Unit IV

4 Linked List

4.1 : Introduction to Static and Dynamic Memory Allocation

Q.1 Give the difference of static memory and dynamic memory.

☞[SPPU : June-22, Marks 2]

Ans. :

Sr. No.	Static memory	Dynamic memory
1.	The memory allocation is done at compile time.	Memory allocation is done at dynamic time.
2.	Prior to allocation of memory some fixed amount of it must be decided.	No need to know amount of memory prior to allocation.
3.	Wastage of memory or shortage of memory.	Memory can be allocated as per requirement.
4.	e.g. Array.	e.g. Linked list.

4.2 : Introduction to Linked List

Q.2 What is Linked List ?

Ans. : A linked list is a set of nodes where each node has two fields 'data' and 'link'. The 'data' field stores actual piece of information and 'link' field is used to point to next node. Basically 'link' field is nothing but address only.

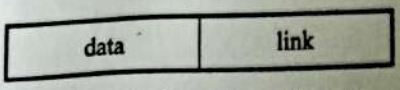


Fig. Q.2.1 Structure of node

Hence link list of integers 10, 20, 30, 40 is

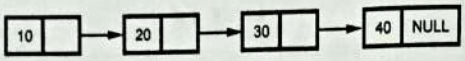
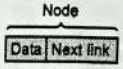
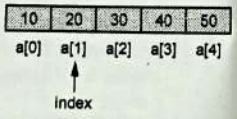


Fig. Q.2.2

Note that the **'link' field of last node consists of NULL which indicates end of list.**

Q.3 Give the difference between Linked list and Arrays.

Ans. :

Sr. No.	Linked list	Array
1.	The linked list is a collection of nodes and each node is having one data field and next link field. For example 	The array is a collection of similar types of data elements. In arrays the data is always stored at some index of the array. For example 
2.	Any element can be accessed by sequential access only.	Any element can be accessed randomly i.e. with the help of index of the array.
3.	Physically the data can be deleted.	Only logical deletion of the data is possible.
4.	Insertions and deletion of data is easy.	Insertions and deletion of data is difficult.

5.	Memory allocation is dynamic. Hence developer can allocate as well as deallocate the memory. And so no wastage of memory is there.	The memory allocation is static. Hence once the fixed amount of size is declared then that much memory is allocated. Therefore there is a chance of either memory wastage or memory shortage.

4.3 : Realization of Linked List using Dynamic Memory Management

Q.4 Explain the new and delete operators of dynamic memory management

Ans. : For performing the linked list operations we need to allocate or deallocate the memory dynamically. The dynamic memory allocation and deallocation can be done using new and delete operators in C++.

The dynamic memory allocation is done using an operator **new**. The syntax of dynamic memory allocation using new is

new data type;

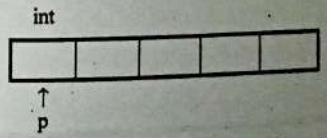
For example :

```
int *p;
p=new int;
```

We can allocate the memory for more than one element. For instance if we want to allocate memory of size in for 5 elements we can declare.

```
int *p;
p=new int[5];
```

In this case, the system dynamically assigns space for five elements of type *int* and returns a pointer to the first element of the sequence, which is assigned to p. Therefore, now, *p* points to a valid block of memory with space for five elements of type *int*.



The program given below allocates the memory for any number of elements and the memory for those many number of elements get deleted at the end of the program.

The memory can be deallocated using the **delete** operator. The syntax is

delete variable_name;

For example

delete p;

4.4 : Linked List as ADT

Q.5 Give an ADT for linked list.

Ans. : In ADT the implementation details are hidden. Hence the ADT will be - .

Abstract DataType List

{

Instances : List is a collection of elements which are arranged in a linear manner.

Operations : Various operations that can be carried out on list are -

1. Insertion : This operation is for insertion of element in the list.
2. Deletion : This operation removed the element from the list.
3. Searching : Based on the value of the key element the desired element can be searched.
4. Modification : The value of the specific element can be changed without changing its location.
5. Display : The list can be displayed in forward or in backward manner.

}

4.5 : Representation of Linked List

Q.6 Give representation of linked list.

Ans. :

```
class node
{
```



```
public:
   int data;
   node *next;
};
class sll
{
private:
   node *head;
public:
   void create();
   void print();
   .
   .
   .
};
```

Data members of list

Operations on list

4.6 : Primitive Operations on Linked List

Q.7 Write a function to insert a node in a linked list.

[SPPU : June-22, Marks 9]

Ans. : There are three possible cases when we want to insert an element in the linked list

1) Insertion of a node as a head node

2) Insertion of a node as a last node

3) Insertion of a node after some node

```
/*
-------------------------------------------------------------
   Function to insert at end
-------------------------------------------------------------
*/
void sll::insert_last()
{
node *New,*temp;
cout<<"\nEnter The element which you want to insert";
cin>>New->data;
```

```
if(head==NULL)
   head=New;
else
{
   temp=head;
   while(temp->next!=NULL)
      temp=temp->next;
   temp->next=New;
   New->next=NULL;
}
}

/*
-----------------------------------------------------------------
     Function to insert after a node
-----------------------------------------------------------------
*/
void sll:: insert_after()
{
 int key;
 node  *temp,*New;
 New= new node;
 cout<<"\n Enter The element which you want to insert";
 cin>>New->data;
 if(head==NULL)
 {
  head=New;
 }
 else
 {
  cout<<"\n Enter The element after which you want to insert the
node";
  cin>>key;
  temp=head;
  do
```



```
{
  if(temp->data==key)
  {
        New->next=temp->next;
        temp->next=New;
        break;

  }
  else
    temp=temp->next;
 }while(temp!=NULL);
}
}
/*
-----------------------------------------------------------------
Function to insert at the beginning
-----------------------------------------------------------------
*/

void sll:: insert_head()
{
 node *New,*temp;
 New=new node;
 cout<<"\n Enter The element which you want to insert";
 cin>>New->data;
 if(head==NULL)
    head=New;
 else
 {
  temp=head;
  New->next=temp;
  head=New;
 }
}
```

Q.8 Write and explain deletion operation of a node from linked list.

Ans. :

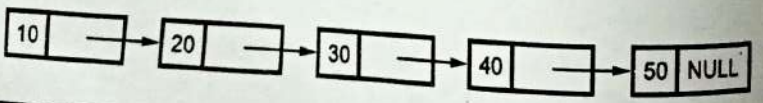
```
void sll:: dele()
{
   node *temp, *prev ;
   int key;
   temp=head;
   clrscr();
   cout<<"\nEnter the data of the node you want to delete: ";
   cin>>key;
   while(temp!=NULL)
   {
    if(temp->data==key)
      break;
    prev=temp;
    temp=temp->next;
  }
  if(temp==NULL)
  cout<<"\nNode not found";
  else
  {
     if(temp==head) //first node
       head=temp->next;
     else
       prev->next=temp->next;  //intermediate or end node
    delete temp;
    cout<<"\nThe Element is deleted\n";
  }
   getch();

}
```

Firstly Node to be deleted is searched in the linked list

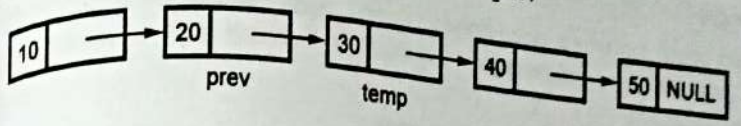
Once the node to be deleted is found then get the previous node of that node in variable **prev**

If we want to delete the head node then set its adjacent node as a new head node and then release the memory

