Introduction

Data Structure is a collection of elements and various operations that can be performed on these elements based on specific rules. It is classified into Primitive data structures and Non Primitive data structures.

Primitive data structures are fundamental data types such as int, char, float. Non Primitive data structures are built by using the fundamental data types. Non Primitive data structures are classified into linear data structure and Non Linear data structure.

Linear data structures are the data structure in which data is arranged in a straight sequence such as arrays, list. Non Linear data structures are the data structure in which data may be arranged in hierarchical manner such as trees, graphs.

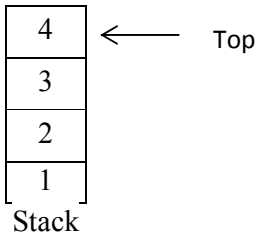
Array: Array is a kind of data structure that can store a collection of elements of same data type.

Ex: Program for reading 3 numbers and displaying them.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[3],i;
clrscr();
printf("Enter the elements\n");
for( i=0;i<3;i++)
scanf("%d",&a[i]);
printf("The elements are\n");
for(i=0;i<3;i++)
printf("%d",a[i]);
getch();
}
```

Stacks: It is an ordered list in which all the insertion and deletion of elements are taken place at one end, called top.

Ex: If we consider the elements 1, 2, 3, 4 in the stack, then they are stored in the stack as shown below.



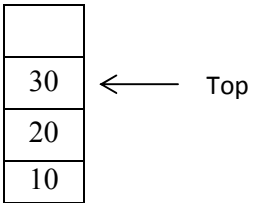
Push and pop are the 2 operations performed on the stack.

Push: By this operation we can insert an element into the stack, but before performing push operation stack full condition has to be checked. If the stack is full we can't insert an element.

Push function:

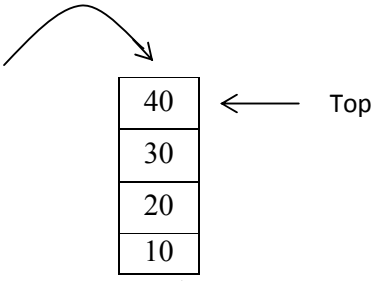
```
void push(int item)
{
if(top==MAX-1)  // Checking whether the stack is full, here MAX is the size of the stack
printf("Stack overflow\n");
else
a[++top]=item;  // Pushing an element to stack if stack is not full
}
```

Ex: If we have a stack of size 4 and we have the following elements 10, 20, 30 in the stack which will be represented as below.



Stack

Now we are performing push operation, inserting the element 40 into the stack. Now the stack will be as below.



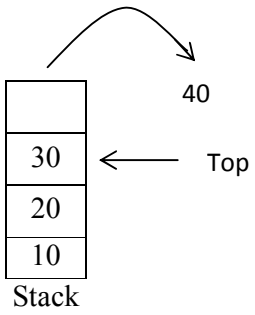
Stack

Pop: By this operation we can delete an element from the stack, but before performing pop operation stack empty condition has to be checked. If the stack is empty nothing will be there to delete.

Pop function:

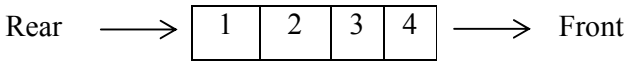
```
int pop()
{
int item;
if(top==-1)  // Checking whether the stack is empty
printf(" Stack underflow");
else
item=a[top--];  // Popping an element from stack is stack is not empty
}
```

Now we are performing pop operation, deleting the top most element of the stack. Now the stack will be as below.



Queue: It is an ordered collection of elements that has 2 ends named as front and rear. The rear end is used for inserting an element and the front end is used for deleting an element.

Ex: If we consider the elements 1, 2, 3, 4 in the queue, then they are stored in the queue as shown below.



Queue

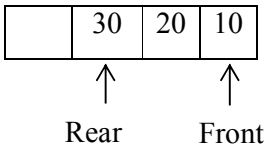
Insert and delete are the two operations performed on the queue.

Insert: Inserting an element into the queue will always takes place at the rear end, but before inserting an element queue full condition has to be checked. If the queue is full we can't insert an element.

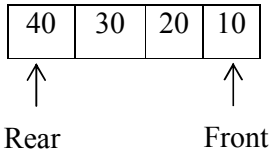
Insert function:

```
void insert( int item)
{
if(rear>=MAX-1)
printf("Queue overflow");
else
{
if(front==-1)
front=0;
else
{
rear++;
a[rear]=item;
}
}
}
```

Ex: If we have a queue of size 4 and we have the following elements 10, 20, 30 in the queue which will be represented as below.



Now inserting element 40 into the queue

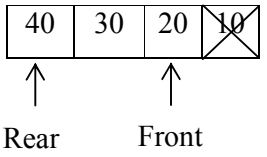


Delete: Deleting an element from the queue will always takes place at the front end, but before deleting an element, queue empty condition has to be checked. If the queue is empty nothing will be there to delete.

Delete function:

```
del()
{
if(front==-1|| front>rear)
printf("Queue underflow");
else
{
item=a[front];
front++;
}
}
```

Now deleting an element from the queue where front is pointing.

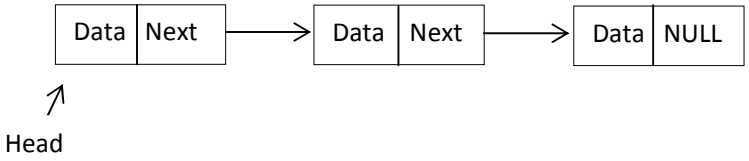


Linked List: In linked list elements are not stored in continuous locations; the elements are linked using pointers.

Types of linked list

- Singly linked list
- Doubly linked list
- Circular linked list

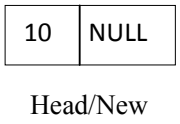
Singly linked list: It contains nodes which have data field and next field (or pointer field). In the data field value will be stored and in the next field address of the next node will be stored.



Inserting an element at front

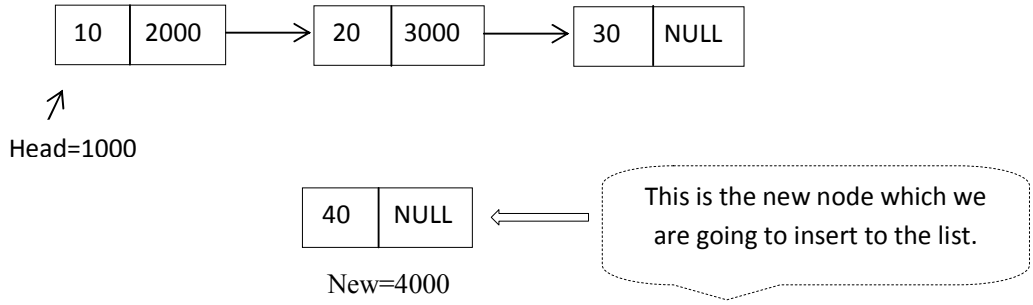
If there is no node in the list then the newly inserted node will be the head node.

Ex:

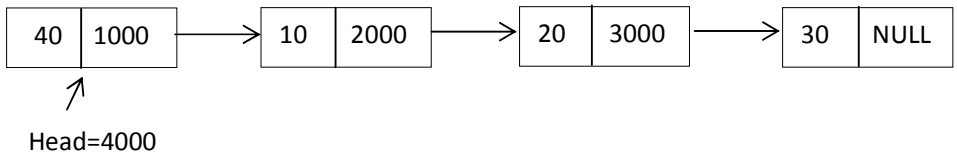


If linked list is already created, consider following is the linked list which already exists.

Ex:



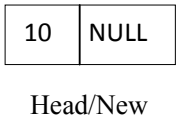
After inserting the new node the list is as below.



Inserting an element at end

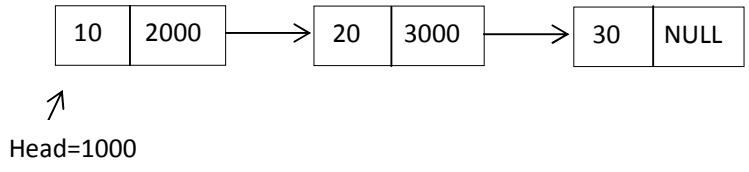
If there is no node in the list then the newly inserted node will be the head node.

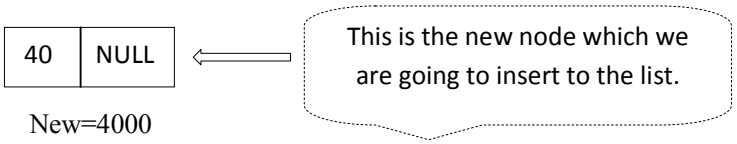
Ex:



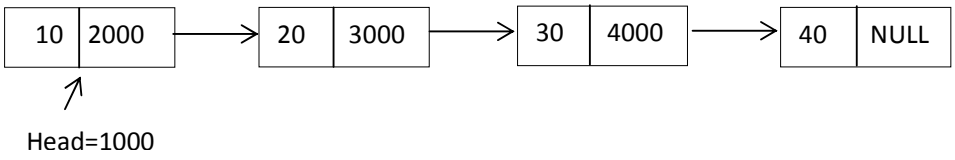
If linked list is already created, consider following is the linked list which already exists.

Ex:





After inserting the new node the list is as below.

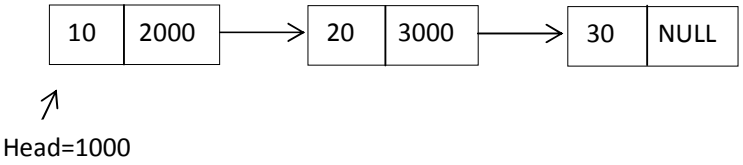


Deleting an element at front

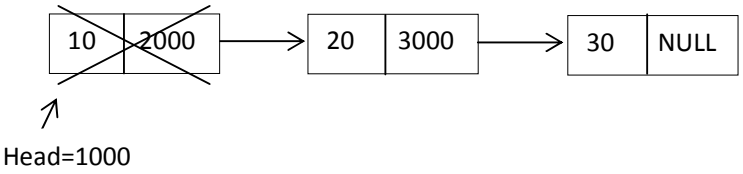
If there is no node in the list then nothing will be there to delete.

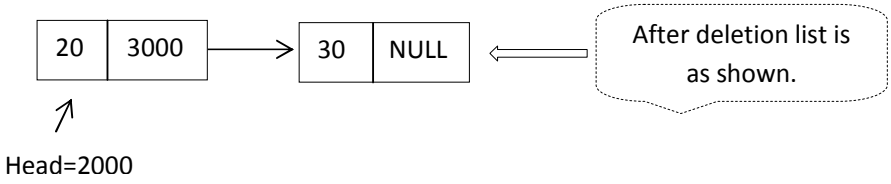
Consider following is the linked list which already exists.

Ex:



In the above list we are going to delete the head node.

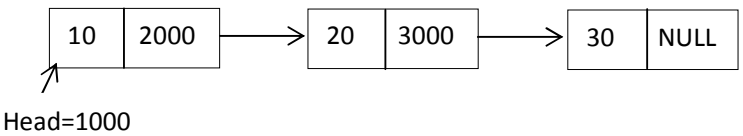




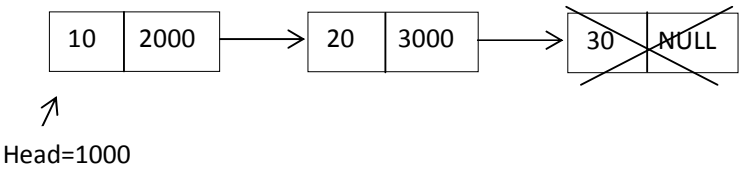
Deleting an element at the end

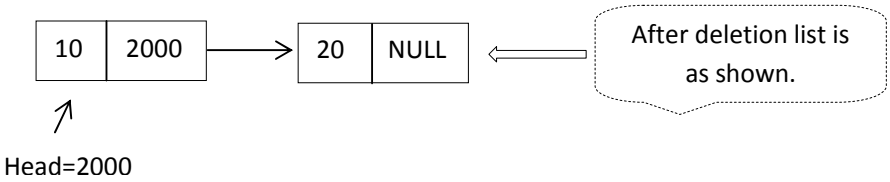
Consider following is the linked list which already exists.

Ex:

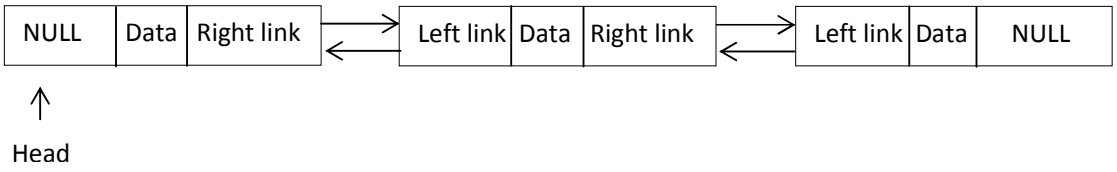


In the above list we are going to delete the end node.



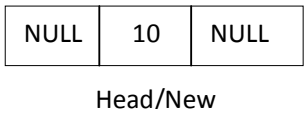


Doubly linked list: It contains nodes which have data field and two pointer field, left link and right link. Left link holds the address of previous node and right link holds the address of next node.



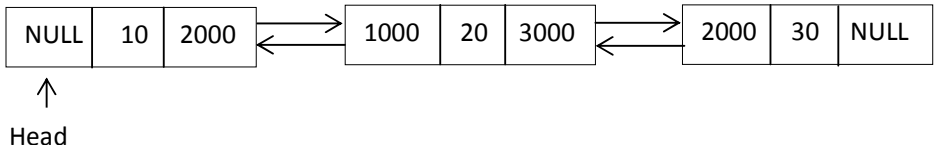
Inserting an element at front

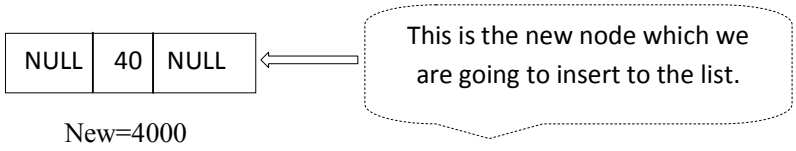
If there is no node in the list then the newly inserted node will be the head node



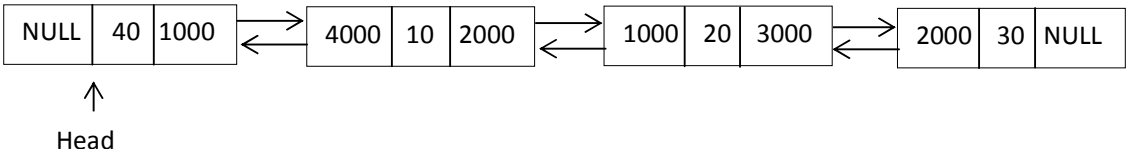
Consider following is the doubly linked list which already exists.

Ex:



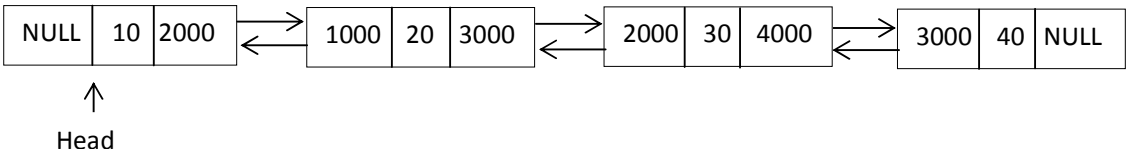


After inserting the new node, the list is as below.



Inserting an element at end

After inserting the new node, the list is as below.

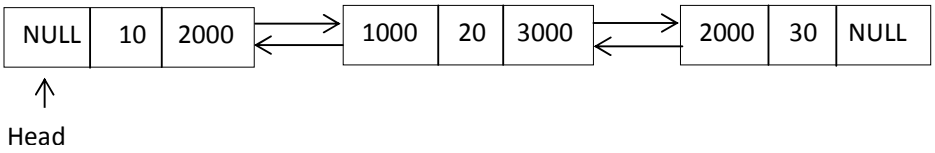


Deleting an element at front

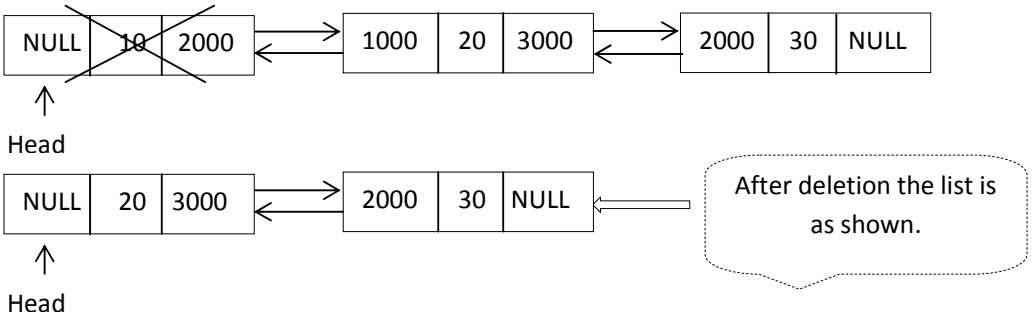
If there is no node in the list then nothing will be there to delete.

Consider following is the doubly linked list which already exists.

Ex:

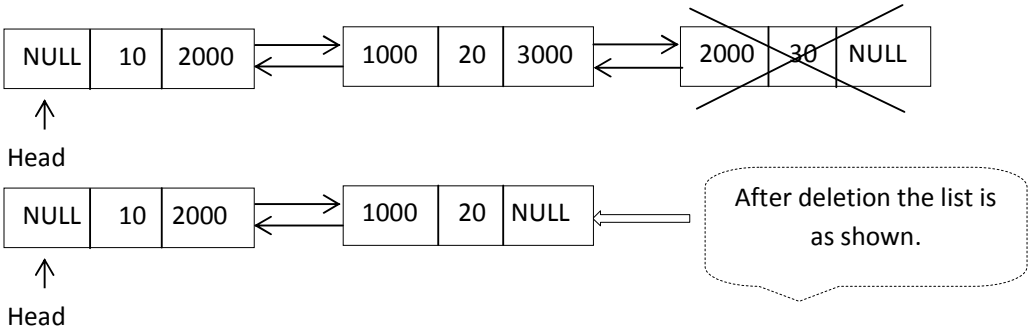


In the above list we are going to delete the head node.

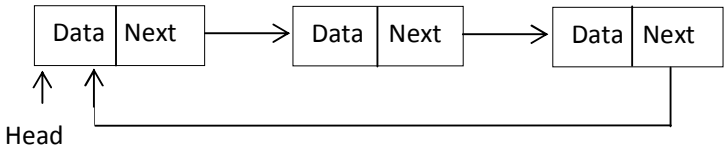


Deleting an element at end

In the above example list we are going to delete the last node.



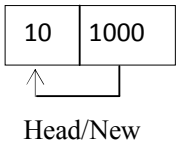
Circular linked list: It is the variation of singly linked list in which the last element is linked to the first element.



Inserting an element at front

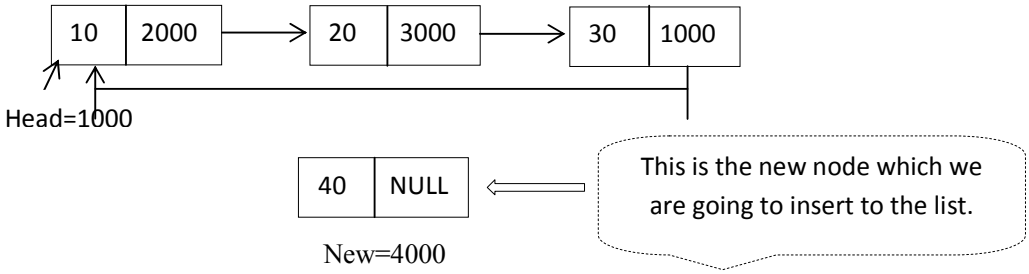
If there is no node in the list then the newly inserted node will be the head node.

Ex:

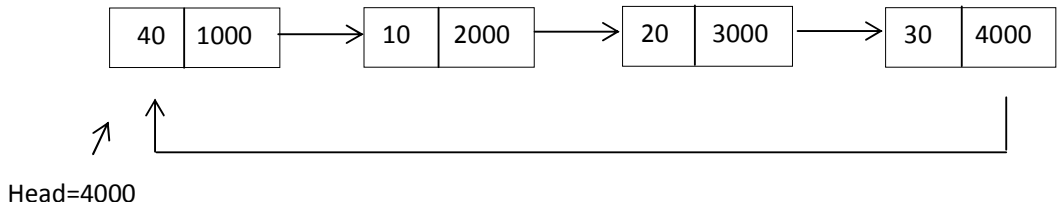


Consider following is the linked list which already exists.

Ex:



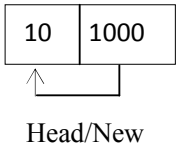
After inserting the new node the list is as below.



Inserting an element at end

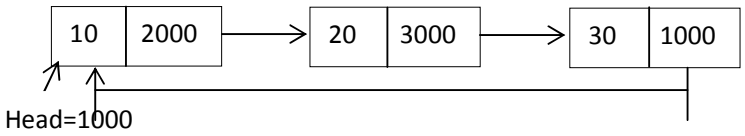
If there is no node in the list then the newly inserted node will be the head node.

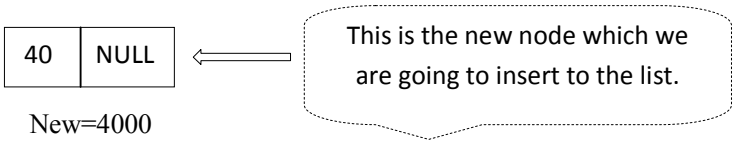
Ex:



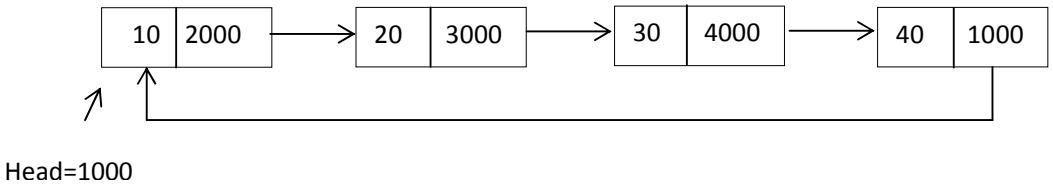
Consider following is the linked list which already exists.

Ex:





After inserting the new node the list is as below.

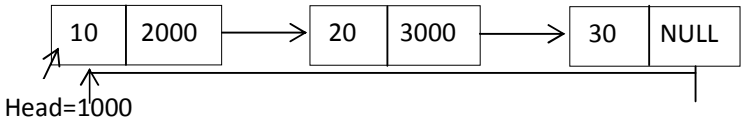


Deleting an element at front

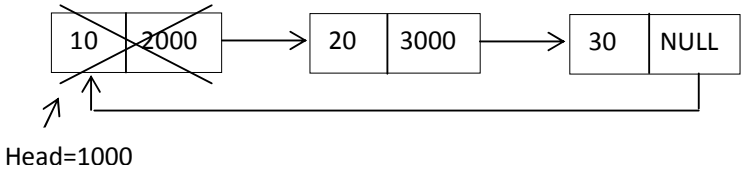
If there is no node in the list then nothing will be there to delete

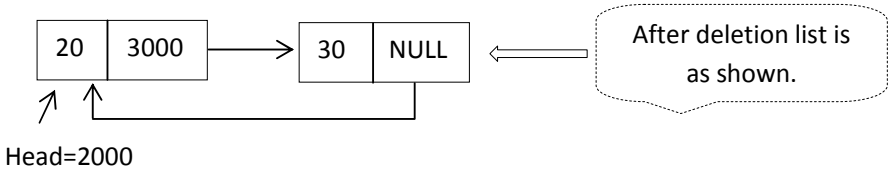
Consider following is the linked list which already exists.

Ex:



In the above list we are going to delete the head node.

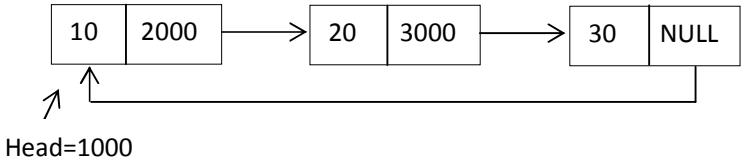




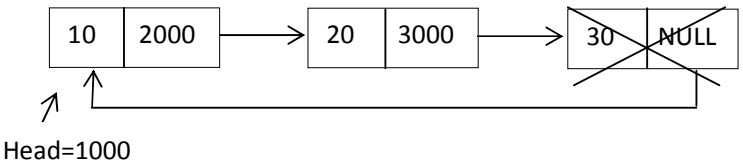
Deleting an element at the end

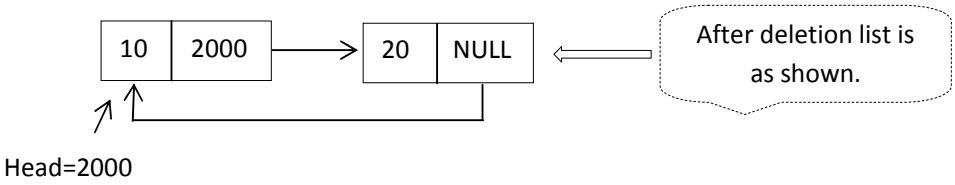
Consider following is the linked list which already exists.

Ex:



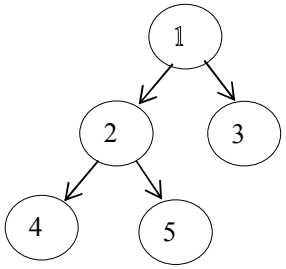
In the above list we are going to delete the end node.





Trees: A tree is a data structure made up of nodes or vertices and edges without having any cycle. The tree with no nodes is called the **null** or **empty** tree.

Ex:



Terminologies used in Trees

Root: The top node in a tree. In the above tree the root node is 1.

Child: A node directly connected to another node when moving away from the Root.

Ex: 2, 3 are the children of node 1.

4, 5 are the children of node 2.

