp->llink = head;
                temp->rlink  =cur;
                cur->llink = temp;

                printf("Do you want to enter another node?(y/n)");
                ch = getchar();
        }
                return head;
}


void display(NODE head)
{
        NODE temp;

        if(head -> rlink == head)
        {
                printf("List is empty\n");
                return;
        }

        printf("Contents of the list in reverse order are\n");
        temp = head->llink;
```

```
        while(temp != head)
        {
                printf("%d\n",temp->info);
                temp=temp->llink;
        }
}

void main()
{
        NODE head;
        char  choice;
        int item;

        head= (NODE)malloc(sizeof(struct node));
        head->rlink = head;
        head->llink = head;

        head = create(head);

        display(head);
}
```

b. Write a C program to create a linked list and interchange the elements to the list at position m and n and display contents of the list before and after interchanging the elements. 10

soln:

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

struct node()
{
        int info;
        struct node *rlink;
        struct node *llink;
};
typedef struct node *NODE;

NODE create_front(NODE head)
{
        NODE temp,cur;
        int item;
        char ch = 'y';
```

```
        while(ch == 'y')
        {
                temp = (NODE)malloc(sizeof(struct node));
                printf("Enter the item\n");
                scanf("%d",&temp->info);

                cur = head->rlink; // obtain address of first node

                head->rlink = temp;
                temp->llink = head;
                temp->rlink  =cur;
                cur->llink = temp;

                printf("Do you want to enter another node?(y/n)");
                ch = getchar();
        }
                return head;
}

void interchange(NODE head,int m, int n)
{
        NODE first,second,cur;
        int temp,count;

        if(head->rlink == head)
        {
                printf("List is empty\n");
                return;
        }

        count = 1;
        cur = head->rlink;

        while(count!= m)
        {
                cur = cur->rlink;
                count++;
        }

        first = cur; /* first points to the node at position m */

        while(count != n)
        {
                cur = cur->rlink;
                count++;
```

```
        }

        second = cur;        /* second points to the node at position n */


        /* Interchange the elements */
        temp = first->info;
        first->info = second->info;
        second->info = temp;

}


void display(NODE head)
{
        NODE temp;

        if(head -> rlink == head)
        {
                printf("List is empty\n");
                return;
        }

        printf("Contents of the list in reverse order are\n");
        temp = head->rlink;

        while(temp != head)
        {
                printf("%d\n",temp->info);
                temp=temp->rlink;
        }
}

void main()
{
        NODE head;
        int m,n;

        head = (NODE)malloc(sizeof(struct node));
        head->rlink = head;
        head->llink = head;

        head = create_front(head);

        printf("Enter two valid positions in the list m,n such that m<n \n");
```

```
        scanf("%d%d",&m,&n);

        printf("Before swapping\n");

        display(head);

        interchange(head,m,n);

        printf("After swapping\n");
        display(head);
}
```

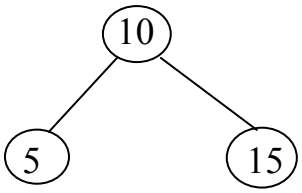
7.a. Define the following:

i) Binary tree
ii) Complete binary tree
iii) Almost complete binary tree
iv) Binary search tree
v) Depth of a tree 10

soln:

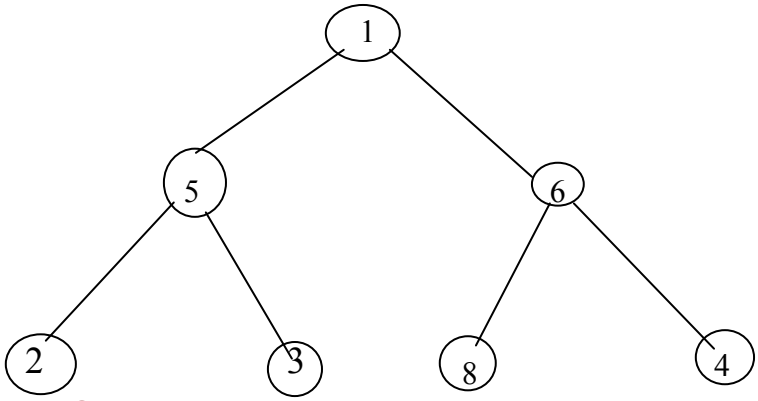
i)Binary tree – A binary tree is a tree which is a collection of zero or more nodes and finite set of directed lines called branches that connect the nodes. A tree can be empty or partitioned into three subgroups, namely :

* Root – If tree is not empty, the first node is called root node.
* Left subtree – It is a tree which is connected to the left of root.
* Right subtree - It is a tree which is connected to the right of root.



