Pointers :- Data structures ①

Reading 2-D Arrays using pointers

type arrayname [exp1][exp2]

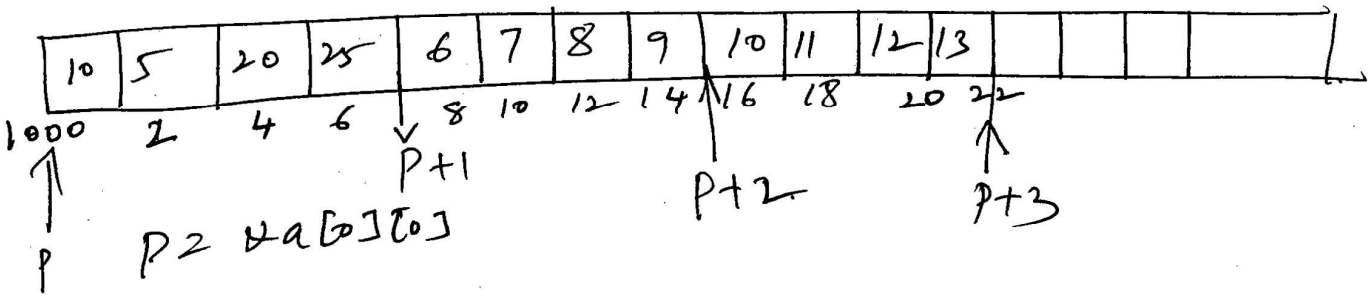
Prof. Kallinatha H. D.
Assistant Professor
Department of Computer Science & Engg.
Siddaganga Institute of Technology
TUMKUR - 572 103, Karnataka, INDIA

int a[2][3]

int (*P)[3]

	0	1	2	3
0	10	5	20	25
1	6	7	8	9
2	10	11	12	13

→ is stored in memory as follows.



P or P+0 → points to 0-row.

P+i → points to i-row.

*(P+i) → points to 0th element in the ith row.

to read → scanf("%d", *(a+i)+j);

to print → printf("%d", *(*(a+i)+j));

```
void addmatrix(int m, int n, int (*a)[10],
                              int (*b)[10], int (*c)[10])
{
    int i,j;
    for(i=0; i<m; i++)
    {
        for(j=0; j<n; j++)
        {
            *(*(c+i)+j) = *(*(a+i)+j) + *(*(b+i)+j)
        }
    }
}
```

Functions Returning pointers

```
int *larger(int *, int *);
main()
{ int a[] = {10,20,5,-5,10,};  *p
                                  → p = larger(a,5);
    printf("large elem is =%d", *p);
}
int * larger(int *x, int n);
{   int *large;
    large = x;
    for(i=1; i<n; i++)
    {   if(*large > *(x+i))
            large = (x+i);
    }
    return large;
}
```

pointers to functions

A function like a variable, has a type & can address location in the memory. It is therefore possible to declare a pointer to a function, which can then be used as an argument in another function.

```
type (*fptr) ();
```

() → are necessary because

type *gptr() → has a different meaning that is gptr is function returning pointer to a type value

Prof. Kallinatha H. D.
Assistant Professor
Department of Computer Science & Engg.
Siddaganga Institute of Technology
TUMKUR - 572 103, Karnataka, INDIA

```
int mul(int, int);
int (*p1)(int, int);
main()
{ int x, y, z;
  x=10;
  y=20;
  p1=mul;
  z=(*p1)(x, y);
printf("Result = %d", z);
}
```

```
int mul(int a, int b)
{ return(a*b);
}
```

Applications Function pointers

1. Driver Frame work for OS
2. call backs – at runtime for event handling
3. State Machines
4. Signal Handlers
5. KEIL / 8051 program for ARM
6. Array of functions

> newty.de/ fpt/ index. html

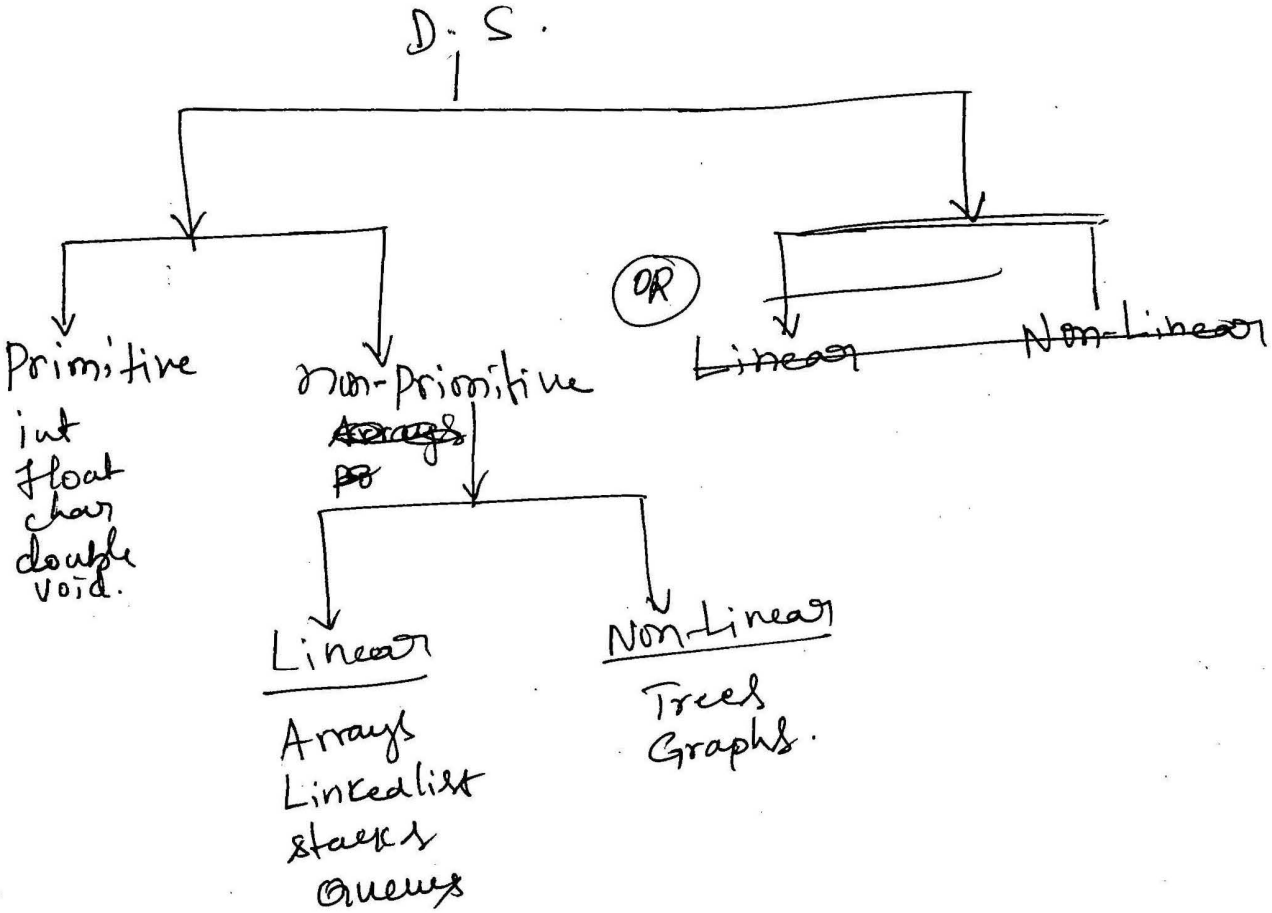
Do It yourself (DIY)

1. Implement the following ~~function~~ using function pointers
   (1) strcpy
   (2) strcat
   (3) strstr
   (4) strrev
   (5) strlen

2.

Data Structures

"Data structures is a particular way of storing and organizing data in a computer, so that it can be used efficiently."



Structures and Unions

Arrays can be used to represent a group of data items that belong to the same type; we can not use array, ~~for~~ if we want to represent a collection of data items of different types using a single name.

C supports a constructed data type known as structures, a mechanism for packing data of different types.

A structure is a convenient group of logically related data items.

"A structure is a group of items in which each item is identified by it's own identifier, each of which is known as a 'member'.

Ex:- Time : Sec, min, Hr
Date : Day, Month, Year
Book : Author, title, Price, year.

Defining a structure

Structures must be defined first for their format that may be used later to declare structure variables.

```
struct book_bank[5] → structure tag @ name
{
    char title[20];
    char author[15];
    int pages;
    float price;
};
```

Keyword { } structure elements

Structure definition describes a format called a template to represent information, shown below:

Title	Array of 20 chars
author	Array of 15 chars
pages	integer
price	float

General format of a structure definition is as follows:

```
struct        tag_name
{
        data_type        member1
                         member2
        .                .
        .                .
        .                .
};
```

(1) It is terminated by a semicolon.

(2) Entire statement defining structure is considered as a single statement.

(3) 'Tag_name' is used to declare structure variables of it's type.

Arrays	Structure
1. Collection of elements of same data type	1. Can have elements of different type
2. Derived data type	2. User-defined datatype.
3. Array behaves like a built-in data type	3. We have to design & declare the data structure.

(3) Declaring a Structure variable

```
struct book_bank, book1, book2, book3;
```

(OR)

```
struct
{
  - - -
  - - -
  - - -
} book1, book2, book3;
```

Type-defined Structures

```
typedef struct
{
  type .. member1
  :
  type  members
} type_name;
```

type name v1, v2, ...

(4) Accessing structure members

(dot) . → operator

period (or) member operator.

(5) Initialization

Rules for Initializing Structures

(1) can't Initialize Structure template.
(2) The order of values enclosed in braces & must match order of members.
(3) partial initialization is possible
(4) Uninitialized members will be assigned default values.

⑥ Copying & Comparing structure variables.

```
struct class
{   int number
    char name[20];
    float marks;
};
```

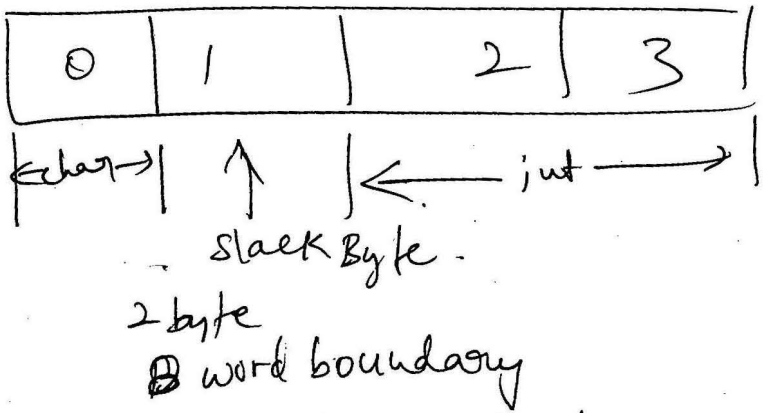
S1 = S2 ;

S1 == S2 ✗
S1 != S2 ✗

WAP to illustrate the comparison of str variable

Slack Bytes → word boundary

The unoccupied byte between 2 word boundaries is known as Slack Byte.



2 byte
⊞ word boundary

⑦ operations on individual members

(3) Declaring Structure Variables

It is similar to the declaration of variables.

```
struct book_bank book1, book2;
```

book_bank → Tag_name; book1, book2 → variables.

"Members of a structure themselves are not variables. They do not occupy any memory until they are associated with the structure variables."

"Where the compiler comes across a declaration statement, it reserves memory space for the structure variables."

We can define & declare a structure as follows. as well.

```
struct book_bank
{
    char title[20];
    char author[15];
    int pages;
    float price;
} book1, book2;
```

```
struct (optional ?)
{
    ---
    ---
    ...
} book1, book2;
```

Type defined structures:-

```
typedef struct
{
    ...
    datatype member;
    ...
} type_name;
```

```
type_name book1, book2;
...
```

④ Accessing Structure Members.

```
struct personal
{
    char name[20];
    int day;
    char month[10];
    int year;
    float salary;
};
main()
{
    struct personal person;
    printf("Input values \n");
    scanf("%s %d %s %d %f", person.name, &person.day
    person.month, &person.year
    &person.salary);

    printf("%s %d %s %d %f", person.name, person.day,
    person.month, person.year,
    person.salary);
}
```

⑤ Structure Initialization

Operations on Individual members

```
student.marks += 10.00;
student.num++;
++student.num;
```

3-ways of Accessing Members

```
typedef struct
{
    int x;
    int y;
} VECTOR;
```

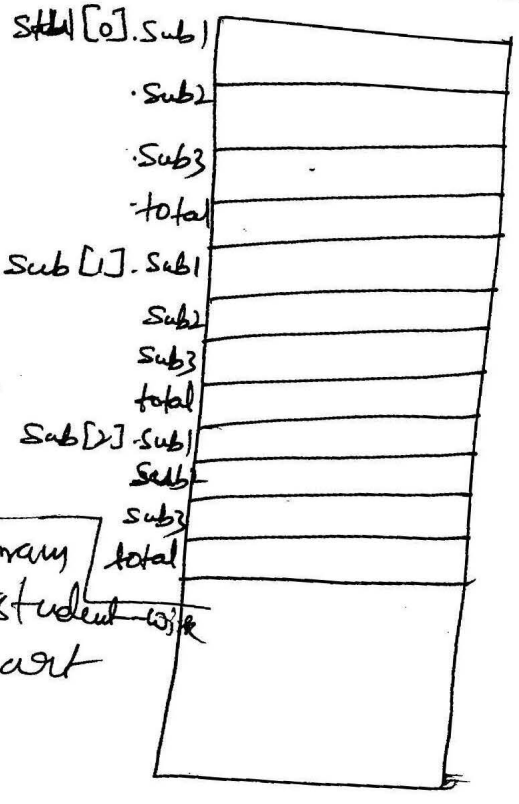
```
VECTOR v, *ptr;
ptr = &v;
```

(1) v.x
(2) (*ptr).x
(3) ptr->x

selection operator (.)

Arrays of Structures

```
Struct marks
{
    int sub1;
    int sub2;
    int sub3;
    int total;
};
struct marks student[3];
```



For the student array, write a program to calculate the subject-wise & student-wise total & store them as a part of the structure.

```
{   int i;
    printf("Enter 3 student marks");
     for(i=0; i<3; i++)
        scanf("%d%d%d", &student[i].sub1, &student[i].s
struct marks total;                    .&student[i].sub
  for(i=0; i<3; i++)
  {  student[i].total = student[i].sub1 + student[i].sub2 +
                                          student[i].sub3;
     total.sub1 += student[i].sub1;
     total.sub2 += student[i].sub2;      } Individual subjects
     total.sub3 += student[i].sub3;
     total.total += student[i].total; //grandtotal.
  }
printf("STUDENT          Total \n\n");
     for(i=0; i<3; i++)
        printf("student[%d]      %d\n", i+1, student[i].tot
printf("\n subject        total \n\n");
printf("%s    %d\n %s    %d\n %s    %d\n",
       "sub 1", total.sub1,
       "sub2", total.sub2,
       "sub3", total.sub3);
printf("\n Grand Total = %d\n", total.total);
}
```

(10.9) Arrays within Structures

```
int main()
{
    struct marks
    {
        int number;
        float subject[3];
    }student[2];

struct marks student[3] = { {30,40,70,0},
                            {40,70,80,0},
                            {50,60,70,0} };

    struct marks total;
    int i, j;

for(i=0; i<3; i++)
  {  for(j=0; j<3; j++)
      {
          ~~total.subject[i]~~
          student[i].total += student[i].subject[j];

          ~~total.total~~
          total.subject[i] += student[i].subject[j];
      }
      total.total += student[i].total;
  }
```

Structures within structures

```
struct salary
{
    char name;
    char dept;
    int basic_pay;
    int da;
    int hra;
    int CA;
} employee;
```

⇒

```
struct salary
{   char name;
    char department;
    struct
    {   int da;
        int hra;
        int CA;
    } allowance;
} employee;
```

To Access `employee.allowance.dearness;`

```
struct pay
{   int da;
    int HRA;
    int CA;
};
```

```
struct salary
{   char name;
    char dept;
    struct pay allowance;
    struct pay arrears;
};
```

```
struct salary employee[100];
employee[i].allowance.da;
```

(11) Structures & functions :-

3 ways to pass values of structures to functions. (9)

1. To pass each member of the structure member individually.
2. passing a copy of the entire structure to the called function.
3. pointers to pass the structure as an argument

---

(2) function_name (structure variable name);

```
data_type fun_name( struct_type st_name)
 {
    . . .
    . . .
    . . .
    return(expression);
 }.
```

---

(1)

```
typedef struct
 {   int x, y;
 }POINT;
void display(int, int);
int main()
 {  POINT P1 = {2,3};
    display(P1.x, P1.y);
   return 0;
 }
void display(int a, int b)
 {
   printf("The coordinates of the point are: %d %d", a, b);
 }
```

(2) passing the Entire structure

```
void display(POINT);
int main()
  {
     POINT P1 = {10,20};
      display(P1);
      return 0;
  }
void display(POINT P)
  {
    printf("%d %d", P.x, P.y);
  }
```

(3) passing structure address

```
struct student
  {
     int r_no;
     char name[20];
     char course[20];
     int fees;
  };
void display(struct student *);
int main()
  {
     struct student S1 = {123, "Gandhi", CSE, 12345};
      display(&S1);
     return 0;
  }
```

WAP to read, display, add & subtract two distances. Distance must be defined using kms & mts.

```
void display(struct Student *p)
{
    printf("Roll no. = %d", p->r_no);
    printf("Name = %s", p->name);
    printf("Course = %s", p->course);
    printf("Fees = %d", p->fees);
}
```

Union

```
typedef struct PT1
{ int x, y;
};
typedef union PT2
{ int x, y;
};
int main()
{ point P1 = {2,3};
  Point P2;
  P2.x = 4;
  P2.y = 5;
  printf("P1(x,y) = (%d,%d)", P1.x, P1.y);
  printf("P2(x,y) = (%d %d", P2.x, P2.y);
}
```

WAP to create a Generic array which can take char @ int @ float type data at run time.

WAP to store info. of a student either name (or) USN & marks using data provided at runtime.

```
struct student
{   union
    {  char name[20];
       int roll_no;
    };
   int marks;
};
```

Size of structures

Total no. of records in a DB.

Bit fields :- is a set of adjacent bits whose size can be from 1 to 16 bits in length.

- Datatype - int, unsigned (or) signed

points
1. starts at first bit of the word
2. No overlapping
3. Bits can be unnamed - padding : usigned : bit-len
4. bits can be unused
5. address of bit fields can't be obtained
6. Bits field can't be arrayed
7. values of bits field should be in range

File Management in C

File: is a collection of bytes stored on a secondary storage device.

operations
1. naming
2. open
3. read
4. write
5. closing

(2) Defining & opening a file.

To use file we need to specify to O.S:
(1) Filename – primary name + optional period.
(2) D.S. – FILE in library of stdio.h
(3) purpose.

```
FILE *fp;
fp = fopen ("filename", "mode");
```

r, w, a.
r+ w+ a+

(3) closing a file

```
fclose(file_ptr);
```

(4) I/O on files

```
putc(c, fp) – output to file
c = getc(fp);
```

"WAP to read data from the keyboard, write it to a file called INPUT, again read the same data from the INPUT file & display it on the screen".

```
main()
{
    FILE *fp1;
    char c;
    printf("Data Input \n\n");
    f1 = fopen("Input", "w");
    while ((c = getchar()) != EOF)
        putc(c, f1);
    fclose(f1);
    f1 = fopen("Input", "r");
    while ((c = getc(f1)) != EOF)
        printf("%c", c);
    fclose(f1);
}
```

getW & putW for integer

File named Data contains a series of integer numbers. code a program to read these numbers & then write all 'odd' numbers to a file to be called 'odd' & all even numbers to a file 'even'

```
main()
{ FILE *f1, *f2, *f3;
  int no, i;
  printf("Contents of Data file\n\n");
```

```
f1 = fopen("Data", "w");
for(i=1; i<=30; i++)
{   scanf("%d", &no);
    if(no == -1) break;
    putw(no, f1);
}
fclose(f1);
f1 = fopen("Data", "r");
f2 = fopen("ODD", "w");
f3 = fopen("EVEN", "w");
while((no = getw(f1)) != EOF)
{   if(no%2 == 0)
        putw(no, f3);
    else
        putw(no, f2);
}
fclose(f1);
fclose(f2);
fclose(f3);
f2 = fopen("ODD", "r");
f3 = fopen("EVEN", "r");
printf("contents of ODD file\n");
while(no = getw(f2)) != EOF)
    printf("%4d", no);
printf("EVEN file\n");
while((no = getw(f3)) != EOF)
    printf("%4d", no);
```

```
fclose(f2);
fclose(f3);
```

fprintf & fscanf functions.

fprintf(*fp, "control string", list);

WAP to open a file named INVENTORY, store in it the following data

~~struct INVENTORY~~

INAME INUM, price, Qua

~~INVENTORY item[10];~~

typedef struct

{

~~int~~ char item[10];

~~char filename[20];~~

int ina, qty;

float price;

} INVENTORY.

Extend the program to read this data from the file INVENTORY and display the inventory table with the value of each item.

(5) Error Handling During I/O operations (13)

Error may occur in the following situations:

(1) Trying to read beyond the end-of-file mark;
(2) Device overflow.
(3) Trying to use a file that has not been opened.
(4) Trying to do operation for which file was not opened.
(5) Opening a file with an invalid filename.
(6) Attempting to write to a write-protected file.

feof(fp) → 0 → not EOF

non-zero → if EOF has occured.

ferror(fp) → 0 → no error

non-zero → if Error

```
main()
{  char *filename;
   FILE *fp1, *fp2;
   int i, num;
   fp1 = fopen("TEST", "w");
    for(i=10; i<=100; i+=10)
       putw(i, fp1);
   fclose(fp1);
  printf("\n Input filename\n");
open file:
   scanf("%s", filename);
```

```
if((fp2 = fopen(filename,"r"))==NULL)
{
    printf("cannot open the file.\n");
    printf("Type filename again.\n\n");
    goto Openfile;
}
else
 for(i=1; i<=20; i++)
 {   num=getw(fp2);
     if(feof(fp2))
     { printf("\nRan out of data.\n");
       break;
     }
     else
       printf("%d\n", num);
 }
 fclose(fp2);
}
```

⑥ Random Access to files

① n = ftell(fp);

② rewind(fp) → beginning

③ fseek(fp, offset, position);

offset → displacement

position → 0 – Beginning of file
1 – current position
2 – End of file.

Ex: fseek(fp, 0L, 0) → rewind
— " — 1) → current pos
— " — 2) → End

fseek(fp, m, 0) → (m+1)th byte from beginning.
fseek(fp, m, 1) → from current pos m bytes
−m → backward

```
main
{
  FILE *fp;
  long n; char c;
  fp=fopen("Random", "w");
  while((c=getchar())!=EOF)
      putc(c,fp);
  printf(" NO. of char = %ld \n", ftell(fp));
  fclose(fp);
  fp=fopen("Random", "r");

  n=0L;
```

```
while (feof(fp) == 0)
{
   fseek(fp, n, 0);
   printf("pos of %c is %d\n", getc(fp), ftell(fp));
   n = n + 5L;
}
fseek (fp, -1L, 2);
do {
   putchar(getc(fp));
   } while( ! fseek (fp, -2L, 1));
   fclose(fp);
}
```

WAP to append additional items to the INVENTORY created earlier & print total Contents of the file

Command Line Arguments.

```
void main(int argc, char *argv[])
{
    . . . . .
    . . . . . .
    . . . . .
}
```

argc → Count

argv → Array of char pointers

```
$copy x.c y.c
```

argc = 3

argv[0] → copy
argv[1] → x.c
argv[2] → y.c.

```
#include<stdio.h>
#include <process.h>
int main(int argc, char *argv[])
{
    FILE *fp1, *fp2;
    int ch;
    if(argc<3)
    { printf("src/dest files missing in the Cmd line\n");
      exit(0);
    }
    fp1 = fopen(argv[1], "r");
    if(fp1==NULL)
    { printf("The source file can not be opened for reading\n");
      exit(0);
    }
    fp2 = fopen(argv[2], "w");
    if(fp2==NULL)
    { printf("Dest file not opened\n");
      exit(0);
    }
```

```
while( (ch = getc(fp1)) != EOF)
        putc(ch, fp2);

    fclose(fp1);
    fclose(fp2);

return 0;
}
```

(16)

WAP to create a str called Student (USN, name, M1, M2, M3) & Write functions to (1) Search record based USN.
(2) append record.
(3) display record

```
typedef struct
{
    int usn;
    char name[20];
    int m1;
    int m2;
    int m3;
}STUDENT;
```

```
int search record(int Key, FILE *fp);
void append record(FILE *fp);
void display_records(FILE *fp);
```

```
void int main()
{
    STUDENT st;
    char fname[10];
    FILE *fp;
    int key, usn, flag, ch;
printf("\n Enter the file name\n");
scanf("%s", fname);

for(;;)
{   printf(" 1: Insert Record 2: Search record\n");
    printf(" 3. Display record 4. Quit\n");
    printf(" Enter the choice\n");
    scanf("%d", &ch);
```

```
switch(choice)
{   case 1:
        fp = fopen(fname, "a+");
        if(fp == NULL)
        {   printf("Fopen failed \n");
            break;
        }
        append_record(fp);
        fclose(fp);
        break;
    case 2: fp = fopen(fname, "r");
        if(fp == NULL)
        {   printf("File open failed \n");
            break;
        }
        printf("Enter USN:");
        scanf("%d", &key_usn);
        flag = search_record(key_usn, fp);
        if(flag == 0)
            printf("unsuccessful search \n");
        else
            printf("Successful search \n");
        fclose(fp); break;
    case 3: fp = fopen(fname, "r");
        display_record(fp);
        fclose(fp);
        break;
} } } default: exit(0);
```

1.Employee database

```
#include <stdio.h>
#include <stdlib.h>
#include<string.h>
typedef struct
{
   unsigned  id;
   char name[20];
   unsigned salary;
   char dept[10];
   unsigned age;
}EMPLOYEE;

void fnAddData(FILE *fp,int n)
{
   EMPLOYEE e;
   int i;
   for(i=0;i<n;i++)
   {
    printf("\nenter the record no:%d\n",i+1);
    printf("\nenter EMP-id:");
    scanf("%d",&e.id);
    getchar();
    printf("\nenter Emp-name:");
    scanf("%s",e.name);
     printf("\nenter Emp-Salary:");
    scanf("%d",&e.salary);
    getchar();
    printf("\nenter Emp-dept:");
    scanf("%s",e.dept);
    printf("\nenter EMP-age:");
    scanf("%d",&e.age);

      fprintf(fp,"%d\t%s\t%d\t%s\t%d\n",e.id,e.name,e.salary,e.dept,e.age);

   }

}

void display(FILE *fp)
{
   EMPLOYEE e;
     printf("\n Emp-id       Emp-name    Emp-salary   DEPT.      AGE\n");
   while(fscanf(fp,"%d%s%d%s%d",&e.id,e.name,&e.salary,e.dept,&e.age)!=EOF)
   {

      printf("\n%u          %s          %u          %s          %u \n",e.id,e.name,e.salary,e.dept,e.age);
   }
}

int search_id(FILE *fp,unsigned id)
{
```

```
  EMPLOYEE e;
  printf("\n Emp-id      Emp-name    Emp-salary   DEPT.     AGE\n");
  while(fscanf(fp,"%u%s%u%s%u",&e.id,e.name,&e.salary,e.dept,&e.age)!=EOF)
  {

    if(id==e.id)
    {

      printf("\n%u        %s        %u        %s        %u
\n",e.id,e.name,e.salary,e.dept,e.age);
      return 1;
    }
  }

  return 0;
}


int search_sal(FILE *fp,unsigned sal)
{
  EMPLOYEE e;
  int count=0;
      printf("\n Emp-id      Emp-name    Emp-salary   DEPT.     AGE\n");
  while(fscanf(fp,"%u%s%u%s%u",&e.id,e.name,&e.salary,e.dept,&e.age)!=EOF)
  {

    if(sal==e.salary)
    {

      printf("\n%u\t%s\t%u\t%s\t%u \n",e.id,e.name,e.salary,e.dept,e.age);
      count++;
    }
  }

 if(count==0)
 return 0;
}

int search_age(FILE *fp,unsigned age)
{
  EMPLOYEE e;
  int count=0;
    printf("\n Emp-id      Emp-name    Emp-salary   DEPT.     AGE\n");
  while(fscanf(fp,"%u%s%u%s%u",&e.id,e.name,&e.salary,e.dept,&e.age)!=EOF)
  {
    if(age==e.age)
    {

      printf("\n%u     %s     %u     %s     %u \n",e.id,e.name,e.salary,e.dept,e.age);
      count++;
    }
  }
```

```
  if(count==0)
  return 0;
}

int search_dept(FILE *fp,char dept[])
{
   EMPLOYEE e;
   int count=0;
      printf("\n Emp-id  \tEmp-nam  \tEmp-salar\t  DEPT.\t      AGE\n");
   while(fscanf(fp,"%u%s%u%s%u",&e.id,e.name,&e.salary,e.dept,&e.age)!=EOF)
   {

      if(strcmp(e.dept,dept)==0)
      {

         printf("\n%u \t%s \t %u\t %s\t%u \n",e.id,e.name,e.salary,e.dept,e.age);
      }
   }

    if(count==0)
     return 0;
}
int main()
{
   int n,ch,flag;
   EMPLOYEE key;
   FILE *fp;
   printf("MENU for Employee DTA base");
    while(1)
   {
   printf("\n 1.ADDdata\n2.display\n3.search record-id\n 4.search record-salary\n5.search record-
age\n6.search record-dept\n");
   printf("Enter your chpice\n");
   scanf("%d",&ch);

      switch(ch)
      {
         case 1:fp=fopen("EMP_DATA","a+");
                  if(fp==NULL)
                  {
                     printf("file operation error\n");
                     break;
                  }
               printf("enter the num of records to be entered\n");
               scanf("%d",&n);
               fnAddData(fp,n);
               fclose(fp);
               break;
         case 2: fp=fopen("EMP_DATA","r");
                  if(fp==NULL)
                    {
```

```
            printf("file operation error\n");
            break;
          }
          display(fp);
           fclose(fp);
          break;

case 3: fp=fopen("EMP_DATA","r");
            if(fp==NULL)
            {
            printf("file operation error\n");
            break;
            }
           printf("enter the emp-id to be searched\n");
           scanf("%u",&key.id) ;
           flag= search_id(fp,key.id);
           if(flag==1)
           printf("\n serach succesful");
           else
            printf("\n serach UNsuccesful");
             fclose(fp);
            break;
  case 4:fp=fopen("EMP_DATA","r");
            if(fp==NULL)
        {
            printf("file operation error\n");
            break;
        }
           printf("enter the emp-salary to be searched\n");
           scanf("%u",&key.salary) ;


           flag=search_sal(fp,key.salary);

           if(flag==1)
           printf("\n serach succesful");
           else
            printf("\n serach UNsuccesful");
             fclose(fp);
            break;
    case 5:fp=fopen("EMP_DATA","r");
            if(fp==NULL)
            {
            printf("file operation error\n");
            break;
            }
           printf("enter the emp-age to be searched\n");
           scanf("%u",&key.age) ;
           flag= search_age(fp,key.age);
           if(flag==1)
           printf("\n serach succesful");
           else
```

```
                printf("\n serach UNsuccesful");

              fclose(fp);
                break;


            case 6:fp=fopen("EMP_DATA","r");
                if(fp==NULL)
              {
                printf("file operation error\n");
                break;
              }
                printf("enter the emp-dept to be searched\n");
                scanf("%s",key.dept) ;
                flag= search_dept(fp,key.dept);
                if(flag==1)
                printf("\n serach succesful");
                else
                 printf("\n serach UNsuccesful");
                 fclose(fp);
                 break;
            default:exit(0);
    }

  }
  return 0;
}
```

2.Stack operations.

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 5
#define MAXLEN 30
typedef enum {TRUE=1,FALSE=0}bool_t;

typedef struct{
        int iStArr[SIZE];
        int iTop;
}stack_t;

void fnPush(int, stack_t*);
int fnPop(stack_t*);
int fnPeek(stack_t);
bool_t isEmpty(stack_t);
bool_t isFull(stack_t);
void fnDisplay(stack_t);
int main(void)
{
        int iChoice, iElem;
        stack_t myStack;
        myStack.iTop = -1;
        for(;;)
        {
                printf("\n1.PUSH\n2.POP\n3.PEEK\n4.DISPLAY\n5.EXIT\n");
                printf("\nEnter your choice : ");
                scanf("%d", &iChoice);

                switch(iChoice)
                {
                        case 1: if(!isFull(myStack))
                                        {
                                                printf("\nEnter element to push on to the stack\n");
                                                scanf("%d", &iElem);

                                                fnPush(iElem, &myStack);
                                        }
                                        else
                                                printf("\nSTACK OVERFLOW\n");
                                        break;
                        case 2: if(!isEmpty(myStack))
                                        {
                                                iElem = fnPop(&myStack);
                                                printf("\nElement popped is %d\n", iElem);
                                        }
                                        else
                                                printf("\nSTACK UNDERFLOW\n");
                                        break;

                        case 3: if(!isEmpty(myStack))
```

```
                                        {
                                        iElem = fnPeek(myStack);
                                        printf("\nElement at the top of the stack is %d\n", iElem);
                                        }
                                        else
                                                printf("\nSTACK IS EMPTY\n");

                                        break;
                        case 4: if(!isEmpty(myStack))
                                        {
                                                fnDisplay(myStack);
                                        }
                                        else
                                                printf("\nSTACK IS EMPTY\n");
                                        break;

                        case 5: exit(0);
                }
        }
        return 0;
}

void fnPush(int iVal, stack_t *pStk)
{
        (pStk->iTop)++;
        pStk->iStArr[pStk->iTop] = iVal;
}

int fnPop(stack_t *pStk)
{
        int iElem;
        iElem = pStk->iStArr[pStk->iTop];
        (pStk->iTop)--;
        return iElem;
}

int fnPeek(stack_t stack)
{
        return stack.iStArr[stack.iTop];
}

bool_t isEmpty(stack_t stack)
{
        if(stack.iTop == -1)
                return TRUE;
        else
                return FALSE;
}
bool_t isFull(stack_t stack)
{       if(stack.iTop == SIZE -1)
                return TRUE;
        else
```

```
        return FALSE;
}

void fnDisplay(stack_t stack)
{
        int i;
        printf("\nContents of the stack are\n");
        for(i=stack.iTop; i>=0; i--)
                printf("%d\t",stack.iStArr[i]);

        printf("\n");
}
```

7 - ((x * ((x+y) / (5-3)) + y / (4 - 2 * 5))

Dynamic Memory Management

The process of allocating memory at run time is known as DMA allocation.

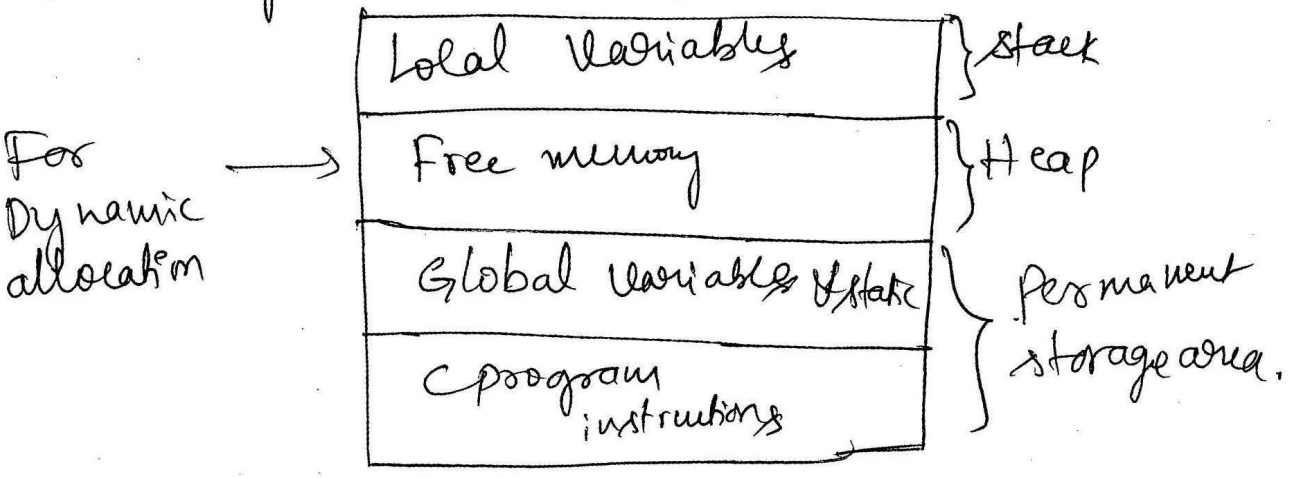
(1) malloc → Allocates request size of bytes & returns a ptr to the first byte of the allocated space

(2) calloc → Allocates space for array of elements, initializes them to zero. & returns pointer to memory.

(3) free → Frees allocated memory

(4) realloc → Modifies the size of previously allocated space.

(1) Memory allocation process



Prof. Kallinatha H. D.
Assistant Professor
Department of Computer Science & Engg.
Siddaganga Institute of Technology
TUMKUR - 572 103, Karnataka, INDIA

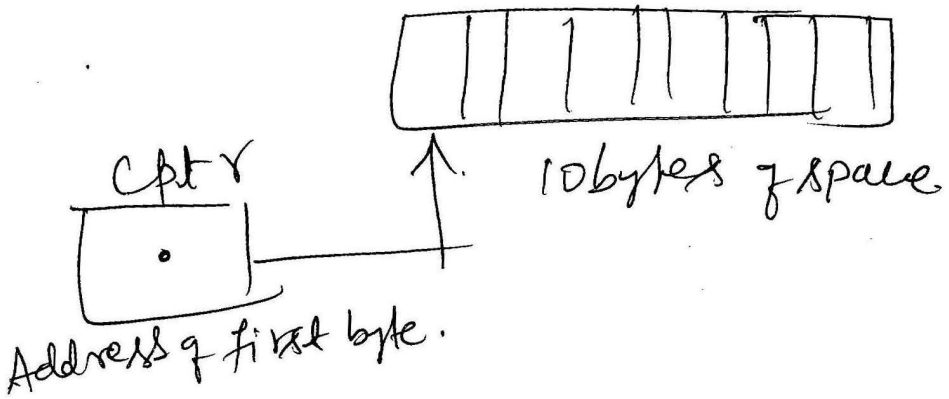
② Allocating a Block of Memory: Malloc

Malloc function ~~returns~~ reserves a block of memory of specified size & returns a pointer of type void. This mean that we need to assign it to any type of pointer

```
ptr = (cast-type*) malloc(byte_size);
```

```
x = (int*) malloc(100*sizeof(int));
```

```
cptr = (char*) malloc(10);
```




```
st_var = (struct store*) malloc(sizeof(struct store));
```

```
void * malloc( size_t size)
```

↓ Defined alloc.h/malloc.h
(unsigned int)

. W A P that uses a table of integers whose size will be specified interactively at runtime.

```
#define NULL 0

main()
{   int *p, *table;
    int size;
    printf("what is the size of table?");
    scanf("%d", &size);
    if((table = (int*) malloc(size * sizeof(int))) == NULL)
      {
        printf("No Space for allocation\n");
        exit(0);
      }
    printf("\n Address of the first byte is %u\n", table);
    printf("Input table value\n");
    for(p=table; p<table+size; p++)
      scanf("%d", p);

    for(p=table+size-1; p>=table; p--)
        printf("%d is stored at address %u\n", *p, p);

    return 0;
}
```

(13) Allocating multiple Blocks of Memory : calloc

- used to store derived data types such as arrays & structures.
- calloc allocates multiple blocks of storage.

```
ptr = (cast-type*) calloc(n, elem-size);
```

Ex:-

```
typedef struct
{
    char name[25];
    float CGPA;
    int Age;
    long int id;
} student;
student *st_ptr;
int class_size=30;
st_ptr=(record*) calloc(class_size, sizeof(record));
```

`void *calloc(size_t n, size_t size);`

```
#include<malloc.h>
int main()
{ int *p, i;
  p=(int*)calloc(3, sizeof(int));
  for(i=0; i<=2; i++)
      *(p+i) = 10*(i+1);

  for(i=0; i<=2; i++)
    printf("The value at the index %d is %d\n", i, *(ptr+i));
} return 0;
```

(3)
program to find maximum of n-numbers using dynamic array.

```
int main()
{
    int *a, i, j, n;
    printf("Enter the no. of elements \n");
    scanf("%d", &n);
    a = (int *) calloc(n, sizeof(int));
    if(a == NULL)
    {
        printf("Insufficient memory \n");
        return;
    }
    printf("Enter %d ele \n", n);
    for(i=0; i<n; i++)
    {
        scanf("%d", &a[i]);
    }                    (a+i)
    j=0;
    for(i=1; i<n; i++)
        if(a[i] > a[j]) j = i;
    printf("The biggest = %d is found in pos = %d \n",
                                    a[j], j+1);
    free(a);
}
```

Allocate memory space dynamically for a string 'str' and a substring 'substr' with 'm' & n chars.
WAP to check whether the substring is present.
If found, delete & display str.

(3) realloc

```
int main()
{ int *p,i;
   p=(int*) calloc(3, sizeof(int));
  printf("Enter 3 elements\n");
    for(i=0; i<=2; i++)
        scanf("%d", (p+i) );

  p=(int*) realloc(p, 5*sizeof(int));

  p[3]=23;
  p[4]=40;

for(i=0; i<=4; i++)
printf("value at index %d is %d \n", i, *(p+i));
 realloc(p,0);
}
```

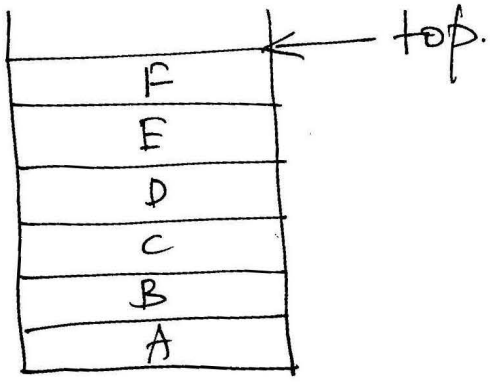
(4) Memory Leakage

- Memory reserved dynamically but not accessible to any of the program is called `memory leakage'.
- Happens when a memory is not deallocated.

```
int main()
{
  int *a;
  a=(int*) malloc(sizeof(int));
    *a=100;
  a=(int*) malloc(sizeof(int));
    *a=200;
}
```

The Stack

A stack is an ordered collection of items into which new items may be inserted and from which items may be deleted at one end, called the top of the ~~stack~~ stack.



- Adding an item 'i' on top of the stack is called 'push' & It is written as push(S, i).
- Removing an item 'i' from top of the stack is called 'pop'.
- Stack is called as 'pushdown list'.
- Empty Stack → before pop
- Just knowing the item on top of the stack without removing is called 'stacktop'(S) @ 'peek'
- Illegal attempt to pop @ access an item from an empty stack is called 'Underflow'.

Ex:- Solving mathematical expressions having several nested parenthesis.

Ex A + B

A + B − C

(A+B) * (C−D)

High | Exponentiation

* / %

low | + −

A $ B * c − D + E / F / (G + H)

((A + B) * c − (D − E)) $ (F + G)

A − B / ((* D $ E)

① 7 − ((x * ((k + y) / (J − 3)) + 4) / (4 − 2.5))

② ((A + B)

③ A + B (

④) A + B (− (

⑤ (A + B)) − (C + D

Valid = true;
S = the empty stack;
while (we have not read entire string)
{ read the next symbol of the string;
if (symb == '(' || symb == '[' || symb == '{')
push(S, symb);
if (symb == ')' ② ']' ④ '}')
if (empty(S))
valid = false;
else { i = pop(S);
if (i is not the matching opener of symb)
valid = false;
}
if (! empty(S))
valid = false;
if (valid)
else

{x + (y − [a + b]) * c − [(d + e)]} / (h − (j − (k − [l − m])))

(17)

```
#define <stdio.h>
#define <stdlib.h>
#define TRUE 1
#define FALSE 0
#define MAX 20

int top=-1;
int stkempty( )
{   if(top==-1)
        return TRUE;
    else
        return FALSE;
}

int main( )
{ char ex[MAX], st[MAX];
  int valid=TRUE;
  int i;
  char c, symb;
  printf("\nEnter expression\n");
  scanf("%s", ex);
  i=0;
while(ex[i]!='\0')
{ symb=ex[i];
  if(symb=='(' || symb=='[' || symb=='{')
        push(st, symb);
  if(symb==')' || symb==']' || symb=='}')
        if(stkempty(st))
          valid=FALSE;
        else
        {
            c=pop(st);
```

```
void push(char s[], char ch
{ if(top==MAX-1)
  { printf("\n STACK
                OVERflow\n");
    exit(0);
  }
  else {
        top++;
        s[top]=ch;
  }
}
```

```
char pop(char s[])
{ char ch;
  if(top==-1)
     printf(" UNDERFlow");
  else {
        ch=s[top];
        top--;
  } return ch;
}
```

```
if(c=='(' || c=='[' || c=='{')
    push(st, c);
else
    valid = FALSE;
}
i++;
}
if(stempty(st)==0)
    valid = FALSE;
if(valid)
    printf("%s is valid", ex);
else
    printf("%s is Invalid in", ex);
return 0;
}
```

---

① A+B — AB+ — +AB

② A+B*C — AB+C* — +*ABC

③ (A+B)*(C-D) — AB+ CD- *

④ A$B*C-D+E/F/(G+H) — AB$C*D-EF/GH+/+

⑤ ((A+B)*C-(D-E))$(F+G) — AB+C*DE--FG+$

⑥ A-B/E*D$E) — ABCDE$*/-

A+B*C

(A+B)*C

```
#define MAXSIZE 30
struct STACK
{   int top;
    int s[MAXSIZE];
} stack;
int main()
{ char postfix[20]; float eval(char *);
  printf("\nEnter a valid postfix expression:\n");
    gets(postfix);
  printf("\nResult of Evaluation = %f\n", eval(postfix));
} return 0;
float eval(char *postfix)
{   float x1, x2;
    int i=0; char ch; stack.top=-1;
  while((ch=postfix[i++]) != '\0')
  {  if(isdigit(ch))
        stack.s[++stack.top] = ch-'0';
     else
     {  x2 = stack.s[stack.top--];
        x1 = stack.s[stack.top--];
        switch(ch)
        { case '+': stack.s[++stack.top] = x1+x2;
                    break;
          case '-': stack.s[++stack.top] = x1-x2;
                    break;
          case '*': stack.s[++stack.top] = x1*x2;
                    break;
          case '/': stack.s[++stack.top] = x1/x2;
                    break;
          default: printf("Invalid operator!!!\n");
                   exit(1);
        }
     }
  } return(stack.s[stack.top]);
}
```

0 - 48
1 - 49
2 - 50
:
9 - 57

Evaluation of postfix expression

Why Evaluate an postfix expression?

(1) To Evaluate infix expression requires the knowledge of precedence of operators & associativi

(2) parenthesis in the expression makes it couple

(3) we have to scan from L to R & R to L, make it complex.

Algorithm

while end of input is not reached
BEGIN

```
symbol = read next char()
If (symbol is an operand)
        push(symbol, S);
else
        op2 = pop(S);
        op1 = pop(S);
        res = op1 optr op2;
        push(res, S);
```

END while ENDIF

Ex:- 623 + - 382 / + * 2 $3 +

Ans = 52

Algorithm to Convert from Infix to postfix (Su

1. stk = empty
2. while (not end of input) {
   symb = next char;
   if (symb is an operand)
   ~~postfix~~ add ~~it~~ symb to postfix string
   else {
   while (stack not empty && preced(symb) ≤
   preced(stack
   {
   topsymb = pop(stk);
   add topsymb to the postfix string;
   }
   push (stk, symb);
   }
   }
3. while (stack.top ~~!=~~ not Empty)
   {
   topsymb = pop(opstk);
   add topsymb to postfix
   }

A + B * C

A * B + C

symb	postfix	stack
A	A	~~+~~
+	A	+
B	AB	+
*	AB	+*
C	ABC	+
	ABC*	
	ABC*+	

	A	
	~~*~~A	~~*~~
	AB	~~*~~
	AB	
+	AB*	+
	ABC	+
	ABC*+	

(2) (A+B)*C

20

Symb	postfix string	stack
(		(
A	A	(
+	A	(+
B	AB	(+
)	AB+	
*	AB+	*
	AB+C	*
C	AB+C*	

(3) ((A-(B+C))*D)$(E+F)

Symb	postfix	stack
(		(
(		((
A	A	((.
-	A	((-
(	A	((-(
B	AB	((-(
+	AB	((-(+
C	ABC	((-(+
)	ABC+	((-
)	ABC+-	(
*	ABC+-	(*
D	ABC+-D	(*
)	ABC+-D*	

$	ABC+-D*	$(
(	ABC+-D*	$(
E	ABC+-D*E	$(
+	ABC+-D*E	$C+
F	ABC+-D*EF	$C+
)	ABC+-D*EF+	$
	ABC+-D*EF+$	

H.W

(4) (A+B)*(C-D)$E*F

(5) (A+B)*(C$(D-E)+F)-G.

Prof. Kallinatha H. D.
Assistant Professor
Department of Computer Science & Engg.
Siddaganga Institute of Technology
TUMKUR - 572 103, Karnataka, INDIA

Infix to postfix

```
struct stack {
  char item[30];
  int top;
};
int main()
{ char infix[30], suffix[20], c, ch;
  int i=0, j=0;
  int precedence(char);
  struct stack st;
  st.top=-1;
  printf("\n Enter a valid infix expression:");
    gets(infix);
    st.items[++st.top]='#';
while((ch=infix[i++])!='\0')
{ if(isalpha(ch)||isdigit(ch))
      suffix[j++]=ch;

  else
  {
    switch(ch)
    { case '(': st.items[++st.top]=ch;
      break;
      case ')': while((c=st.items[st.top--])!='(')
        suffix[j++]=c;
      break;
      default: while(prec(ch)<=prec(c=st.items[st.top]))
      { suffix[j++]=c;
        st.top--;
      }
      st.items[++st.top]=ch;
      break;

} } }
```

```
while(st->top>0)
 suffix[i++]=st.items[st.top--];
 suffix[i]='\0';
 printf("postfix is:%s", suffix);
return 0;
}
int preced(char ch)
{
 if(ch=='#'||ch=='(')
   return 0;
 if(ch=='+'||ch=='-')
   return 1;
 if(ch=='*'||ch=='/'||ch=='%')
  return 2;
  if(ch=='$'||ch=='^')
   return 3;
}
```

Infix to prefix

(24)

① A + B − C

Symb	prefix	stack.
C	C	
−	C	−
B	CB	−
+	CB	− +
A	CBA	− +
	CBA+	−
	CBA+−	

←

② (A+B) * (C−D) $ E ↑ F

symb	prefix	stack.
F	F	
↑	F	↑
E	FE	↑
$	FE	↑ $
)	FE	↑ $)
D	FED	↑ $)
−	FED	* $) −
C	FEDC	* $) −
(	FEDC−	* $
*	FEDC−$	* *
)	FEDC−$	* *)

B
+
A
(

FEDC-$B **)
FEDC-$B **)+
FEDC-$BA **)+
FEDC-$BA+ **
FEDC-$BA+* *
FEDC-$BA+**

**+AB$-CDEF

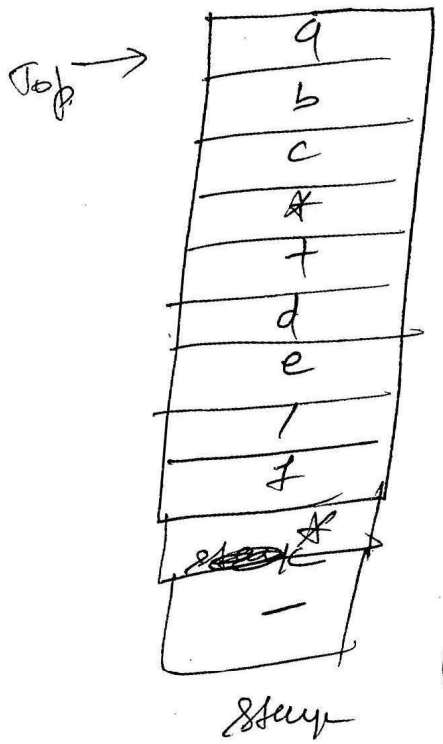
(3) (A+B)*(C$(D-E)+F)-G.

```
{ char infix[20];
  int top;
}st;
main()
{
  char prefix[20], stk[20], c, ch;
  int i, j=0;
  int precd(char);
 st.top = -1;
 printf(" Enter valid infix ");
  gets(st.infix);
  stk[++st.top] = '#';
  i = strlen(st.infix)-1;
while(i>=0)
{ ch = st.infix[i];
  if(isdigit(ch) || isalpha(ch))
  prefix[j++] = ch;
  else
  {
    switch(ch);
    {case ')' : stk[++st.top] = ch;
                break;
     case '(' : while ((c=stk[st.top--]) != ')')
                  prefix[j++] = c;
                break;
     default: while(p(c) < p(c=stk[st.top]))
              { prefix[j++] = c;
                st.top--;
              }stk[++st.top] = ch;
              break;
    }
  }
  i--;
}
```

```
while(st->top>=0)
    prefix[j++] = stk[st->top--];
    prefix[j] = '\0';

printf("\n prefix is\n");
for(i=strlen(prefix)-1; i>=0; i--)
    printf("%c", prefix[i]);
}
```

③ Conversion from postfix to Infix

= abc * + de / f * −



① a, b, c *

② a, (b*c), +

a + (b*c), d, e, /

③ a + (b*c), d/e,

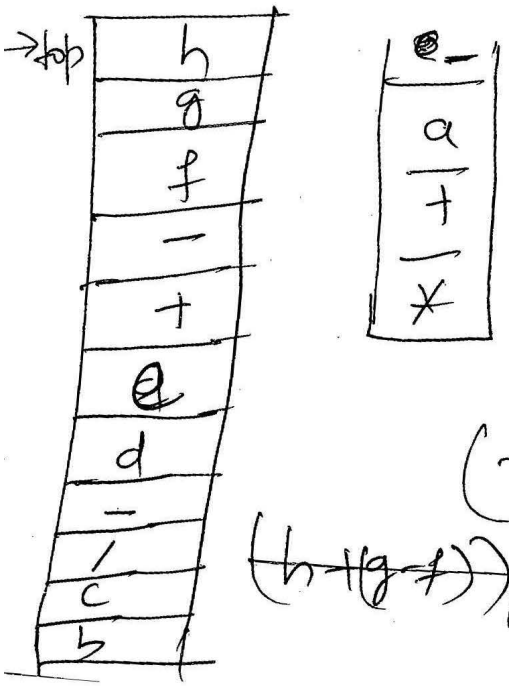
④ a + (b*c), d/e, f, *

⑤ (a+(b*c)), (d/e)*f, −

⑥ ((a+(b*c)) − ((d/e)*f))

④ Prefix to Infix

*+a−bc/−de+−fgh ←



h, g, f −

h, (g−f), +

(h+(g−f)), e, d, −

(h+(g−f)), (e−d), /

(h+(g−f))/(e−d), c, b, −

(h+(g−f))/(e−d), (c−b), −

~~(h+(g-f))/(e-d)-(c-b), a, +~~

~~(h+(g-f))/(e-d)-(c-b)+a,~~

---

h, g, f, −

h, (f-g), +

f-g+h, e, d, −

f-g+h, d-e, /

~~(~~(d-e)/(f-g+h), c, b, −

, b-c, a, +

(d-e)/(f-g+h) , a+b-c, *

~~(d-e)/(f-g+h) * (a+b-c)~~

$$= \boxed{(a+b-c) * (d-e)/(f-g+h)}$$

postfix

① AB+C−

② ABC+−

③ ABCDE−+$*EF*−

prefix

+−ABC

+A−BC

++A−*$BCD/+EF*GHI

Recursion

It is a technique for defining a problem in terms of one or more versions of the same problem. A function, which contains a call to itself is called 'recursive function'.

① Iterative function calls for the explicit repetition of some process until a certain is met.

$$fact(n) = \begin{cases} 1 & \text{if } n=0 \\ n*(n-1)*(n-2)\ldots 3\times2\times1 & \text{if } n>0 \end{cases}$$

```
fact = 1
fact = fact * 1
     = fact * 2
     = fact * 3
     = fact * 4
     :     :
fact = fact * n
```

```
int factorial (int n)
{
    int fact, i;
    fact = 1;
    for(i=1; i<=n; i++)
        fact *= i;

    return fact;
}
```

② Recursive factorial function

$$\begin{cases} n! = 1 & \text{if } n=0 \longrightarrow \text{Base case} \\ n! = n*(n-1)! & \text{if } n>0 \rightarrow \text{recursive case} \end{cases}$$

A definition, which defines an object in terms of a simpler case of itself is called 'recursive definition'.

Ex:- $5! = 5*4!$

$4! = 4*3!$

$3! = 3*2!$

$2! = 2*1!$

$1! = 1*0!$

$0! = 1$

$1! = 1*1$

$2! = 2*1! = 2$

$3! = 3*2 = 6$

$4! = 4*3! = 4*6 = 24$

$5! = 5*4!$

$= 5*24 = 120$

```
int fact(int n)
{
    if(n==0)
        return 1;
    else
    return n*fact(n-1);
}
```

Multiplication of Natural numbers

Iterative definition

a * b = 'a' added to itself b times

a + a + a ....

Recursive definition

a * b = a if b = 1;

a * b = a * (b-1) + a if b > 1

Ex :- 6 * 3

= 6 * 2 + 6

= 6 * 1 + 6 + 6

= 6 + 6 + 6

= 18

```
int MNNI(int a, int b)
{ int i;
  int sum=0;
  for(i=1 ; i<=b ; i++)
     sum= a;
  return sum;
}
```

```
int MNNR(int a, int b)
{ int i;
  int sum=1
  sum = a * MNNR(b-1) + a
  if( b==0 )
     return a
  return a + MNNR(a, b-1)
}
```

6 * 0 + 6

6 + 6

6 * 1 + 6

6 + 6

6 * 2 + 6

① GCD – Euclid's algorithm

$$GCD(a,b) = \begin{cases} b & \text{if } b \text{ divides } a \\ GCD(b, a \% b), & \text{otherwise} \end{cases}$$

Ex: a = 62 b = 8

GCD(62, 8)

rem = 62 % 8 = 6

GCD(8, 6)

rem = 8 % 6 = 2

GCD(6, 2)

rem = 6 % 2 = 0

return 2

Iteration

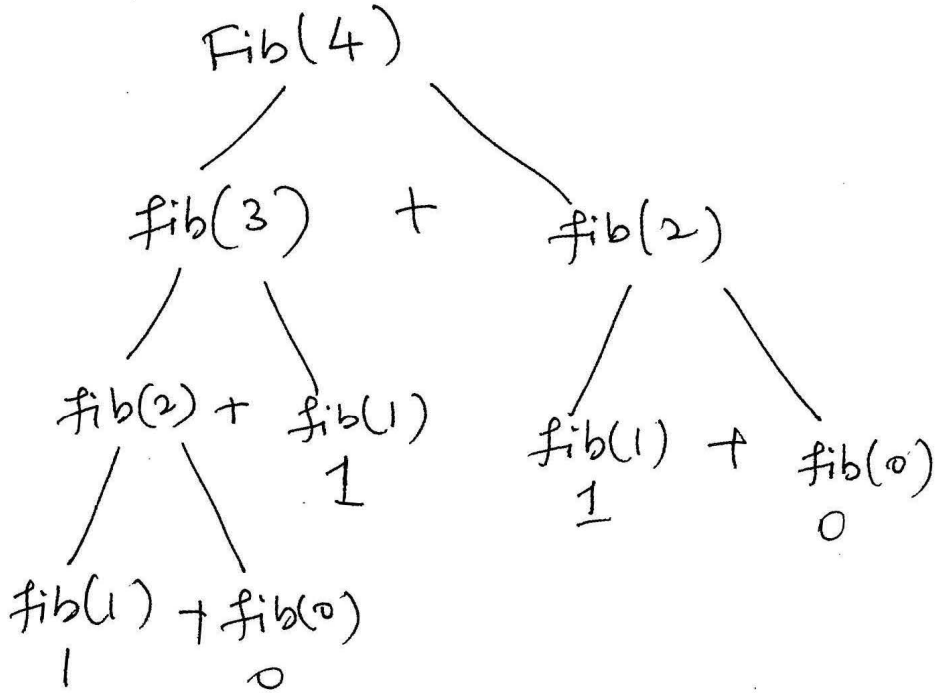
```
int GCD(int m, int n)
{ int r;
  if((m==0)||(n==0))
    return 0;
  if((m<0)||(n<0))
    m=-m  n=-n

  do {
    r = m % n;
    if(r==0)
      return n;
    m = n;
    n = r;
  }while(true);
  return n;
}
```

Recursion

```
int GCD(int x, int y)
{ int rem;
  rem = x % y;
  if(rem==0)
    return y
  else
    return(GCD(y, rem));
}
```

(4) Fibonacci Series

$$fib(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ fib(n-1) + fib(n-2) & \text{otherwise} \end{cases}$$




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

(5) Binary Search :-

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

(6) Find Maximum of n elements

```
int max(int a[], int n);
{
    int big;
    if(n == 0)
        return a[n];
    big = large(a, n-1);
    if(a[n] > big)
        return a[n];
    return big;
}
```

(7) Min of an array

```
int min(int a[], int n)
{   int small;
    if(n==0) return a[n];
    small = min(a, n-1);
    if(a[n]<small)
        return a[n];
    return small;
}
```

(8) Sum of n elements of arrays

```
int Sum(int a[], int n)
{   if(n==0) return(a[n]);
    return a[n] + Sum(a, n-1);
}
```

(9) product of n elements of array

```
int prod(int a[], int n)
{   if(n==0)
        return a[n];
    return a[n] * prod(a, n-1);
}
```

(10) Print a given no. in reverse order.

```
void reverse(int n)
{
    printf("%d", n%10);
    if(n/10==0) return;
    reverse(n/10);
}
```

(11) print array elements in reverse order

```
void rev_arr(int n, int a[])
{
    if(n==-1) return;
    printf("%d\n", a[n]);
    reverse(n-1, a);
}
```

Ancient prophecy - Rules Brahma. (legend.

$2^{64}-1$ seconds @ 585 Billion years @ 127 times. age the time.

Tower of Hanoi

1. n=1 Move A to C
2. n-1 A to B using C
3. n A to C
4. n-1 B to C using A

TOH(3, A, B, C) ④

2, A, C, B ②

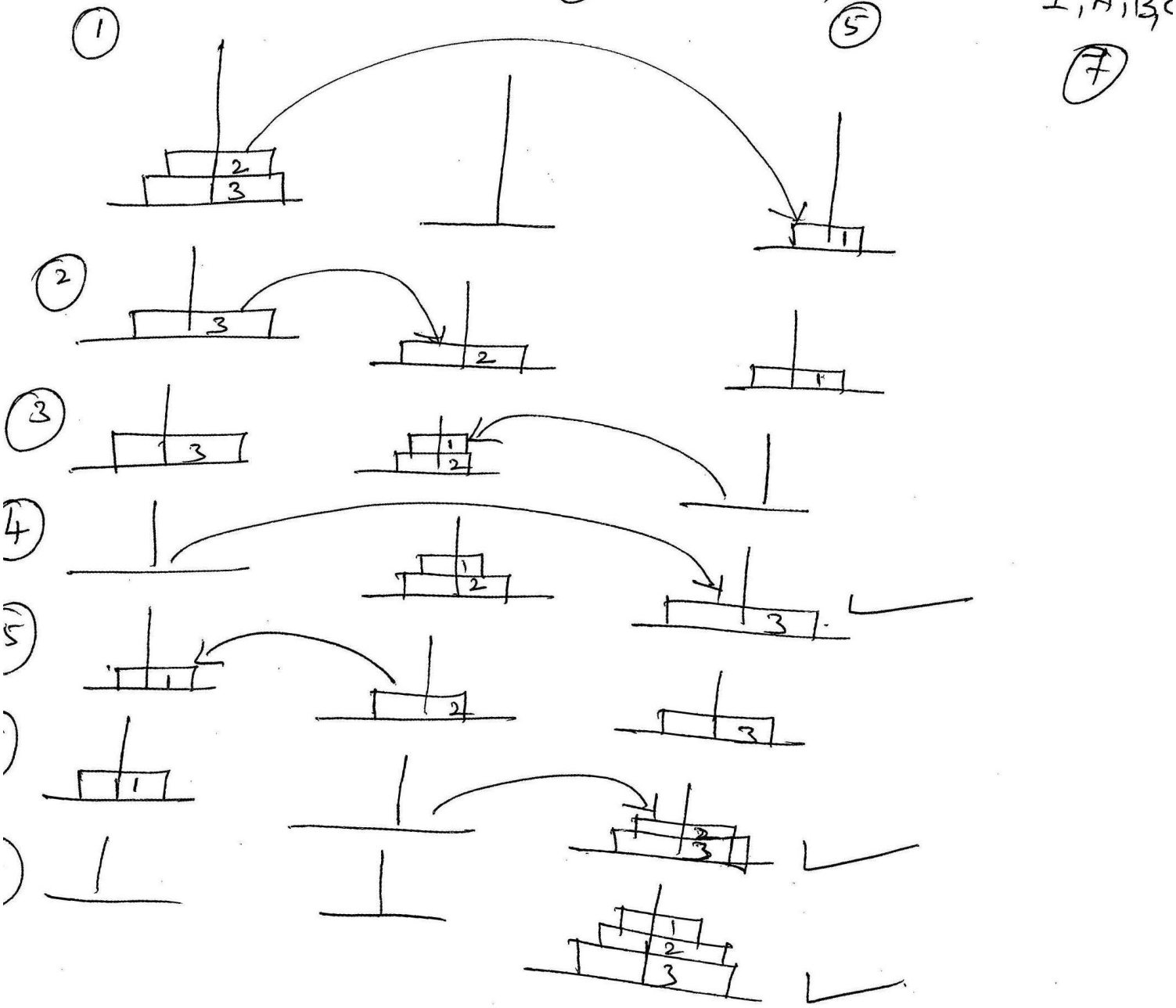
1, A, B, C ①

1, C, A, B ③

2, B, A, C ⑥

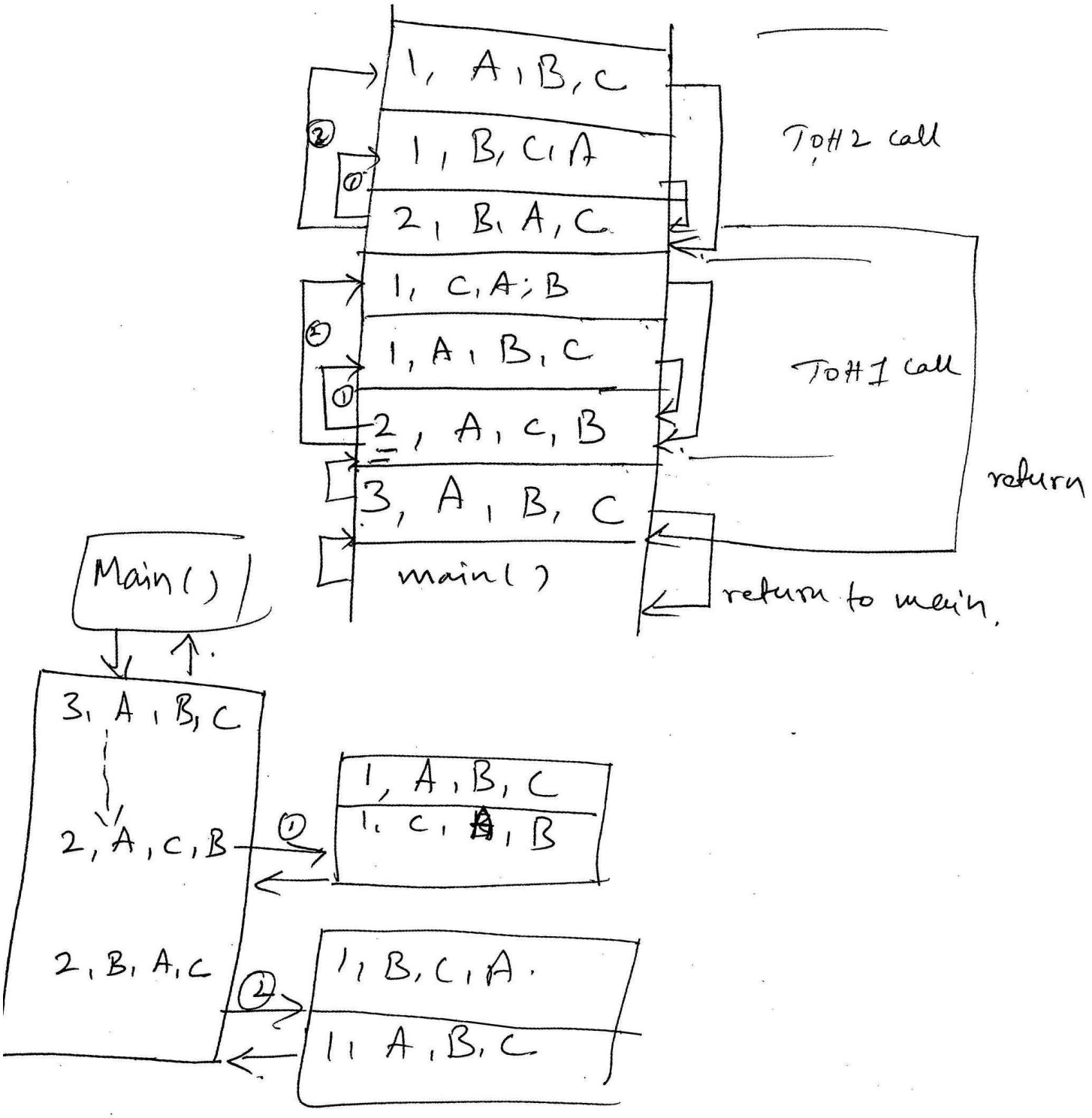
1, B, C, A ⑤

1, A, B, C ⑦



```
void TOH(int n, char from, char aux, char to)
{
    if(n==1)
    {
        printf("Move %d disk from %c to %c", n, from, to);
        return;
    }
    TOH(n-1, from, to, aux);
    printf("Move %d disk from %c to %c", n, from, to);
    TOH(n-1, aux, from, to);
}
int main()
{
    int n;
    printf("Enter the num of disks...\n");
    scanf("%d", &n);
    printf("The sequence of moves involved in TOH
            are...\n");
    TOH(n, 'A', 'B', 'C');
}
```

Stack frames for TOH



Iteration	Recursion
1) Control structure	Selection structure
2) uses repetition	repetition indirectly through recursive calls
3) Terminated when the loop condition fails	3) Terminated when the base case is satisfied.
4) Infinite if condition never fails	4) Infinite if there is no base case (or) if base case never reaches
5) Not suited for all problems (Tower of Hanoi, tree traversals)	5) Recursion is best suited.
6) Faster & occupy less memory & easy to design	6) slower, occupy more memory. & design is not straight forward.

Queues

Queues are ordered collection of items from which items can be deleted from one end (front) & added from other end called rear.

Insert

step1: If rear = MAX-1
overflow
ENDIF Goto step4

2: IF front = -1 & rear-1
f=r=0;
else
r=r+1;

3: q[r] = num;

Delete

1: IF f=-1 & f->R
underflow

else
val = q[R];
f=f+1;

2: [Exit]

Prof. Kallinatha H. D.
Assistant Professor
Department of Computer Science & Engg.
Siddaganga Institute of Technology
TUMKUR - 572 103, Karnataka, INDIA

```
#define MAX 10
int q[MAX];
int f=-1, r=-1;
void insert(void);
int dele_el(void);
void display();

void insert()
{
    int num;
    printf("Enter");
    scanf("%d", &num);
    if(rear==MAX-1)
        printf("\n overflow");
        return;
    else if(f==-1 && r==-1)
        f=r=0;
    else
        r++;
    q[r]=num;
}
int del_el()
{   int val;
    if(f==-1 || f>r)
    {   printf("under flow");
        return -1;
    }
    else  val=q[f];
          f++;
          if(f>r)
              f=rear=-1
    return val;
}
```

```
void display( )
{  int i;
   if(f==-1 ; f>r)
        printf("Empty");

   else
   { for(i=f; i<=r; i++)
        printf("\n%d", q[i]);
   }
}
```

(2) struct

```
typedef struct
{  int item[100];
   int r, f;
}QUEUE;

QUEUE q;

q.r = q.f = -1;  int n
void insert( );
int del( );
void display( );

insert(
{   if(q.r==MAX-1)
        over

    elsif( q.r==-1 && q.f==-1)
        q.r = q.f = 0;

    else r++;
```

```
#include <stdbool.h>
#define MAX 5
typedef struct
{   int front, rear;
    int items[MAX];
} Q_t;
void fnQIn(Q_t *, int);
int fnQDel(Q_t *);
void fnDisplay(Q_t);
bool isQEmpty(Q_t);
bool isQFull(Q_t);

int main()
{  int ich, ival;
   Q_t myQ;
   myQ.front = 0;
   myQ.rear = -1;
  for(;;)
  {  printf("\n Q operations \n");
     printf("-----------\n");
     printf("1 Insert 2. delete 3. display 4. exit \n");
     printf("Enter your choice");
      scanf("%d", &ich);
     switch(ich)
     {  case 1: if(isQFULL(myQ))
                  printf("Q FULL");
              else
              {  printf("Enter");
                 scanf("%d", &ival);
                 fnQIn(&myQ, ival);
              }
              break;
```

```
case 2: if(isQEmpty(myQ));
            printf("Empty");
        else {
            ival = fnDeQ(&myQ);
            printf("ele del = %d", ival);
        }
        break;
case 3: if(isQEmpty(myQ))
            printf("Queue is empty\n");
        else
        { printf("\n The ele in the Q are\n");
          fnDisplay(myQ);
        }
        break;
case 4: exit(0);
case 5: default: printf("\n wrong choice\n");
    }
}
return 0;
}
```

```
void fnQIn(Q_t *pq, int val)
{
    (pq->rear)++;
    pq->items[pq->rear] = val;
}
int fnDeQ(Q_t *pq)
{
    int val;
    val = pq->items[pq->front];
    if(pq->rear == pq->front)
    {
        pq->rear = -1;
        pq->front = 0;
    }
    else
    {
        (pq->front)++;
    }
    return val;
}
void fnDisplay(Q_t, Q);
{ int i;
    for(i=Q.front; i<=Q.rear; i++)
        printf("%d\t", Q.items[i]);
}
bool fnQEmpty(Q_t Q)
{
    if(Q.rear == Q.front-1)
        return true;
    else
        return false;
}
```

```
bool isQFull(Q_t q)
{
    if(Q.rear==MAX-1)
        return true;
    else return False;
}
```

---

priority Queue →

It is a data structure in which the intrinsic ordering of the elements does determine the results of it's basic operations

1. Ascending PQ. – min
2. Descending PQ. – MAX

Handling deletion
1. empty indicator then shift
2. Insert at empty
3. Compaction after delete
4. ordered Insert

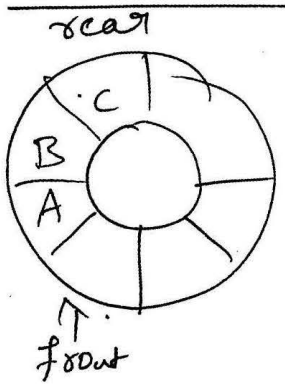
```
int fnPQMinDel(Q_t *pq)
{
    int small, minpos, i;
    if(pq->rear == pq->front)
    {
        small = pq->items[pq->front];
        pq->rear=-1;
        pq->Front=0;
    }
    else
    {
        minpos = pq->front;
        for(i=pq->front+1; i<=pq->rear; i++)
        {
            if(pq->items[minpos] > pq->items[i])
                minpos=i;
        }
```

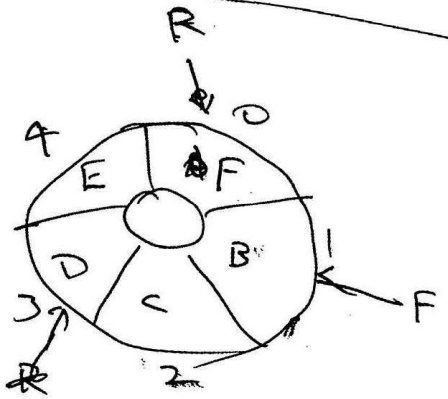
```
        small = PQ->items[minpos];
        for(i=minpos; i<=PQ->rear; i++)
            PQ->items[i] = PQ->items[i+1];
        (PQ->rear)--;
    }
    return small;
}
```

(4)

Circular Queue

(1)






```
void EnQ(int myq[], int *f, int *r, int val)
{
    if(*f == -1)
        *f = *r = 0;
    else
        *r = (*r+1) % MAX;
}

int DeQ(int myq[], int *f, int *r)
{
    int val;
    val = myq[*f];
    if(*f == *r)
        *f = *r = -1;
    else
        *f = (*f+1) % MAX;
}
```

```
void fnDis(int myq[], int *f, int *r)
{
  int i;
  if(*f<=*r)
  {
      for(i=*f; i<=*r; i++)
        printf("%d\t", myq[i]);
  }
  else
  {
      for(i=*f; i<=MAX-1; i++)
        printf("%d\t", myq[i]);
      for(i=0; i<=*r; i++)
        printf("%d\t", myq[i]);
  }
  printf("\n Front-ele = %d\n", myq[*f]);
  printf("\n Front-ele = %d\n", myq[*r]);
}

Empty(int myq, int *f, int *r)
{
  if(*f==-1)
    return true;
  else
    return false;
}
```

```
Full
if(*f==(*r+1)%MAX
  return true
else
  return false;
{
```

Priority Queue

It is a DS in which the intrinsic ordering of the elements does determine the results of it's basic ~~et~~ operations.

① APQ (MIN) ② DPQ (MAX)

Handling deletion ① Delete 'min' element use empty indicator then shift elements ~~&~~ then

② Delete & Insert to first empty ~~&~~ position.

③. Compaction after deletion.

④ Ordered Insert (Inefficient)

```
①void PQInS_ASC(Q_t *PQ, int val)
 {  int j;
    j = PQ->rear;
 while( j>=0 && val<PQ->items[j])
  {  pq->items[j+1] = PQ->items[j];
     j--;
  }
 PQ->items[j+1] = val;
 (PQ->rear)++;} --
}
```

② DPQ → change condition to (>) greater than

Double Ended Queues (Dequeue) – Linear array Implementation.

```
void insert_front(int q[], int *f, int *r, int val)
{
    if(*f==-1)
    {
        *f=*r=0;
        q[*r]=val;
    }
    if(*f!=0)
    {
        *f=--(*f);
        q[*f]=val;
    }
}
int del_rear(int q[], int *f, int *r)
{
    int val;
    val=q[*r];
    if(*f==*r)
    {
        *f=*r=-1;
    }
    else
        --(*r);
    return val;
}
```

```
void display(int a[], int *f, int *r)
{ int i;
  for(i=*f; i<2*r; i++)
    printf("%d\t", a[i]);
}
```

Double Ended Queue

Circular Q - Implementation of Deque

```
#include <stdbool.h>
#include <stdio.h>
#define MAX 5

typedef struct
{
    int front, rear;
    int items[MAX];
} cqueue_t;

void fnEnQRear(cqueue_t *, int);
int fnDeQFront(cqueue_t *);
void fnEnQFront(cqueue_t *, int);
int fnDeQRear(cqueue_t *);
void fnDisplay(cqueue_t);
bool isQEmpty(cqueue_t);
bool isQFull(cqueue_t);

int main()
{
    int ich, ival;
    cqueue_t myq;
    myq.front = -1;
    myq.rear = -1;
```

```
for(;;)
{
  printf("\n Q:-operation\n");
  printf("----------");
  printf("\n 1. Q insert Rear \n 2. Q insert Front \n 3. Q delete Front \n
          4. Q delete Rear \n 5. Q display \n 6. Exit \n");
  printf("Enter your choice \n");
   scanf("%d", &ichoice);

switch(ich)
  { case 1: if(isQFull(myq)
              printf("\n Q is full \n");
            else
            { printf("\n Enter the ele to be inserted");
              scanf("%d", &iVal);
              fnEnQRear(&myq, iVal);
            }
            break;

    Case 2: if(isQFull(myq))
              printf("\n Q is Full\n");
            else
            { printf("\n Enter the element to be
                      inserted");
              scanf("%d", &iVal);
              fnEnQFront(&myq, ival);
            }
            break;
```

```
Case 3: if( isQEmpty(myq))
            printf("inQ is Empty");
        else{
            ival = fnDeqFront(&myq);
            printf("\n element Deleted is: %d", ival);
        }
        break;
Case 4: if( isQEmpty(myq))
            printf("\n Q is Empty");
        else{
            ival = fnDeqRear(&myq);
            printf(" element Deleted is: %d\n", ival);
        }
        break;
Case 5: if(isQEmpty(myq))
            printf("\n Q is Empty");
        else
        {
            printf("\n The elems in the Q are");
            fnDisplay(myq);
        }
        break;
 case 6: exit(0);
 default: printf("\n wrong choice\n");
  }
 }
 return 0;
}
```

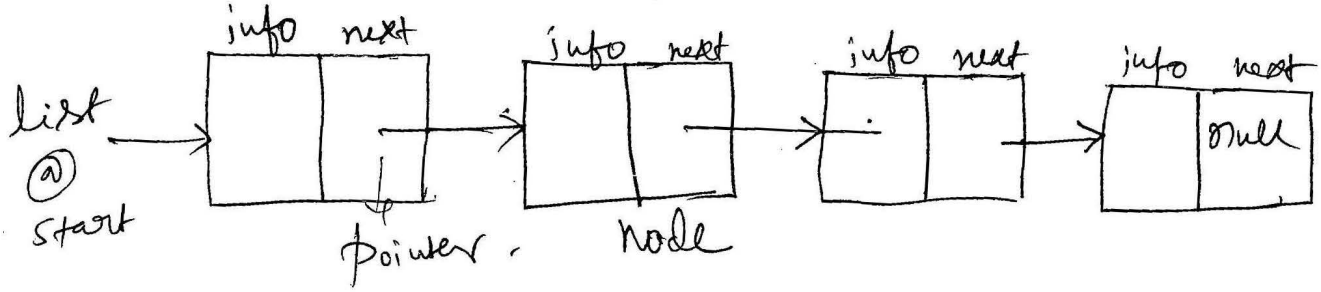
```
void fnEnQRear(cqueue_t *pq, int val)
{
    if(pq->front == -1)
        pq->front = pq->rear = 0;
    else
        pq->rear = (pq->rear+1) % MAX;
    pq->items[pq->rear] = val;
}
void fnEnQFront(cqueue_t *pq, int val)
{
    if(pq->front == -1)
        pq->front = pq->rear = 0;
    else
        pq->front = (pq->front-1 + MAX) % MAX;
    pq->items[pq->front] = val;
}
int fnDeQFront(cqueue_t *pq)
{
    int val;
    val = pq->items[pq->front];
    if(pq->front == pq->rear)
    {
        pq->rear = -1;
        pq->front = -1;
    }
    else
        pq->front = (pq->front+1) % MAX;
    return val;
}
```

```
int fnDeqRear(cqueue_t *pq)
{
    int val;
    val = pq->items[pq->rear];
    if(pq->front == pq->rear)
    {
        pq->rear = -1;
        pq->front = -1;
    }
    else
        pq->rear = (pq->rear - 1 + MAX) % MAX;
    return val;
}

void fnDisplay(cqueue_t myq)
{
    int i;
    if(myq.front <= myq.rear)
    {
        for(i=myq.front; i<=myq.rear; i++)
            printf("%d\t", myq.items[i]);
    }
    else {
        for(i=myq.front; i<=MAX-1; i++)
            printf("%d\t", myq.items[i]);
        for(i=0; i<=myq.rear; i++)
            printf("%d\t", myq.items[i]);
    }
    printf("\n Front is = %d", myq.items[myq.front]);
    printf("\n Rear is = %d\n", myq.items[myq.rear]);
}
```

```
bool isQEmpty(Queue_t myq)
{
    if(myq.front == -1)
        return true;
    else
        return false;
}
bool isQFull(Queue_t myq)
{
    if(myq.front == (myq.rear+1) % MAX)
        return true;
    else
        return false;
}
```

Linked Lists

Linked list is a linear collection of data elements. (items)
These data elements are called 'nodes'.
Each node contains two fields, an information field and a next address field.




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

typedef struct node *nodeptr;
caps

Every node contains a pointer to another node which is of the same type, it is also called a 'self-referential data type'

Array	Linked list
1) Array does store.	1) LL does not store its nodes in consecutive memory locations.
2) Array allows random access of data	2) LL does not allow
3) No. of elements in an array are static	3) we can add any number of elements in the list (node).

(1) Algorithm for Traversing a SLL.

1. SET PTR = START
2. Repeat Step 3 & 4 while PTR != NULL
3. Apply processing to PTR → Data
4. SET PTR = PTR → NEXT
5. Exit

(2) For Counting no. of nodes

same as above step 3 ⇒ C = C + 1

(3) Insert front

1: Temp = getnode( )
   if (Temp == NULL)
   write Node can't be created
2: Set Temp → Data = val
3: Set Temp → Next = START
4: Set START = Temp;
7: Exit.

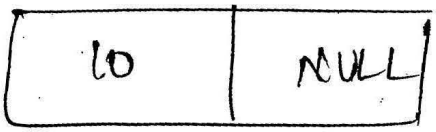
Types of linked lists

1. SLL
2. DLL
3. CLL < SSLL / DLL

SLL is a collection of zero or more nodes where each node has two or more fields & only one link field.
A chain is a singly linked list with zero or more nodes.

1. Creating a node

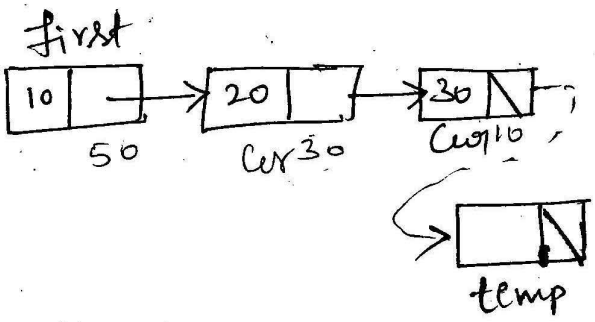
```
nodeptr first;
first = NULL; // empty list
first = (nodeptr) malloc (sizeof(struct node));
first -> info = 20;
first -> next = NULL;
```



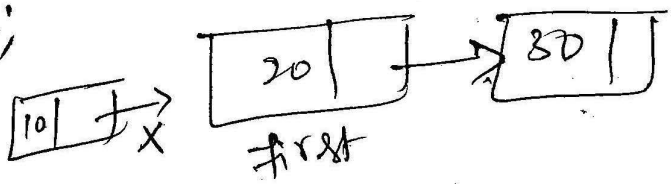
2. operations on SLL
   1. Inserting a node into the list
   2. Deleting a node from the list
   3. Search list
   4. Display list

```
struct NODE
{   int info;
    struct NODE *next;
};
typedef struct NODE *NODEPTR;
NODEPTR allocNode(void)
{   NODEPTR newborn;
    newborn=(NODEPTR) malloc(sizeof(struct NODE));
    if(newborn==NULL)
    {  printf("\nInsufficient Heap memory");
       exit(0);
    }
    return newborn;
}
Void ReleaseNode(NODEPTR x)
{
}  free(x);
}
NODEPTR insertFront(NODEPTR first, int value)
{  NODEPTR temp;
    temp=allocNode();
    temp->info=value;
    temp->next=first;
   return temp;
}
```

```
NODEPTR insertAtEnd(NODEPTR first, int value)
{
    NODEPTR temp, cur;
    temp = allocNode();
    temp->info = value;
    temp->link = NULL;
    if(first == NULL)
        return temp;
    cur = first;
    while(cur->link != NULL)
    {
        cur = cur->next;
    }
    cur->next = temp;
    return first;
}
```

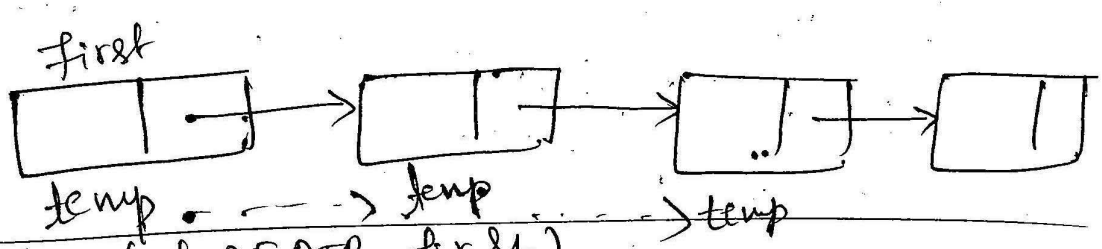



```
NODEPTR deleteFirst(NODEPTR first)
{
    NODEPTR temp;
    if(first == NULL)  // empty list
    { printf("inList is Empty can't delete\n");
      return first;
    }
    temp = first;
    first = first->link;
    return first;
}
printf("deleted ele is %d\n", temp->info);
ReleaseNode(temp);
```



NODEPTR deleteRear(NODEPTR first)

```
{   NODEPTR temp;
    if(first == NULL)
    {  printf("\n List is empty Cannot deleted\n");
    } return first;
    temp = first;
    while( temp->link->link != NULL)
            temp = temp->link;
    printf("\n Element deleted is %d\n", temp->link->info);
    ReleaseNode( temp->link);
    temp->link = NULL;
    return first;
}
```

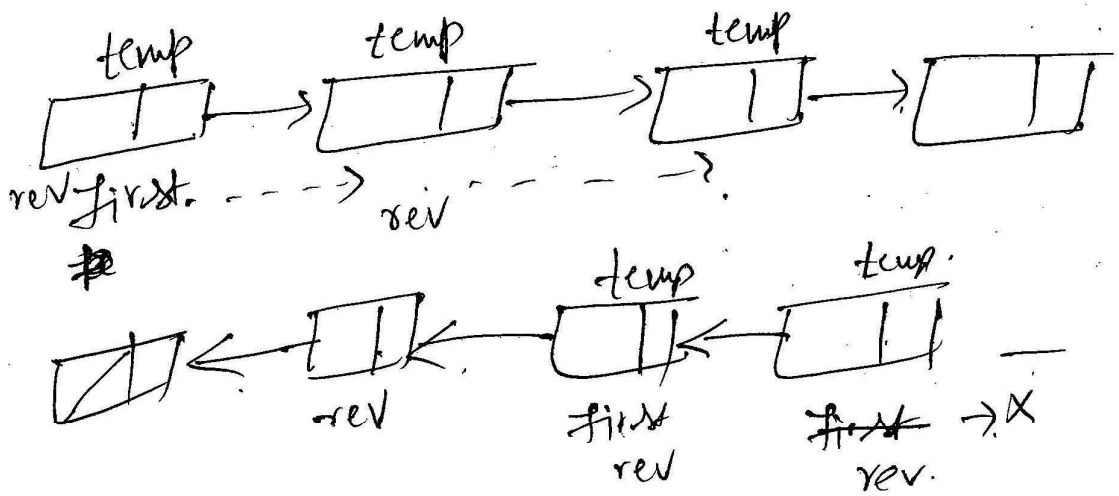



```
Void displist(NODEPTR first)
{   NODEPTR curr;
    int i=1;
    if( first == NULL)
      {  printf("\n List is Empty\n");
      } return;
    printf("\n The Contents of the list are: \n");
    printf("\t Address \t\t Value \t Next Address");
    curr = first;
```

```
while(curr != NULL)
{
 printf("\n Node %d \t %p \t %3d \t %p", i++, curr, curr->info,
                                          curr->link);
  curr = curr->link;
}
 printf("\n");
}
```

```
NODEPTR insertAtpos (NODEPTR first, int value, int pos)
{   NODEPTR temp, prev, cur;
    int count;
    temp = allocNode();
    temp->info = value;
    temp->link = NULL;
   if( first == NULL && pos == 1)
       return temp;
    if( first == NULL)
    {  printf("\n Invalid position !!! \n");
     } return first;

    if(pos == 1)
    {  temp->link = first;
     } return temp;

     count = 1;
     prev = NULL;
     cur = first;
while( cur != NULL && count != pos)
   {  prev = cur;
      cur = cur->link;
    }  count++;
```

```
if(count == POS)
{
    prev -> link = temp;
    temp -> link = cur;
    return first;
}
printf("\n Invalid pos.!!");
return first;
}
```




```
NODEPTR reverselist(NODEPTR first)
{
    NODEPTR rev = NULL, temp;
    if( first == NULL)
        printf(" Empty list");
    else
    { while( first != NULL)
      {
          temp = first;
          first = first -> link;
          temp -> link = rev;
          rev = temp;
      }
    }
    return rev;
}
```

```
NODEPTR deleteAtPos(NODEPTR first, int pos)
{
    NODEPTR prev, cur;
    int count, len;
    if(first == NULL)
    {
        printf("\n Empty list \n Invalid pos");
    } return first;

    ~~count=1~~;
  if(pos==1)
   {  first = first->link;
      printf("\n Element deleted is %d\n", cur->info);
       ReleaseNode(cur);
   } return first;
 len=0; cur=first;
 while(cur!=NULL)
 {  cur=cur->link;
     len++;
 }
 count=1;
 prev=NULL;
 cur = first; //Reinitialize

if(pos>len || pos<0)
    printf("\n Invalid position");
```

```
while(cur != NULL && count != pos && pos <= len)
{
    prev = cur;
    cur = cur->link;
    count++;
}
if(count == pos)
{
    prev->link = cur->link
    printf("\n Element deleted is %d\n", cur->info);
    ReleaseNode(cur);
}
return first;
}
```

```
NODEPTR POP(NODEPTR first)   delete front.
{
    NODEPTR temp;
    if(first == NULL)
    {
        printf("\n List is Empty Cannot delete\n");
        return first;
    }
    temp = first;
    first = first->link;
    printf("\n ele popped = %d", temp->info);
    ReleaseNode(temp);
    return first;
}
NODEPTR PUSH(NODEPTR first, int value)   (Insfront)
{
    NODEPTR temp;
    temp = allocaNode();
    temp->info = value;
    temp->link = first;
    return temp;
}
bool isEmpty(NODEPTR first)
{
    if(first == NULL)
    return 1;
    else
    return 0;
```

SLL-Queue

- Insert-rear function
- Delete-front function.
- Display

Search

```
NODEPTR search(NODEPTR first, int x)
{ NODEPTR p;
for(p=list; p!=NULL; p=p->next)
    if(p->info==x)
        return(p)
return(NULL);
}
```

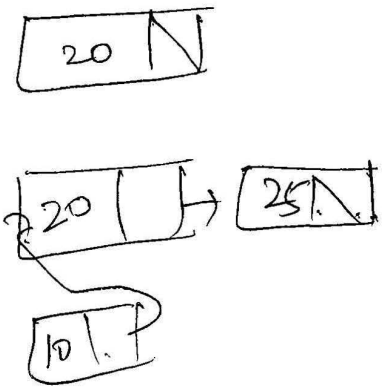
Lists in C

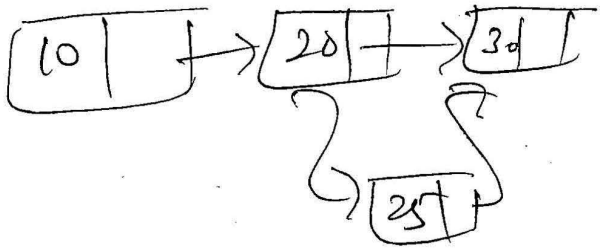
Array Implementation of Lists.

```
#define NumNodes 500
struct nodetype{
        int info;
        int next;
};
struct nodetype node[NUMNODES];
```

Ordered list

```
NODEPTR insNode(NODEPTR list, int val)
{
    NODEPTR temp, cur, prev;
    temp = allocNode();
    temp->info = val;
    if (list == NULL)
    {   temp->next = NULL;
        return temp;
    }
    if(list->info > val)
    {   temp->next = list;
        list = temp;
    }   return list;
prev = NULL;
 cur = list;
while(cur != NULL && cur->info <= val)
 {      prev = cur;
        cur = cur->next;
 }
prev->next = temp;
 temp->link = cur;
 return list;
}
```





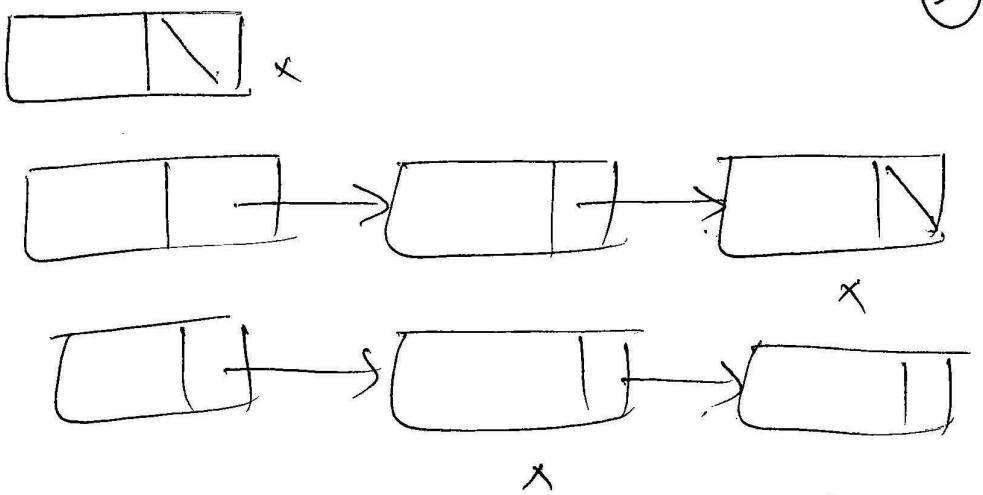
```
void displist(NODEPTR list)
{
    NODEPTR cur;
    if(list==NULL)
    {
        printf("\nList is Empty\n");
    }   return;

    for(cur=list; cur!=NULL; cur=cur->next);
    {
        printf("%d", cur->info);
    }
    printf("\n");
}

NODEPTR UnionList(NODEPTR list1, NODEPTR list2)
{   NODEPTR p1=list1, p2=list2, p3=NULL;

    p3 = mergeList(p1, p3);
    p3 = remDuplicate(p3);
    return p3;
}
```

```
NODEPTR mergeList(NODEPTR list1, NODEPTR list2)
{
    NODEPTR p1 = list1, p2 = list2, p3 = NULL;
    if(list1 == NULL)
        return list2;
    else if(list2 == NULL)
        return list1;
    while(p1 != NULL)
    {
        p3 = insNode(p3, p1->info);
        p1 = p1->next;
    }
    while(p2 != NULL)
    {
        p3 = insNode(p3, p2->info);
        p2 = p2->next;
    }
    return p3;
}
```

```
NODEPTR remDuplicate(NODEPTR list)
{
    NODEPTR prev, cur;
    prev = list; prev->next
    cur = ~~cur~~->next;
    while(prev->next != NULL)
    {
        if(prev->info == cur->info)
        {
            prev->next = cur->next;
            releaseNode(cur);
            cur = ~~node~~ prev->next;
        }
        Continue;
    } prev = cur; cur = cur->next;
    return list;
}
```

```
NODEPTR interseclist(NODEPTR list1, NODEPTR list2)
{
    NODEPTR p1=list1, p2=list2, p3=NULL, s1, s2;
                                          NULL NULL
    if(list1==NULL || list2==NULL)
        return NULL;
    while(p1!=NULL)
    {
        s1=insNode(s1, p1->info);
        p1=p1->next;
    }
    while(p2!=NULL)
    {
        s2=insNode(s2, p2->info);
        p2=p2->next;
    }
    s1=remDuplicate(s1);
    s2=remDuplicate(s2);

p1=s1;
while(p1!=NULL);
{   p2=s2;
        while(p2!=NULL)
        {
            if(p1->info==p2->info)
            {   p3=insNode(p3, p1->info);
                break;
            }
            p2=p2->next;
        }
    p1=p1->next;
}
} return p3;
```

Delete an element in a given ordered list



```
NODEPTR delKeyNode(NODEPTR list, int key)
{
    NODEPTR prev, cur;
    if(list == NULL)
    {
        printf("Empty list");
        return;
    }
    if(key == list->info)
    {
        cur = list;
        list = list->link;
        return list;
        releasenode(cur);
    }
    prev = NULL;
    cur = list;
    while(cur != NULL)
    {
        if(cur->info == key)
        {
            break;
        }
        prev = cur;
        cur = cur->link;
    }
```

```
    if( cur->info == key)
    {
        prev->link = cur->link;
        releaseNode(cur);
    }
}
return list;
}
```

Evaluation & Addition of polynomial

$P = 3x^2 - 2x + 5xy - 2$

$Q = -3x^2 + 3x + 4y^2 + 8$

$x + 5xy + 4y^2 + 6$

```
struct node
{   float coeff;
    float px;
    float py;
    int flag;
}   struct node *next;
typedef struct node *NODEPTR;
NODEPTR read_P(NODEPTR);
Void print_P(NODEPTR);
float Eval_P(NODEPTR);
NODEPTR Add_Poly(NODEPTR, NODEPTR, NODEPTR);
NODEPTR insert_Poly(NODE, float, float, float);
int main()
{
   int ichoice;
   NODEPTR P1=NULL, P2=NULL, P3=NULL;
printf("\nEnter first poly coeff, type -999 to terminate");
  P1 = read_poly(P1);
printf("\n Enter second polynomial type -999 for coeff
          termin");
  P2 = read_poly(P1);
printf("\n\nThe First polynomial: \n");
  print_poly(P1);
printf("\n\nThe Second poly: \n");
  Print_Poly(P2);
```

```
printf("\n1: Evaluate_polynomial\n2: ADD_poly\n");
printf("Enter your choice");
scanf("%d", &ich)
switch(ich)
{ case 1: printf("\n1st poly Evaluation\n");
          printf("\nResult=%f\n", Eval_Poly(P1));
          printf("\n Second poly Evaluation :\n");
          printf(" Result=%f\n", Eval_poly (P2));
          break;
  case 2: P3=add_Poly (P1, P2, P3);
          printf("\nResultant polynomial After Addition:\n");
          printPoly(P3);
          break;
  default: printf(" Invalid choice");
}
return 0;
}
```

```
void print_poly(NODEPTR p)
{
    if(p==NULL)
    {
        printf("\n polynomial doesn't exist\n");
        return;
    }
    while(p->next!=NULL)
    {
        printf("%6.1fx^%3.1fy^%3.1f+", p->coeff, p->px,
                                        p->py);
        p=p->next;
    }
printf("%6.1fx^%3.1fy^%3.1f+", p->coeff, p->px, p->py);
}
```

```
NODEPTR read_P(NODEPTR p)
{
    float cf, px, py;
    printf("Enter coefficient value:");
    scanf("%f", &cf);
    while(cf!=-999)
    {
        printf("\nEnter power of x:");
        scanf("%f", &px);
        printf("\nEnter power of y:");
        scanf("%f", &py);
        p=insert_poly(p, cf, px, py);
        printf("\nEnter coffiecient:");
        scanf("%f", &cf);
    }
} return(p);
```

```
NODEPTR insertpoly( NODEPTR P, float cf, float px,
                                              float py);
{
    NODEPTR newnode, temp;
    newnode = allocnode();
    newnode->Coeff = cf;
    newnode->px = px;
    newnode->py = py;
    newnode->flag = 0;
    newnode->next = NULL;
if( P == NULL)
    P = newnode;
else
{   temp = P;
    while( temp->next != NULL)
        temp = temp->next;
    temp->next = newnode;
}
return P;
}
```

```
NODEPTR Add_Poly(NODEPTR P1, NODEPTR P2, NODEPTR P3)
{
    float cf;
    NODEPTR temp;
    temp=p2;
while(P1!=NULL)
{
    while(p2!=NULL)
    {
        if((P1->px==p2->px)&&(P1->py==p2->py))
            break;
        p2=p2->next;
    }
    if(P2!=NULL)
    {
        cf=p1->coeff * p2->coeff;
        p2->flag=1;
        if(cf!=0)
        p3=insert_poly(P3,cf,P1->px,p1->py);
    }
    else
    p3=insert_poly(P3,P1->coeff,p1->px,P1->py);
    p1=p->next;
    p2=temp;
}
}
```

=>

```
float Evaluate_Poly( NODEPTR p)
{
   float x,y, sum=0;
   printf("\n Enter the value of x \n");
   scanf("%f", &x);
   printf("\n Enter the values of y \n");
   scanf("%f", &y);
  while(p!=NULL)
  {
    sum=sum+p->coeff * pow(x, p->px) * pow(y,
                                        p->py);
    p=p->next;
  }
  return (sum);
}
```

```
while(p2!=NULL)
{
  if(p2->flag==0)
  {
   p3=insert_p(p3, p2->coeff, p2->px, p2->py);
  }
   p2=p2->next;
}
return(p3);
}
```

SLL with Diff. Header

```
//SLL with header nodes operations
#include <stdio.h>
#include <stdlib.h>

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

typedef struct NODE* NODEPTR;

NODEPTR allocNode(void);
void ReleaseNode(NODEPTR);
NODEPTR insertFront(NODEPTR, int);
NODEPTR insertAtEnd(NODEPTR, int);
NODEPTR deleteFirst(NODEPTR);
NODEPTR deleteRear(NODEPTR);
void dispList(NODEPTR);
NODEPTR insertAtPos(NODEPTR, int, int);
NODEPTR reverseList(NODEPTR);

int main()
{
    NODEPTR head = NULL;
    int iChoice, iVal, iPos;
    head=allocNode();
    head->info=0;
    head->link=NULL;

        for(;;)
        {
                printf("\nLIST OPERATIONS\n");
                printf("====================");
                printf("\n0.Insert Front\n1.Insert Rear\n2.Delete
Front\n3.Delete Rear\n4.Display\n5.Insert at Position\n6.Reverse
List\n7.Exit\n");
                printf("\nEnter your choice\n");
                scanf("%d",&iChoice);

                switch(iChoice)
                {
                        case 0: printf("\nEnter Element to be inserted\n");
                                scanf("%d",&iVal);
                 head= insertFront(head,iVal);
                                break;

                        case 1: printf("\nEnter Element to be inserted\n");
                                scanf("%d",&iVal);
                                head = insertAtEnd(head,iVal);
                                break;

                        case 2: head = deleteFirst(head);
                                break;

                        case 3: head = deleteRear(head);
                                break;

                        case 4: dispList(head);
                                break;
```

struct HEAD
{ int info; // count
  struct NODE *next;
};
typedef struct HEAD *HEADPTR;

HEADPTR head = NULL;

```
                        case 5: printf("\nEnter Element to be inserted\n");
                                scanf("%d",&iVal);
                                printf("\nEnter position at which element is to
be inserted\n");
                                scanf("%d",&iPos);
                                head = insertAtPos(head,iVal,iPos);
                                break;

                        case 6: head = reverseList(head);
                 break;

                        case 7: exit(0);
                }
        }

    return 0;
}


NODEPTR allocNode(void)
{
        NODEPTR newborn;
        newborn = (NODEPTR)malloc(sizeof(struct NODE));

        if(newborn == NULL)
        {
                printf("\nInsufficient Heap Memory");
                exit(0);
        }
        return newborn;
}

void ReleaseNode(NODEPTR x)
{
        free(x);
}


//first = insertFront(first,iVal);
NODEPTR insertFront(NODEPTR head, int value)
{
        NODEPTR temp;

        temp = allocNode();
        temp->info = value;
        temp->link = head->link;
        head->link=temp;
    head->info++;

        return head;

}

//first = insertAtEnd(first,iVal);
NODEPTR insertAtEnd(NODEPTR head, int value)
{
        NODEPTR temp,cur;

        temp = allocNode();
```

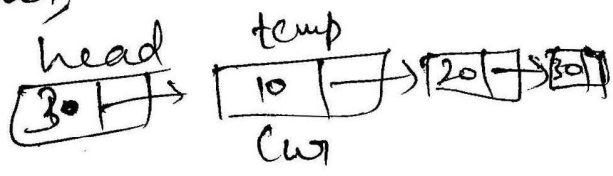
```
        temp->info = value;
        temp->link = NULL;

    //if(head->link == NULL)
      //        return temp;

    cur = head;
    while(cur->link != NULL)
    {
            cur = cur->link;
    }
    cur->link = temp;
    head->info++;
    return head;
}

//first = deleteFirst(first);
NODEPTR deleteFirst(NODEPTR head)
{
    NODEPTR temp;
        if(head->link == NULL)
        {
                printf("\nList is empty cannot delete\n");
                return head;
        }
        temp = head->link;
        //temp = temp->link;

        printf("\nElement deleted is %d \n",temp->info);
        head->link=temp->link;
        ReleaseNode(temp);
    head->info--;
        return head;
}
```

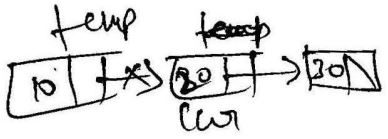
(Head)

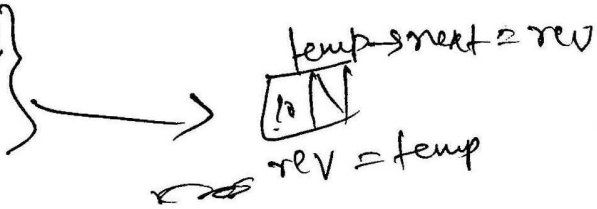
```
NODEPTR insAtPos(NODEPTR first, int value, int pos)
{
    NODEPTR temp, prev, cur;
     int count;
     temp=allocNode();
      temp->info=value;
       temp->next=NULL;

    count=1;
    prev=first;
    cur=first->next
  while(cur!=NULL && count!=pos)
   {    prev=cur;
        cur=cur->next;
        count++;
   }
```

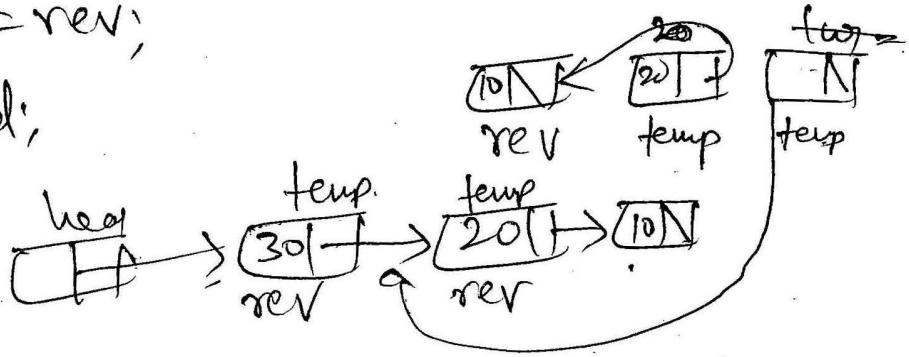
```
if(count == pos)
{
    prev->next = temp;
    temp->next = cur;
    first->info++;
    return first;
}
printf("\nInvalid Position !!!\n");
return first;
}
```

```
NODEPTR revList(NODEPTR head)
{
    NODEPTR rev=NULL, temp, cur;
    cur = head->next;
    while(cur!=NULL)
    {
        temp=cur;
        cur=cur->next;
        temp->link=rev;
        rev=temp;
    }
    head->link=rev;
    return head;
}
```









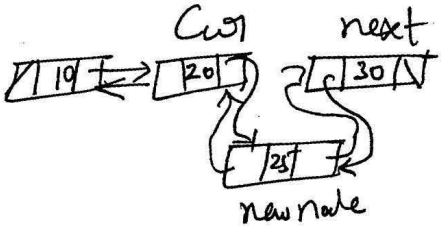
Doubly Linked List


```
struct node
{
    int info;
    struct node *left;
    struct node *right;
};
typedef struct node *NODEPTR;
NODEPTR allocNode(void){
{
    NODEPTR NewBorn;
    NewBorn=(NewBorn) malloc(sizeof(struct node));
    if(NewBorn==NULL)
    {
        printf("Heap full...NODE creation not possible");
        exit(1);
    }
    return NewBorn;
}
void ReleaseNode(NODEPTR x)
{
    free(x);
}
NODEPTR insertFront(NODEPTR first, int val)
{
    NODEPTR newNode;
    newNode=allocNode();
    newnode->info=val;
    newnode->left=NULL;
    newnode->right=NULL;
    if(first==NULL)
        first=newnode;
```

```
        else
        {
            newnode->right = first;
            first->left = newnode;
        }
    return newnode;
}

NODEPTR insRear(NODEPTR first, int val)
{
    NODEPTR newnode, last;
    newnode = allocNode();
    newnode->info = val;
    newnode->right = NULL;
    last = first;
    while (last->rlink != NULL)
        last = last->rlink;
    last->right = newnode;
    newnode->left = last;
} return first;

NODEPTR delFront(NODEPTR first)
{
    if(first == NULL)
    { printf("Empty list");
      return first;
    }
    first = first->right;
    printf("Deleted element is:%d", first->left->info);
    ReleaseNode(first->left);
    first->left = NULL;
return(first);
```

```
NODEPTR delRear(NODEPTR first)
{
    NODEPTR last;
    if( first ==NULL)
    {
        printf(" Empty list");
    }   return first;

    last = first;
    while(last->right->right!=NULL)
        last =last->right;

    printf("\nDeleted element is =%d\n", last->right->info);
    ReleaseNode(last->right);

    last->right=NULL;

  return first;
}

NODEPTR insertAfter(NODEPTR , int val)
{
  NODEPTR newNode, cur, next;
  if( first == NULL)
  {
      printf("\nList is empty");
  }   return newnode;

  cur=first;
  while( cur->right!=NULL)
  {
      if(item== cur->info) break;
  }     cur=cur->right;

if(cur->right==NULL)
{
  printf(" Key not found \n");
} return first;
```

newNode2 allocateNode();
newNode->info = val;



```
next = Cur->right;

Cur->right = newNode;
newNode->left = Cur;
next->left = newnode;
newNode->right = next;

return first;
}
```

```
NODEPTR insertbefor (NODEPTR first, int val)
{
    NODEPTR newnode, Cur, prev;
                    ——> new node = allocNode();
    if( first == NULL)        new node->info = val;
    {  puf(" List Empty");
       return head;
    }
  Cur = head->right;
  while ( Cur->right != NULL)
  {
     if(item == cur->info) break;
       Cur = cur->right;
  }
 if ( cur->right == NULL)
 {  printf("Key not found\n");
    return ~~0~~ first;
 }
 prev = Cur->left;
 prev->right = newNode;
  temp->left = prev;
  Cur->left = temp;
newNode~~temp~~->right = Cur;

return first;
```

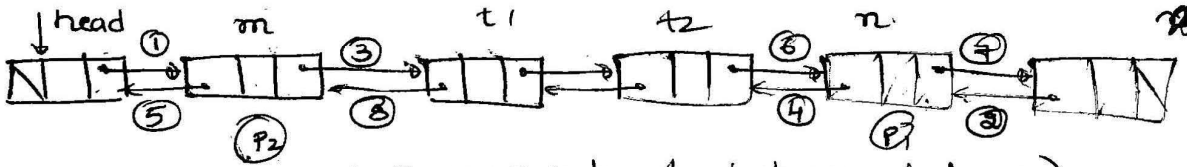
DLL - Header

① Insert after a key element.

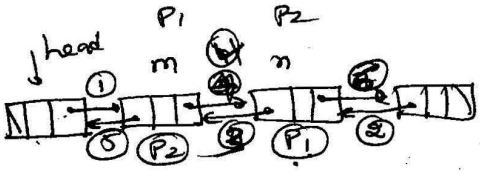
```
NODEPTR InsAfter(NODEPTR head, int data)
{
    int key;
    NODEPTR newnode, temp;
    newnode = allocNode();
    newnode->info = data;
    newnode->right = newnode->left = NULL;
  if(head->right == NULL)
  {
    head->right = newnode;
    newnode->left = head;
    head->info++;
  }
  else
  {
    printf("\n Enter the key element:");
    scanf("%d", &key);
    temp = head->right
    while (temp!=NULL && key!=temp->info)
        temp = temp->right;
    if(temp==NULL)
        printf("\n key %d is not found in list", key);
    else
    {
        newnode->right = temp->right;
        if(temp->right!=NULL)
            temp->right->left = newnode;
```

```
        temp->right = newnode;
        newnode->left = temp;
        head->info++;
    }
}
return head;
}

NODE Del_First_with_Last(NODEPTR head, NODE temp)
{
    while(temp->left != head)
        temp = temp->left;
    head->right = temp->right;
    if(temp->right != NULL)
        temp->right->left = head;
    head->info--;

    printf("First Node deleted is %d\n", temp->info);
    Releasenode(temp);
    return head;
}
```

```
NODEPTR Del_Pos (NODEPTR head, int pos)
{
    int i; NODEPTR temp;
    if(head->right == NULL)
    { Empty list;
      return NULL;
    }
    if(pos<1 || pos>head->info)
        printf("\nInvalid Input\n");
    else
    {
        temp = head->right;
        for(i=1; temp!=NULL && i!=pos; i++)
            temp = temp->right;
        temp->left->right = temp->right;
        if(temp->right!=NULL)
            temp->right->left = temp->left;
        head->info--;
        printf("%d from pos %d is deleted\n",
                                temp->info, pos);
        Releasnode(temp);
    }
    return head;
}
```




```
NODE *swap_node( NODE * head, int m, int n)
{
    NODE *P1, *P2, *t1, *t2;
    int i;
    if((m>=1 && m<=head->info) && (n>=1 && n<=head->info))
    {
        P1 = head->rptr;
        P2 = head->rptr;

        for(i=1; i!=m; i++)
            P1 = P1->rptr;

        for(i=1; i!=n; i++)
            P2 = P2->rptr;
```




```
        if(P1->rptr != P2) // if not adjacent nodes
        {
            t1 = P1->rptr;
            t2 = P2->lptr;

            1. P1->lptr->rptr = P2;
            2. if(P2->rptr != NULL)
                   P2->rptr->lptr = P1;
            3. P1->rptr = P2->rptr;
            4. P2->lptr = P1->lptr;
            5. P1->lptr = t2;
            6. t2->rptr = P1;
            7. P2->rptr = t1;
            8. t1->lptr = P2;
        }
        else
        {
            1. P1->lptr->rptr = P2;
            2. if(P2->rptr != NULL)
                   P2->rptr->lptr = P1;
            3. P1->rptr = P2->rptr;
            4. P2->lptr = P1->lptr;
            6 P2->rptr = P1;
            5 P1->lptr = P2;
        }
    }
    else
        printf("\nInvalid");
    return(head);
}
```

```
NODEPTR SWAP_NODES (NODEPTR head, int m, int n)
{
    NODEPTR p1, p2, t1, t2;
    int count;
    if(((m>=1) && (m<=head->info)) && n>=1 && n<=head->info)
    {
        count=1;
        p1=p2=head->right;
        while(count!=m)
        {
            p1=p1->right;
            count++;
        }
        count=1;
        while(count!=n)
        {
            p2=p2->right;
            count++;
        }
            if(p1->rptr!=p2)
            {
                t1=p1->right;
                t2=p2->left;
                p1->left->right=p2;
                        if(p2->right!=NULL)
                            p2->right->left=p1;
                p1->right=p2->right;
                p2->left=p1->left
                p1->left=t2;
                t2->right=p1
                p2->right=t1;
                t1->left=p2;
            }
```

```
else
{
    1  p1->left->right = p2;
                if(p2->right!=NULL)
    2  p2->right->left = p1;

    3  p1->right = p2->right;
    4  p2->left = p1->left;
    5  p1->left = p2;
    6  p2->right = p1;
}
printf("\n Swapped %d & %d nodes Successfully\n", m,n);
}
else
    printf("\n Invalid number for Swapping");
return head;
}
void display(NODEPTR head)
{
NODEPTR temp;
if(head->right==NULL)
 { printf("\n Empty List\n");
   return;
 }
 temp=head->right;
 printf("\n List contents:\n");
while(temp!=NULL)
  { printf("%d", temp->info);
    temp=temp->right;
  }
printf("Total no. of node %d",
                head->info);
}
```

```
//CLL-operations.
#include <stdio.h>
#include <stdlib.h>

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

typedef struct NODE* NODEPTR;
NODEPTR deleteAtPos(NODEPTR first,int pos);
NODEPTR allocNode(void);
void ReleaseNode(NODEPTR);
NODEPTR insertFront(NODEPTR, int);
NODEPTR insertAtEnd(NODEPTR, int);
NODEPTR deleteFirst(NODEPTR);
NODEPTR deleteRear(NODEPTR);
void displist(NODEPTR);
NODEPTR insertAtPos(NODEPTR, int, int);
NODEPTR reverseList(NODEPTR);

int main()
{
    NODEPTR first = NULL;
    int iChoice, iVal, iPos;

      for(;;)
      {
            printf("\nLIST OPERATIONS\n");
            printf("====================");
            printf("\n0.Insert Front\n1.Insert Rear\n2.Delete
Front\n3.Delete Rear\n4.Display\n5.Insert at Position\n6.Reverse
List\n7.Exit\n8.del at pos\n");
            printf("\nEnter your choice\n");
            scanf("%d",&iChoice);

            switch(iChoice)
            {
                  case 0: printf("\nEnter Element to be
inserted\n");
                        scanf("%d",&iVal);
                        first = insertFront(first,iVal);
                        break;

                  case 1: printf("\nEnter Element to be
inserted\n");
```

```
//CLL-operations.
#include <stdio.h>
#include <stdlib.h>

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

typedef struct NODE* NODEPTR;
NODEPTR deleteAtPos(NODEPTR first,int pos);
NODEPTR allocNode(void);
void ReleaseNode(NODEPTR);
NODEPTR insertFront(NODEPTR, int);
NODEPTR insertAtEnd(NODEPTR, int);
NODEPTR deleteFirst(NODEPTR);
NODEPTR deleteRear(NODEPTR);
void dispList(NODEPTR);
NODEPTR insertAtPos(NODEPTR, int, int);
NODEPTR reverseList(NODEPTR);

int main()
{
    NODEPTR first = NULL;
    int iChoice, iVal, iPos;

      for(;;)
      {
            printf("\nLIST OPERATIONS\n");
            printf("====================");
            printf("\n0.Insert Front\n1.Insert Rear\n2.Delete
Front\n3.Delete Rear\n4.Display\n5.Insert at Position\n6.Reverse
List\n7.Exit\n8.del at pos\n");
            printf("\nEnter your choice\n");
            scanf("%d",&iChoice);

            switch(iChoice)
            {
                  case 0: printf("\nEnter Element to be
inserted\n");
                        scanf("%d",&iVal);
                        first = insertFront(first,iVal);
                        break;

                  case 1: printf("\nEnter Element to be
inserted\n");
```

```
                        scanf("%d",&iVal);
                        first = insertAtEnd(first,iVal);
                        break;

                case 2: first = deleteFirst(first);
                        break;

                case 3: first = deleteRear(first);
                        break;

                case 4: dispList(first);
                        break;

                case 5: printf("\nEnter Element to be
inserted\n");
                        scanf("%d",&iVal);
                        printf("\nEnter position at which
element is to be inserted\n");
                        scanf("%d",&iPos);
                        first = insertAtPos(first,iVal,iPos);
                        break;

                case 6: first = reverseList(first);
              break;

                case 7: exit(0);
                case 8:
                        printf("\nEnter position at which
element is to be deleted\n");
                        scanf("%d",&iPos);
                        first = deleteAtPos(first,iPos);
                        break;

            }
      }

    return 0;
}


NODEPTR allocNode(void)
{
      NODEPTR newborn;
      newborn = (NODEPTR)malloc(sizeof(struct NODE));

      if(newborn == NULL)
      {
```

```
            printf("\nInsufficient Heap Memory");
            exit(0);
      }
      return newborn;
}

void ReleaseNode(NODEPTR x)
{
      free(x);
}


//first = insertFront(first,iVal);
NODEPTR insertFront(NODEPTR last, int value)
{
      NODEPTR temp;

      temp = allocNode();
      temp->info = value;
      if(last==NULL)
      last=temp;
      else
        temp->link = last->link;

      last->link=temp;

      return last;

}

//first = insertAtEnd(first,iVal);
NODEPTR insertAtEnd(NODEPTR last, int value)
{
      NODEPTR temp,cur;

      temp = allocNode();
      temp->info = value;
      temp->link = NULL;

    if(last == NULL)
      last=temp;
    else
        temp->link=last->link;

last->link=temp;

    return temp;
```

```
}

//first = deleteFirst(first);
NODEPTR deleteFirst(NODEPTR last)
{
    NODEPTR first;
      if(last == NULL)
      {
            printf("\nList is empty cannot delete\n");
            return NULL;
      }
      if(last->link==last)
      {
          printf("\nElement deleted is %d \n",last->info);
        ReleaseNode(last);
        return NULL;

      }
      first = last->link;
      last->link = first->link;

      printf("\nElement deleted is %d \n",first->info);
      ReleaseNode(first);

      return last;
}

NODEPTR deleteRear(NODEPTR last)
{
    NODEPTR prev;
      if(last == NULL)
      {
            printf("\nList is empty cannot delete\n");
            return NULL;
      }
      if(last->link==last)
      {
          printf("\nElement deleted is %d \n",last->info);
        ReleaseNode(last);
        return NULL;

      }
      prev=last->link;
      while(prev->link!=last)
            prev=prev->link;

    prev->link=last->link;
    printf("deleted element is %d\n", last->info);
    ReleaseNode(last);
} return prev;
```

```
void dispList(NODEPTR last)
{
	NODEPTR curr;
	int i=1;
	if(last == NULL)
	{
		printf("\nList is empty\n");
		return;
	}

	printf("\nThe contents of the list are :\n");
	printf("\tAddress\t\tValue\tNextAddress");
	curr = last->link;
	while(curr!= last)
	{
		printf("\nNode %d\t%p\t%3d\t%p",i++,curr,curr-
>info,curr->link);
		curr = curr->link;
	}
	printf("\nNode %d\t%p\t%3d\t%p",i++,last,last->info,last-
>link);
	printf("\n");
}
```

```
NODEPTR insertAtPos(NODEPTR last, int value, int pos)
{
      NODEPTR temp,prev,cur;
      int count;

      temp = allocNode();
      temp->info = value;
      temp->link = NULL;
    if(last == NULL && pos == 1)
    {
        last=temp;
        last->link=temp;
        return last;
    }
    if(last == NULL)
      {
            printf("\nInvalid Position!!!\n");
            return last;
      }
if(pos == 1)
      {
            temp->link = last->link;
            last->link=temp;
            return last;
      }

      count = 1;
      prev = NULL;
      cur = last->link;
      while(cur != last && count != pos)
      {
            prev = cur;
            cur = cur->link;
            count++;
      }

      if(count == pos&&cur!=last)
      {
            prev->link = temp;
            temp->link = cur;
            return last;
      }
      else
      {
           temp->link=last->link;
           last->link=temp;
           return temp;

      }

      printf("\nInvalid Position!!!\n");
      return last;
}
```

```
SLL with header nodes operations (CLL)-Header
nclude <stdio.h>
nclude <stdlib.h>

ruct NODE

   int info;
   struct NODE *link;


pedef struct NODE* NODEPTR;

DEPTR allocNode(void);
id ReleaseNode(NODEPTR);
DEPTR insertFront(NODEPTR, int);
DEPTR insertAtEnd(NODEPTR, int);
DEPTR deleteFirst(NODEPTR);
DEPTR deleteRear(NODEPTR);
id dispList(NODEPTR);
DEPTR insertAtPos(NODEPTR, int, int);
DEPTR reverseList(NODEPTR);

: main()

  NODEPTR head = NULL;
  int iChoice, iVal, iPos;
  head=allocNode();
  head->info=0;
  head->link=head;

    for(;;)
    {
          printf("\nLIST OPERATIONS\n");
          printf("====================");
          printf("\n0.Insert Front\n1.Insert Rear\n2.Delete Front\n3.Delete Rear\n4.Display
5.Insert at Position\n6.Reverse List\n7.Exit\n");
          printf("\nEnter your choice\n");
          scanf("%d",&iChoice);

          switch(iChoice)
          {
                case 0: printf("\nEnter Element to be inserted\n");
                      scanf("%d",&iVal);
              head= insertFront(head,iVal);
                      break;

                case 1: printf("\nEnter Element to be inserted\n");
                      scanf("%d",&iVal);
                      head = insertAtEnd(head,iVal);
                      break;

                case 2: head = deleteFirst(head);
                      break;

                case 3: head = deleteRear(head);
                      break;

                case 4: dispList(head);
                      break;

                case 5: printf("\nEnter Element to be inserted\n");
                      scanf("%d",&iVal);
                      printf("\nEnter position at which element is to be inserted\n");
                      scanf("%d",&iPos);
                      head = insertAtPos(head,iVal,iPos);
                      break;

                case 6: head = reverseList(head);
              break;

                case 7: exit(0);
```

```
        }
    }

  return 0;
```



```
TR allocNode(void)

    NODEPTR newborn;
    newborn = (NODEPTR)malloc(sizeof(struct NODE));

    if(newborn == NULL)
    {
          printf("\nInsufficient Heap Memory");
          exit(0);
    }
    return newborn;


id ReleaseNode(NODEPTR x)

    free(x);



first = insertFront(first,iVal);
DEPTR insertFront(NODEPTR head, int value)

    NODEPTR temp,cur;

    temp = allocNode();
    temp->info = value;
    temp->link = head->link;
    head->link=temp;
    //cur=head;
    //while(cur->link!=head)
   //    cur=cur->link;
 // cur->link=head;
  head->info++;
    return head;



first = insertAtEnd(first,iVal);
DEPTR insertAtEnd(NODEPTR head, int value)

    NODEPTR temp,cur;

    temp = allocNode();
    temp->info = value;
    temp->link = NULL;

  //if(head->link == NULL)
    //     return temp;

  cur = head;
  while(cur->link != head)
  {
          cur = cur->link;
  }
  cur->link = temp;
  temp->link=head;
  head->info++;
  return head;


first = deleteFirst(first);
DEPTR deleteFirst(NODEPTR head)

  NODEPTR temp;
    if(head->link == head)
```

```
    {
          printf("\nList is empty cannot delete\n");
          return head;
    }
    temp = head->link;
    //temp = temp->link;

    printf("\nElement deleted is %d \n",temp->info);
    head->link=temp->link;
    ReleaseNode(temp);
  head->info--;
    return head;


)EPTR deleteRear(NODEPTR head)

  NODEPTR temp;
    if(head->link == head)
    {
          printf("\nList is empty cannot delete\n");
          return head;
    }
    temp = head;
    while(temp->link->link!= head)
    {
          temp = temp->link;
    }

    printf("\nElement deleted is %d \n",temp->link->info);

    ReleaseNode(temp->link);
    temp->link=head;

ad->info--;
    return head;


id dispList(NODEPTR head)

    NODEPTR curr;
    int i=1;
    if(head->link == NULL)
    {
          printf("\nList is empty\n");
          return;
    }

    printf("\nThe contents of the list are :\n");
    printf("\tAddress\t\tValue\tNextAddress");
    curr = head->link;
  printf("\nHead-Node \t%p\t%3d\t%p",head,head->info,head->link);

    while(curr != head)
    {
          printf("\n\tNode %d\t%p\t%3d\t%p",i++,curr,curr->info,curr->link);
          curr = curr->link;
    }
    printf("\n");


)DEPTR insertAtPos(NODEPTR head, int value, int pos)

    NODEPTR temp,prev,cur;
    int count;

    temp = allocNode();
    temp->info = value;
    temp->link = NULL;

    /*if(first->link == NULL && pos==1)
    {
        first->link=temp;
```

```
        first->info++;
        return first;
    }

    if(first->link == NULL && pos>1)
    {
          printf("\nInvalid Position!!!\n");
          return first;
    }*/


    count = 1;
    prev = head;
    cur = head->link;

    while(cur != head && count != pos)
    {
          prev = cur;
          cur = cur->link;
          count++;
    }

    if(count == pos)
    {
          prev->link = temp;
          temp->link = cur;
          head->info++;
          return head;
    }

    printf("\nInvalid Position!!!\n");
    return head;


DEPTR reverseList(NODEPTR head)

    NODEPTR cur,temp;*/
    NODEPTR newlist = NULL;*/
    cur = first;*/
    while(cur != NULL)*/
    {*/
        temp = allocNode();*/
        temp->info = cur->info;*/
        temp->link = newlist;*/
        newlist = temp;*/
        cur = cur->link;*/
    }*/
    return newlist;*/
  NODEPTR rev = NULL,prev,cur;
  cur=head->link;
    while(cur!=head)
    {
          prev = cur;
          cur = cur->link;
          prev->link = rev;
          rev = prev;
    }
    head->link=rev;
    prev->link=head;
    return head;
```

```
/*** PROGRAM 13: ADDITION OF TWO LONG POSITIVE
INTEGERS USING CIRCULAR
                    LINKED LIST WITH HEADER NODE
***/
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
#include<stdlib.h>
struct node
{
   int info;
   struct node *next;
};
typedef struct node *NODE;
void main()
{
     NODE *insert(NODE*,int);
     NODE *add(NODE*,NODE*,NODE*);
     void display(NODE *);
     void display1(NODE*);
     NODE *head1,*head2,*head3;
     char ch;
     clrscr();

     head1=(NODE*)malloc(sizeof(NODE));
     head2=(NODE*)malloc(sizeof(NODE));
     head3=(NODE*)malloc(sizeof(NODE));
     head1->next=head1;
     head2->next=head2;
     head3->next=head3;

    printf("Enter the first number(press enter
to terminate)\n");
     while((ch=getchar())!='\n')
```

```
    {
       if(isdigit(ch))
           head1=insert(head1,ch-'0');
       else
       {
           printf("Invalid digit");
           exit(1);
        }
    }

   printf("Enter the second number(press
enter to terminate)\n");
    while((ch=getchar())!='\n')
    {
       if(isdigit(ch))
           head2=insert(head2,ch-'0');
       else
       {
           printf("\nInvalid digit\n");
           exit(1);
        }
     }

     printf("\nFIRST NUMBER:  ");
     display1(head1);

     printf("\nSECOND NUMBER: ");
     display1(head2);

     head3=add(head3,head1,head2);
     printf("\nRESULTANT SUM: ");
     display(head3);
     getch();
}
```

PROG13

```
 NODE *insert(NODE *head,int digit)
{
        NODE *newnode,*first;
        newnode=(NODE*)malloc(sizeof(NODE));
        newnode->info=digit;

        newnode->next=head->next;
        head->next=newnode;
        return(head);
    }

     NODE * add(NODE *head3,NODE *head1,NODE
*head2)
      {
            int total,carry=0,rem;
            NODE *num1,*num2;
            num1=head1->next;
            num2=head2->next;

            while(num1!=head1 && num2!=head2)
            {
                total=num1->info + num2->info
+carry;
                rem=total%10;
                head3=insert(head3,rem);
                num1=num1->next;
                num2=num2->next;
                carry=total/10;
             }

             while(num1!=head1)
            {
                total=num1->info + carry;
```

```
            rem=total%10;
            head3=insert(head3,rem);
            num1=num1->next;
            carry=total/10;
         }

         while(num2!=head2)
        {
            total=num2->info +carry;
            rem=total%10;
            head3=insert(head3,rem);
            num2=num2->next;
            carry=total/10;
         }
          if(carry==1)
             head3=insert(head3,carry);
          return(head3);
}

void display(NODE *head)
{
     NODE *temp;
     if(head==head->next)
     {
        printf("\nEmpty list\n");
        return;
     }
     temp=head->next;
     while(temp!=head)
     {
         printf("%d",temp->info);
         temp=temp->next;
     }
 }
```

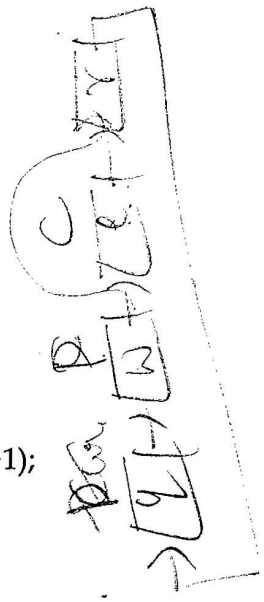
output

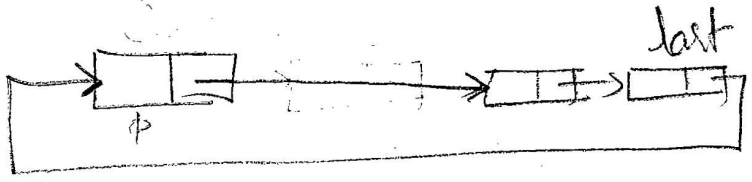
84

```
Input
void display1(NODE *head)
{
    NODE *temp1,*temp2;
    if(head==head->next)
    {
       printf("\nEmpty list\n");
       return;
    }
    temp2=head;
    while(temp2!=head->next)
    {
       temp1=head->next;
       while(temp1->next!=temp2)
          temp1=temp1->next;
        printf("%d",temp1->info);
        temp2=temp1;
    }

}
```

Josephus problem


```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#define MAXLEN 30

struct NODE{
   char name[MAXLEN];
   struct NODE *link;
};

typedef struct NODE* NODEPTR;

NODEPTR allocNode();
void releaseNode(NODEPTR);
NODEPTR insRear(NODEPTR , char []);

int main()
{
   NODEPTR last = NULL, cur,prev;
        int i, iNum, iSpNum;
        char soldierName[MAXLEN];

        printf("\nEnter the number of soldiers : ");
        scanf("%d",&iNum);

        for(i=0;i<iNum;i++)
        {
                printf("\nEnter the name of soldier %d : ",i+1);
                scanf("%s",soldierName);
                last = insRear(last, soldierName);
        }
//   srand(time(NULL));
//      iSpNum = (rand()%10)+2;
   printf("\nEnter a value larger than 1 for the counter\n");
   scanf("%d",&iSpNum);
        printf("\nCounter value is : %d\n",iSpNum);
        cur = last->link;
        while(last->link != last)
        {
                for(i=0;i<iSpNum;i++)
                {
                        prev =cur;
```





```
                cur = cur->link;
            }
            printf("\nEliminated Soldier is : %s\n", cur->name);
            getchar();
            if(last == cur)
        last = prev;
            prev->link = cur->link;
            releaseNode(cur);
        }
        printf("\nSoldier chosen to bring in reinforcements is : %s\n",last->name);
        return 0;
}

NODEPTR insRear(NODEPTR last, char name[])
{
  NODEPTR temp, first;
  temp = allocNode();
       strcpy(temp->name,name);

  if(last == NULL)
  {
    last = temp;
    temp->link = last;
    return last;
  }
  first = last->link;
  last->link = temp;
  temp->link = first;
  return temp;
}

NODEPTR allocNode()
{
  NODEPTR newborn;
  newborn = (NODEPTR) malloc(sizeof(struct NODE));
  if(newborn == NULL)
  {
    printf("\nInsufficient Heap Memory\n");
    exit(0);
  }
  return newborn;
}
```

```
void releaseNode(NODEPTR p)
{
   free(p);
}
```

Trees

A tree is a data structure which is collection of zero or more nodes & finite set of directed lines called branches that connect the nodes.

The first node in the tree is called root node & remaining nodes are called subtrees.

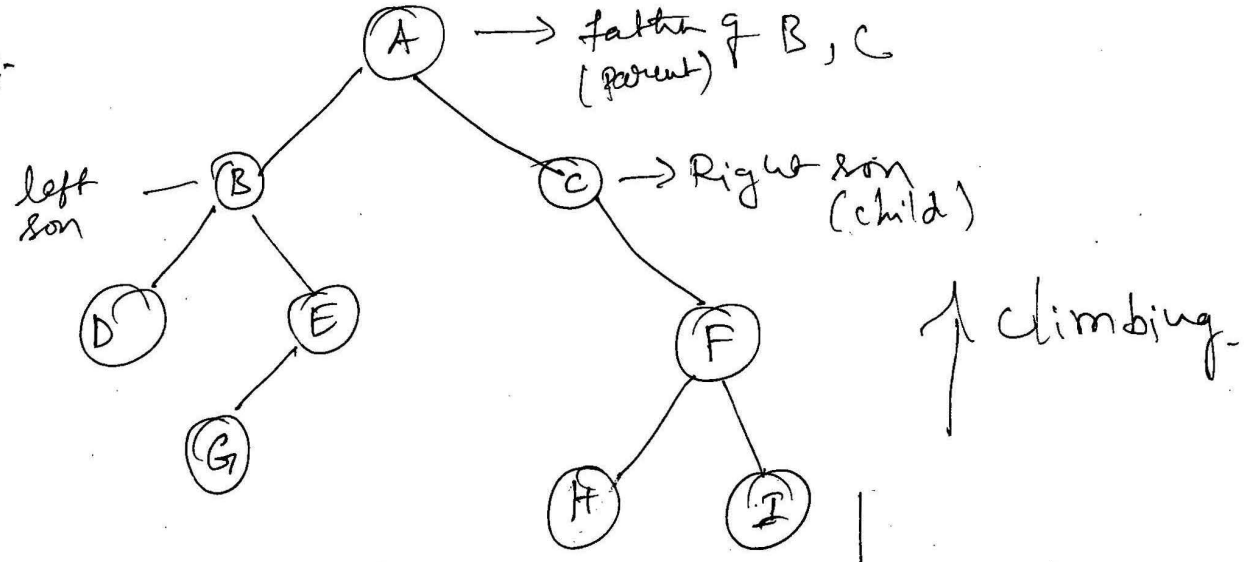
Trees ← General trees
Binary trees.

Binary Tree

It is a finite set of elements that is either empty or is partitioned into three disjoint subsets:

- Root: First node of the tree
- other two sets are themselves binary tree called left & right subtrees of the original tree.
- A left or right subtree can be empty. Each element of a binary tree is called a node of the tree.

Ex:



'A' is ancestor of G.

- left descendant - 'D'
- Right de-" - H, F.

G, H, I - leaf nodes

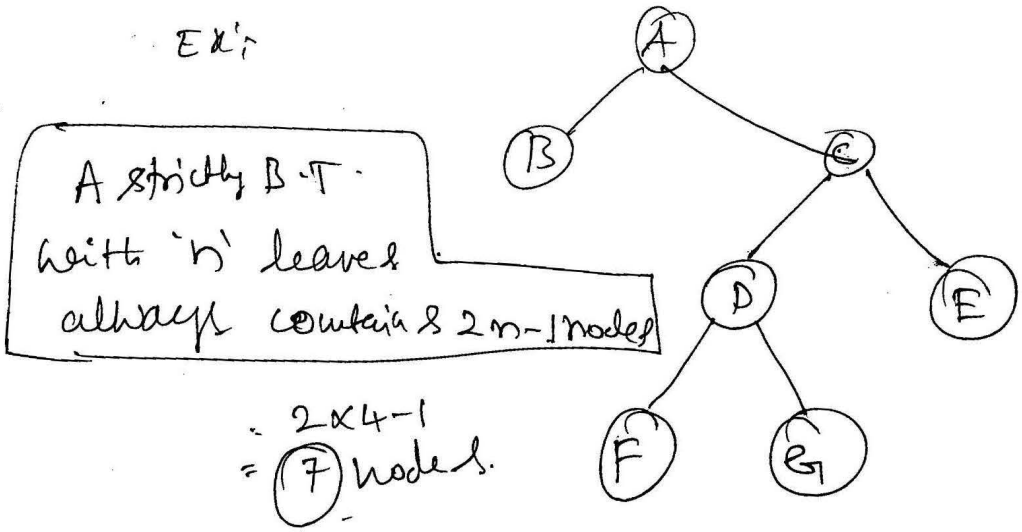
Siblings: 2 or more nodes having the same parent are called siblings.

Types of B-Trees

(1) Strictly B.T

If every nonleaf node in a B.T has non Empty left & right subtrees.

Ex:



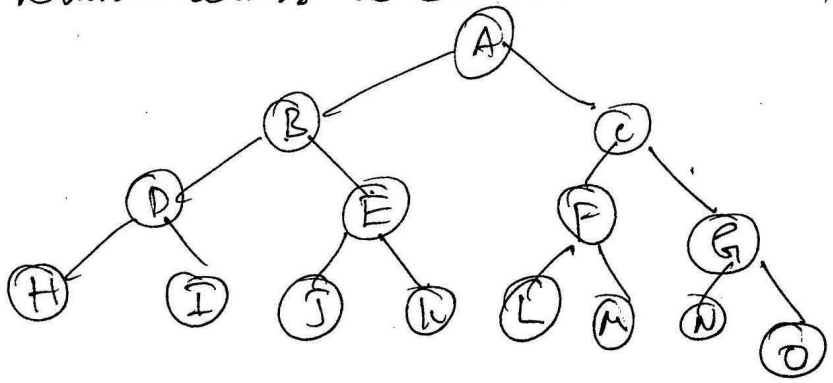
A strictly B.T with 'n' leaves always contains 2n-1 nodes.

∴ 2×4-1
= 7 nodes.

Level - Root level 0 & level of any node is the (of a node) tree is one more than the level of it's father.

Depth of a tree: B.T is the max. level of any leaf in the tree.

(2) A Complete B.T of depth d is the strictly B.T of all of whose leaves are at level d.



The Compft total no. of nodes in complete B.T. (2)

$$tn = 2^0 + 2^1 + 2^2 + \cdots + 2^d = \sum_{j=0}^{d} 2^j \quad (89)$$

$$\boxed{tn = 2^{d+1} - 1} \quad \text{(By induction)}$$

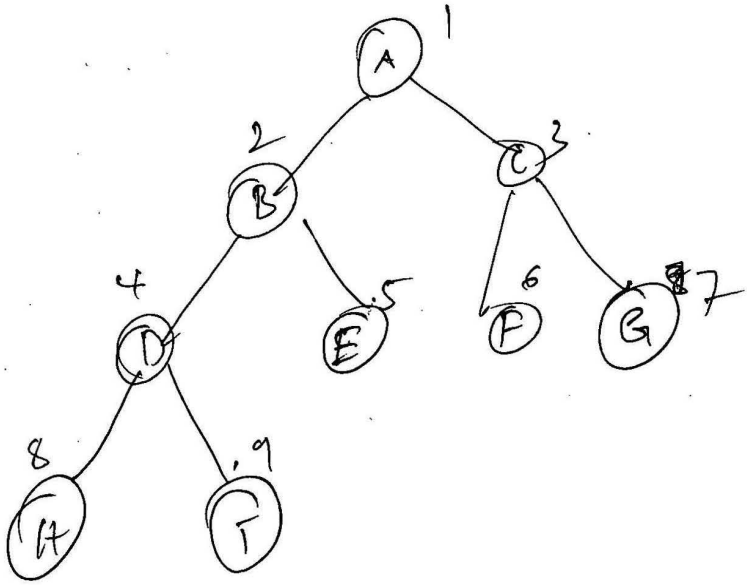
$$\boxed{\text{no. of leaf nodes} = 2^d}$$

$$\therefore \boxed{\text{non-leaf nodes} = 2^d - 1}$$

(3) Almost Complete B.T

It it (1) Any node `nd' at level less d-1 has two sons @ child

(2.) For any node nd in the tree with a right descendant at level d, nd' must have a left son @ every left descendant of nd is either a leaf at level d @ has two sons.



Degree of a node : The no. of branches associated with each node.

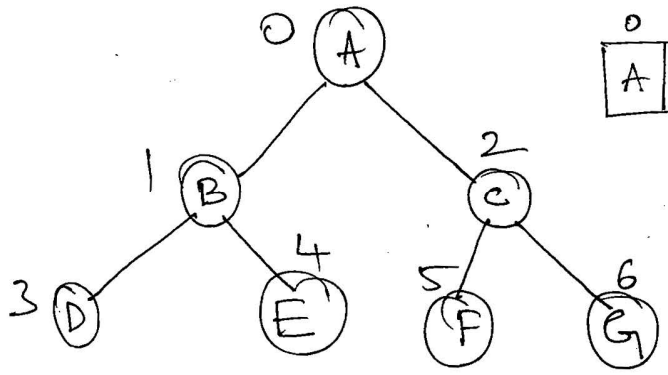
Indegree : no. of branch directed toward the node

outdegree : no. of branch is directed away from the node

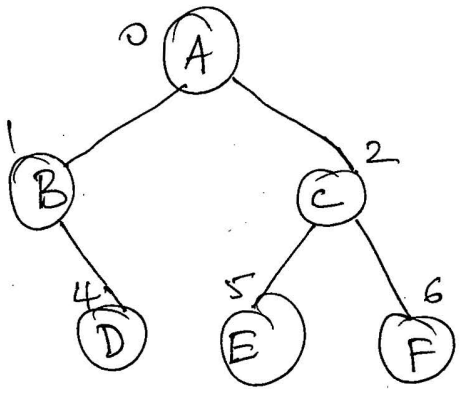
Storage representation of a binary tree

(1) Sequential allocation – arrays – static
(2) Linked – " – – LL – Dynamic.

(1) Array representation



0	1	2	3	4	5	6
A	B	C	D	E	F	G



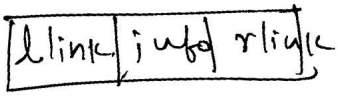
0	1	2	3	4	5	6
A	B	C		D	E	F

Prof. Kallinatha H D.
Assistant Professor
Department of Computer Science & Engg.
Siddaganga Institute of Technology
TUMKUR - 572 103, Karnataka, INDIA

```
#define MAX_SIZE 200
struct node
{
    int info;
    int used;
};
typedef struct node NODE;
NODE a[MAX_SIZE];
```

---

(2) Linked representation.



(1) Insertion
(2) Traversal
(3) Search.
(4) Copying.
(5) Deletion.

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

TREES

- DEFINITION
- TERMINOLOGIES
- BINARY TREE REPRESENTATIONS
- TREE TRAVERSALS
- BINARY SEARCH TREE
- THREADED BINARY TREE
- EXPRESSION TREES

Prof. Kallinatha H. D.
Assistant Professor
Department of Computer Science & Engg
Siddaganga Institute of Technology
TUMKUR - 572 103, Karnataka, INDIA

1

TREES

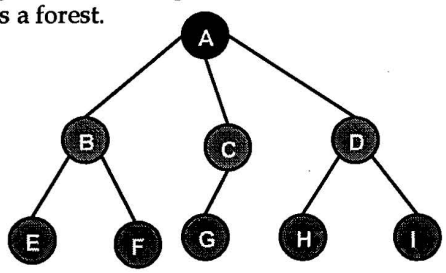
DEFINITION :

- A tree is a data structure that represents hierarchical relationships between individual data items.
- A tree is non-primitive, non-linear data structure.

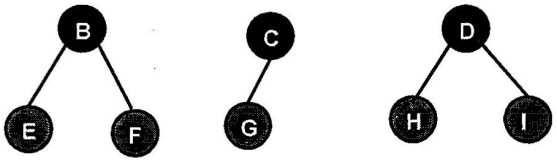
2

FOREST

It is a set of disjoint trees. In a given tree, if the root node is removed, then it becomes a forest.



In the above tree, we have forest with three trees .



3

TREES

- Consists of a finite set of elements called ***nodes***, and a finite set of lines called ***branches***, that connects the nodes.
- The number of branches associated with a node is the ***degree*** of the node.
- The number of branches (edges) leaving a node is called an ***outdegree*** of that node.
- The number of branches (edges) entering a node is called an ***indegree*** of that node.

4

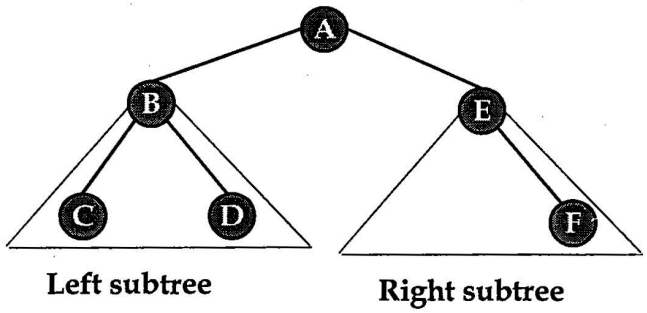
93

TERMINOLOGIES

- **Binary Tree**: A binary tree is a finite set of elements that is either empty or is partitioned into three disjoint subsets:
  - The first subset contains a single element called the **ROOT** of the tree.
  - The other two subsets are themselves binary trees, called the **LEFT** and **RIGHT** subtrees of the original tree.

- **NODE** : Each element of a binary tree is called a node of the tree.

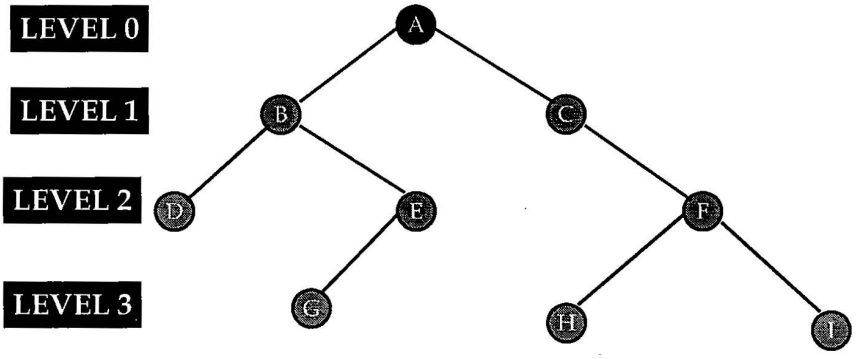
5



BINARY TREE

6

TREES



7

TERMINOLOGIES

Parent: A node is a parent if it has successor nodes i.e, if it has an outdegree greater than zero.

Child: A node with a predecessor is a child. A child node has an indegree of one.

Root: A node with no predecessor is called as a root node.

Leaf /Terminal Node : A leaf is any node with an outdegree of zero, (External Node) i.e, a node with no successors.

or

A Node that has no child is called a ***leaf node***.

8

TERMINOLOGIES

- **Ancestor:** Node n_1 is an ancestor of node n_2 if n_1 is either the parent of n_2 or parent of some ancestor of n_2.
- **Descendants:** The nodes in the path below the parent .

9

TERMINOLOGIES

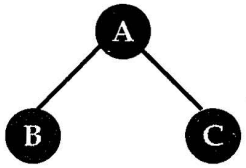
- **Left Descendant:** Node n_2 is a left descendant of node n_1 if n_2 is either the left child of n_1 or a descendant of left child of n_1.
- **Right Descendant:** Node n_2 is a right descendant of node n_1 if n_2 is either the right child of n_1 or a descendant of right child of n_1.

10

TERMINOLOGIES

- **Siblings** : Two or more nodes having the same parent.

Ex:



Here, A is the Parent of B and C, B and C are children of A, B and C are siblings.

11

TERMINOLOGIES

- **Level of a node:** It is the distance of a node from the root.
  - The root of a tree has level 0(zero)
  - Level of any other node in the tree is one more than level of its parent.

- **Depth /Height of a Binary tree:** The maximum level of any leaf in the tree.

12

TYPES OF BINARY TREES

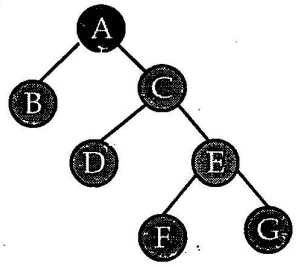
- Strictly Binary tree
- Complete Binary tree
- Almost complete binary tree
- Binary Search tree
- Expression tree

13

TERMINOLOGIES

- **Strictly Binary Tree** : (Fully B.T.)
  If every non-leaf node in a binary tree has non empty left and right sub trees, the tree is termed as strictly binary tree.

Ex:

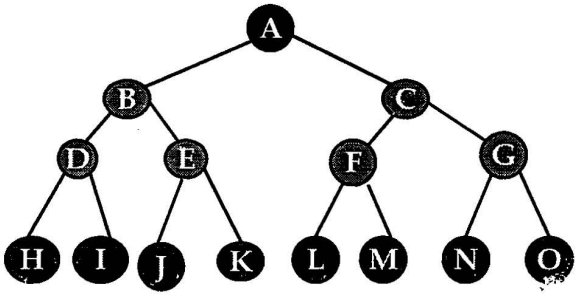


14

98

TERMINOLOGIES

- **Complete Binary Tree:**
  A completely binary tree of depth d is the strictly binary tree, all of whose leaves are at level d.
  Ex: Complete binary tree of depth 3.

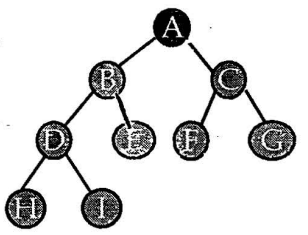


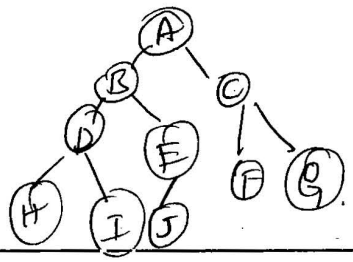
15

TERMINOLOGIES

- **Almost Complete Binary Tree:**
  A binary tree of depth d is an almost complete binary tree if:
  (1) leaf at d@d-
- Any node n_d at level less than d-1 has two children.
  (Total no. of nodes at level d-1 should be 2^{d-1}).
- The total number of nodes at level d may be equal to 2^d.
  If total number of nodes at level d is less than 2^d, then the nodes should be present only from left to right.

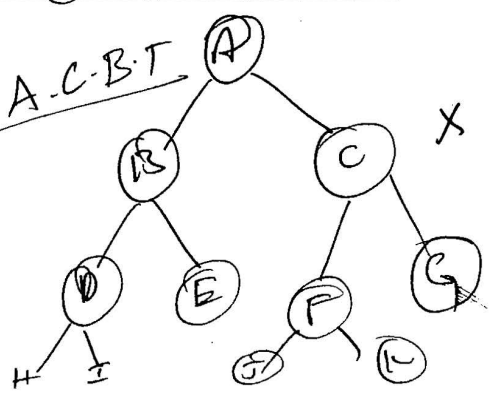
Ex:





16

Not A-C-B-T



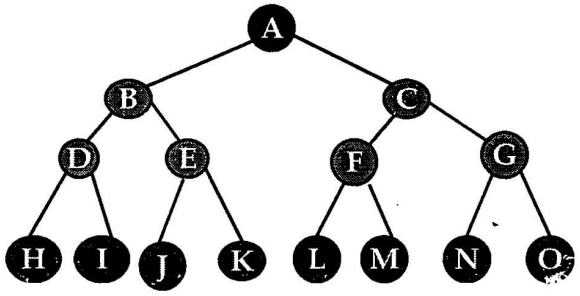
X

TERMINOLOGIES

- **Complete Binary Tree:**

  A completely binary tree of depth d is the strictly binary tree, all of whose leaves are at level d.

  Ex: Complete binary tree of depth 3.



15

TERMINOLOGIES

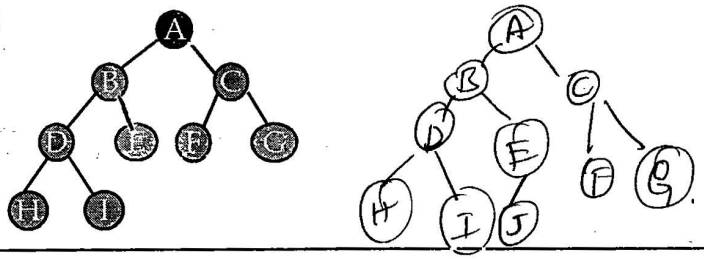
- **Almost Complete Binary Tree:**

  A binary tree of depth d is an almost complete binary tree if:

  (1) leaf at d@d-1

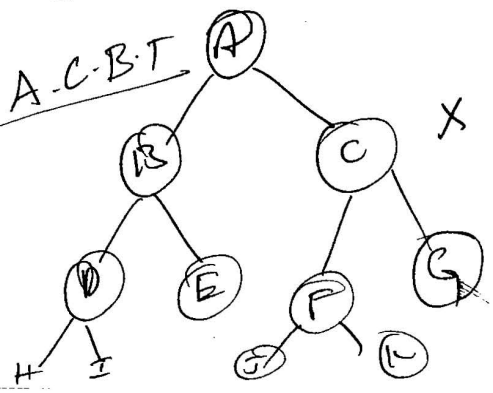
  - Any node n_d at level less than d-1 has two children.
    (Total no. of nodes at level d-1 should be 2^{d-1}).
  - The total number of nodes at level d may be equal to 2^d.
    If total number of nodes at level d is less than 2^d,then the nodes should be present only from left to right.

Ex:



16

Not A.C.B.T



X

TERMINOLOGIES

NOTE:

1. If a binary tree contains m nodes at level L , then it contains atmost $2m$ ***nodes at level L+1***.

2. Since binary tree can contain atmost one node at level 0(root node), it can contain exactly 2^L ***nodes at level L***.

3. A strictly binary tree with n leaves always contain $2n-1$ ***nodes***.

4. The total number of nodes in a complete binary tree of depth *d* equals the sum of number of nodes at each level between 0 (zero) and d.

5. In a complete binary tree of depth d, there are 2^d ***leaf nodes*** and 2^d-1 ***nonleaf nodes***.

17

TREE REPRESENTATION

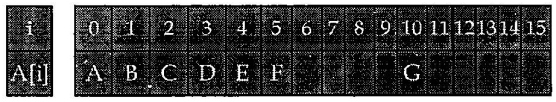
- Binary trees can be represented 2 ways in memory. They are:

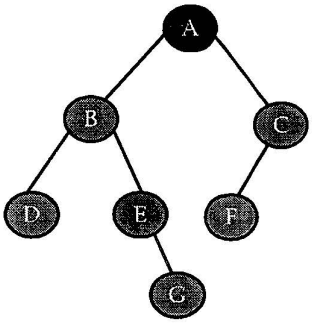
1. Array representation
2. Linked list representation.

18

TREE REPRESENTATION

- **Array Representation:**





If *P* is the parent, then the left and right child are at the position 2*P*+1 and 2*P*+2 respectively.

19

TREE REPRESENTATION

- **Linked List Representation:**

In linked representation of trees, each node *N* of the tree will contain three fields:

1. Info
2. Left child
3. Right child

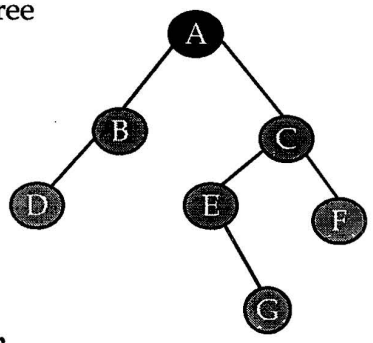
Logical Representation of the tree :

```
struct treenode
{
   int data;
   struct treenode *rchild;
   struct treenode *lchild;
};
```

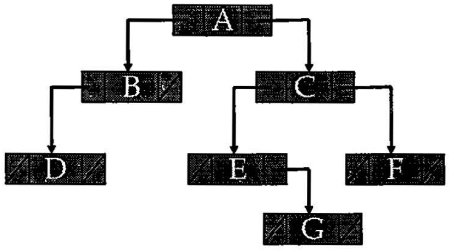
20

TREE REPRESENTATION

- **Linked List Representation:**
  consider the following binary tree



Linked representation :



21

TREE TRAVERSALS

- *Traversing* is a method of visiting each node of a tree exactly once in a some order.

- Three Methods for Binary Tree traversal:
  - ***Inorder*** Traversal
  - ***Preorder*** Traversal
  - ***Postorder*** Traversal

22

102

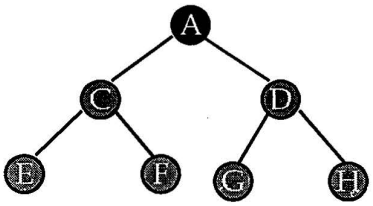
TREE TRAVERSALS

- **Inorder Traversal (Symmetric order) :**

To traverse a non empty binary tree in inorder :

1. Traverse the left subtree in inorder
2. Visit the root
3. Traverse the right subtree in inorder.

Example:



Inorder : E C F A G D H

23

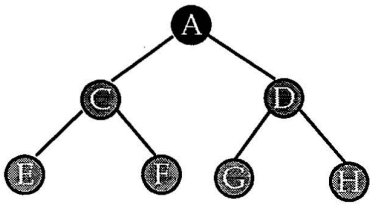
TREE TRAVERSALS

- **Preorder Traversal (Depth-First order) :**

To traverse a non empty binary tree in preorder :

1. Visit the root
2. Traverse the left subtree in preorder
3. Traverse the right subtree in preorder.

Example:



Preorder : A C E F D G H

24

103

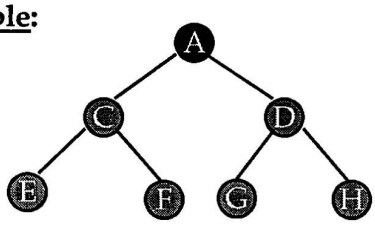
TREE TRAVERSALS

- **Postorder Traversal :**

To traverse a non empty binary tree in postorder :

1. Traverse the left subtree in postorder.
2. Traverse the right subtree in postorder.
3. Visit the root.

Example:



Postorder : E F C G H D A

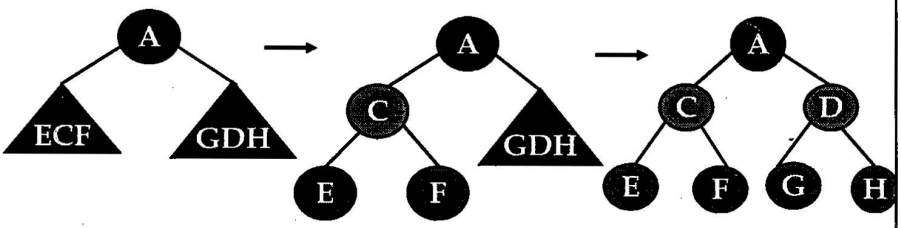
25

PROBLEM:

Given the following Inorder and predorder traversals, construct the binary tree:

Inorder: ECFAGDH

Preorder: ACEFDGH



26

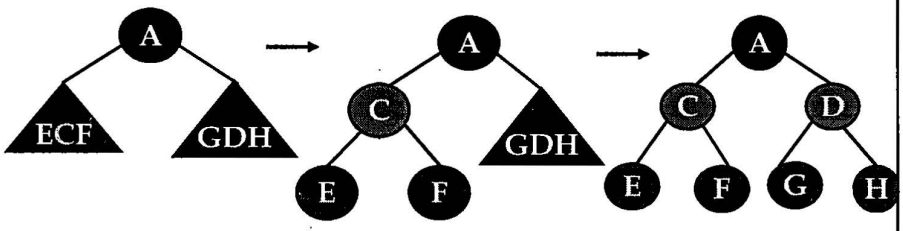
104

PROBLEM:

Given the following Inorder and postorder traversals, construct the binary tree:

Inorder: ECFAGDH

Postorder: EFCGHD**A**



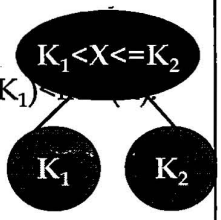
27

BINARY SEARCH TREE

Binary Search Tree (Binary Sorted Tree) :

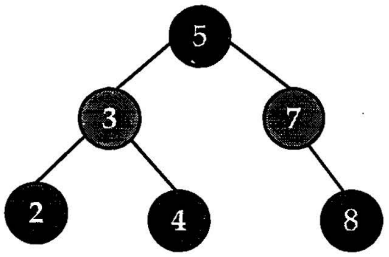
- Suppose *T* is a binary tree. Then *T* is called a binary search tree if each node of the tree has the following property :
  the value of each node is greater than every value in the left subtree of that node and is less than or equal to every value in the right subtree of that node.

- Let X be a node in a binary search tree .
  If K_1 is a node in the left subtree of x,then info(K_1)<info(X)
  If K_2 is a node in the right subtree of X , then
  info(X)<=info(K_2)



105

- **Example BST:**

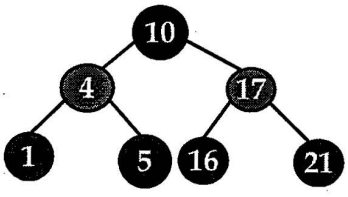


29

OPERATIONS ON BST

- *Traversal*
- *Searching*
- *Insertion*
- *Deletion*

TRAVERSAL:

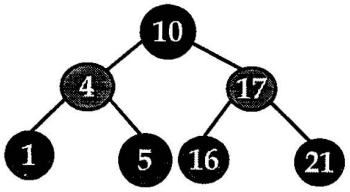


INORDER: 1, 4, 5, 10, 16, 17, 21
PREORDER: 10, 4, 1, 5, 17, 16, 21
POSTORDER: 1, 5, 4, 16, 21, 17, 10

30

106

SEARCHING:



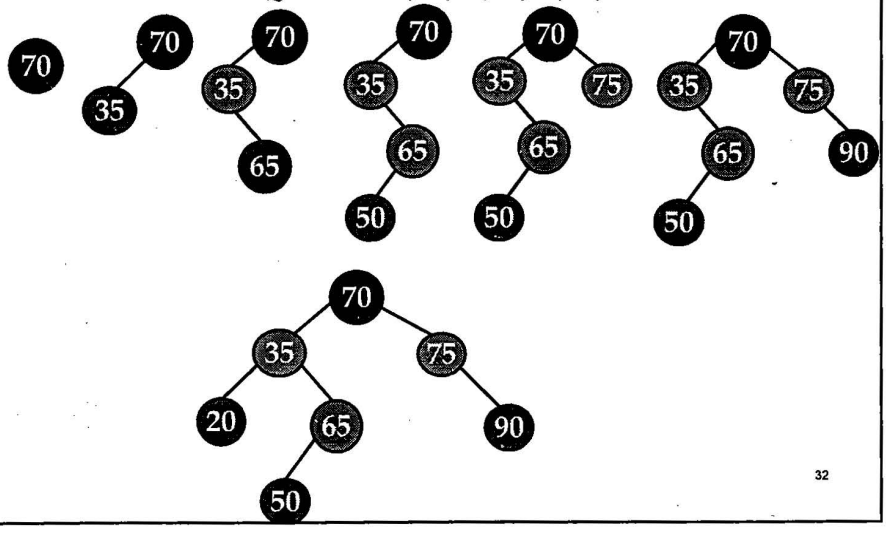
KEY: 5

```
while(ROOT!=NULL)
{
   if(KEY<ROOT->INFO)
      ROOT=ROOT->LCHILD;
   else if(KEY>ROOT->INFO)
      ROOT=ROOT->RCHILD;
   else
      return(1);
}
```

31

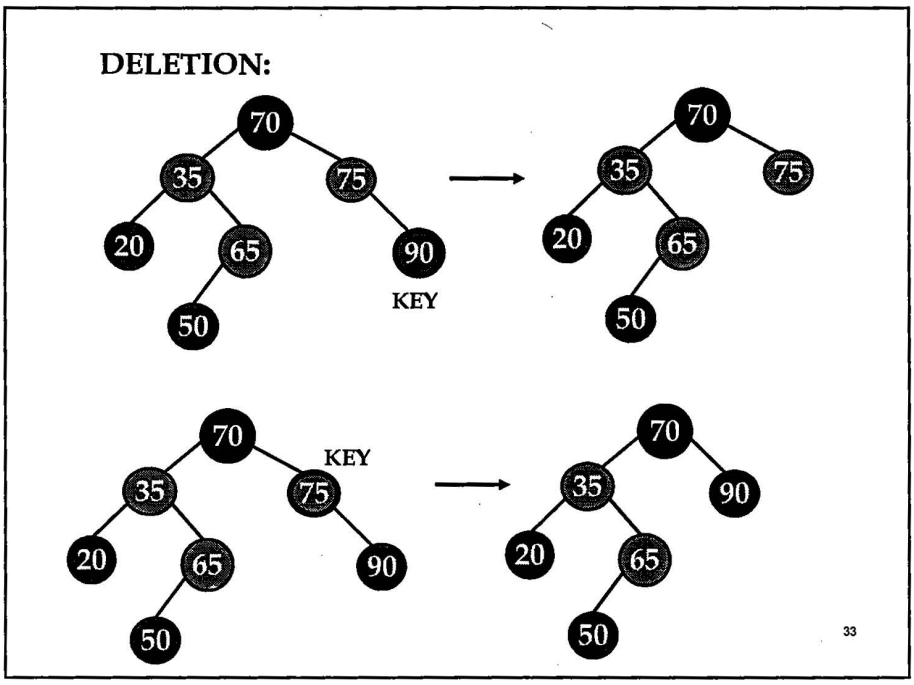
INSERTION:

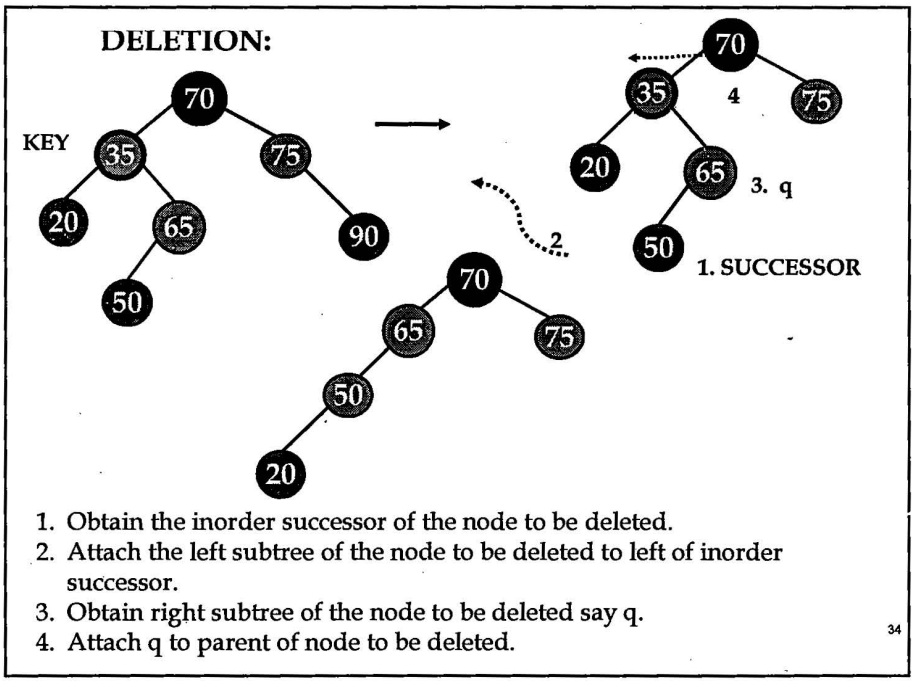
GIVE THE VARIOUS STEPS INVOLVED IN CREATING A BST FOR THE FOLLOWING SEQUENCE: 70, 35, 65, 50, 75, 90, 20



32

107


DELETION:
70
35
75
20
65
90
KEY
50
70
35
75
20
65
50
70
KEY
35
75
20
65
90
50
70
35
90
20
65
50
33


DELETION:
KEY
70
35
75
20
65
90
50
70
4
35
75
20
65
3. q
50
1. SUCCESSOR
2
70
65
75
50
20
1. Obtain the inorder successor of the node to be deleted.
2. Attach the left subtree of the node to be deleted to left of inorder successor.
3. Obtain right subtree of the node to be deleted say q.
4. Attach q to parent of node to be deleted.
34

108

THREADED BINARY TREE

- In linked representation of a binary tree, approximately half of the entries in the pointer fields i.e, lchild and rchild contain NULL . This space may be more efficiently used by replacing certain NULL pointers by some ***special pointers*** which points to the successor or to the predecessor.
- These special pointers are called THREADS, and binary trees with such pointers are called **Threaded Binary Trees**.

35

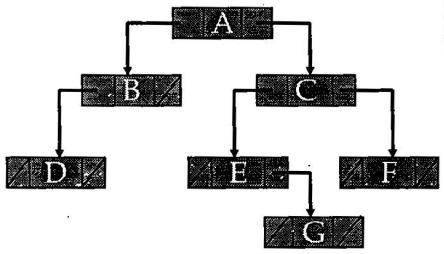


Fig (a)

Inorder : D B A E G C F

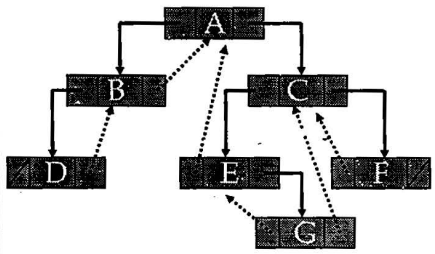


Fig (b)

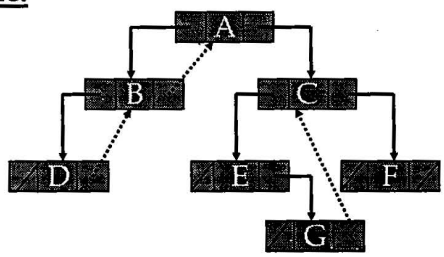
Inorder : D B A E G C F

In-threaded Binary tree 36

AN INORDER THREADING OF A BINARY TREE / IN-THREADED BINARY TREE

A binary tree is threaded based on the traversal technique.

- If the right link of a node is NULL and if it is replaced by the address of the inorder successor then the tree is said to be **Right in-threaded binary tree**.
- **Example:**

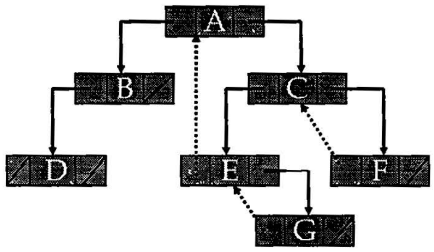


Inorder : D B A E G C F

37

AN INORDER THREADING OF A BINARY TREE / IN-THREADED BINARY TREE

- If the left link of a node is NULL and if it is replaced by the address of the inorder predecessor, then the tree is said to be **Left in-threaded binary tree**.
- **Example:**



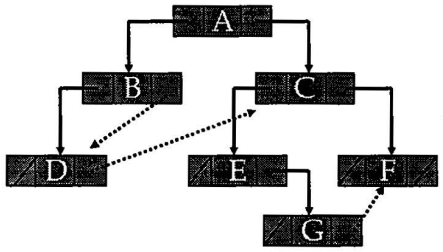
Inorder : D B A E G C F

38

110

A PREORDER THREADING OF A BINARY TREE / PRE-THREADED BINARY TREE

- If the right link of a node is NULL and if it is replaced by the address of the preorder successor then the tree is said to be **Right Pre-threaded binary tree**.
- **Example:**

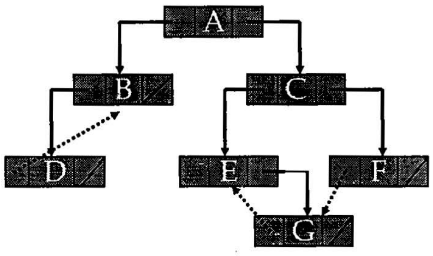


Preorder : A B D C E G F

39

A PREORDER THREADING OF A BINARY TREE / PRE-THREADED BINARY TREE

- If the left link of a node is NULL and if it is replaced by the address of the preorder predecessor, then the tree is said to be **Left Pre-threaded binary tree**.
- **Example:**



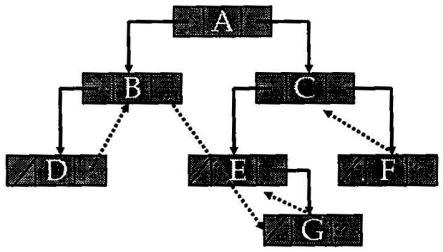
Preorder : A B D C E G F

40

41

A POSTORDER THREADING OF A BINARY TREE / POST-THREADED BINARY TREE

- If the right link of a node is NULL and if it is replaced by the address of the postorder successor then the tree is said to be **Right Post-threaded binary tree**.
- **Example:**

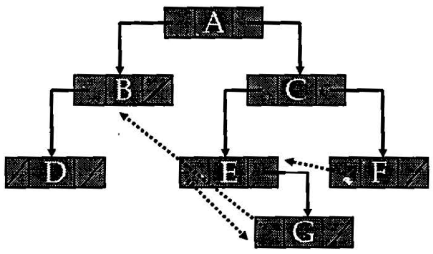


Postorder : D B G E F C A

41

A POSTORDER THREADING OF A BINARY TREE / POST-THREADED BINARY TREE

- If the left link of a node is NULL and if it is replaced by the address of the postorder predecessor, then the tree is said to be **Left Post-threaded binary tree**.
- **Example:**



Postorder : D B G E F C A

42

12

REPRESENTING AN EXPRESSION CONTAINING OPERANDS AND BINARY OPERATORS (+, -, *, /, ^ ETC) :

- An Infix expression consisting of operators and operands can be represented using a binary tree with root as the operator that is to be applied to the results of evaluations of expressions represented by left or right subtrees.
- A node representing an operator is a nonleaf node , whereas a node representing an operand is a leaf node.

43

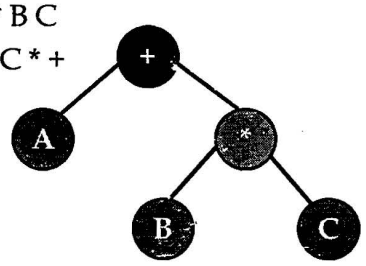
REPRESENTING AN EXPRESSION

- **Example1:**

Inorder : A + B * C

Preorder : + A * B C

Postorder : A B C * +



44

1) (A + B) * C

2) A + (B + C) * D $ (E * F)

3) (A + B * C) $ ((A + B) * C)

113

CREATION OF BINARY TREE FOR POSTFIX EXPRESSION

Scan the symbol from left to right and perform the following operations:
1. Create a node for the scanned symbol.

2. If the scanned symbol is an operand then push the corresponding node onto stack.

3. If the scanned symbol is an operator, then pop top two nodes from stack. First popped is attached to right of the node with scanned operator. Second popped is attached to the left of the node with scanned operator.

4. Push the node corresponding to the operator onto stack.

5. Repeat step1 to step4 till all the symbols of postfix expression are scanned. Once end of input is encountered, the address of root node of the expression tree is on the top of the stack.

45

CREATION OF BINARY TREE FOR PREFIX EXPRESSION

Scan the symbol from right to left and perform the following operations:
1. Create a node for the scanned symbol.

2. If the scanned symbol is an operand then push the corresponding node onto stack.

3. If the scanned symbol is an operator, then pop top two nodes from stack. First popped is attached to left of the node with scanned operator. Second popped is attached to the right of the node with scanned operator.

4. Push the node corresponding to the operator onto stack.

5. Repeat step1 to step4 till all the symbols of postfix expression are scanned. Once end of input is encountered, the address of root node of the expression tree is on the top of the stack.

46

K-BST

```
/*** PROGRAM 15: CREATION, DELETION AND TRAVERSAL OF NODES IN A BINARY
SEARCH TREE(BST) ***/
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct node
{
   int info;
   struct node *lchild,*rchild;
};
typedef struct node *NODE;
 void main()
{
    int choice,data,key;
    NODE root=NULL;
    NODE CREATE(NODE,int);
    void INORDER(NODE),POSTORDER(NODE),PREORDER(NODE);
    NODE DELETE_NODE(NODE,int);
    //clrscr();
    while(1)
   {

printf("\n1:CREATE\n2:INORDER\n3:PREORDER\n4:POSTORDER\n5.DELETION\n6:EXI
T");
            printf("\nEnter your choice\n");
            scanf("%d",&choice);
            switch(choice)
            {
               case 1: printf("\nEnter data to be inserted\n");
                       scanf("%d",&data);
                       root=CREATE(root,data);
                       break;
               case 2: if(root==NULL)
                                printf("\nEMPTY TREE\n");
                       else
                       {
                                printf("\nINORDER TRAVERSAL:\n");
                                INORDER(root);
                       }
                       break;
               case 3: if(root==NULL)
```

```
                        printf("\nEMPTY TREE\n");
                    else
                    {
                        printf("\nPREORDER TRAVERSAL:\n");
                        PREORDER(root);
                    }
                    break;
            case 4: if(root==NULL)
                        printf("\nEMPTY TREE\n");
                    else
                    {
                        printf("\nPOSTORDER TRAVERSAL:\n");
                        POSTORDER(root);
                    }
                    break;
            case 5: printf("\nenter the key to delete:\n");
                    scanf("%d",&key);
                    root=DELETE_NODE(root,key);
                    break;
            case 6: exit(0);
          }
    }
}

NODE CREATE(NODE root,int data)
{
    NODE newnode,x,parent;
    newnode=(NODE)malloc(sizeof(struct node));
    newnode->lchild=newnode->rchild=NULL;
    newnode->info=data;

    if(root==NULL)
        root=newnode;
    else
    {
        x=root;
        while(x!=NULL)
        {
          parent=x;
          if(x->info<data)
              x=x->rchild;
```

```
            else if(x->info>data)
                    x=x->lchild;
            else
            {
                    printf("\nNode is already present in the tree\n");
                    return(root);
            }
          }

          if(parent->info<data)
            parent->rchild=newnode;
          else
            parent->lchild=newnode;
   }
   return(root);
}

void INORDER(NODE root)
{
    if(root!=NULL)
    {
            INORDER(root->lchild);
            printf("%d ",root->info);
            INORDER(root->rchild);
          }
}

void PREORDER(NODE root)
{
          if(root!=NULL)
          {
            printf("%d ",root->info);
            PREORDER(root->lchild);
            PREORDER(root->rchild);
          }
}

 void POSTORDER(NODE root)
{
          if(root!=NULL)
          {
```

```
                POSTORDER(root->lchild);
                POSTORDER(root->rchild);
                printf("%d ",root->info);
              }
}
NODE  DELETE_NODE(NODE root, int key)
{
              NODE cur,q,parent,successor;
              if(root==NULL)
              {
                        printf("\nTree is empty\n");
                        return root;
              }
              parent=NULL,cur=root;
              while(cur!=NULL)
              {
                        if(key==cur->info)
                          break;
                        parent=cur;
                        cur= (key<cur->info)?cur->lchild:cur->rchild;
              }
              if(cur==NULL)
              {
                        printf("\nData is not found\n");
                        return root;
              }

              if(cur->lchild==NULL)
                        q=cur->rchild;
              else if(cur->rchild==NULL)
                        q=cur->lchild;
              else
              {
                        successor = cur->rchild;
                        while(successor->lchild != NULL)
                            successor = successor->lchild;

                        successor->lchild = cur->lchild;
                        q = cur->rchild;
              }
```

```
        if (parent == NULL)
                return q;
        if(cur == parent->lchild)
                parent->lchild = q;
        else
                parent->rchild = q;
        printf("\n%d is deleted from BST\b",key);
        free(cur);
        return  root;
}

NODEPTR searchNode(NODEPTR root, int item)
{
  if(root == NULL)
    return root;
  if(item == root->info)
    return root;
  if(item < root->info)
    searchNode(root->lchild, item);
  else
    searchNode(root->rchild, item);

}
```

K-Post fix Expression tree

```
/*** PROGRAM 17A: CREATION OF EXPRESSION TREE FOR POSTFIX EXPRESSION AND
EVALUATION ***/
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>
#include<math.h>
struct node
{
    char info;
    struct node *lchild,*rchild;
};
typedef struct node *TREE;
float eval(TREE);
TREE  create_tree(char *);

    void main( )
   {
            char postfix[30];
            float res;
            TREE * root = NULL;
            printf("\nEnter a valid Postfix expression\n");
            scanf("%s",postfix);
            root = create_tree(postfix);
            res = eval (root);
            printf("\nResult = %f\n",res);
  }

     TREE  create_tree(char postfix[ ])
    {
            TREE newnode, stack[20];
            int i=0, top = -1;
            char ch;
            while( (ch=postfix[i++])!='\0')
            {
               newnode = ( TREE )malloc( sizeof ( struct node ));
               newnode->info = ch;
               newnode->lchild = newnode->rchild = NULL;
               if(isalnum(ch))
                        stack[++top]=newnode;
               else
               {
```

K-Post fix Expression tree

```
                    newnode->rchild = stack[top--];
                    newnode->lchild = stack[top--];
                    stack[++top]=newnode;
                }
              }
               return(stack[top--]);
    }

    float  eval (TREE root)
    {
            float num;
            switch(root->info)
            {
              case  '+' : return (eval(root->lchild) + eval(root->rchild));
              case  '-' : return (eval(root->lchild) - eval(root->rchild));
              case  '*' : return (eval(root->lchild) * eval(root->rchild));
              case  '/' : return (eval(root->lchild) / eval(root->rchild));
              case  '^' : return (pow(eval(root->lchild), eval(root->rchild)));
              default:   if(isalpha(root->info))
                                {
                                  printf("\n%c = ",root->info);
                                  scanf("%f",&num);
                                  return(num);
                                }
                                else
                                return(root->info - '0');
    }
}
```