Suppose we have,

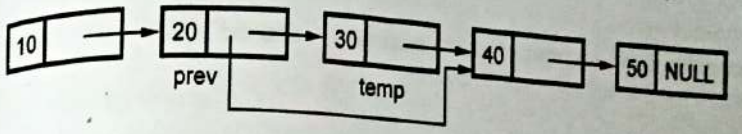


Suppose we want to delete node 30. Then we will search the node containing 30. Mark the node to be deleted as **temp**. Then we will obtain previous node of **temp** Mark previous node as **prev**

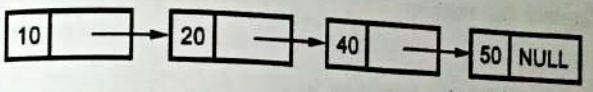


Then,

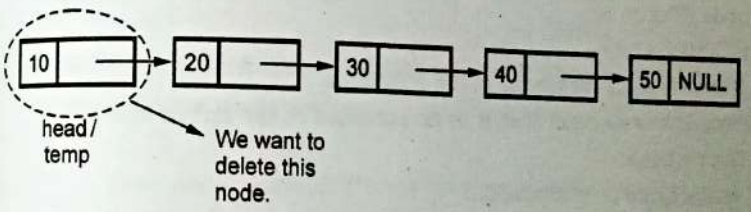
$$prev \rightarrow next = temp \rightarrow next$$



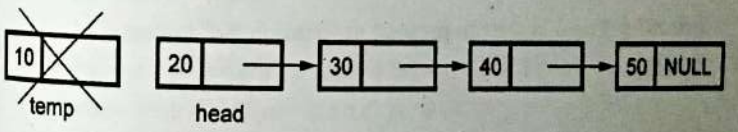
Now we will **delete** the **temp** node. Then the linked list will be



Another case is, if we want to delete a head node then -



This can be done using following statements




```
head = temp → next;
delete temp;
```

Q.9 Write an algorithm to count the number of nodes between given two nodes in a linked list.

Ans. :

```
void count_between_nodes(node *temp1, node *temp2)
{
 int count=0;
 //temp1 represents the starting node
 //temp2 represents the end node
 //we have to count number of nodes between temp1 and temp2
 while(temp1->next!=temp2)
  count++;
 printf("\n Total number of nodes between % and %d is %d ",
temp1->data, temp2->data, count);
}
```

Q.10 Write an algorithm to find the location of an element in the given linked list. Is the binary search will be suitable for this search ? Explain the reason.

Ans. :

```
void search_element(node *head,int key)
 {
  node *Temp;
  int count=1;
  //head is a starting node of the linked list
  //key is the element that is to be searched in the linked list.
  Temp=head;
  while(Temp->next!=NULL)
  {
   if(temp->data==key)
   {
     printf("\n The element is present at location %d",count);
     break;
   }
   count++;
   Temp=Temp->next;
 }
}
```


The binary search can be suitable for this algorithm only if the elements in the linked list are arranged in sorted order. Otherwise this method will not be suitable because sorting the linked list is not efficient as it requires the swapping of the pointers.

Q.11 Write a C++ function to perform the merging of two linked lists.

Ans. :

```
node *merge(node *temp1,node *temp2)
{
   node *prev_node1,*next_node2,*head;
   if(temp1==NULL)
   {
      temp1=temp2;
      head=temp2;
   }
if(temp1->data<temp2->data)
      head=temp1;
else
      head=temp2;
while(temp1!=NULL && temp2!=NULL)
{
/*while data of 1st list is smaller then traverse*/
      while((temp1->data<temp2->data) && (temp1!=NULL))
      {
         prev_node1=temp1; /*store prev node of Sll1 */
         temp1=temp1->next;
      }
      if(temp1==NULL)
         break;
   /*when temp1's data>temp1 it will come out
   Of while loop so adjust links with temp1's prev node*/
      prev_node1->next=temp2;
      next_node2=temp2->next; /*store next node of Sll2*/
      temp2->next=temp1;
      temp1=temp2;
```

```
        temp2=next_node2;
    }
    if(temp1==NULL&&temp2!=NULL) /*attach rem nodes of Sll2*/
    {
        while(temp2!=NULL)
        {
            prev_node1->next=temp2;
            prev_node1=temp2;
            temp2=temp2->next;
        }
    }
    return head;
    }
```

Q.12 Write a program in C++ to return the position of an element X in a list L.

Ans. : The routine is as given below -

```
Return_position(node *head,int key)
{
 /* head represents the starting node of the List*/
 /* key represents the element X in the list*/
 int count=0;
 node *temp;
 temp=head;
 while(temp->data!=key)&&(temp!=NULL)
 {
  temp=temp->next;
  count=count+1;
 }
 if(temp->data==key)
  return count;
else if(temp==NULL)
  return -1; /* -1 indicates that the element X is not present in the list*/
}
```


Q.13 Write a C function to concatenate two singly linked list.

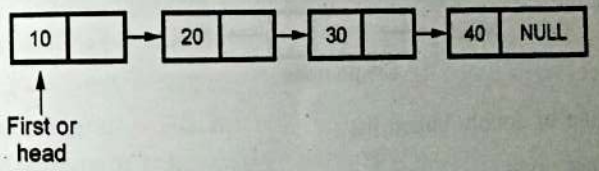
Ans. :

```
void concat(node *head1,node *head2)
{
 node *temp1,*temp2;
 temp1=head1;
 temp2=head2;
 while(temp1->next!=NULL)
    temp1=temp1->next;/*searching end of first list*/
 temp1->next=temp2;/*attaching head of the second list*/
 printf("\n The concatenated list is ...\n");
 temp1=head1;
 while(temp1!=NULL)
 { /*printing the concatenated list*/
    printf(" %d",temp1->Data);
    temp1=temp1->next;
 } }
```

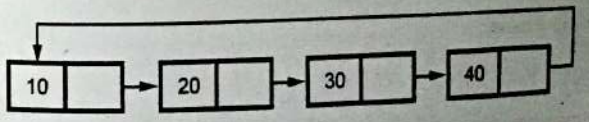
4.7 : Types of Linked List

Q.14 What are the types of linked lists ?

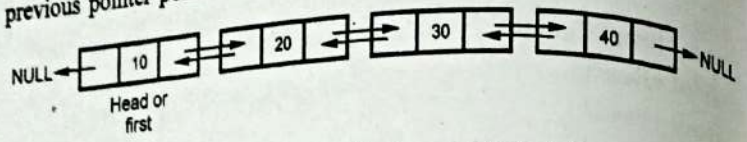
Ans. : 1. Singly linked list : It is called singly linked list because it contains only **one link** which points to the next node. The very first node is called **head** or **first.**



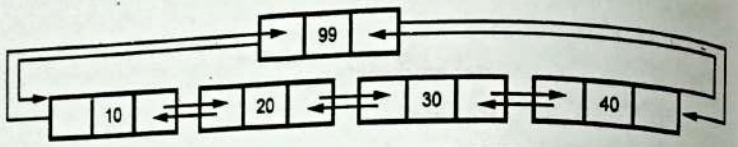
2. Singly circular linked list : In this type of linked list the next link of the last node points to the first node. Thus the overall structure looks circular. Following figure represents this type of linked list.



3. Doubly linear list : In this type of linked list there are two pointers associated with each node. The two pointer are - **next** and **previous.** As the name suggests the next pointer points to the next node and the previous pointer points to the previous node.



4. Doubly circular linked list : In this type of linked list there are two pointers **next** and **previous** to each node and the next pointer of last node points to the first node and previous pointer of the first node points to the last node making the structure circular.



4.8 : Doubly Linked List

Q.15 What is doubly linked list ? Give the node structure of it.

Ans. : The typical structure of each node in doubly linked list is like this.