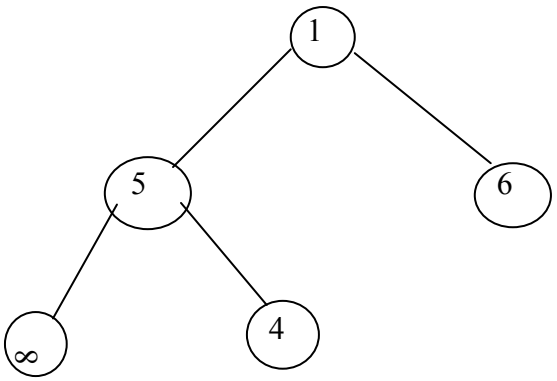
ii)Complete binary tree - A strictly binary tree in which the number of nodes at any level i is 2^i, is said to be a complete binary tree.

Ex:



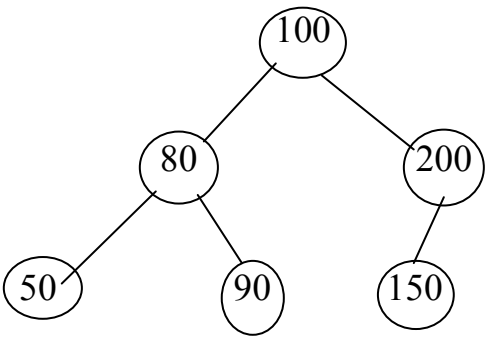
iii) Almost complete binary tree - A tree of depth d is an almost complete binary tree if following two properties are satisfied -
1. If the tree is complete upto the level d-1, then the total number of nodes at the level d-1 shoud be 2^{d-1} .
2. The total number of nodes at level d may be equal to 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:



iv)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.

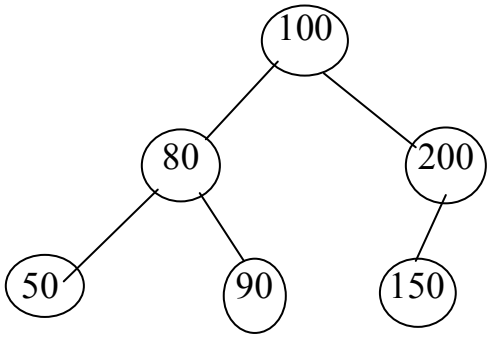
Ex:



v)**Depth of a tree** – Depth of a tree is defined as the length of the longest path from root to a leaf of the tree. Length of a path is the number of branches encountered

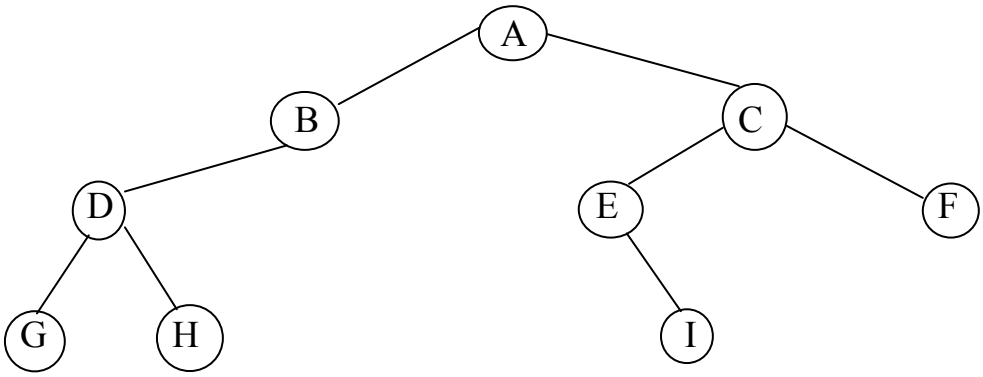
while traversing the path.

Ex:



The depth of the tree = 2

b. Given the following graph, write the inorder, preorder and postorder traversals. 06



SOLN: INORDER = LEFT ROOT RIGHT
= G D H B A E I C F

PREORDER = ROOT LEFT RIGHT
= A B D G H C E I F

POSTORDER = LEFT RIGHT ROOT
= G H D B I E F C A

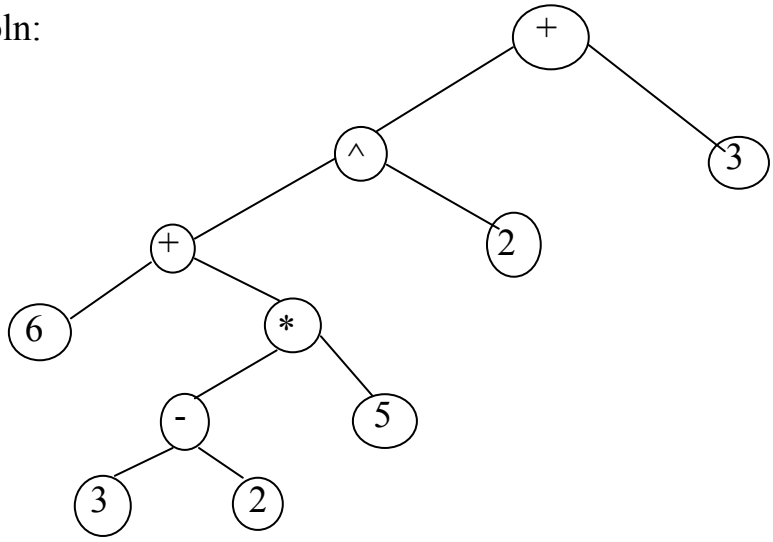
c. In brief describe any 4 applications of trees. 04

soln: Some of the applications of binary search tree are :

- Searching and sorting.
- Manipulation of arithmetic expressions.
- Constructing symbol table.
- The trees are also used in syntax analysis of compiler design and are used to display the structure of a sentence in a language.