Parent: The node which is predecessor of any node is called as parent node. In other words, the node which has branch from it to any other node is called as parent node.

Ex: 1 is the parent of 2 and 3

2 is the parent of 4 and 5

Siblings: A group of nodes with the same parent.

Ex: 2, 3 are siblings because the parent for them is same, that is 1

4, 5 are siblings because the parent for them is same, that is 2

Leaf: A node with no children.

Ex: 3, 4, 5

Internal node: A node with at least one child.

Ex: 1, 2

Degree: The number of sub trees of a node.

Ex: Degree of node 1 is 2

Degree of node 3 is 0

Edge: The connection between one node and another.

Path: A sequence of nodes and edges connecting a node with a descendant.

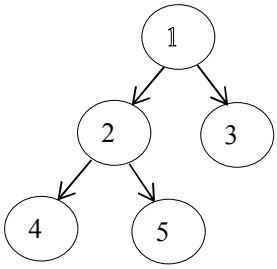
Level: The number of connections between the node and the root. The root node is always at level zero.

Ex: The level of nodes 2 and 3 is 1

The level of nodes 4 and 5 is 2

Binary tree: In binary tree each node can have at most two children, which are referred to as the left child and the right child.

Ex:



Binary search tree: A Binary Search Tree (BST) is a tree in which all the nodes follow the following rules:

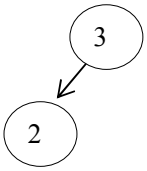
- The left sub-tree of a node has a key less than to its parent node's key.
- The right sub-tree of a node has a key greater than to its parent node's key.

Ex: Construct the Binary search tree for the following input 3, 2, 1, 5

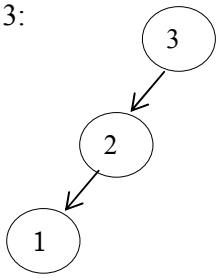
The first node will be the root node and the remaining nodes have to be arranged as per the above rules.

Step 1:

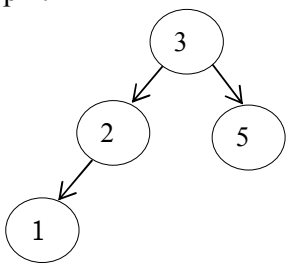
Step 2:



Step 3:

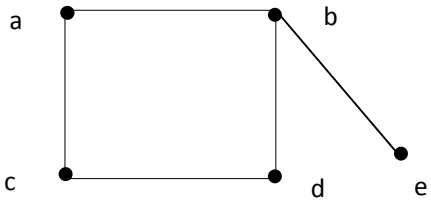


Step 4:



Graph: A graph is a pair of sets (V, E), where V is the set of vertices and E is the set of edges, connecting the pairs of vertices.

Ex:



V = {a, b, c, d, e}

E = {ab, ac, bd, be, cd}

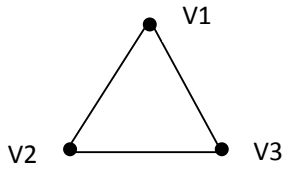
There are 2 types of graphs Directed and Undirected graphs.

In directed graphs the directions will be shown by the edges but in undirected graphs only edges will be there between the vertices without the directions.

Terminologies used in graph

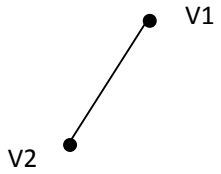
Complete graph: A graph of n vertices with n(n-1)/2 number edges is a complete graph.

Ex:



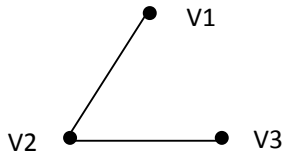
Sub graph: The sub graph S of Graph G is the set of vertices and edges are the subsets of graph G.

Ex:



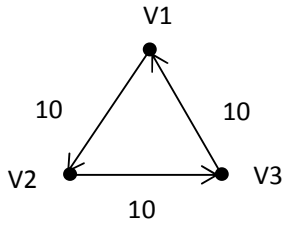
Connected graph: An undirected graph is said to be connected if there is an edge between every pair of distinct vertices

Ex:



Weighted graph: The graph which contains weights with its edges.

Ex:



Path: It is a sequence of vertices and there exists an edge between the vertices.

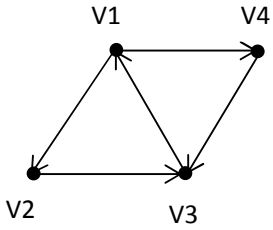
Ex: The path for the above graph is V1-V2-V3

Cycle: A closed walk through the graph where the starting and ending vertex is same.

Ex: The cycle for the above graph is V1-V2-V3-V1

Indegree and Outdegree: The number of edges that are incident to that vertex is the indegree. The number of edges that are going away from that vertex is the outdegree.

Ex:



Vertices	Indegree	Outdegree
V1	1	2
V2	1	1
V3	2	1
V4	1	1

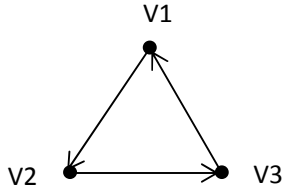
Self Loop: The edge that connects a vertex to itself.

Ex:

Representation of graph in Adjacency matrix

Consider a graph G of n vertices and matrix M, if there is an edge between vertices V_i and V_j then M[i][j]=1 else M[i][j]=0.

Ex:



$$\begin{array}{c} \\ V1 \\ V2 \\ V3 \end{array} \begin{array}{c} \begin{array}{ccc} V1 & V2 & V3 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \end{array}$$

1. Design, Develop and Implement a menu driven Program in C for the following Array operations
a. Creating an Array of N Integer Elements
b. Display of Array Elements with Suitable Headings
c. Inserting an Element (ELEM) at a given valid Position (POS)
d. Deleting an Element at a given valid Position (POS)
e. Exit.
Support the program with functions for each of the above operations.

Program:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int a[50],n,pos,ele;  // Global variables declaration
void create()  // Function for creating an array with n number of elements
{
int i;
printf("Enter the number of elements\n");
scanf("%d",&n);
printf("Enter the array elements\n");
for(i=0;i<n;i++)
scanf("%d",&a[i]);
}
void insert()  // Function for inserting an element at the given position
{
int i;
printf("Enter the element and position\n");
scanf("%d%d",&ele,&pos);
for(i=n-1;i>=pos;i--)
{
a[i+1]=a[i];
}
a[pos]=ele;
n++;  // After insertion number of elements (n) has to increase
}
void del()  // Function for deleting an element at the given position
{
int i;
printf("Enter the element position you want to delete\n");
scanf("%d",&pos);
printf("The deleted element is %d\n",a[pos]);
for(i=pos;i<n;i++)
{
a[i]=a[i+1];
}
n--;  // After deletion the number of elements (n) has to decrease
}
```

```
void display()  // Function for displaying the array elements
{
int i;
printf("The array elements are:\n");
for(i=0;i<n;i++)
printf("%d\t",a[i]);
printf("\n");
}
void main()
{
int ch;
clrscr();
create();
while(1)
{
printf("1. Insert\n2. Delete\n3. Display\n4. Exit\n");
printf("Enter the choice\n");
scanf("%d",&ch);
switch(ch)
{
case 1: insert();
        break;
case 2: del();
        break;
case 3: display();
        break;
case 4: exit(0);
default: printf("Invalid choice\n");
}
}
}
```

Output:

```
Enter the number of elements
2
Enter the array elements
10
20
1. Insert
2. Delete
3. Display
4. Exit
Enter the choice
1
Enter the element and position
30
```

```
1
1. Insert
2. Delete
3. Display
4. Exit
Enter the choice
2
Enter the element position you want to delete
2
The deleted element is 20
1. Insert
2. Delete
3. Display
4. Exit
Enter the choice
3
The array elements are:
10      30
1. Insert
2. Delete
3. Display
4. Exit
Enter the choice
4
```

2. Design, Develop and Implement a Program in C for the following operations on Strings
a. Read a main String (STR), a Pattern String (PAT) and a Replace String (REP)
b. Perform Pattern Matching Operation: Find and Replace all occurrences of PAT in STR with REP if PAT exists in STR. Report suitable messages in case PAT does not exist in STR
Support the program with functions for each of the above operations. Don't use Built-in functions.

Program:

```
#include<stdio.h>
#include<conio.h>
void findandreplace(char str[50],char pat[50],char rep[50])
{
int s,c,p,i,j,occ=0;
char res[50];
s=p=c=j=0;
while(str[c]!='\0')  // Checking for end of the main string
{
if(str[s]==pat[p])  // Comparing each character of main string and pattern string
{
p++;
s++;
if(pat[p]=='\0')  // End of the pattern string
{
occ++;  // Increase the number of occurrences
printf("Pattern found at %d\n",c);
for(i=0;rep[i]!='\0';i++,j++)  // Copy the replace string to result string
res[j]=rep[i];
p=0;  // Initializing pattern string from first character
c=s;
}
}
else  // Characters are not matching
{
res[j] = str[c];  // Copy the main string character to result string
j++;
c++;
s=c;
p=0;
}
}  // End of while
res[j]='\0';
if(occ)
{
printf("Number of occurrences=%d\n",occ);
printf("The resultant string is:%s" ,res);
}
```

```
else
printf("Pattern not found");
}
void main()
{
char str[50],pat[50],rep[50];
clrscr();
printf("Enter a string \n");
gets(str);
printf("Enter a pattern string \n");
gets(pat);
printf("Enter a replace string \n");
gets(rep);
findandreplace(str,pat,rep);
getch();
}
```

Output:

```
Enter a string
hi hello
Enter a pattern string
hello
Enter a replace string
hi
Pattern found at 3
Number of occurrences=1
The resultant string is : hi hi
```

3. Design, Develop and Implement a menu driven Program in C for the following operations on STACK of Integers (Array Implementation of Stack with maximum size MAX)
a. Push an Element on to Stack
b. Pop an Element from Stack
c. Demonstrate how Stack can be used to check Palindrome
d. Demonstrate Overflow and Underflow situations on Stack
e. Display the status of Stack
f. Exit
Support the program with appropriate functions for each of the above operations

Program:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#define MAX 5
int top=-1,a[MAX];
void push(int item)  // Push function
{
if(top==MAX-1)
printf("Stack overflow\n");
else
a[++top]=item;  // Pushing an element to stack
}
int pop()  // Pop function
{
int itemdel;
if(top==-1)
return 0;
else
{
itemdel=a[top--];  // Poping an element from stack
return itemdel;
}
}
void display()  // Display function
{
int i;
if(top==-1)
printf("Stack empty\n");
else
{
printf("The elements are:\n");
for(i=top;i>=0;i--)
printf("%d\n",a[i]);
```

```
}
}
void palindrome(int num)  //Palindrome function
{
int count=0,rem,i,rev=0,n,item;
n=num;
while(n!=0)
{
rem=n%10;
push(rem);  // The last digit of a number n pushing into the stack
n=n/10;
count++;  // Count gives the total number of digits pushed into the stack
}
for(i=0;i<count;i++)
{
item=pop();
rev=item*pow(10,i)+rev;
}
printf("Reversed number = %d\n",rev);
if(num==rev)
printf("The number is palindrome\n");
else
printf("The number is not a palindrome\n");
}
void main()
{
int ch,item,num,itemdel;
clrscr();
while(1)
{
printf("1.Push\n2.Pop\n3.Display\n4.Palindrome\n5.Exit\n");
printf("Enter the choice\n");
scanf("%d",&ch);
switch(ch)
{
case 1: printf("Enter the item to be inserted\n");
        scanf("%d",&item);
        push(item);
        break;
case 2: itemdel=pop();
        if(itemdel)
        printf("Deleted item is : %d\n",itemdel);
        else
        printf("Stack underflow\n");
        break;
```

```
case 3: display();
        break;
case 4: printf("Enter the number:\n");
        scanf("%d",&num);
        palindrome(num);
        break;
case 5: exit(0);
}
}
}
```

Output:

```
1.Push
2.Pop
3.Display
4.Palindrome
5.Exit
Enter the choice
3
Stack underflow
1.Push
2.Pop
3.Display
4.Palindrome
5.Exit
Enter the choice
1
Enter the item to be inserted
10
1.Push
2.Pop
3.Display
4.Palindrome
5.Exit
Enter the choice
2
Deleted item is : 10
```

1.Push
2.Pop
3.Display
4.Palindrome
5.Exit
Enter the choice
3
Stack empty
1.Push
2.Pop
3.Display
4.Palindrome
5.Exit
Enter the choice
4
Enter the number:
121
Reversed number = 121
The number is palindrome
1.Push
2.Pop
3.Display
4.Palindrome
5.Exit
Enter the choice
5

4. Design, Develop and Implement a Program in C for converting an Infix Expression to Postfix Expression. Program should support for both parenthesized and free parenthesized expressions with the operators: +, -, *, /, %(Remainder), ^(Power) and alphanumeric operands.

Program:

```
#include<stdio.h>
#include<conio.h>
char a[25];
int top=-1;
void push(char symbol)  // Push function
{
a[++top]=symbol;
}
char pop()  // Pop function
{
char item;
item=a[top--];
return(item);
}
int pre(char op)  // Precedence function gives the precedence of the operator
{
int p;
switch(op)
{
case '^':p=3;
         break;
case '*':
case '/':
case '%':p=2;
         break;
case '+':
case '-':p=1;
         break;
case '(':p=0;
         break;
case '$':p=-1;
         break;
}
return p;
}
void infixtopostfix(char infix[],char postfix[])  // Infix to postfix conversion function
{
int i,j=0;
char symbol,item;
```

```
push('$');  // first $ is pushed to indicate the end of the stack
for(i=0;infix[i]!='\0';i++)
{
symbol=infix[i];
switch(symbol)
{
case '(':push(symbol);
          break;
case ')':item=pop();
          while(item!='(')  // Till the '(' is reached pop the item from stack and copy to postfix
          {
           postfix[j++]=item;
           item=pop();
          }
          break;
case '+':
case '-':
case '*':
case '/':
case '^':
case '%':while(pre(a[top])>=pre(symbol))  // Checks the precedence of symbol
          {
           item=pop();
           postfix[j++]=item;
          }
          push(symbol);
          break;
default:postfix[j++]=symbol;  // If the symbol is operand copy to postfix
         break;
}
}
while(top>0)  // The remaining symbols in the stack copying to postfix
{
item=pop();
postfix[j++]=item;
}
postfix[j]='\0';
}
void main()
{
char infix[25],postfix[25];
clrscr();
printf("Enter the infix expression:\n");
scanf("%s",infix);
infixtopostfix(infix,postfix);
```

```
printf("Postfix expression is : %s\n",postfix);
getch();
}
```

Output:

```
Enter the infix expression:
(a+b)*c
Postfix expression is : ab+c*
```

5. Design, Develop and Implement a Program in C for the following Stack Applications
a. Evaluation of Suffix expression with single digit operands and operators: +, -, *, /, %, ^
b. Solving Tower of Hanoi problem with n disks

Program: 5a

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
float a[25];
int top=-1;
void push(float symbol)  // Push function
{
a[++top]=symbol;
}
float pop()  // Pop function
{
return(a[top--]);
}

float eval(char op,float op1,float op2)  // Function for evaluating postfix expression
{
switch(op)
{
case '^': return(pow(op1,op2));
case '*': return(op1*op2);
case '/': return(op1/op2);
case '%': return((int)op1%(int)op2);  // Finding remainder so typecast op1 and op2 to integer
case '+': return(op1+op2);
case '-': return(op1-op2);
}
return 0;
}
void main()
{
char postfix[25],symbol;
float op1,op2,res;int i;
clrscr();
printf("Enter the postfix expression\n");
scanf("%s",postfix);
for(i=0;postfix[i]!='\0';i++)
{
symbol=postfix[i];
if(isdigit(symbol))
push(symbol-'0');  /* isdigit() function will return ASCII value so to get the actual value of a symbol
```

```
                    subtracting symbol from ASCII value of zero */
else
{
op2=pop();
op1=pop();
res=eval(symbol,op1,op2);
push(res);
}
}
res=pop();
printf("Result = %.3f",res);
getch();
}
```

Output:

```
Enter the postfix expression
23+4-3*
Result = 3.000
```

Program: 5b

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<stdlib.h>
#include<limits.h>  // Used for the macro INT_MIN
struct stack  // Structure for stack
{
int capacity;
int top;
int *array;
};
struct stack* create(int capacity)  // Function for creating a stack of given size 'n'
{
struct stack* s1=(struct stack*)malloc(sizeof(struct stack));
s1->capacity=capacity;
s1->top=-1;
s1->array=(int*)malloc(s1->capacity*sizeof(int));
return s1;
}
void push(struct stack *s1, int item)  // Push function
{
if(s1->top==s1->capacity-1)  // Condition to check stack overflow
return;
```

```
s1->array[++s1->top]=item;
}
int pop(struct stack* s1)  // Pop function
{
if(s1->top==-1)  // Condition to check stack underflow
return INT_MIN;
return s1->array[s1->top--];
}
void move(struct stack *src,struct stack *dest,char s,char d)  /* Function for moving the disks
                                                                between poles*/
{
int p1=pop(src);
int p2=pop(dest);
if(p1==INT_MIN)  // If pole1 is empty
{
push(src, p2);  // Moving the p2 to source
printf("Move the disk %d from \'%c\' to \'%c\'\n",p2,d,s);
}
else if(p2==INT_MIN)  // If pole2 is empty
{
push(dest,p1);  // Moving the p1 to destination
printf("Move the disk %d from \'%c\' to \'%c\'\n",p1,s,d);
}
else if(p1>p2)  // If top disk of pole1 is greater than top disk of pole2
{
push(src,p1);  // Moving the p1 to the same pole(source)
push(src,p2);  // Moving the p2 to the source
printf("Move the disk %d from \'%c\' to \'%c\'\n",p2,d,s);
}
else  // If top disk of pole1 is less than top disk of pole2
{
push(dest,p2);  // Moving the p2 to the same pole(destination)
push(dest,p1);  // Moving the p1 to the destination
printf("Move the disk %d from \'%c\' to \'%c\'\n",p1,s,d);
}
}
void tower(int n,struct stack *src,struct stack *aux,struct stack *dest)
{
int i,t;
char s='S',d='D',a='A';
//If number of disks is even, then interchange destination pole and auxiliary pole
if(n%2==0)
{
char temp=d;
d=a;
```

```
a=temp;
}
t=pow(2,n)-1;  // Calculating total number moves
for(i=n;i>=1;i--)  // Larger disks will be pushed first
push(src,i);
for(i=1;i<=t;i++)  // Within the total number moves (t) the tower of hanoi problem has to be solved
{
if(i%3==1)
move(src,dest,s,d);
else if(i%3==2)
move(src,aux,s,a);
else if(i%3==0)
move(aux,dest,a,d);
}
}
void main()
{
int n;
struct stack *src,*dest,*aux;
clrscr();
printf("Enter the number of disks: ");
scanf("%d",&n);  // n is the number of disks
src = create(n);  // Creating stack for source pole
aux = create(n);  // Creating stack for auxiliary pole
dest = create(n);  // Creating stack for destination pole
tower(n,src,aux,dest);
getch();
}
```

Output:

```
Enter the number of disks: 3
Move the disk 1 from 'S' to 'D'
Move the disk 2 from 'S' to 'A'
Move the disk 1 from 'D' to 'A'
Move the disk 3 from 'S' to 'D'
Move the disk 1 from 'A' to 'S'
Move the disk 2 from 'A' to 'D'
Move the disk 1 from 'S' to 'D'
```

6. Design, Develop and Implement a menu driven Program in C for the following operations on Circular QUEUE of Characters (Array Implementation of Queue with maximum size MAX)
a. Insert an Element on to Circular QUEUE
b. Delete an Element from Circular QUEUE
c. Demonstrate Overflow and Underflow situations on Circular QUEUE
d. Display the status of Circular QUEUE
e. Exit
Support the program with appropriate functions for each of the above operations

Program:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define MAX 5
int a[MAX],f=0,r=-1,count=0;
void insert(int item)  // Function for inserting an element into the queue
{
if(count==MAX)
printf("Queue overflow\n");
else
{
r=(r+1)%MAX;
a[r]=item;
count=count+1;
}
}
void del()  // Function for deleting an element from queue
{
int itemdel;
if(count==0)
printf("Queue underflow\n");
else
{
itemdel=a[f];
f=(f+1)%MAX;
count=count-1;
printf("The deleted item is %d\n",itemdel);
}
}
void display()  // Function for displaying the contents of queue
{
int i,j;
if(count==0)
printf("Queue empty\n");
```

```
else
{
i=f;
for(j=1;j<=count;j++)
{
printf("%d\t",a[i]);
i=(i+1)%MAX;
}
}
}
void main()
{
int ch,item,itemdel;
clrscr();
while(1)
{
printf("\n1.Insert\n2.Delete\n3.Display\n4.Exit\n");
printf("Enter the choice\n");
scanf("%d",&ch);
switch(ch)
{
case 1: printf("Enter the item to be inserted\n");
        scanf("%d",&item);
        insert(item);
        break;
case 2: del();
        break;
case 3: display();
        break;
case 4: exit(0);
}
}
}
```

Output:

```
1.Insert
2.Delete
3.Display
4.Exit
Enter the choice
1
Enter the item to be inserted
10
1.Insert
2.Delete
3.Display
```

4.Exit
Enter the choice
2
The deleted item is 10
1.Insert
2.Delete
3.Display
4.Exit
Enter the choice
3
Queue empty
1.Insert
2.Delete
3.Display
4.Exit
Enter the choice
4

7. Design, Develop and Implement a menu driven Program in C for the following operations on Singly Linked List (SLL) of Student Data with the fields: USN, Name, Branch, Sem, PhNo
a. Create a SLL of N Students Data by using front insertion.
b. Display the status of SLL and count the number of nodes in it
c. Perform Insertion / Deletion at End of SLL
d. Perform Insertion / Deletion at Front of SLL(Demonstration of stack)
e. Exit

Program:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct student  // Structure for student details
{
char usn[12],name[25],branch[25];
int sem,phone_no;
struct student *link;
};
typedef struct student STUD;
STUD *read()  // Function for reading the student details
{
char usn[12],name[25],branch[25];
int sem,phone_no;
STUD *temp;
temp=(STUD *)malloc(sizeof(STUD));
printf("Enter the students details:\n");
printf("Enter USN:");
scanf("%s",temp->usn);
printf("Enter name:");
scanf("%s",temp->name);
printf("Enter branch:");
scanf("%s",temp->branch);
printf("Enter semester:");
scanf("%d",&temp->sem);
printf("Enter phone number:");
scanf("%d",&temp->phone_no);
temp->link=NULL;
return temp;
}
STUD *insert_front(STUD *head)  // Function for inserting node at front
{
STUD *New;
New=read();
New->link=head;
```

```
return New;
}
STUD *insert_end(STUD *head)  // Function for inserting node at end
{
STUD *New,*temp;
New=read();
if(head==NULL)
return New;
temp=head;
while(temp->link!=NULL)
temp=temp->link;
temp->link=New;
return head;
}
STUD *delete_front(STUD *head)  // Function for deleting node at front
{
STUD *temp;
if(head==NULL)
{
printf("List is empty\n");
return head;
}
temp=head;
head=head->link;
free(temp);
return head;
}
STUD *delete_end(STUD *head)  // Function for deleting node at end
{
STUD *prev,*temp;
if(head==NULL)
{
printf("List is empty\n");
return head;
}
prev=NULL;
temp=head;
while(temp->link!=NULL)
{
prev=temp;
temp=temp->link;
}
prev->link=NULL;
free(temp);
return head;
```

```
}
void display(STUD *head)
{
STUD *temp;
int count=0;
if(head==NULL)
{
printf("List is empty\n");
return;
}
printf("USN\tNAME\tBRANCH\tSEM\tPHONE NO.\n");
temp=head;
while(temp!=NULL)
{
printf("%s\t%s\t%s\t%d\t%d\n",temp->usn,temp->name,temp->branch,temp->sem,temp->
phone_no);
temp=temp->link;
count++;
}
printf("The number of nodes in SLL=%d\n",count);
}
void stack()
{
int ch;
STUD *head=NULL;
while(1)
{
printf("1.Push\n2.Pop\n3.Display\n4.Exit\n");
printf("Enter choice\n");
scanf("%d",&ch);
switch(ch)
{
case 1: head=insert_front(head);
        printf("Node inserted \n");
        break;
case 2: head=delete_front(head);
        printf("Node deleted\n");
        break;
case 3: display(head);
        break;
case 4: return;
}
}
}
void main()
```

```
{
int ch,i,n;
STUD *head=NULL;
clrscr();
printf("Creation of SLL of N students\n");
printf("Enter the number of students\n");
scanf("%d",&n);
for(i=1;i<=n;i++)
head=insert_front(head);
printf("SLL created successfully…..\n");
while(1)
{
printf("1.Display\n2.Insert Front\n3:Insert End\n4.Delete Front\n5.Delete End\n6.Stack\n7.Exit\n");
printf("Enter the choice\n");
scanf("%d",&ch);
switch(ch)
{
case 1: display(head);
        break;
case 2: head=insert_front(head);
        printf("Node inserted at front\n");
        break;
case 3: head=insert_end(head);
        printf("Node inserted at end\n");
        break;
case 4: head=delete_front(head);
        printf("Node deleted at Front\n");
        break;
case 5: head=delete_end(head);
        printf("Node deleted at end\n");
        break;
case 6: stack();
        break;
case 7: exit(0);
}
}
}
```

Output:

```
Creation of SLL of N students
Enter the number of students
1
Enter the student details:
Enter USN:1
Enter name:aa
Enter branch: cs
Enter semester:3
Enter phone number:345
SLL created successfully…..
1.Display
2.Insert Front
3.Insert End
4.Delete Front
5.Delete End
6.Stack
7.Exit
Enter the choice
1
USN   NAME   BRANCH   SEM   PHONE NO
1     aa     cs       3     345
The number of nodes in SLL=1
1.Display
2.Insert Front
3.Insert End
4.Delete Front
5.Delete End
6.Stack
7.Exit
Enter the choice
7
```

8. Design, Develop and Implement a menu driven Program in C for the following operations on Doubly Linked List (DLL) of Employee Data with the fields: SSN, Name, Dept, Designation, Sal, PhNo

a. Create a DLL of N Employees Data by using end insertion.
b. Display the status of DLL and count the number of nodes in it
c. Perform Insertion and Deletion at End of DLL
d. Perform Insertion and Deletion at Front of DLL
e. Demonstrate how this DLL can be used as Double Ended Queue
f. Exit

Program:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>
struct emp
{
char SSN[12];
char name[25];
char dept[25];
char designation[25];
float salary;
int phone_no;
struct emp *left;
struct emp *right;
};
typedef struct emp EMP;
EMP *read()
{
float salary;
EMP *temp;
temp=(EMP *)malloc(sizeof(EMP));
printf("Enter the employees details:\n");
printf("Enter SSN\n");
scanf("%s",temp->SSN);
printf("Enter Name\n");
scanf("%s",temp->name);
printf("Enter Department\n");
scanf("%s",temp->dept);
printf("Enter Designation\n");
scanf("%s",temp->designation);
printf("Enter Salary\n");
scanf("%f",&salary);
temp->salary=salary;
```

```
printf("Enter Phone Number\n");
scanf("%d",&temp->phone_no);
temp->right=temp->left=NULL;
return temp;
}
EMP *insert_front(EMP *head)
{
EMP *New;
New=read();
if(head==NULL)  // If there is no node in the list then new node will be the head node
head=New;
else
{
head->left=New;
New->right=head;
head=New;
}
return head;
}
EMP *insert_end(EMP *head)
{
EMP *temp,*New;
New=read();
if(head==NULL)
head=New;
else
{
temp=head;
while(temp->right!=NULL)  // Till the end of the list is reached keep on moving to the right node in the list
temp=temp->right;
temp->right=New;
New->left=temp;
}
return head;
}
EMP *delete_front(EMP *head)
{
EMP *temp,*next;
if(head==NULL)
printf("List is empty\n");
else
{
temp=head;
if(temp->right==NULL)
```

```
head=NULL;
else
{
next=temp->right;
temp->left=NULL;
head=next;
}
free(temp);
}
return head;
}

EMP *delete_end(EMP *head)
{
EMP *temp,*prev;
if(head==NULL)
printf("List is empty\n");
else
{
temp=head;
if(temp->right==NULL)
head=NULL;
else
{
while(temp->right!=NULL)
{
temp=temp->right;
}
prev=temp->left;
prev->right=NULL;
}
free(temp);
}
return head;
}
void display(EMP *head)
{
EMP *temp;
int count=0;
if(head==NULL)
printf("List is empty\n");
else
{
temp=head;
printf("SSN\tName\tDept\tDesignation\tSalary\t\tPhone_No.\n");
```

```
while(temp!=NULL)
{
printf("%s\t%s\t%s\t%s\t\t\t%f\t%d\n",temp->SSN,temp->name,temp->dept,temp->designation,temp->salary,temp->phone_no);
temp=temp->right;
count++;
}
printf("Number of nodes in DLL=%d\n",count);
}
}
EMP *deq(EMP *head)
{
int ch;
while(1)
{
printf("1.Insert front\n2.Insert end\n3.Delete front\n4.Delete end\n5.Display\n6.Exit\n");
printf("enter the choice\n");
scanf("%d",&ch);
switch(ch)
{
case 1: head=insert_front(head);
        break;
case 2: head=insert_end(head);
        break;
case 3: head=delete_front(head);
        break;
case 4: head=delete_end(head);
        break;
case 5: display(head);
        break;
case 6: return head;
}
}
}
void main()
{
int ch,n,i;
EMP *head;
clrscr();
head=NULL;
printf("Creation of DLL\n");
printf("Enter the number of employees\n");
scanf("%d",&n);
for(i=1;i<=n;i++)
head=insert_end(head);
```

```
while(1)
{
printf("1.Display\n2.Insert front\n3:Insert end\n4.Delete front\n5.Delete end\n6.Double ended queue\n7.Exit\n");
printf("Enter the choice\n");
scanf("%d",&ch);
switch(ch)
{
case 1: display(head);
        break;
case 2: head=insert_front(head);
        printf("Node Inserted at front\n");
        break;
case 3: head=insert_end(head);
        printf("Node inserted at end\n");
        break;
case 4: head=delete_front(head);
        printf("Node deleted at front\n");
        break;
case 5: head=delete_end(head);
        printf("Node deleted at end\n");
        break;
case 6: head=deq(head);
        break;
case 7: exit(0);
}
}
}
```

Output:

```
Creation of DLL
Enter the number of employees
1
Enter the employees details:
Enter SSN
1
Enter Name
Shiva
Enter Department
CS
Enter Designation
Asst Professor
Enter Salary
30000
Enter Phone Number
555
1. Display
```

2. Insert front
3. Insert end
4. Delete front
5. Delete end
6. Double ended queue
7. Exit
Enter the choice
1

SSN	Name	Dept	Designation	Salary	Phone_No
1	Shiva	CS	AP	30000	555

Number of nodes in DLL=1
1. Display
2. Insert front
3. Insert end
4. Delete front
5. Delete end
6. Double ended queue
7. Exit
Enter the choice
2
Enter the employees details:
Enter SSN
2
Enter Name
Rama
Enter Department
CS
Enter Designation
Asst Professor
Enter Salary
40000
Enter Phone Number
666

1. Display
2. Insert front
3. Insert end
4. Delete front
5. Delete end
6. Double ended queue
7. Exit
Enter the choice
1

SSN	Name	Dept	Designation	Salary	Phone_No
2	Rama	CS	AP	40000	666

```
1       Shiva   CS      AP              30000  555
Number of nodes in DLL=2
1. Display
2. Insert front
3. Insert end
4. Delete front
5. Delete end
6. Double ended queue
7. Exit
Enter the choice
7
```

9. Design, Develop and Implement a Program in C for the following operations on Singly Circular Linked List (SCLL) with header nodes

a. Represent and Evaluate a Polynomial P(x,y,z) = 6x2y2z-4yz5+3x3yz+2xy5z-2xyz3

b. Find the sum of two polynomials POLY1(x,y,z) and POLY2(x,y,z) and store the result in POLYSUM(x,y,z)

Support the program with appropriate functions for each of the above operations

Program:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<math.h>
typedef struct
{
int coeff,x_exp,y_exp,z_exp;
struct polynode *link;
}polynode;
polynode *create(int coeff,int x_exp,int y_exp,int z_exp)
{
polynode *node;
node=(polynode*)malloc(sizeof(polynode));
node->coeff=coeff;
node->x_exp=x_exp;
node->y_exp=y_exp;
node->z_exp=z_exp;
node->link=NULL;
return node;
}
polynode *attach(polynode *node,polynode *poly)
{
polynode *cur;
cur=poly->link;
while(cur->link!=poly)
{
cur=cur->link;
}
cur->link=node;
node->link=poly;
return poly;
}
polynode *read()
{
int n,i;
int coeff,x_exp,y_exp,z_exp;
```

```
polynode *temp;
polynode *poly=(polynode*)malloc(sizeof(polynode));
poly->link=poly;
printf("Enter number of terms\n");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Term %d:\n",i+1);
printf("Enter the coefficient\n");
scanf("%d",&coeff);
printf("Enter exponent values for x,y and z\n");
scanf("%d%d%d",&x_exp,&y_exp,&z_exp);
temp=create(coeff,x_exp,y_exp,z_exp);
poly=attach(temp,poly);
}
return poly;
}
void display(polynode *poly)
{
polynode *cur;
cur=poly->link;
while(cur!=poly)
{
if(cur->coeff>=0)
printf("+%dx^%dy^%dz^%d",cur->coeff,cur->x_exp,cur->y_exp,cur->z_exp);
else
printf("%dx^%dy^%dz^%d",cur->coeff,cur->x_exp,cur->y_exp,cur->z_exp);
cur=cur->link;
}
}
void evaluate(polynode *poly1)
{
polynode *poly;
int x,y,z,xval,yval,zval,res=0;
poly=poly1->link;
printf("\nEnter the values for x, y and z:\n");
scanf("%d%d%d",&x,&y,&z);
while(poly!=poly1)
{
xval=pow(x,poly->x_exp);
yval=pow(y,poly->y_exp);
zval=pow(z,poly->z_exp);
res+=poly->coeff*xval*yval*zval;
poly=poly->link;
}
```

```
printf("Result=%d\n",res);
}
polynode *add(polynode *poly1,polynode *poly2,polynode *poly)
{
int comp;
polynode *a,*b,*temp;
a=poly1->link;
b=poly2->link;
while(a!=poly1 && b!=poly2)
{
if(a->x_exp==b->x_exp && a->y_exp==b->y_exp && a->z_exp==b->z_exp)
comp=0;
else if(a->x_exp>b->x_exp)
comp=1;
else
{
if(a->x_exp==b->x_exp && a->y_exp>b->y_exp)
comp=1;
else if(a->x_exp==b->x_exp && a->y_exp==b->y_exp && a->z_exp>b->z_exp)
comp=1 ;
else
comp=-1;
}
switch (comp)
{
case 0: temp=create(a->coeff+b->coeff,a->x_exp,a->y_exp,a->z_exp);
        poly=attach(temp,poly);
        a=a->link;
        b=b->link;
        break;
case 1: temp=create(a->coeff,a->x_exp,a->y_exp,a->z_exp);
        poly=attach(temp,poly);
        a=a->link;
        break;
case -1: temp=create(b->coeff,b->x_exp,b->y_exp,b->z_exp);
        poly=attach(temp,poly);
        b=b->link;
        break;
}
}
while(a!=poly1)
{
temp=create(a->coeff,a->x_exp,a->y_exp,a->z_exp);
poly=attach(temp,poly);
a=a->link;
```

```
}
while(b!=poly2)
{
temp=create(b->coeff,b->x_exp,b->y_exp,b->z_exp);
poly=attach(temp,poly);
b=b->link;
}
return poly;
}
void main()
{
int ch;
polynode *poly1,*poly2,*poly3;
clrscr();
poly3=(polynode*)malloc(sizeof(polynode));
poly3->link=poly3;
while(1)
{
printf("\n1.Represent & Evaluate a Polynomial\n2.Add 2 Polynomial\n3.Exit\n");
scanf("%d",&ch);
switch(ch)
{
case 1: printf("Enter polynomial:\n");
        poly1=read();
        display(poly1);
        evaluate(poly1);
        break;
case 2: printf("Enter polynomial 1:\n");
        poly1=read();
        display(poly1);
        printf("\nEnter polynomial 2:\n");
        poly2=read();
        display(poly2);
        poly3=add(poly1,poly2,poly3);
        printf("\nThe resultant polynomial is:\n");
        display(poly3);
        break;
case 3: exit(0);
}
}
}
```

Output:

```
1. Represent & Evaluate a Polynomial
2. Add two Polynomial
3. Exit
1
Enter polynomial:
Enter number of terms
2
Term 1:
Enter the coefficient
2
Enter exponent values for x, y and z
1 1 1
Term 2:
Enter the coefficient
3
Enter exponent values for x, y and z
2 3 1
+2x^1y^1z^1+3x^2y^3z^1
Enter the values for x, y and z:
1 1 1
Result=5
1. Represent & Evaluate a Polynomial
2. Add two Polynomial
3. Exit
2
Enter polynomial 1:
Enter number of terms
1
Term 1:
Enter the coefficient
```

```
4
Enter exponent values for x, y and z
1 1 1
+4x^1y^1z^1
Enter polynomial 2:
Enter number of terms
2
Term 1:
Enter the coefficient
5
Enter exponent values for x, y and z
1 1 1
Term 2:
Enter the coefficient
6
Enter exponent values for x, y and z
1 2 2
+5x^1y^1z^1+6x^1y^2z^2
The resultant polynomial is:
+9x^1y^1z^1+6x^1y^2z^2
1. Represent & Evaluate a Polynomial
2. Add two Polynomial
3. Exit
3
```

10. Design, Develop and Implement a menu driven Program in C for the following operations on Binary Search Tree (BST) of Integers
a. Create a BST of N Integers: 6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2
b. Traverse the BST in Inorder, Preorder and Post Order
c. Search the BST for a given element (KEY) and report the appropriate message
d. Delete an element(ELEM) from BST
e. Exit

Program:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct node
{
int info;
struct node *left;
struct node *right;
};
typedef struct node NODE;
NODE *insert(int item,NODE *root)  // Function for inserting the elements
{
NODE *temp,*cur,*prev;
temp=(NODE *)malloc(sizeof(NODE));
temp->info=item;
temp->left=NULL;
temp->right=NULL;
if(root==NULL)
return temp;
prev=NULL;
cur=root;
while(cur!=NULL)
{
prev=cur;
cur=(item<=cur->info)?cur->left:cur->right;
}
if(item<prev->info)
prev->left=temp;
else
prev->right=temp;
return root;
}
NODE *construct(NODE *root)  // This function is for constructing the binary search tree
{
int a,n,i;
```

```
printf("Enter the number of elements\n");
scanf("%d",&n);
printf("Enter the elements\n");
for(i=0;i<n;i++)
{
scanf("%d",&a);
root=insert(a,root);
}
printf("Tree constructed successfully…..\n");
return root;
}
void preorder(NODE *root)
{
if(root!=NULL)
{
printf("%d\t",root->info);
preorder(root->left);
preorder(root->right);
}
}
void inorder(NODE *root)
{
if(root!=NULL)
{
inorder(root->left);
printf("%d\t",root->info);
inorder(root->right);
}
}
void postorder(NODE *root)
{
if(root!=NULL)
{
postorder(root->left);
postorder(root->right);
printf("%d\t",root->info);
}
}
int search(NODE *root,int key)
{
NODE *cur;
int n=0;
cur=root;
if(cur!=NULL)
{
```

```
if(key==cur->info)
{
n=1;
return n;
}
else if(key<cur->info)  // If key is less than the current node then search the key in left subtree
return search(cur->left,key);
else  // If key is greater than the current node then search the key in right subtree
return search(cur->right,key);
}
else
return 0;
}
NODE *minvalue(NODE* node)
{
NODE *current = node;
while(current->left != NULL)  // Loop to find the leftmost leaf
current=current->left;
return current;
}
NODE *del(NODE *root,int key)
{
if(root==NULL)
return root;
if(key<root->info)
root->left=del(root->left, key);
else if(key>root->info)
root->right=del(root->right, key);
else
{
if(root->left==NULL)  // Node with only one child or no child
{
NODE *temp = root->right;
free(root);
return temp;
}
else if(root->right==NULL)
{
NODE *temp = root->left;
free(root);
return temp;
}
else  // Node with two children
{
NODE *temp=minvalue(root->right);
```

```
root->info=temp->info;
root->right=del(root->right, temp->info);
}
}
return root;
}
void main()
{
int item,ch,key,n;
NODE *root;
clrscr();
root=NULL;
while(1)
{
printf("\n1.Construct BST\n2.Preorder\n3.Inorder\n4.Postorder\n5.Search an Element\n6.Delete an Element\n7:Exit\n");
printf("Enter choice\n");
scanf("%d",&ch);
switch(ch)
{
case 1: root=construct(root);
        break;
case 2: preorder(root);
        break;
case 3: inorder(root);
        break;
case 4: postorder(root);
        break;
case 5: if(root==NULL)
        printf("List Empty\n");
        else
        {
        printf("Enter the element\n");
        scanf("%d",&key);
        n=search(root,key);
        if(n)
        printf("Key found\n");
        else
        printf("Not found\n");
        }
        break;
case 6: if(root==NULL)
        printf("List Empty\n");
        else
        {
```

```
            printf("Enter the element\n");
            scanf("%d",&key);
            n=search(root,key);
            if(n)
            {
            root=del(root,key);
            printf("Element deleted\n");
            }
            else
            printf("Not found\n");
            }
            break;
case 7: exit(0);
default: printf("Wrong Choice\n");
}
}
}
```

Output:

```
1. Construct BST
2. Preorder
3. Inorder
4. Postorder
5. Search an Element
6. Delete an element
7. Exit
Enter the choice
1
Enter the number of elements
3
Enter the elements
4
6
1
Tree constructed successfully…..
1. Construct BST
2. Preorder
3. Inorder
4. Postorder
5. Search an Element
6. Delete an element
7. Exit
Enter the choice
2
```

4 1 6
1. Construct BST
2. Preorder
3. Inorder
4. Postorder
5. Search an Element
6. Delete an element
7. Exit
Enter the choice
3
1 4 6
1. Construct BST
2. Preorder
3. Inorder
4. Postorder
5. Search an Element
6. Delete an element
7. Exit
Enter the choice
4
1 6 4
1. Construct BST
2. Preorder
3. Inorder
4. Postorder
5. Search an Element
6. Delete an element
7. Exit
Enter the choice
5

Enter the element
6
Key found
1. Construct BST
2. Preorder
3. Inorder
4. Postorder
5. Search an Element
6. Delete an element
7. Exit
Enter the choice
6
Enter the element
4
Element deleted
1. Construct BST
2. Preorder
3. Inorder
4. Postorder
5. Search an Element
6. Delete an element
7. Exit
Enter the choice
7

11. Design, Develop and Implement a Program in C for the following operations on Graph (G) of Cities

a. Create a Graph of N cities using Adjacency Matrix.

b. Print all the nodes reachable from a given starting node in a digraph using BFS method

c. Check whether a given graph is connected or not using DFS method.

Program:

```
#include<stdio.h>
#include<conio.h>
int a[10][10],q[10],visit[10],n,i,j;
void bfs(int v)
{
static int f=0,r=-1;
for(i=0;i<n;i++)
if(a[v][i]==1 && visit[i]==0)
q[++r]=i;
if(f<=r)
{
visit[q[f]]=1;
bfs(q[f++]);
}
}
void dfs(int u)
{
int i;
visit[u]=1;
for(i=0;i<n;i++)
if(a[u][i]==1 && visit[i]==0)
dfs(i);
}
void main()
{
int v,f=1;
clrscr();
printf("Enter the number of nodes\n");
scanf("%d",&n);
printf("Enter the adjacency matrix\n");
for(i=0;i<n;i++)
for(j=0;j<n;j++)
scanf("%d",&a[i][j]);
for(i=0;i<n;i++)
visit[i]=0;
printf("Enter the starting vertex\n");
scanf("%d",&v);
```

```
visit[v]=1;
bfs(v);  // This function is for finding the reachable nodes from the given starting vertex
printf("Nodes reachable from vertex %d:\n",v);
for(i=0;i<n;i++)
if(visit[i]==1 && i!=v)
printf("%d\n",i);
for(i=0;i<n;i++)
visit[i]=0;
dfs(0);  // This function is for finding whether the graph is connected or not
for(i=0;i<n;i++)
if(visit[i]==0)
{
f=0;
break;
}
if(f==0)
printf("The given graph is not connected\n");
else
printf("The given graph is connected\n");
getch();
}
```

Output 1:

```
Enter the number of nodes
3
Enter the adjacency matrix
0 1 0
1 0 0
0 0 0
Enter the starting vertex
0
Nodes reachable from vertex 0:
1
The given graph is not connected
```

Output 2:

```
Enter the number of nodes
3
Enter the adjacency matrix
0 1 1
1 0 0
1 0 0
Enter the starting vertex
0
Nodes reachable from vertex 0:
1
2
The given graph is connected
```

12. Given a File of N employee records with a set K of Keys(4-digit) which uniquely determine the records in file F. Assume that file F is maintained in memory by a Hash Table(HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT. Let the keys in K and addresses in L are Integers. Design and develop a Program in C that uses Hash function H: K ®L as H(K)=K mod m (remainder method), and implement hashing technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing.

Program:

```
#include<stdio.h>
#define m 20
int HT[10];
int hash(int key)  // This function will return the index where the key has to be placed in the table
{
return key%m;
}
void linear_probe(int hk,int key)  // This function will solve the collision
{
int i,flag=0;
for(i=hk+1;i<m;i++)
{
if(HT[i]==999)
{
HT[i]=key;
flag=1;
break;
}
}
for(i=0;i<hk&&flag==0;i++)
{
if(HT[i]==999)
{
HT[i]=key;
flag=1;
break;
}
}
if(!flag)  // If flag is not set to 1 indicates that the key is not placed in the table
printf("HASH Table is Full!!!\n");
}
void main()
{
FILE *fp;
int N,i,key,hk;
char name[100];
```

```
for(i=0;i<m;i++)
HT[i]=999;  // Initialize all the table values to 999
fp=fopen("emp.txt","r");  // Emp file includes key of an employee and name of employee
while(!feof(fp))  // Till the file reaches end, the contents of file will be read i.e, key and name
{
fscanf(fp,"%d%s",&key,name);
hk=hash(key);
if(HT[hk]==999)
HT[hk]=key;
else
{
printf("Collision for key %d:\n",key);
printf("Collision solved by Linear Probing\n\n");
linear_probe(hk,key);
}
}
printf("Address\tKeys\n");
for(i=0;i<m;i++)
printf("%d\t%d\n",i,HT[i]);
getch();
}
```

Output:

Address	Keys
0	200
1	101
2	999
3	999
4	999
5	999
6	999
7	999
8	999
9	999
10	999
11	111
12	999
13	999
14	999
15	999
16	999
17	999
18	999
19	999

emp.txt file contains

101 aa

111 bb

200 cc