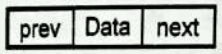
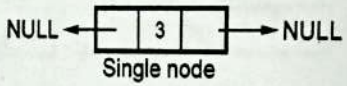


Fig. Q.15.1

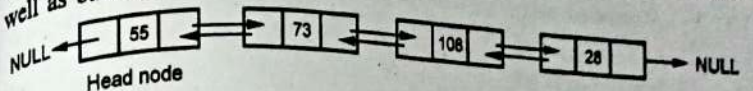


'C++' structure of doubly linked list :

```
typedef struct node
{
   int Data;
   struct node *prev;
   struct node *next;
}
```


The linked representation of a doubly linked list is

Thus the doubly linked list can traverse in both the directions, forward as well as backwards.



Q.16 Write a method to create a doubly linked list.

[SPPU : June-22, Marks 3]

Ans. :

```
void dll :: create()
{
   // Local declarations here
   node  *n1,*last,*temp;
   char ans ='y';
   int flag=0;
   int val;
   do
   {
      cout<<"\nEnter the data :";
      cin>>val;

      // allocate new node
      n1 = new node;
      if ( n1 == NULL )
         cout<<"Unable to allocate memory\n";
      n1 -> Data  = val ;
      n1 -> next = NULL;
      n1 -> prev = NULL;
      if (flag==0) // Executed only for the first time
      {
         temp=n1;
         last=temp;
         flag=1;
      }
      else
```

```
    {
        // last keeps track of the most recently
        // created node

        last->next=n1;
        n1->prev=last;
        last=n1;
    }
    cout<<"\n\nEnter more?";
    ans = getche();
  }while(ans=='y's||ans=='Y');
  cout<<"\nThe List is created\n";
  head=temp;
  getch();
}
```

Q.17 Write C++ function to delete a node from doubly linked list.

☞[SPPU : June-22, Marks 3]

Ans. :

```
void dll:: del()
{
  node *curr, *temp;
  int  data;
  curr=head;
  cout<<"\nEnter the data of the node you want to delete: ";
  cin>>data;
  while(curr!=NULL)
  {
   if(curr->Data==data) //traverse till the required node to delete
   break;               //is found
   curr=curr->next;
}
if(curr==NULL)
cout<<"\nNode not found";
else
{
  if(curr==head)
  {
   if(head->next==NULL&&head->prev==NULL)//only one node
         head=NULL;
   else
```

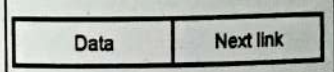
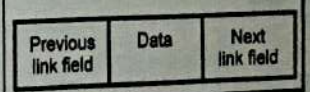


```
   {
   head=curr->next;// first node
   head->prev=NULL;
   }
  }
  else
  {
   temp=curr->prev; //intermediate or end node
   if(curr->next!=NULL)
   (curr->next)->prev=temp;
   temp->next=curr->next;
  }
   delete curr; //free memory
   cout<<"\nThe item is deleted\n";
 }

 getch();

}
```

Q.18 What is the difference between singly and doubly linked list ?

☞ [SPPU : June-22, Marks 3]

Ans. :

Sr. No.	Singly linked list	Doubly linked list
1.	Singly linked list is a collection of nodes and each node is having one data field and next link field. For example : 	Doubly linked list is a collection of nodes and each node is having one data field, one previous link field and one next link field. For example : 

2.	The elements can be accessed using next link.	The elements can be accessed using both previous link as well as next link.
3.	No extra field is required; hence node takes less memory in SLL.	One field is required to store previous link hence node takes more memory in DLL.
4.	Less efficient access to elements.	More efficient access to elements.

Q.19 What are the advantages of doubly linked list over the singly linked list ?

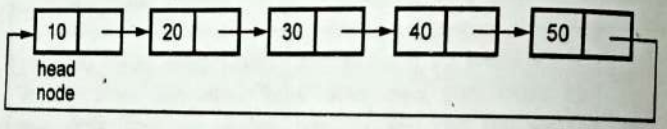
Ans. : • The doubly linked list has two pointer fields. One field is previous link field and another is next link field.

• Because of these two pointer fields we can access any node efficiently whereas in singly linked list only one pointer field is there which stores forward pointer, which makes accessing of any node difficult one.

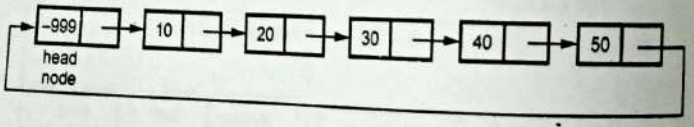
4.9 : Circular Linked List

Q.20 What is circular linked list ?

Ans. : The circular linked list is as shown below -



or



The Circular Linked List (CLL) is similar to singly linked list except that the last node's next pointer points to first node.

Q.21 Write a C++ function to create a circular linked list.

Ans. :

```
void sll ::Create()
{
char ans;
int flag=1;
node *New,*temp;
clrscr();
do
{
  New = new node;
  New->next=NULL;
  cout<<"\n\n\n\tEnter The Element\n";
  cin>>New->data;
  if(flag==1)   /*flag for setting the starting node*/
  {
     head = New;
     New->next=head;
     flag=0;/*reset flag*/
  }
  else          /* find last node in list */
  {
   temp=head;
   while (temp->next != head)/*finding the last node*/
      temp=temp->next; /*temp is a last node*/
   temp->next=New;
   New->next=head; /*each time making the list
                     circular*/
  }
  cout<<"\n Do you want to enter more nodes?(y/n)";
  ans=getch();
  }while(ans=='y'||ans=='Y');
}
```

Single node in the Circular list

Q.22 What is the advantage of circular linked list over singly linked list ?

Ans. : Advantage of circular linked list over singly linked list

- In circular list the next pointer of last node points to head node, whereas in doubly linked list each node has **two pointers** : One previous pointer and another is next pointer.
- The **main advantage** of circular list over doubly linked list is that with the help of single pointer field we can **access head** node quickly.
- Hence some amount of memory get saved because in circular list only one pointer field is reserved.

4.10 : Doubly Circular Linked List

Q.23 What is doubly circular linked list ?

Ans. : The doubly circular linked list can be represented as follows

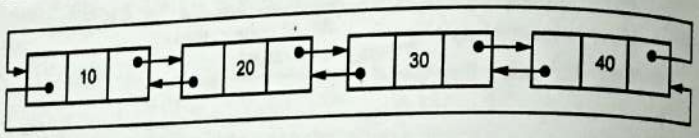


Fig. Q.23.1 Doubly circular list

The node structure will be

```
struct node
{
   int data;
   struct node *next;
   struct node *prev;
};
```

4.11 : Applications of Linked List

Q.24 What are the applications of linked list ?

Ans. : Various applications of linked list are -

1. Linked list can be used to implement linear data structures such as stacks and queues.

2. Linked list is useful for implementing the non-linear data structures, such as tree and graph.
3. Polynomial representation, and operations such as addition, multiplication and evaluation can be performed using linked list.

4.12 : Polynomial Manipulations

Q.25 Explain how to represent a polynomial using linked list ?

Ans. : A polynomial has the main fields as coefficient, exponent in linked list, it will have one more field called **'link'** field to point to next term in the polynomial. If there are n terms in the polynomial then n such nodes has to be created.

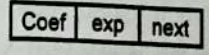
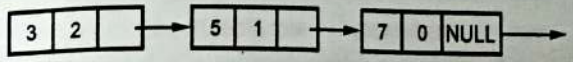


Fig. Q.25.1 Node of polynomial

For example : To represent $3x^2+5x+7$ the link list will be,



In each node, the exponent field will store exponent corresponding to that term, the coefficient field will store coefficient corresponding to that term and the link field will point to next term in the polynomial. Again for simplifying the algorithms such as addition of two polynomials we will assume that the polynomial terms are stored in descending order of exponents.

The node structure for a singly linked list for representing a term of polynomial can be defined as follows :

```
typedef struct Pnode
{
   float coef;
   int exp;
   struct node *next;
} p;
```

Q.26 Write a function for addition of two polynomials. ☞ **[SPPU : June-22, Marks 9]**

Ans. :

```
void Lpadd::add(Lpadd p1,Lpadd p2)
{
  p *temp1, *temp2,*dummy;
  float Coef;
  temp1 =p1.head;
  temp2 =p2.head;
  head = new p;
  if ( head == NULL )
    cout<<"\nMemory can not be allocated";
  dummy = head;//dummy is a start node
  while ( temp1 != NULL && temp2 != NULL )
  {
    if(temp1->exp==temp2->exp)
    {
      Coef = temp1->coef + temp2->coef;
      head = Attach(temp1->exp,Coef,head);
      temp1 = temp1->next;
      temp2 = temp2->next;
    }
    else if(temp1->exp<temp2->exp)
    {
      Coef = temp2 -> coef;
      head =Attach(temp2->exp, Coef,head);
      temp2 = temp2 -> next ;
    }
    else if(temp1->exp>temp2->exp)
    {
      Coef = temp1 -> coef;
      head =Attach(temp1->exp, Coef,head);
      temp1 = temp1 -> next ;
    }
}
```



```
  //copying the contents from first polynomial to the resultant
  //poly
  while ( temp1 != NULL )
  {
    head = Attach(temp1->exp,temp1->coef,head);
    temp1 = temp1 -> next;
  }
  //copying the contents from second polynomial to the resultant
poly.
  while ( temp2 != NULL )
  {
    head = Attach(temp2->exp,temp2->coef,head);
    temp2 = temp2 -> next;
  }
  head ->next = NULL;
  head = dummy -> next;//Now set temp as starting node
  delete dummy;
  return;
}
p *Lpadd::Attach( int Exp, float Coef, p *temp)
{
  p *New, *dummy;

  New = new p;
  if (New == NULL )
    cout<<"\n Memory can not be allocated \n";
  New->exp  = Exp;
  New->coef = Coef;
  New->next = NULL;
  dummy = temp;
  dummy -> next = New;
  dummy = New;
  return(dummy);
}
```

4.13 : Generalized Linked List (GLL)

Q.27 Explain the concept of Generalized Linked List with suitable example.

Ans. : A generalized linked list A, is defined as a finite sequence of n>=0 elements,

$a_1, a_2 a_3, \ldots, a_n$, such that ai are either atoms or the list of atoms. Thus $A = (a_1, a_2 a_3, \ldots, a_n)$

Where n is total number of nodes in the list.

Now to represent such a list of atoms we will have certain assumptions about the node structure

Flag	Data	Down pointer	Next pointer

Flag = 1 means down pointer exists.

= 0 means next pointer exists.

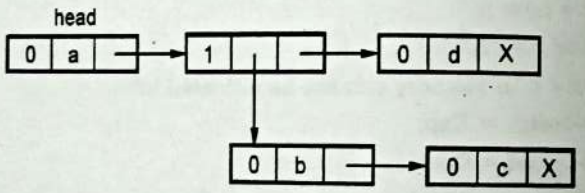
Data means the atom

Down pointer is address of node which is down of the current node.

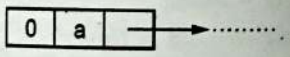
Next pointer is the address of the node which is attached as the next node.

Example of GLL

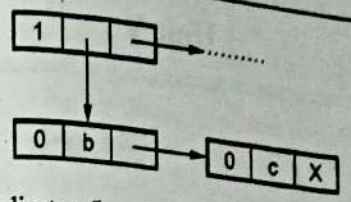
(a , (b , c), d)



In above example the head node is



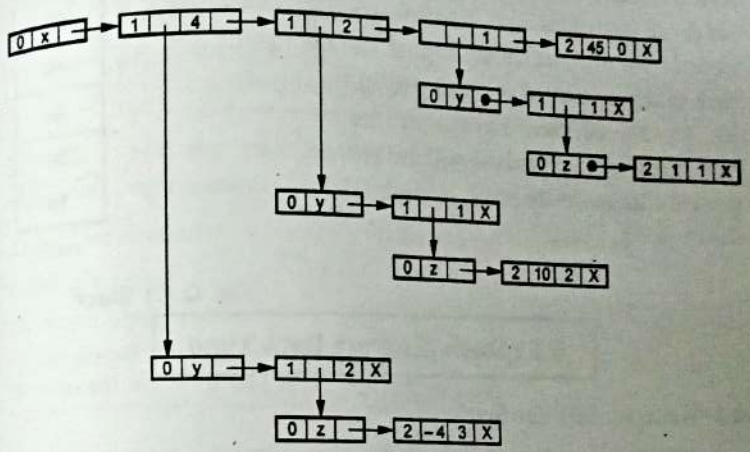
In this case the first field is 0, it indicates that the second field is variable. If first field is 1 means the second field is a down pointer, means some list is starting.



The above figure indicates (b , c). In this way the generalised linked list can be built. The X in the next pointer field indicates NULL value.

Q.28 Consider $-4x^4y^2z^3 + 10x^2yz^2 + 7xyz + 45$ as a polynomial. Represent it as GLL.

Ans. :



END...

Unit V

5 Stack

5.1 : Basic Concept

Q.1 Define the term - Stack.

Ans. : A stack is an **ordered list** in which all insertions and deletions are made at **one end**, called the **top**. If we have to make stack of elements 10, 20, 30, 40, 50, 60 then 10 will be the bottommost element and 60 will be the topmost element in the stack. A stack is shown in Fig. Q.1.1.

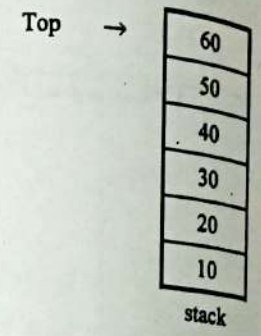


Fig. Q.1.1 Stack

5.2 : Stack Abstract Data Type

Q.2 Write an ADT for Stack.

Ans. : Stack is a data structure which posses LIFO i.e. Last In First Out property. The abstract data type for stack can be as given below.

Abstract DataType stack

{

Instances : Stack is a collection of elements in which insertion and deletion of elements is done by one end called top.

Preconditions :

1. Stfull () : This condition indicates whether the stack is full or not. If the stack is full then we cannot insert the elements in the stack.

2. Stempty () : This condition indicates whether the stack is empty or not. If the stack is empty then we cannot pop or remove any element from the stack.

Operations :

1. Push : By this operation one can push elements onto the stack. Before performing push we must check stfull () condition.
2. Pop : By this operation one can remove the elements from stack. Before popping the elements from stack We should check stempty () condition.

}

5.3 : Representation of Stack using Sequential Organization

Q.3 What are the two methods of representing a stack using sequential organization? Explain them with suitable examples.

Ans. : Declaration 1 :

```
#define size 100
int stack[size], top = -1;
```

In the above declaration stack is nothing but an array of integers. And most recent index of that array will act as a top.

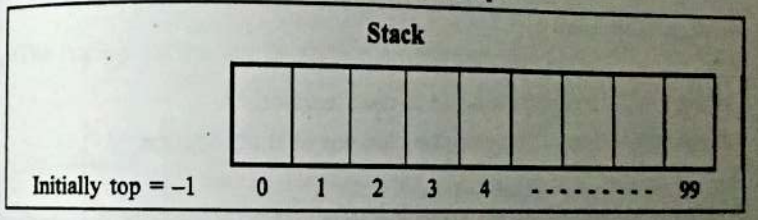


Fig. Q.3.1 Stack using one dimensional array

The stack is of the size 100. As we insert the numbers, the top will get incremented.

Declaration 2 :

```
#define size 10
struct stack {
    int s[size];
```

```
int top;
}st;
```

In the above declaration stack is declared as a structure.

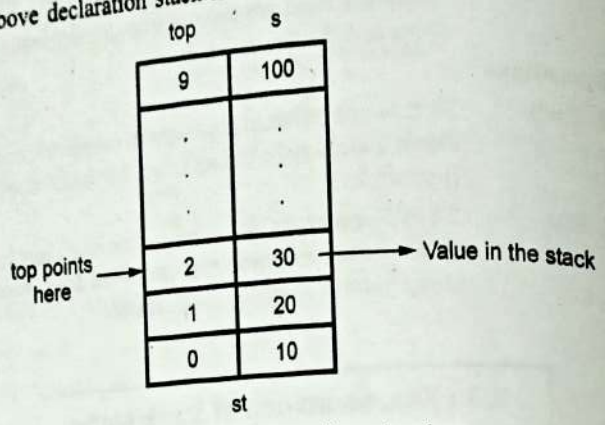


Fig. Q.3.1 Stack using structure

5.4 : Stack Operations

Q.4 Explain the push and pop operations of stack.

☞ **[SPPU : June-22, Marks 8]**

Ans. : 1. Push

Push is a function which inserts new element at the top of the stack. The function is as follows.

```
void push(int item)
{
 st.top++; / * top pointer is set to next location */
 st.s[st.top] =item; /* placing the element at that location */
}
```

The push function takes the parameter **item** which is actually the element which we want to insert into the stack - means we are pushing the element onto the stack.

2. Pop

It deletes the element at the top of the stack. The function pop is as given below -

Note that always top element is to be deleted.


```
int pop()
{
int item;
item = st.s[st.top];
st.top - -;
return(item);
}
```

Push operation can be shown by following Fig. Q.4.1.

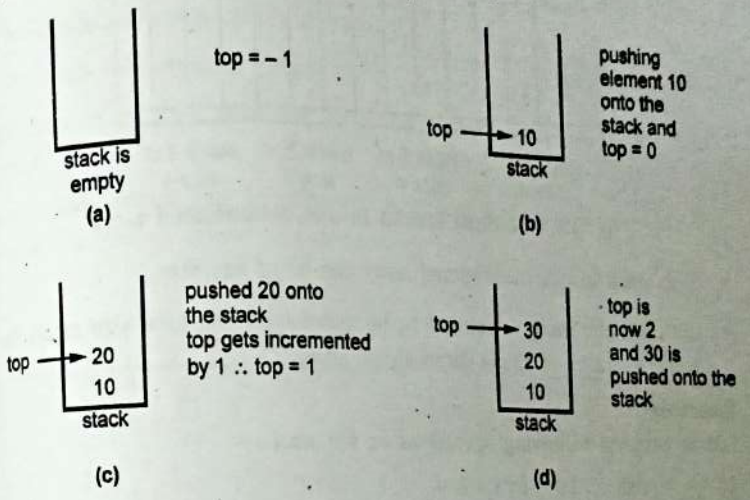


Fig. Q.4.1 Performing push operation

The pop operation can be shown by following Fig. Q.4.2.

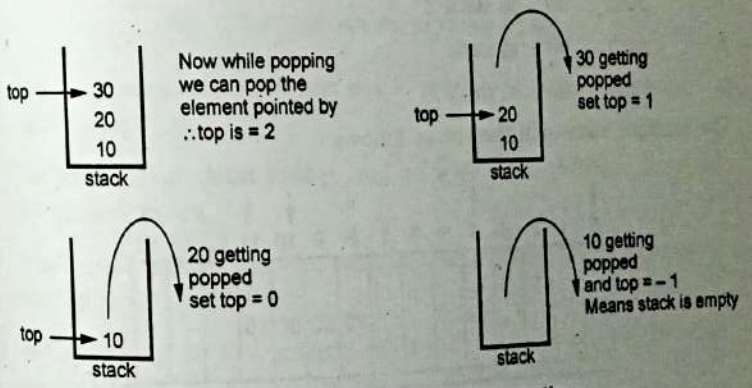


Fig. Q.4.2 Performing pop operation

5.5 : Multiple Stacks

Q.5 Explain the concept of multiple stacks with suitable example.

Ans. : • In a single array any number of stacks can be adjusted. And push and pop operations on each individual stack can be performed.

- The following Fig. Q.5.1 shows how multiple stacks can be stored in a single dimensional array

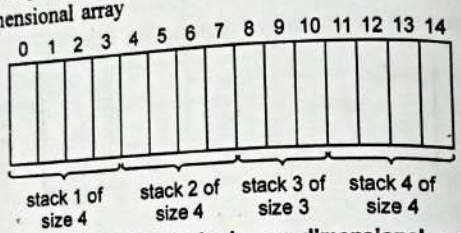


Fig. Q.5.1 Multiple stacks in one dimensional array

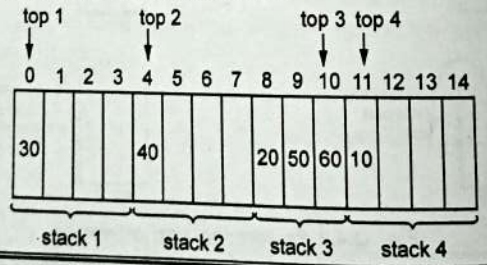
- Each stack in one dimensional array can be of any size.
- The only one thing which has to be maintained that total size of all the stacks < = size of single dimensional array.

Example

Let us perform following operations on the stacks -

1. push 10 in stack 4
2. push 20 in stack 3
3. push 30 in stack 1
4. push 40 in stack 2
5. push 50 in stack 3
6. push 60 in stack 3

The multiple stack will then be as follows -



- Here stack 3 is now full and we can not insert the elements in stack 3.
- The top 1, top 2, top 3, top 4 indicates the various top pointers for stack 1, stack 2, stack 3 and stack 4 respectively.
  - **stack 1**-array [0] will be lower bound and array [3] will be upper bound. That is we can perform stack 1 operation for array [0] to array [3] only.
  - **stack 2**-The area for stack 2 will be from array [4] to array [7]
  - **stack 3**-From array [8] to array [10]
  - **stack 4**-From array [11] to array [14]

Q.6 Give a pseudo code for implementing two stacks in a single array.

Ans. : Two stacks can be adjusted in a single array with n numbers, which is as shown below

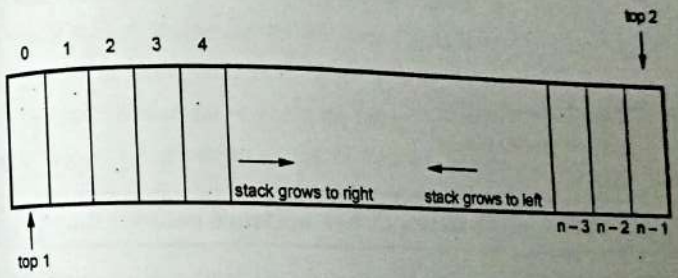


Fig. Q.6.1 Two stacks in single array

- One stack starts at the leftmost end of array and other at the rightmost end of array.
- The insertion in stack1 moves top1 to right while insertion in stack2 moves **top2** to left.
- When stack is full both the **top1** and **top2** positions are adjacent to each other.
- The **advantage** of this arrangement is that every location of the array can be utilized.

- The implementation routines are -

```
# define MAX 80
int stack[MAX];
int top1=-1, top2=n;
void push (int item, int stackno)
{
   if(stackno == 1)
   {
   /* pushing in stack1 */
      if(top1+1 == top2)

         cout<<"stack1 is full";
      top1++;
      stack [top1]=item;
   }
   else
   {
      if (top2-1 == top1)
      cout<<"stack2 is full";
      top2--
      stack[top2]=item;
   }
int pop (int stackno)
{
   int item;
   if (stackno == 1)
   {
      if (top1 == - 1)
         cout<<"stack1 is empty";
      item = stack [top 1];
      top1++
      return(item);
   }
   else
```



```
   {
      if (top2 == MAX)
         cout<<"stack2 is empty";
      item = stack [top2];
      top2--
      return(item);

   }
}
```

5.6 : Applications of Stack

Q.7 Enlist the applications of stack.

Ans. : Various applications of stack are -

1. Stack is used for converting one form of expression to another.
2. Stack is also useful for evaluating the expression.
3. Stack is used for parsing the well formed parenthesis.
4. Decimal to binary conversion can be done by using stack.
5. The stack is used for reversing the string.
6. In recursive routines for storing the function calls the stack is used.

5.7 : Polish Notation and Expression Conversion

Q.8 Write an algorithm for conversion of infix to postfix expression.

Ans. : The algorithm is as follows -

Read an expression from left to right each character one by one

1. If an operand is encountered then store it in postfix array.
2. If '(' is read, then simply push it onto the stack. Because the (has highest priority when read as an input.
3. If ')' is reads, then pop all the operators until (is read. Discard (. Store the popped characters in the postfix array.
4. If operator is read then
   1. If instack operator has greatest precedence (or equal to) over the incoming operator then pop the operator and add it to postfix

expression(postfix array). Repeat this step until we get the instack operator of higher priority than the current incoming operator. Finally push the incoming operator onto the stack.

2. Else push the operator.

5. If stack is not empty then pop all the operators and store in postfix array.

6. Finally print the array of postfix expression.

Q.9 Write a function to convert an infix to postfix expression.

☞ [SPPU : June-22, Marks 9]

Ans. :

```
void postfix()
{
  int i,j=0;
  char ch,next_ch;
  for(i=0;i<strlen(in);i++)
  {
    ch= in[i];
    switch(ch)
    {
      case '(': push(ch);
          break;
      case ')':while((next_ch=pop())!='(')
            post[j++] = next_ch;
          break;
      case '+':
      case '-':
      case '*':
      case '/':
      case
    '^':while(!stempty()&&(precedence(stk[top])>=precedence(ch)))
            post[j++]=pop();
          push(ch);//else part
          break;
     default:post[j++]=ch;//if operand is encountered simply
     add it to postfix
```



```
    }
  }
  while(!stempty())
    post[j++]=pop();
  post[j]='\0';
  for(i=0;i<strlen(post);i++)
    printf("%c",post[i]);
}
```

Q.10 Give the postfix and prefix expression - (a+b*c)/(x+y/z)

Ans. : Infix to postfix conversion

Input read	Action	Stack	Output
(	Push (	(	
a	Print a	(	a
+	Push +	(+	a
b	Print b	(+	ab
*	Push	(+ *	ab
c	Print c	(+ *	abc
)	Pop *, Print. Pop +, Print Pop (		abc * +
/	Push /	/	abc * +
(	Push (	/ (	abc * +
x	Print x	/ (	abc * + x
+	Push +	/ (+	abc * + x
y	Print y	/ (+	abc * + xy
/	Push /	/ (+ /	abc * + xy
z	Print z	/ (+ /	abc * + xyz
)	Pop /, Print. Pop +, Print Pop (	/	abc * + xyz / +
end of input	Pop /, Print		abc * + xyz / + / is required postfix expression.

Infix to Prefix Conversion

Reverse the input as) z/y + x (/) c * b + a (and then read each character one at a time.

Input read	Action	Stack	Output
)	Push)	)	
z	Print z	)	z
/	Push	) /	z
y	Print y	) /	zy
+	Pop /, Print push +	) +	zy /
x	Print x	) +	zy / x
(	Pop +, Print + Pop)		zy / x +
/	Push /	/	zy / x +
)	Push	/)	zy / x +
c	Print c	/)	zy / x + c
*	Push *	/) *	zy / x + c
b	Print b	/) *	zy / x + c
+	Pop *, Print it push +	/)	zy / x + cb *
a	Print	/) +	zy / x + cb * a
(	Pop +, Print + Pop)	/	zy/ x + cb * a +
end of input	Pop/, Print it	empty	zy / x + cb * a + /
	Reverse the output and print it.		/ + a * bc + x / yz is prefix expression.

Q.11 Convert the following expression into postfix form. Show all the steps and stack content : 4$2*3−3+8/4(1+1)

Ans. :

Input symbol	Action	Stack	Postfix expression
4	Print 4	empty	4
$	Push $	$	4
2	Print 2	$	42
*	POP $, Print it. Push *	*	42 $
3	Print 3	*	42 $ 3
−	POP *, Print it Push −	−	42 $ 3*
3	Print it	−	42 $ 3*3
+	POP −, Print it. Then push +	+	42 $ 3*3 −
8	Print 8	+	42 $ 3*3 − 8
/	Push /	+ /	42 $ 3*3 − 8
4	Print 4	+ /	42 $ 3*3 − 84
/	POP/, Print it Then Push/	+ /	42 $ 3*3 − 84 /
(	Push (	+ / (	42 $ 3*3 − 84 /
1	Print 1	+ / (	42 $ 3*3 − 84 / 1
+	Push +	+ / (+	42 $ 3*3 − 84 / 1
1	Print 1	+ / (+	42 $ 3*3 − 84 / 11
)	POP +, Print it POP (	+ /	42 $ 3*3 − 84 / 11 +
end of input	POP/, Print POP +, Print	empty	42 $ 3*3 − 84 / 11 + / +

The postfix expression is **42$3*3 − 84/11+/+**

5.8 : Postfix Expression Evaluation

Q.12 Write an algorithm for evaluation of postfix expression.

Ans. :

1. Read the postfix expression from left to right.

2. If the input symbol read is an operand then push it on to the stack.
3. If the operator is read POP two operands and perform arithmetic operations if operator is
   + then result = operand 1 + operand 2
   – then result = operand 1 – operand 2
   * then result = operand 1 * operand 2
   / then result = operand 1 / operand 2
4. Push the result onto the stack.
5. Repeat steps 1-4 till the postfix expression is not over.

Q.13 Write a function for evaluation of postfix expression.

Ans. :

```
double EVAL::post(char exp[])
{
   char  ch,*type;
   double  result, val, op1, op2;
   int i;
   st.top = 0;
   i=0;
   ch = exp[i];
   while ( ch !='$' )
   {
      if ( ch >= '0' && ch <= '9')
      type ="operand";

      else if ( ch == '+' || ch == '-'||
       ch == '*'|| ch == '/' ||
       ch == '^' )
      type="operator";
      if( strcmp(type,"operand")==0)/*if the character is operand*/
      {
      val = ch – 48;
      push(val);
      }
```

The characters '0', '1', ... '9' will be converted to their values, so that they will perform arithmetic operation.


```
   else
   if (strcmp(type,"operator")==0)/*if it is operator*/
   {
    op2 = pop();
    op1 = pop(); //popping two operands to perform
                 arithmetic operation
    switch(ch)
    {
    case '+' : result = op1 + op2;
                 break;
    case '-' : result = op1 – op2;
                 break;
    case '*' : result = op1 * op2;
                 break;
    case '/' : result = op1 / op2;
                 break;
    case '^' : result = pow(op1,op2);
                 break;
   }/* switch */
   push(result);
   }
      i++;
      ch=exp[i];
   } /* while */
   result = pop(); /*pop the result*/
   return(result);
}
```

Finally result will be pushed onto the stack.

5.9 : Linked Stack and Operations

Q.14 Explain the linked implementation of stack.

☞ [SPPU : June-22, Marks 4]

Ans. : • The advantage of implementing stack using linked list is that we need not have to worry about the size of the stack.

- Since we are using linked list as many elements we want to insert those many nodes can be created. And the nodes are dynamically getting created so there won't be any stack full condition.
- The typical structure for linked stack can be

```
struct stack
{
   int data;
   struct stack *next;
}node;
```

- Each node consists of data and the next field. Such a node will be inserted in the stack. Following figure represents stack using linked list.

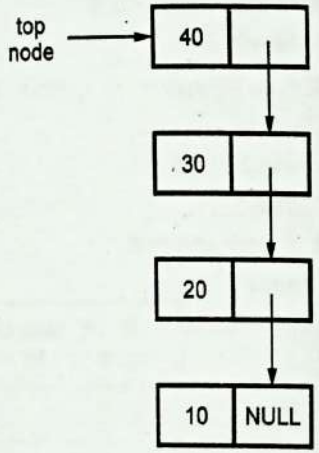


Fig. Q.14.1 Representing the linked stack

Q.15 Write functions for Push and pop operations using the linked implementation. **☞[SPPU : June-22, Marks 4]**

Ans. :

```
/*
   ---------------------------------------------
   The Push function
   ---------------------------------------------
*/
void Lstack::Push(int Item, node **top)
{
```



```
   node *New;
   New = new node;
   New->data=Item;
   New -> next = *top;
   *top = New;
}
/*
   ---------------------------------------------
   The Pop function
   ---------------------------------------------
*/
int Lstack::Pop(node **top)
{
   int item;
   node *temp;
   item = (*top) ->data;
   temp = *top;
   *top = (*top) -> next;
   delete temp;
   return(item);
}
```

5.10 : Recursion-Concept

Q.16 What is recursion ? Also enlist the properties of Recursion.

Ans. : Definition : Recursion is a programming technique in which the function calls itself repeatedly for some input.

By recursion the same task can be performed repeatedly.

Properties of Recursion

Following are two fundamental principles of recursion

1. There must be at least one condition in recursive function which do not involve the call to recursive routine. This condition is called a "way out" of the sequence of recursive calls. The is called base case property.

2. The invoking of **each recursive call must reduce** to some manipulation and must go closer to base case condition.

Q.17 Generate n^{th} term Fibonacci sequence with recursion.

Ans. :

```
#include<iostream>
using namespace std;
int fib(int n)
{
   int x,y;
   if(n<=1)
      return n;
   x=fib(n-1);
   y=fib(n-2);
   return (x+y);
}
int main(void)
{
   int n,num;
   cout<<"\n Enter location in fibonacci series: ";
   cin>>n;
   num=fib(n);
   cout<<"\n The number at "<<n<<" position is "<<num<<" in
   fibonacci series";
   return 0;
}
```

Output

```
Enter location in fibonacci series: 3
The number at 3 position is 2 in fibonacci series
```

Q.18 Explain how stack plays an important role in implementing a recursive function.

Ans. : For illustrating the use of stack in recursive functions, we will consider an example of finding factorial. The recursive routine for obtaining factorial is as given below -

```
int fact (int num)
{
```



```
int a, b;
if (num = = 0)
   return 1 ;
else
{
   a = num – 1 ;
   b = fact (a) ;
   f = a * b;
   return f ;
}
}
/* Call to the function */
ans = fact (4) ;
```

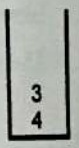
For the above recursive routine we will see the use of stack. For the stack the only principle idea is that – when there is a call to a function the values are pushed onto the stack and at the end of the function or at the return the stack is popped

Suppose we have num = 4. Then in function fact as num ! = 0 else part will be executed. We get

a = num – 1

a = 3

And a recursive call to fact (3) will be given. Thus internal stack stores

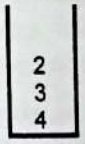


Next fact (2) will be invoked. Then

∵ a = num – 1

a = 3 – 1

b = fact (2)

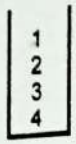


Then,

$\because a = num - 1$

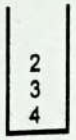
$a = 2 - 1$

$b = fact\ (1)$



Now next as num = 0, we return the value 1.

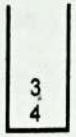
$\therefore a = 1, b = 1$, we return $f = a * b = 1$ from function **fact**. Then 1 will be popped off.



Now the top of the stack is the value of a

Then we get $b = fact\ (2) = 1$

$\therefore f = a * b = 2 * 1 = 2$ will be returned.



Here a = 3

Then $b = fact\ (3)$

$b = 2$

$\therefore f = a * b = 6$ will be returned.

The stack will be

Here a = 4

Then $b = fact\ (4)$

$= 6$

$\therefore f = a * b = 24$ will be returned.

As now stack is empty. We return $f = 24$ from the function **fact**.

Q.19 What direct and indirect recursion ?

☞ [SPPU : June-22, Marks 9]

Ans. : 1. Direct Recursion

Definition : A C function is directly recursive if it contains an explicit call to itself.

For example

```
int fun1(int n)
{
   if(n<=0)
      return n;
   else
      return fun1(n-1);
}
```

Call to itself

2. Indirect Recursion

Definition : A C function is indirectly recursive if function1 calls function2 and function2 ultimately calls function1.

For example

```
int fun1(int n)
{
   if(n<=0)
      return n;
   else
      return fun2(n);
}
int fun2(int m)
{
   return fun1(n-1);
}
```

Q.20 What is recursive tree ? Explain.

Ans. : A tree recursion is a kind of recursion in which the recursive function contains the pending operation that involves another recursive call to the function.

The implementation of **fibonnaci** series is a classic example of tree recursion.

C Function

```
int fib(int n)
{
   if(n==0)
      return 0;
   if(n==1)
      return 1;
   return fib(n-1)+fib(n-2);
}
```

```
fib(4)
= fib(3) + fib(2)
= [fib(2) + fib(1)] + [fib(1) + fib(0)]
= [(fib(1) + fib(0)] + [1] + [1 + 0]
= [1+0] + 1 + 1
fib(4) = 3
```

The call tree can be represented as follows

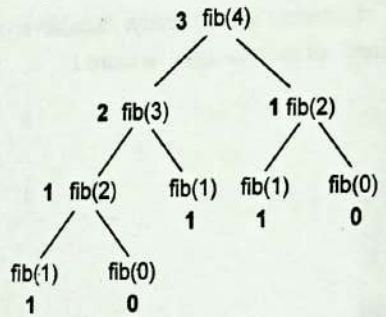


Fig. Q.20.1 Recursive tree

5.11 : Backtracking Algorithmic Strategy

Q.21 What is backtracking ?

Ans. : • Backtracking is a method in which

1. The desired solution is expressible as an n tuple $(x_1, x_2,...x_n)$ where x_i is chosen from some finite set S_i.
2. The solution maximizes or minimizes or satisfies a criterion function C $(x_1, x_2,...x_n)$.

- The basic idea of backtracking is to build up a vector, one component at a time and to test whether the vector being formed has any chance of success.
- The major **advantage** of backtracking algorithm is that we can realize the fact that the partial vector generated does not lead to an optimal solution. In such a situation that vector can be ignored.
- Backtracking algorithm determines the solution by systematically searching the solution space (i.e. set of all feasible solutions) for the given problem.

Q.22 Define the terms - problem state, state space, solution states, Answer states, dead node.

Ans. : Backtracking algorithms determine problem solutions by systematically searching for the solutions using tree structure.

For example -

Consider a 4-queen's problem. It could be stated as "there are 4 queens that can be placed on 4×4 chessboard. Then no two queens can attack each other".

Following Fig. Q.22.1 shows tree organization for this problem.

- Each node in the tree is called **a problem state.**
- All paths from the root to other nodes define the **state space** of the problem.
- The solution states are those problem states s for which the path from root to s defines a **tuple** in the solution space.

  In some trees the leaves define the **solution states.**
- **Answer states :** These are the leaf nodes which correspond to an element in the set of solutions, these are the states which satisfy the implicit constraints.

For example- Refer Fig. Q.22.1 (See Fig. Q.22.1 on next page)

- A node which is been generated and all whose children have not yet been generated is called **live node.**
- The live node whose children are currently being expanded is called **E-node.**

- **A dead node** is a generated node which is not to be expanded further or all of whose children have been generated.

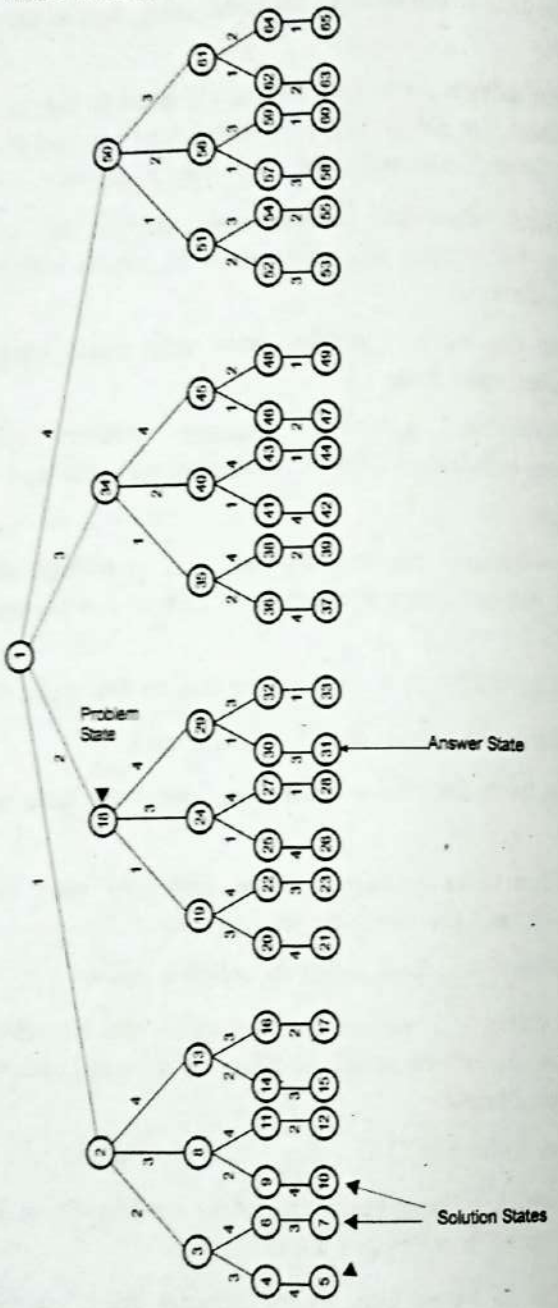


Fig. Q.22.1

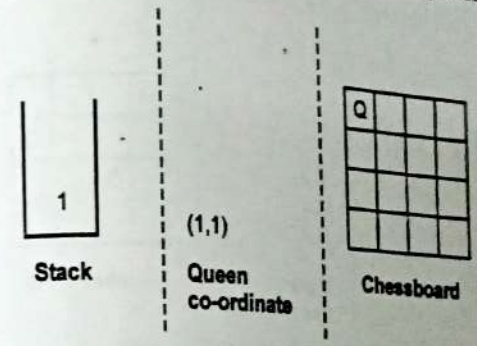
5.12 : Use of Stack In Backtracking

Q.23 With suitable example, explain how stack can be used in backtracking ?

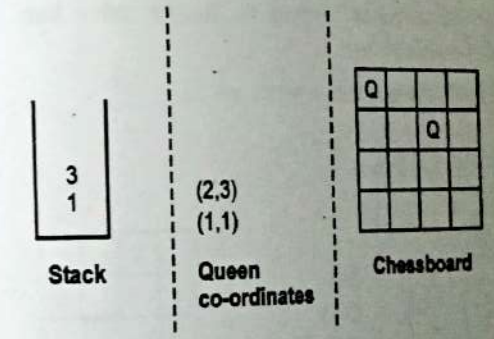
Ans. : The stack can be used in backtracking to handle the recursive procedures.

Let us consider, 4 Queen's problem once again to understand the role of stack in backtracking.

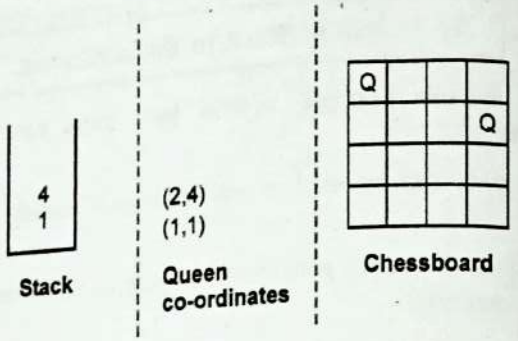
Step 1 : Place queen 1 on 1st row 1st column push only column number onto the stack.



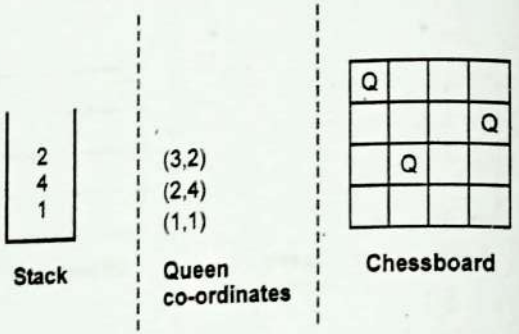
Step 2 : Then place 2 at 2nd row, 3rd column.



Step 3 : This is dead end, because 3rd queen can not be placed in next column as there is no acceptable poistion for queen 3. Hence algorithm backtrack by popping the 2nd queen's position. Now let us place queen 2 at 4th column.

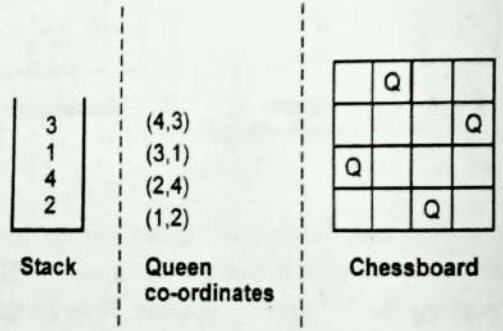


Step 4 : Place 3rd Queen at 2nd column.



Step 5 : Now, there is no valid place for queen 4. So we needed to backtrack all the moves by popping the column index from the stack to try out other possible places.

Step 6 : Finally the possible solution can be.



Algorithm :

1. Start with a queen from first row, first column and search for valid position.
2. If we find a valid position in current row, push the position (i.e. Column number) onto the stack. Then start again on next row.
3. If we don't find a valid position in the current row then we backtrack to previous row i.e. POP the column position for previous row from the stack and search for a valid position.
4. When the stack size is equal to n (for n queens) then that means we have placed n queens on the board. This is a solution to n queen's problem.

END...