8.a. Construct a binary tree for : ((6+(3-2)*5)^2+3) 08

soln:



b. Construct a binary tree from the traversal order given below:

PREORDER = A B D E F C G H L J K

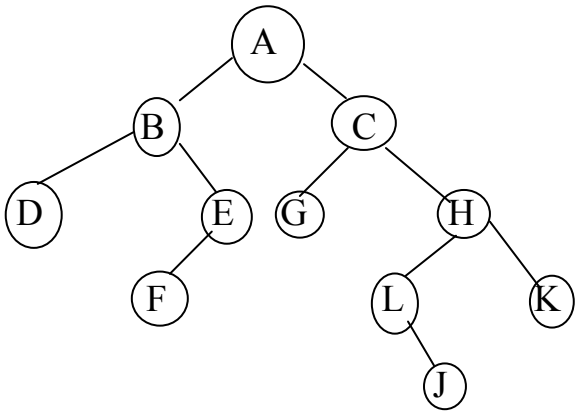
INORDER = D B F E A G C L J H K 08

SOLN:

Here A is root (First in PREORDER) which make the INORDER as

INORDER = |<=LST=>|A|<=RST=>|

Hence the tree can be constructed as:

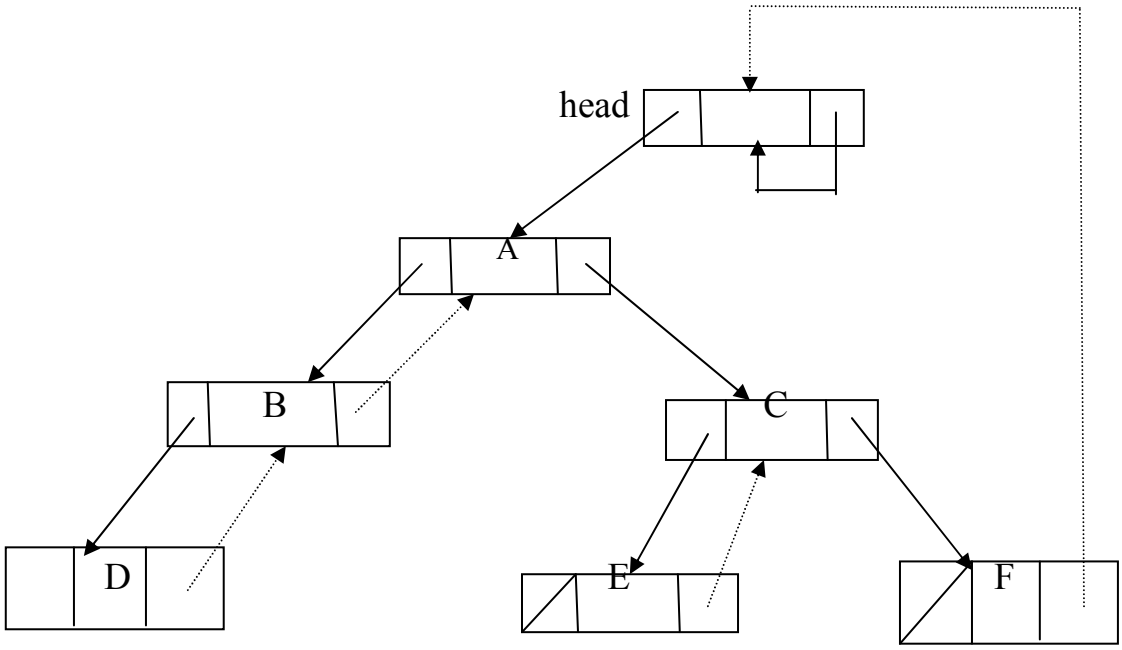


c. What is threaded binary tree? Explain right in and left in threaded binary trees. 04

soln: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.

In-threaded binary tress - In a binary tree, if llink of any node contains null and if it is replaced by address of inorder predecessor, then the resulting tree is called left inthreaded binary tree. If rlink of a node is null and if it is replaced by address of inorder sucessor, the resulting tree is called right inthreaded tree. A inthreaded binary tree is one which is both left and right in-threaded.

EX: Right in-threaded binary tree:



EX: Left in-threaded binary tree: