Structures

```
#include<stdio.h>
#include<conio.h>
struct stud
{   int m1;
    int m2;
    char usn[20];
    char name[50];
};
struct stud s1;
struct stud s[100];
```

Dynamic Memory allocation

malloc()

syntax: ptr = (cast-type*)malloc (byte-size)

ptr = (int*)malloc(100*sizeof(int));

calloc()

syntax: ptr = (float*)calloc(25, sizeof(float));

(cast-type*)calloc(n, element-size)

realloc()

syntax: ptr = realloc(ptr, newsize);

Write a C program to create a structure to store student information (student usn, 5 subject marks).

```
struct stud
{   char usn[10];
    int marks[5];
};
void main()
{   int n,i,j;
    printf("Enter number of students: ");
    scanf("%d",&n);
    struct stud s[100];
    for(i=0;i<n;i++)
    {   printf("Enter USN: ");
        scanf("%s", s[i].usn);
        for(j=0;j<5;j++)
        {   printf("Enter marks %d: ", j+1);
            scanf("%d",&s[i].marks[j]);
        }
    }
    printf("\nThe data received are:");
    for(i=0;i<n;i++)
    {   printf("\nUSN: %s", s[i].usn);
        for(j=0;j<5;j++)
        {   printf("\nMarks %d = %d", j+1, s[i].marks[j]);
        }
    }
    getch();
```

Pointers

Always use %u for printing a pointer (address)

Memory allocation

malloc() [stdlib.h]

```
int *pi;
pi = (int *)malloc(sizeof(int));
```

returns the address of the allocated space to pi.

Q. Create pointer variable for storing height (float) and section name (char) of a student.

```
#include<stdio.h>
#include<conio.h>
int main()
{
    float ht, *h;
    char sec, *s;
    printf("\nEnter height and section:");
    scanf("%f %c", &ht, &sec);
    h = &ht; s = &sec;
    printf("%f %c", *h, *s);
}
getch();
```

Q. Write a macro to allocate memory space.

```
#define MALLOC(p,s) if(p= malloc(sizeof(s))
                    printf("\nMemory insufficient");
```

(p: pointer, s: size)

```
main()
{
    int *pa, a;
    MALLOC(pa, a);
    return 0;
}
```

Q. Write a macro to add two numbers

```
#define ADD(a,b) (return (a)+(b);)   {(c)=(a)+(b), printf("%d",
```

```
main()
{
    int x, y;
    printf("\nEnter two numbers:");
    scanf("%d %d", &x, &y);
    printf("\nSum is:", ADD(x,y));
}
return 0;
```

7. Define a macro to swap two numbers.

```
#define SWAP(a,b) ((a)=(a)+(b), (b)=(a)-(b), (a)=(a)-(b),
  printf("a=%d, b=%d", (a), (b)))
```

8. Write a macro to compare two numbers. If num1 < num2 return -1, if num1 = num2 return 0, if num1 > num2 return 1.

```
#define COMPARE(a,b) ((a)<(b))?-1:((a)==(b))?0:1)
```

9. Write a C program to read an array of elements, perform selection sort and print the sorted array. Use separate functions for all the operations. Pass the array by reference to the functions.

```
#include<stdio.h>
#include<conio.h>
#define COMPARE(a,b) ((a)<(b))?-1:((a)==(b))?0:1)
void input(int arr[100], int n);
void sort(int arr[100], int n);
void print(int arr[100], int n);
void main()
{
    int a[100], n;
    printf("Enter the value of n: ");
    scanf("%d", &n);
    input(arr, n);
    sort(arr, n);
    print(arr, n);
}
void input(int arr[100], int n)
{
    int i;
    printf("Enter array elements: ");
    for(i=0; i<n; i++)
    {
        scanf("%d", &arr[i]);
    }
}
void print(int arr[100], int n)
{
    int i;
    printf("\n The array elements are: ");
    for(i=0; i<n; i++)
        printf("%d", arr[i]);
}
```

```
void sort(int arr[100], int n)
{
  int i, j, temp, pos;
  for(i=0; i<n; i++)
  {
    min = arr[i];
    for(j=i+1; j<n; j++)
    {
      if(arr[j] < min) [ if(COMPARE(arr[j], min) == -1) ]
      { min = arr[j];
        pos = j;
      }
    }
    temp = arr[i];
    arr[i] = arr[pos];
    arr[pos] = temp;
  }
}
```

ith iteration → ith min. [Ascending order]

do this using macro.

Q. Binary search function (write a recursion)

```
int/void binarysearch(int arr[100], int n)   // sorted ascend
{
  int start, end, mid, flag=0;
  start = arr[0];
  end = arr[n-1];
  while(start<=end)
  {
    mid = ((start+end)/2); if(ser==arr[mid] printf("found"
    else if(ser < arr[mid]) flag=1; break;
      end = mid-1;
    else if ser
      start = mid+1;
  }
  return flag;
}
```

0 1 2 3 4 5
1, 10, 15, 17, 18, 19

key = 11

```
int binsearch(int arr[100], int key, int start, int end)
{
  if(start>end) return 0;
  mid = (start+end)/2;
  if(key==arr[mid])
  { printf("found!");
    return (1);
  }
  else if(key<arr[mid])
  { end = mid-1;
    binsearch(arr, key, start, end);
  }
  else if(key) { start = mid+1;
    binsearch(arr, key, start, end);
```

mid = (0+5)/2 = 2
(11 != 15)
end = mid-1 = (1)
mid = (0+1)/2 = 0
arr[mid] = (1==key) ✓
found!

```
  return 0;
}
```

Write using COMPARE macro.

```
#define COMPARE(a,b) ((a)<(b))?-1:((a)>(b))?1:0)
int bin_search(int arr[100], int start, int end, int key)
{
  if(start>end)
    return(0);
  mid=(start+end)/2;
  switch(COMPARE(key, arr[mid])
  { case 0: printf("\n Found!");
            return(1);
            break;
    case -1; end=mid-1;
             bin_search(arr, start, end, key);
             break;
    case 1: start=mid+1;
            bin_search(arr,start,end,key);
            break;
  }
}
```

Write a c function to perform permutation over a list of characters.

```
void perm(char *list, int i, int n)
{
  int j, temp;
  if(i==n)
  { for(j=0;j<=n;j++)
    { printf("%c", list[j]); }
    printf(" ");
  }
  else
  { for(j=i; j<=n;j++)
    { swap(list[i], list[j], temp);
      perm(list, i+1,n);
      swap(list[i], list[j],temp);
    }
  }
}
```

a b c
b a c
b c a
a c b
c a b
c b a

a b c
b c
c b
a c b
c a b
c b a
b c a
b a c

How permute 4 element list

Algorithm

Criteria:

→ input i/p
→ output o/p
→ definiteness (no ambiguity)
→ finiteness
→ effectiveness (execution time should be min, memory constraints should be followed)

Abstraction

ADT - abstract data type.

ADT → Creators
→ Transformers (operations)
→ Reporters (output)

ADT for natural numbers.

Object → begin with 1, end with max memory of system.
$1 \le x \le$ max no. ⊖ representation is a particular machine.

Operations

* addition of 2 natural numbers.
  $x_3 = x_1 + x_2$ x_1 and x_2 satisfy object constraint
  x_3 should be less than the max value to be stored (otherwise it cannot be stored in the machine)
  ° if ($x_1 + x_2 >$ max no) display error
  else $x_3 = x_1 + x_2$.

* increment.
  $x = x + 1$; $x++$;
  x should be $1 \le x \le$ (max no. representation − 1)
  ° if ($x >$ max no) display error (it cannot be stored)
  else $x++$;

HW Write the ADT for a) complex numbers
b) boolean numbers. (only digit) AND, OR, NOT

Arrays.

ADT Arrayname, size, datatype:

Creator:
function create(arrname, size)

Transformer:
function write(arrname, index, element)

Reporter
function print(arrname, index) printf("%d", arrname[index]);

Q Write a C code segment to access one-dimension array using pointers.

```
int a[100], *p, n, i;
   p = a;
printf("\nEnter n: ");
scanf("%d", &n);
for(i=0; i<n; i++)
   scanf("%d", &a[i]);
printf("\nPrinting using pointers: ");
for(i=0; i<n; i++)
{ printf("%d", *p); OR *(p+i)
  p++;
```

Q Using malloc macro allocate the memory space for pointer *pb and this pointer is pointing to an array of 50 elements [dynamic array]

Remember
malloc() – no initial values.
calloc() – initial values are default values.
realloc() – change the number of blocks and block size.

```
#define MALLOC(p,s) (if(!(p=malloc(sizeof(s))))
                     printf("\nMemory insufficient");)

int main()
{
   int arr[50], *pb, i, n;
   MALLOC(pb, arr[0])        n*sizeof(int)
   scanf("%d", &n);
   pb = arr;
   for(i=0; i<n; i++)
      scanf("%d", &arr[i]);
                  (pb+i)
```

```
for(i=0; i<n; i++)
  printf("%d", *(pb+i));
  return;
}
```

Dynamic arrays

Syntax: calloc(p, size)

realloc(p, blocksize)

Hw write macros to paform calloc and realloc.

Q. Write a c code segment to access 2-D array using pointers.

```
int **pa, row, c, i, j
printf
cout<<"Enter number of rows and columns: ";
cin>>r>>c;  scanf("%d %d", &row, &col);
cout<<"Enter array elements: ";
for(i=0; i<r; i++)
  [ for(j=0; j<c; j++)
      cin>> *(*(pa+i)+j); ]

malloc(pa, row * sizeof(*pa));
for(i=0; i<row; i++)
  MALLOC(pa[i], col * sizeof(**pa));   // int.
for(i=0; i<row; i++)
  for(j=0; j<col; j++)
    scanf("%d", &pa[i][j]);
```

structures

user defined

```
typedef struct
{
  fieldnames;
} new-type;
```

```
typedef struct
{
  int rollno;
  char name[50];
} student;
```

Creation of structure variables

student s1, s2;

Unions

```
union
{
  int
  flow
  cha
};
union
```

Q. Write a
within
PhD. Us
student
accordi

```
union
{
  str
  {

  };
  str
  {

  };
  str
  {

  };
union
{

};
int ma
{ in
```

Unions

union eg
{
int a;
float b;
char n[50];
};
union eg e; → 50 bytes

[whichever has max memory requirement, that much memory is allocated to the union variable]

Q. write a c program to create union for student within the union define 3 structures of UG, PG and PhD. Use a variable to identify whether a student is doing UG, PG or PhD. store the data accordingly.

ug - puc, total_marks, state
pg - branch ug, year of completion, college, experience
PhD - area of research
pg, ug - discipline
ug, pg college

```
union stud
{
    struct ug
    {
        char pucollege[100];
        int total_marks;
        char state[50];
    };
    struct pg
    {
        char ugbranch[50];
        int yrcompletion;
        char college[50]
        int exp;
    };
    struct phD
    {
        char areaResearch[50];
        char discipline[50];
        char college[50];
    };
union stud
{
    struct ug u;
    struct pg p;
    struct phD ph;
};
int main()
{
    int ch;
    union studs;
```

```
printf("\n Enter your choice: ");
scanf("%d", &ch);

switch(ch)
{
    case 1: printf("\n Enter ug details: ");
            scanf("%d %p %s", s.u.pucollege, &s.u.total_mar
                s.u.state);
            break;
    case 2: printf("Enter pg details:");
            scanf("%s %d %s %d", s.p.ugbranch,
                &s.p.yrCompletion, s.p.college,
                &s.p.exp);
            break;
    case 3: printf("\n Enter phD details:");
            scanf("%s %s %s", s.phh.areaResearch,
                s.ph.descipline, s.ph.college);
            break;
}
```

Polynomial Addition

Add 2 equations (storing coeff & powers in an array of structures)

$A = 2x^{59} + x^3$

$B = 15x^{58} + 8x^3$

coeff
exp

	0	1	2	3	4	5	6	7	8	9	10
	2	1	15	8	15	2	3				
	59	3	58	3	58	59	3				

startA, endA, startB, endB, endD

keep 2 pointers - startA, endA to keep track of their positions.

keep another pointer avail to keep track of available places

Q. Define a structure to store the polynomial equation. Write 2 functions for the following:

a) To attach elements (terms) into polynomial equation

b) To perform addition of 2 polynomial equations.

```
#define MAX_TERMS 100
struct eqn
{   int coeff;
    int exp;
};
struct eqn Terms[MAX_TERMS];
int avail=0;
void attach(int coeff, int exp)
{
    if(avail>=MAX_TERMS)
    {   printf("\n Insufficient memory space");
        exit(0);
    }
    else
    {   Terms[avail].coeff = coeff;
        terms[avail].exp = exp;
        avail++;
    }
}
void padd(int startA, int endA, int startB, int endB,
          int *startD, int *endD)
{   int coeff;
    *startD = avail;
```

```
while(startA <= endA && startB <= endB)
{
    switch(COMPARE(terms[startA].exp, terms[startB].
                exp)
    {
        case +1: // a>b
                attach(terms[startA].coeff,
                    terms[startA].exp);
                startA++;
                break;
        case -1: // a<b
                attach(terms[startB].coeff,
                    terms[startB].exp);
                startB++;
                break;
        case 0: // a==b
                attach(terms[startA].coeff +
                    terms[startB].coeff, terms[startA].
                        exp);
                startA++;
                startB++;
                break;
    }
}
while(startA <= endA)
{
    attach(terms[startA].coeff, terms[startA].
                exp);
    startA++;
}
while(startB <= endB)
{
    attach(terms[startB].coeff, terms[startB].
                exp);
    startB++;
}
}
```

ADT

object ⇒ terms - coeff - float
- exp - int

functions
↳ create()
↳ attach()
↳ zero()
→ iszero()
padd()
You can also do
pmultiply()
psubtract()

Sparse

A = [

#define
typedef
{ int
int
int
y term;
term (a

Q. write

void
{ in
n
b
b
b
q
}

y

A = [1 0
0 0
0 0
2 3

Sparse Matrix (Non-zero elements are less) [More zero elements]

$$A = \begin{bmatrix} 0 & 2 & 0 \\ 0 & 0 & 5 \\ 0 & 30 & 0 \end{bmatrix}$$

A[0][1] = 2
A[1][2] = 5
A[2][1] = 30

No of non zero elements
Total no of rows and columns

```
#define max 10
typedef struct
{
   int row;
   int col;
   int val;
}term; → sparse matrix
term a[max];
```

1st element of sparse matrix total no. of rows, total no. of cols and no of non zero elements in the matrix is maintained.

	row	col	val
a[0]	3	3	3
a[1]	0	1	2
a[2]	1	2	5
a[3]	2	1	30

Q. Write a c function to transpose sparse matrix.

$$A' = \begin{bmatrix} 0 & 0 & 0 \\ 2 & 0 & 30 \\ 0 & 5 & 0 \end{bmatrix}$$

	row	col	value
b[0]	3	3	3
b[1]	1	0	2
b[2]	1	2	30
b[3]	2	1	5

```
void transpose (term a[], term b[])
{ int n, i, j, currentb;
  n = a[0].val;
  b[0].row = a[0].col;
  b[0].col = a[0].row;
  b[0].value = n;
  if(n > 0)
  {
    currentb = 1;
    for(i=0; i<a[0].col; i++)
      for(j=1; j<=n; j++)
        if(a[j].col == i)
        { b[currentb].row = a[j].col;
          b[currentb].col = a[j].row;
          b[currentb].value = a[j].val;
          currentb++;
```

y y

\# y

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 28 \\ 0 & 0 & 0 \\ 8 & 3 & 0 \end{bmatrix}$$

	row	col	val
a[0]	4	3	4
a[1]	0	0	1
a[2]	1	2	28
a[3]	3	0	8
a[4]	3	1	3

	row	col	val
b[0]	3	4	4
b[1]	0	0	1
b[2]	0	3	8
b[3]	1	3	3
b[4]	2	1	28

Fast transpose matrix

Given:

	row	col	val
a[0]	6	6	8
a[1]	0	0	15
a[2]	0	3	22
a[3]	0	5	-15
a[4]	1	1	11
a[5]	1	2	3
a[6]	2	3	-6
a[7]	4	0	91
a[8]	5	2	28

	row	col	val
b[0]	6	6	8
b[1]	0	0	15
b[2]	0	4	91
b[3]	1	1	11
b[4]	2	1	3
b[5]	2	5	28
b[6]	3	0	22
b[7]	3	2	-6
b[8]	5	0	-15

Index →	0	1	2	3	4	5
rowterms[]	2	1	2	2	0	1

(No. of non-zero terms in each row)

```
void fasttranspose (term a[], term b[])
{
    int rowterms[max_col], startingpos[max_col];
    int i, j, numcols = a[0].col, numterms = a[0].value;
    b[0].row = numcols;
    b[0].col = a[0].row;
    b[0].val = numterms;
    if(numterms > 0)
    {   for(i=0; i<numcols; i++)
            rowterms[i] = 0;
        for(i=1; i<=numterms; i++)
            rowterms[a[i].col]++;
        startingpos[0] = 1;
        for(i=1; i<numcols; i++)
            startingpos[i] = startingpos[i-1] +
                             rowterms[i-1];
        for(i=1; i<=numterms; i++)
        {   j = startingpos[a[i].col]++;
            b[j].row = a[i].col;
            b[j].col = a[i].row;
            b[j].val = a[i].val;
        }
    }
}
```

Index
rowterms[]
startingpos[]

§ Tracing
Index →
rowterms[]
startingpos[]

Pattern m
algorit

Multi-d

2 way
co

Row m

a[i][j

Row m

a[i][j

n-D A

a[i][j]

Strings

ADT: m
r
s

Opera

Index	0	1	2	3	4	5
rowterms[]	2	1	2	2	0	1
startingpos[]	1	3	4	6	8	8

[No of non zero elements in each row)

[Each row term is stored in the transpose matrix at startingpos]

8 Tracing for matrix on previous page (*)

Index →	0	1	2
rowterms[] →	2	1	1
startingpos []	1 ③	3	4 ⑤

B[] (Transpose)

	row	col	val
b[0]	3	4	4
b[1]	0	0	1
b[2]	0	3	8
b[3]	1	3	3
b[4]	2	1	28

Pattern matching algorithm - (KMP)

Multidimensional arrays representation

2 ways of representation - row major and column major.

Row major formal - (2D)

a[i][j] = base address + (i * max_col + j) (sizeof(datatype))

Row major formal - (3D)

a[i][j][k] = base address + (i * max_colj + j * maxkk + k) (sizeof(datatype))

n-D Generalization

a[i][j][k]....[n] = base address + (i * maxjj + j * maxk + ... n) (sizeof(datatype))

Strings

ADT: maxsize
ch char
string

Operation: copy
compare
concatenate
length
reverse
null
unull.
constant

Declare a string variable and initialize it to a name

char name[50] = {"msn"};

Predefined string functions: [string.h]

```
int strlen(string);
    strcpy(destin, source);
    strncpy(destin, source, n);
int strcmp(string1, string2);
int strncmp(string1, string2, n);
int strcmpi(string1, string2);
    strcat(string1, string2);
int strchr(string, ch); // it searches for character
                        ch in string and its
                        location is returned
int strrchr(string, ch); // returns the index of the
                         last occurrence of ch in the
                         string.
char* strtok(string, delimiters); // returns token between
                                  delimiters
    Eg strtok("ab,a.b,ab.", ",;.") If "," it returns
                                   "a.b"
char* strstr(string, pattern); // searches for a pattern
                               in a string and returns
                               a reference to that
                               substring.
```

Q. Give address in a string and search for city name in the address.

```
if(strstr("No10, Ramamurthynagar, Bangalore",
          "Bangalore"))
    printf("In search successful");
else printf("In Not found");
```

$2x^3 + 3x^2$
$x^2 + 5x$

```
int strspn(s, sparset); s="aefgh" sparset="abc" =①.
```
(returns length of s that matches the sparset from the beginning)

s="ababef" sparset="abc" =④.

s="abefab" sparset="abc" =②

int strcspn(cs, sparset);

[complementary string span]

returns the length of the string for which it is not from the sparset from the beginning.

s = "efghijab" sparset = "abc" (6).

strpbrk(s, sparset);

char* string pointer break:

returns a pointer to the position from where the sparset starts in s.

s = "efghijab" sparset = "abc"

9. write a string insertion function using predefined functions.

```
s1 = "mst";
s2 = "ri";
s3 = strncpy(s3, s1, 2);
     strcat(s3, s2);
     strcat(s3, s1+2);

char* insert(char s1[50], char s2[50], int inspos)
{
    char s3[100];
    strncpy(s3, s1, inspos);
    strcat(s3, s2);
    strcat(s3, s1+inspos);
    return s3;
}
```

Pattern search

n-find pattern search (match the last character of pattern with the string and then break the string)

The last character of the pattern will match p If there is a mismatch, no further comparisons are made.

s = "ab ab d abc"

p = "bab"

1st pos = no pattern match

2nd pos = "bad" → does not match (no pattern)

3rd pos "ad@" → does not match.

4th pos: "da(b)" → matches.
Compare the 1st pos now, (a) does not match
No pattern match.

5st pos: "ab(c)" → does not match No pattern match.

Q. Write n-find pattern search function.

```
int nfind(char s[100], char p[100])
{
    char sub[100] = NULL;
    int lp = strlen(p), ls = strlen(s), i;
    for(i=0; i <=(ls-lp); i++)
    {
        strcat(sub, s+i);
        if(s[i+lp-1] == p[lp-1])
        {
            for(j=0; j<lp; j++)
            {
                if(s[i+j] != lp[j])
                {
                    flag=0;
                    break;
                }
            }
        }
        if(flag)
            break;
    }
    return flag;
}
```

```
int nfind(char *s, char *p)
{
    int i, j, start=0;
    int lasts = strlen(s)-1;
    int lastp = strlen(p)-1;
    int endmatch = lastp;
    for(i=0; endmatch <= lasts; endmatch++, start++)
    {
        if(s[endmatch] == p[lastp])
            for(j=0; i = start; j<lastp && s[i] ==
                p[j]; i++, j++);
        if(j == lastp) return start;
    }
    return -1;
}
```

Pattern m

1. p = a a
   s = a b
   1st step start
   2nd step start
   3rd step start
   4th step
   5th step sta
   6th step

2. pat = "
   str = "

1st step: sta
2nd step: st
3rd step: s
4th step:

KMP alg
Knuth
Kweth

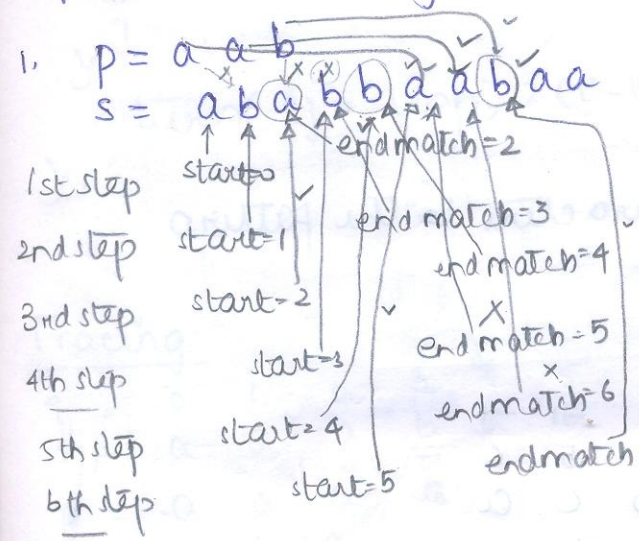
2 func

Computi
void fe
{
int
fai
for
{

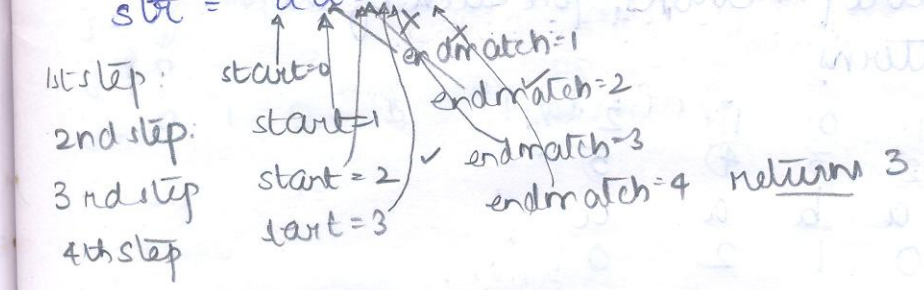
}

pattern matching using ~~end marker~~ four cases (contd)

1. p = a a b
   s = a b a b b a a b a a



2. pat = " c a "
   str = " a a b c a b "



KMP algorithm.

~~Knuth~~ ~~Kweth~~ Morris Pratt algorithm

2 functions: pattern matching
failure. (for pattern and not string)

Computing failure function:

```
void fail(char *pat)
{
    int n = strlen(pat);
    failure[0] = -1;
    for(j=1; j<n; j++)
    {
        i = failure[j-1];
        while((pat[j] != pat[i+1]) && (i >= 0))
            i = failure[i];
        if(pat[j] == pat[i+1])
            failure[j] = i+1;
        else
            failure[j] = -1;
    }
}
```

j	0	1	2	3	4
p	a	a	a	a	b
f	-1	0	1	2	-1

It checks for repetition of characters in the pattern

	0	1	2	3	
p	a	a	a	a	b
f	-1	0	1	2	-1

Eg ②

i		-1	-1	0	-1	-1	-1
j	0	1	2	3	4	5	6
p	a	b	a	a	b	c	a
f	-1	-1	0	-1	-1	-1	0

Q. Compute the failure functions for each of the following patterns

1.

i:	-1	-1	-1	0	1	2	(0) -1
j:	0	1	2	3	4	5	
p:	a	b	a	b	a	a	
f:	-1	-1	0	1	2	0	

2.

i	-1	-1	-1	0 (-1)	0	1	2	3
j:	0	1	2	3	4	5	6	7
p:	a	b	a	a	b	a	a	b
f:	-1	-1	0	0	1	2	3	4

3.

j:	0	1	2	3	4	5	6	7	8
p:	c	h	a	y	a	n	i	k	a

Pattern match:

```
int pmatch(char *s, char *p)
{
    int i=0, j=0;
    int lens = strlen(s);
    int lenp = strlen(p);
    while(i<lens && j<lenp)
    {
        if(s[i] == p[j])
        {
            i++;
            j++;
        }
        else if(j==0)
            i++;
```

```
else
    j = failure[j-1]+1;
}
return((j==lenp)?(i-lenp):-1);
}
```

j	0	1	2	3	4
p	a	a	a	a	b
f	-1	0	1	2	-1

Tracing:

i =	0	1	2	3	4	5	6	7	8	9
s =	a	a	a	b	a	a	a	a	b	
p =	a	a	a	a	b					
j =	0	1	2							
j =			2							
j =			1							
j =			0							
j =	0	1	2	3	4	5				

not equal

Return (9-5) = 4

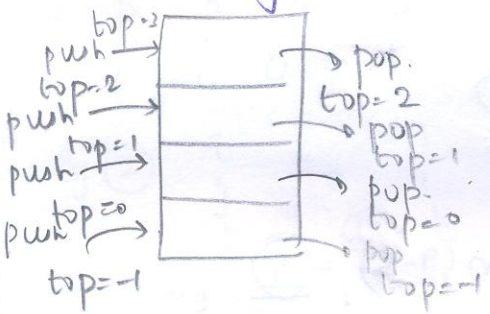
Stacks and Queues.

Stack

Last in first out (LIFO)

inserting elements into the stack - pushing

removing elements from the stack - popping.



ADT for stack.

Datatype of elements
maximum size of stack
top pointer.

Operations.

1) ~~Push~~ Creation of stack.
2) Push
3) Pop
4) display

creation of stack.

```
int stack[max_size];
```

Push.

```
if(top != max_size)
{   top++;
    stack[top] = item;
}
else
    printf("\n Overflow. Stack is full");
```

Pop.

```
if(top != -1)
{   ~~top~~ top--;
}
else
    printf("\n Underflow. stack is empty");
```

Q. Define the stack using structures. Define the functions for stack operations.

```
#define MAX 50
typedef struct
{   int key;
} element;
element stack[MAX];
int top = -1;
void push(int k)
{   if(top != MAX)
    {   top++;
        stack[top].key = k;
```

You can also write functions isfull() and isempty() to check overflow and underflow conditions.

```
    else
      printf("\n Overflow! Stack is full");
}
void pop()
{
    if(top!=-1)
    { printf("\n The item popped is %d", stack[top].key);
      top--;
    }
    else
      printf("\n Underflow condition! Stack is empty");
}
void display()
{
    int t=top; if(t==-1) printf("\n Underflow");
    else
      while(t!=-1)
      {
        printf("\n %d", stack[t].key);
        t--;
      }
}
int isfull()
{
    if(top==MAX)
      return 1;
    else
      return 0;
}
int isempty()
{
    if(top==-1)
      return 1;
    else
      return 0;
}
```

Q. Dynamically allocate memory space for stack using MALLOC macro.
If the stack is full, double the capacity of stack using realloc function.

```
#define MALLOC(p,s) (if(!~~malloc~~(p=malloc(s)))
        printf("\n Insufficient memory"); )

element *p;
MALLOC(p, sizeof(*p)); ~~int top=-1;~~
void ifull()
{
    if(top==MAX)
      { ~~p=~~ realloc(p, sizeof(*p)*2);
```

Queue

ADT.

Object: element datatype, maxsize.

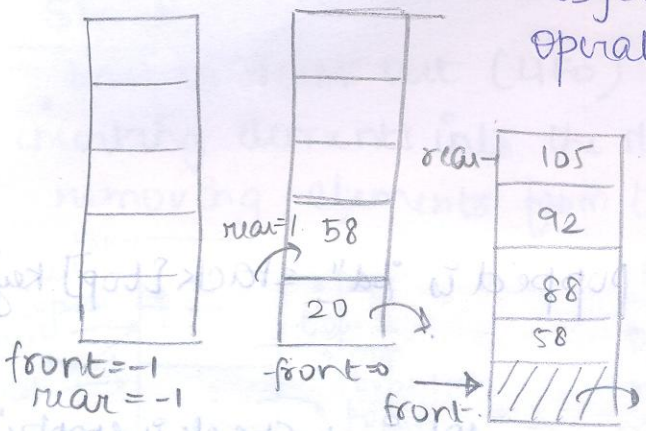
Operations: → creating the queue

→ add() check whether the queue is full()

```
if(rear==(MAX-1))
{"overflow"}
```

→ removal()

```
if(front==rear)
{"empty"}
```



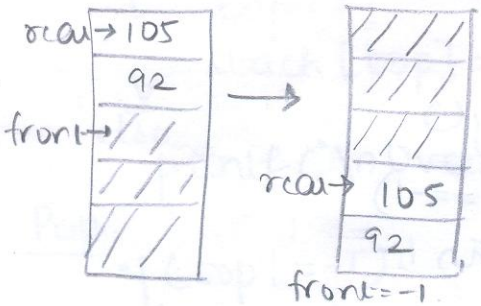
Drawback of queue

Q. Write C functions to add elements into the queue and remove elements from the queue.

Methods of overcoming the shortcoming of queues

→ compaction → circular queue.

Compaction



Each removal from the queue should lead to shifting of each element ahead by one.

Q. Write the C function to perform compaction of queue elements after removing an element from the queue. HW

Define the element data

Queue with structures

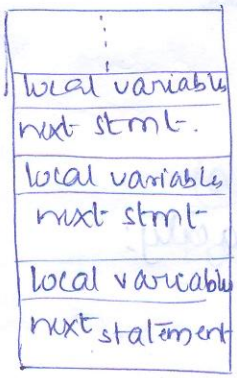
```
#define MAX 50
typedef struct
{
    int key;
} element;
element queue[MAX]; int front=-1; int rear=-1;
int isfull()
{
    if(rear==(MAX-1))
        return 1;
    else
        return 0;
}
```

```
int isempty()
{
    if(front==rear)
      return 1;
    else
      return 0;
}
void add(int item)
{
    if(isfull())
      printf("\n Addition into the queue is difficult");
    else
      {
        rear++;
        queue[rear].~~element~~ key = item;
      }
}
void remove()
{
    if(isempty())
      printf("\nUnderflow! No removal is possible");
    else
      {
        front++;
        printf("\n Element removed is %d", queue[front].
                                    ~~ele~~ key);
      }
}
```

Application of stack.

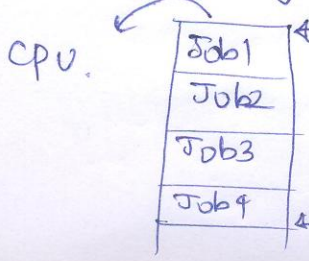
processor-stack()

Recursive factorial function.

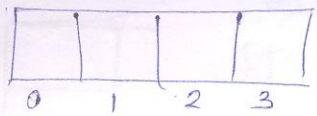


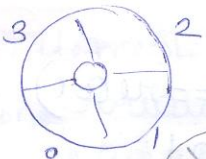
Application of queues

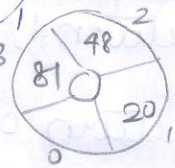
Job scheduling



Circular Queue







front = 0, rear = 0.

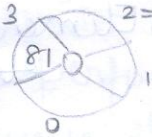
```
add()
{   rear_copy = rear;
    rear = (rear+1) % max;
    if(rear == front)
      {printf("CQ is full \n overflow"); rear = rear_copy;}
    else
      queue[rear] = item;
}

remove()
{   if(rear == front)
      {printf("\n CQ is empty \n Underflow");
       return -1;
      }
    else
      {front = (front+1) % max;
       return(queue[front]);
}
```

In circular queue to differentiate full circular queue and empty circular queue, we leave one location as empty.

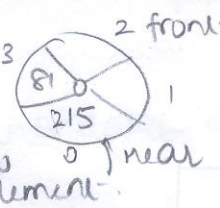
Remove 2 items + add 1

front = 1
return 20

front = 2
return 48



Add front = 2
rear = 0




```
void add(element item)
{   rear = (rear+1) % capacity;
    if(front == rear)
      {queuefull(); }
    queue[rear] = item;
}
void queuefull()
{   element *newqueue;
```

```
MALLOC(newqueue, 2*sizeof(*queue));
int start=(front+1)%capacity;
if(start<2) //no wrap around in CQ
   copy(queue+start, queue+start+capacity-1, newqueue);
else  // wrap around in CQ.
 { copy(queue+start, queue+capacity, newqueue).
   copy(queue, rear+1, newqueue+capacity-start);
 }
front = 2*capacity-1;
rear = capacity-2;
capacity *=2;
free(queue);
queue = newqueue;
}
```

Application of stacks

Evaluation of expressions.

① Convert: $(a+b)*(c/(d+k))$

$(ab+)*(c/(dk+))$

$(ab+)*(cdk+/)$

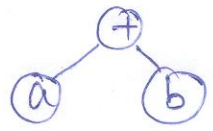
Postfix: $ab+cdk+/*$

Prefix: $(+ab)*(c/(+dk))$

$(+ab)*(/c+dk)$

$*+ab/c+dk$.

② Exp.

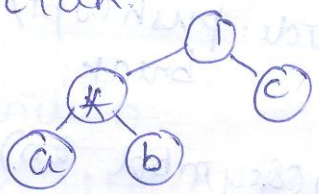


Infix: $a+b$

Postfix: $ab+$.

Prefix: $+ab$.

③ $a*b/c$

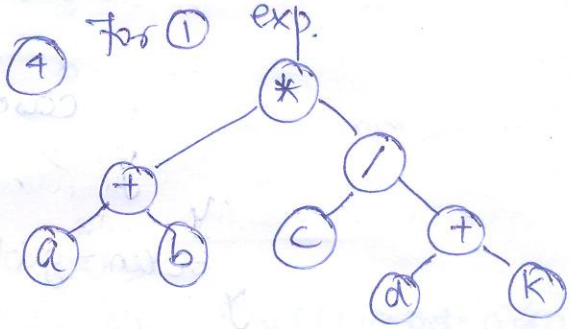


$a*b/c$.

Postfix: $ab*c/$

Prefix: $/*abc$.

④ For ① exp.



$(a+b)*(c/(d+k))$

Postfix: $ab+$ ~~$*dk+$~~ $cdk+/*$

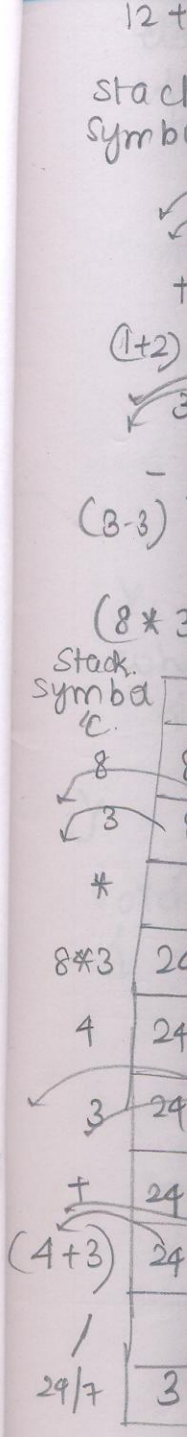
$ab+cdk+/*$

Prefix: $*+ab/c+dk$.

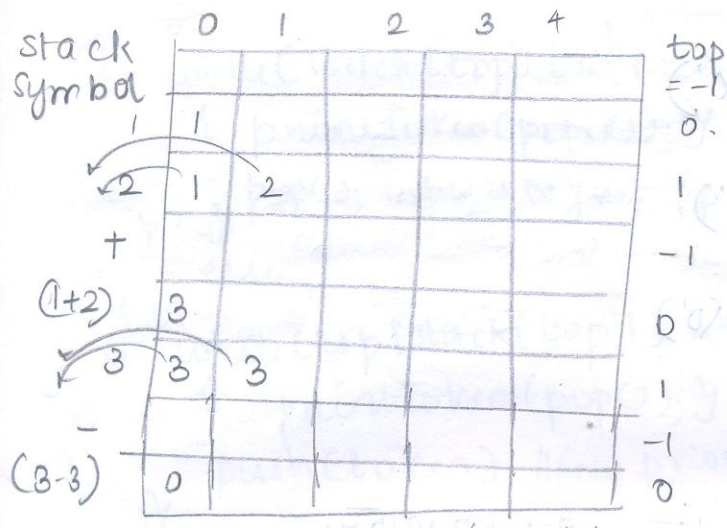
```
int eval(void)
{
  typedef enum {lparen, rparen, plus, minus, mul,
      divide, mod, eos, operand} precedence;
  int eval (void)
  {
    precedence token;
    char symbol;
    int op1, op2;
    int n=0;
    int top=-1;
    token = gettoken (&symbol, &n);
    while (token != eos)
    {
      if (token == operand)
        push (symbol-'0');
      else
      {
        op2 = pop();
        op1 = pop();
        switch (token)
        {
          case plus: push(op1+op2);
                     break;
          case minus: push(op1-op2);
                      break;
          case mul: push(op1*op2);
                    break;
          case divide: push(op1/op2);
                       break;
        }
      }
      token = gettoken (&symbol, &n);
    }
    return pop();
  }
```

Priority:

( )	17
+ −	12
* / %	13



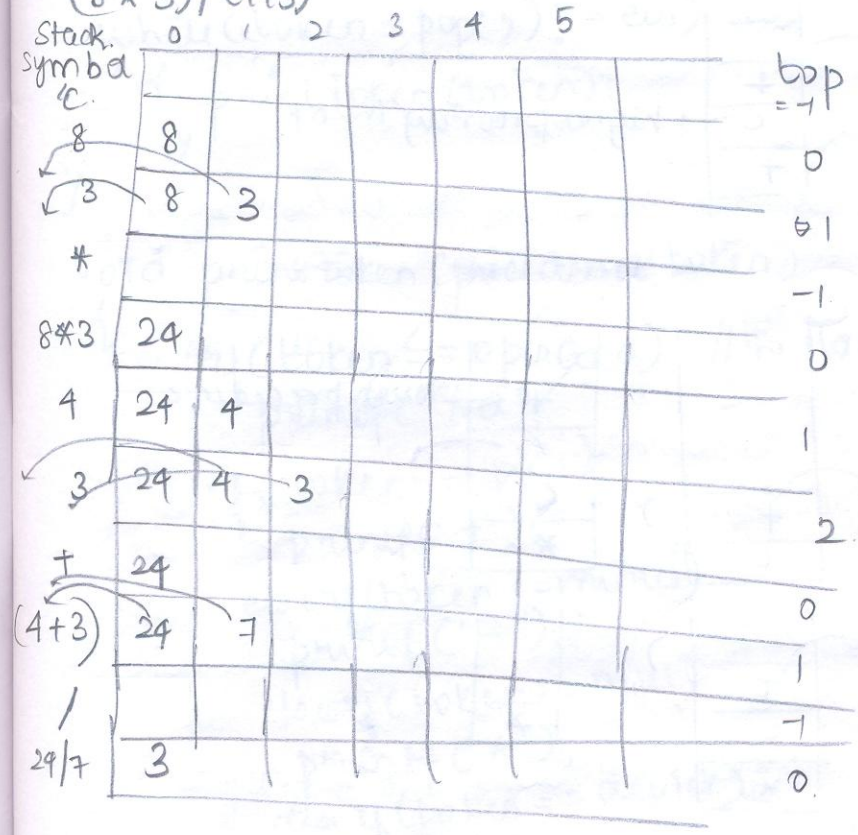
12+3-



(83*)/(43+) 83*43+/

29/7

(8*3)/(4+3)



gettoken function

```
precedence gettoken(*symbol, *n)
{   if( symbol<'0' && isdigit(symbol)) if((symbol-'0')>=0 &&
                                          symbol-'0')<=9)
        return operand;
    else
    {   if( symbol=='+')
            return plus;
        else if(symbol=='-')
            return minus;
        else if(symbol=='*')
            return mul;
        else if(symbol=='/')
            return divide;
```

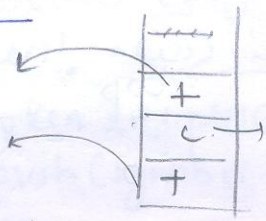
```
else if(symbol == '%')
    return mod;
else if(symbol == '(')
    return lparen;
else if(symbol == ')')
    return rparen;
else if(symbol == '\0')
    return eos;
}
}
```

Infix to postfix

stack stores the opuralors.

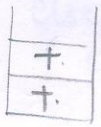
a+(b+c)

abc++



(a+(b+c))

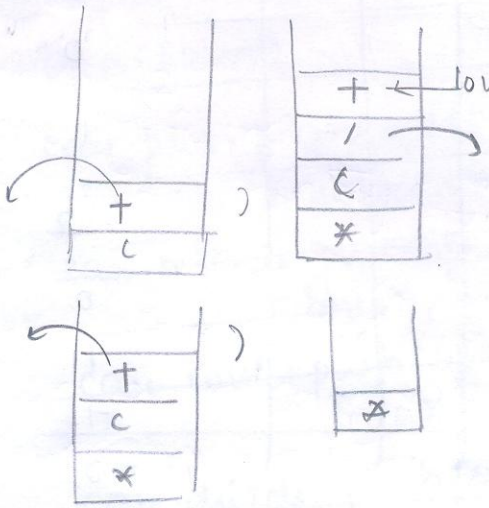
abc++



(a+b) * (b/c+a)

ab+ bc/a+

ab+ bc/a+ *



Postfix conversion

Here the stack is always for 8 oprators.

```
int isp[] = {0, 19, 12, 12, 13, 13, 13, 0};
int icp[] = {20, 19, 12, 12, 13, 13, 13, 0};
void postfix()
{
    char symbol;
    precedence token;
    int n=0;
    stack[0] = eos;
    for(token = gettoken(&symbol, &n); token != eos; token =
            gettoken(&symbol, &n))
    {
        if(token == operand) printf("%c", symbol);
```

Trace:

(b*b)

bb*

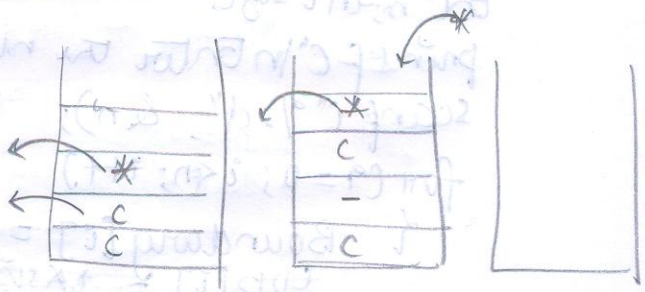
```
else if (token == rparen)
{ while(stack[top] != lparen)
  { printToken(pop());}
  pop(); //this is to just pop the lparen and discard
}
else
{ while(isp[stack[top]] >= icp[token])
  { printToken(pop());}
  push(token); //only priority level is stored
}
}
while((token = pop()) != eos)
{ printToken(token);
}
}
void printToken(precedence token)
{ if(token == operand) //The stack only has operators
  printf("%d");
  if(token == plus)
  printf("+");
  else if(token == minus)
  printf("-");
  else if(token == mul)
  printf("*");
  else if(token == devide)
  printf("/");
  else if(token == mod)
  printf("%");
}
```

isp- in stack.
icp- incoming

left to right - associativity

1. -5 + (2+3)
-15- 23++

Trace:
(b*b) - (4*a*c)) / (2*a)
bb*4a*



Multiple stacks and Queues

n = no of stacks (input from user)

For each stack, memory = max_mem_size/n.

Things to be cared for: stack bottom (max size of previous stack = stack bottom of the new stack).

An array is maintained to keep track of boundary

Boundary[]

For empty stack i,

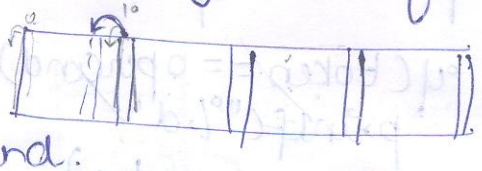
```
if (stack[top[i]] == Boundary[i-1])
    stack empty.
```

Overflow of

```
if (stack[top[i]] == Boundary[i])
    overflow.
```

When one stack is full and its previous stack has empty space, then shift its boundary to the left.

Left shifting of stacks



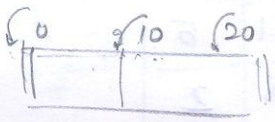
If i-1 is not free, check i-2 and so on until space is found.

If left-shifting is not possible then try right shifting of stacks.

Q. Define multiple stacks using static array. Initialize top pointer values and boundary values for all n stacks. Define push and pop functions for multiple stacks.

```
#define MAX 100
int main()
{
    int stack[MAX], Boundary[MAX], top[MAX];
    int n, int size;
    printf("\n Enter the number of stacks: ");
    scanf("%d", &n);   size = MAX/n;
    for(i=0; i<=n; i++)
    { Boundary[i] = i*size; }
      top[i] = i*size;
      top[i] = Boundary[i]-1;
    }
```

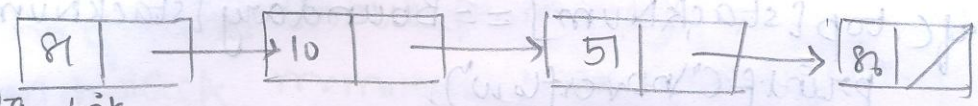
```
void push (int item, int stackNum)
{
    if (top[stackNum] == Boundary[stackNum+1])
        printf("\nOverflow");
    else
    { stack[++top[stackNum]] = item;
    } //top[stackNum]++; }
```




```
int pop (int stackNum)
{
    int item;
    if (top[stackNum] == Boundary[stackNum]-1;
    { printf("\nUnderflow"); exit(0); }
    else item =
        { stack[top[stackNum]--];
        }
    return item;
}
```

```
void leftshift (int stackNum)
{
    int num, flag=1;
    num = stackNum-1;
    while(flag) num >=0)
    {
        if (top[num] != Boundary[num+1]-1)
            { // flag=0;
              break;
            } num--;
    }
    // this lets me find the stacknumber of stack that is
    // empty to the left of the concerned stack
```

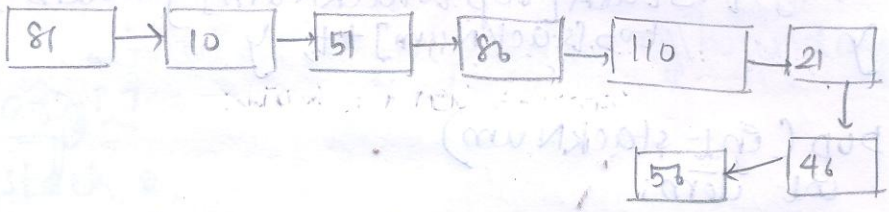
Linked list



	data	link
0	46	6
1	51	4
2	110	7
3	81	5
4	86	X
5	10	1
6	56	/
7	21	0

Create 3 more data nodes 110, 21, 46, 56
(2) (7) (0) (6)

headerNodeIndex = 3



Drawback (Static linked list).

When the static array is full, new nodes cannot be added

To overcome use dynamic memory allocation.

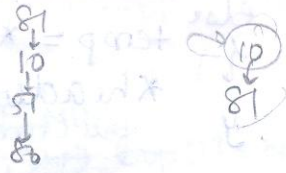
Self referential structures – pointer to its own kind.

```
typedef struct
{  int data;
   node *link;
}node;
```

Q. Write a function to create a linked list. If the linked list is empty the node created is the 1st node in the list. If it is not empty, insert the node in the beginning of the linked list.

```
void createlinklist(int d)
{  node *node1, *temp;
   if(**headerNode->link == NULL)
   {
      *headerNode = MALLOC(node1, sizeof(*node));
      *node1->data = d;
      *node1->link = NULL;
      *headerNode = d;
      *node1->link = NULL;
   }  *headerNode = node1;
   }
   else
   {  node1 = headerNode;
      do while(node1->link != NULL)
      {  node1 = node1->link;
      } while(node1->link != NULL);
```

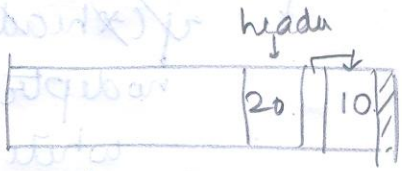
```
while (node1 -> link != NULL)
  { node1 = node1 -> link;
  }
MALLOC (temp, sizeof(*temp));
 node1 -> link = temp;
 node1 =  temp -> data = d;
          temp -> link = NULL;
       node1 = temp;
}
```




```
node *temp
{ MALLOC(temp, sizeof(*temp))
    temp -> link = node1;
    headNode = temp;
}
```

(stack type LS)

```
typedef struct node *nodeptr;
typedef struct
{
   int data;
   nodeptr link;
}node;

nodeptr* header=NULL;
void insert(nodeptr* header, int item)
{
   nodeptr temp ;
   MALLOC(temp, sizeof(*temp))
   temp->data = item;
   if((*header))
     { temp->link = NULL;
       *header= temp;
     }
   else //linked list of nodes already exists.
     { temp-> link = *header;
       *header = temp;
     }
}
```



HW Write the function for inserting in the middle of the linked list
→ end of the linked list.

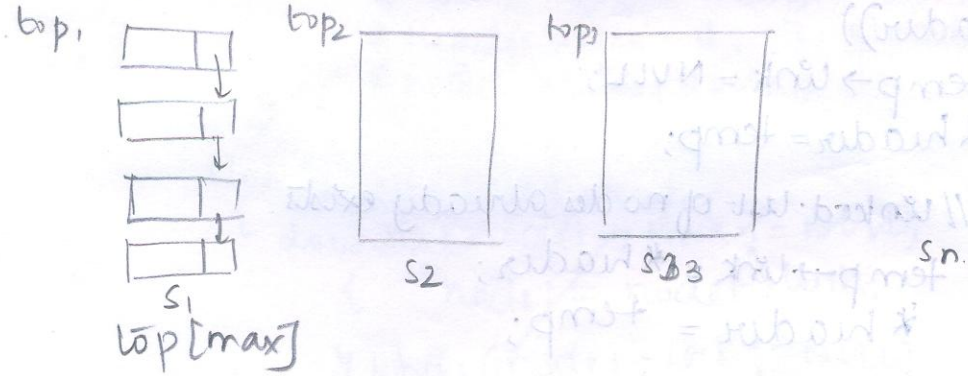
2. Deletion from front of the linked list. (stack type)

```
void delete (nodeptr *header)
{   if(*header==NULL) { printf("\nEmpty"); }
    if(*header->link = NULL) nodeptr temp;
      { *header= NULL;
      }
    else
      { temp = *header; free(temp);
        *header = (*header)-> link;
      }   free(temp);
}
```

To find a node with a particular data

```
void find (nodeptr *header, int item)
{   if(*header) int flag=0;
    nodeptr temp = *header;
      while (temp)
      {
          if(temp->data ==item)
          { printf("\nData found!"); flag=1;
            break;
          }
        temp= temp->link;
      }
    if(!flag)
        printf("\nSorry data not found");
}
```

Multiple stacks *

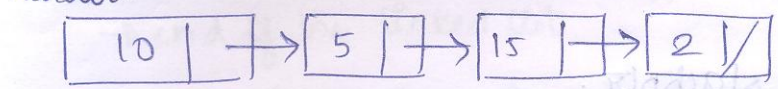


```
typedef struct
{
    int key;
} element;
typedef struct stack * stackptr;
typedef struct
{
    element data;
    stackptr link;
} stack;
```

a) Pushing an element into one of the stacks
b) Popping element from one of the stacks.

```
stackptr top[100]; //initialize all to NULL
                   for(i=0; i<n; i++) top[i] = NULL;

void push(int stackNum, element item)
{ stackptr temp; MALLOC(temp, sizeof(*temp));
  temp->data = item;
  // check if stack is empty or not.
    if(top[stackNum]==NULL)
    {
        temp->link = NULL;
        top[stackNum] = temp;
    }
    else
    {
        temp->link = top[stackNum];
        top[stackNum] = temp;
    }
}

void pop(int stackNum)
{
    stackptr temp;
    if(top[stackNum])
    {
        temp = top[stackNum];
        top[stackNum] = top[stackNum]->link;
        free(temp);
    }
    else
        printf("\nstack is empty");
}
```

Insertion in middle of linked list



Find key and insert new node after key node.

insert

```
void (nodeptr *header int key, int item)
{   nodeptr temp ; MALLOC(temp, sizeof(*temp));
    nodeptr temp keynode = *header;
    while(keynode)
    {   if(keynode->data == key)
        { break; }
        else
            keynode->link = keynode->link;
    }
    nodeptr temp
    temp->data = item;
    temp->link = keynode->link;
    keynode->link = temp;
}
```

Deletion in middle

(with key value)

keep track of previous node

```
void deleteNode(nodeptr *header, int key)
{   nodeptr keynode = *header;
    while(keynode)
    {
    nodeptr prevnode = NULL;
    if(*header->data == key) // deleting 1st node.
    {   keynode = *header;
        *header = *header->link;
        free(keynode);
    }
    else
    {   keynode = *header->link;
        prevnode = *header;
        while(keynode)
        {
```

Insertion into multiple queues

```
void push (int queueNum, element *item)
{
    queuepte temp;
    MALLOC (temp, sizeof (*temp));
    temp->data = item; temp->link = NULL;
    else {rear[queueNum]->link = temp;}
        rear[queueNum] = temp;
    if (front[queueNum] = NULL)
    {   front[queueNum] = temp}
}
```

```
typedef struct queue *queuepte;
typedef struct
{
    element data;
    queuepte link;
} queue;
queuepte front[100], rear[100];
// Initialize all
// of them to NULL.
```

Deletion from multiple queues

```
element remove
void pop(int queueNum)
{   queuepte temp;
    if (!front[queueNum] == NULL)
    {   printf("\n The queue is empty and deletion
            is not possible"); exit(1);
    }
    else
    {   MALLOC (temp, sizeof (*temp));
        temp = front[queueNum];
        printf( front[queueNum] = front[queueNum]->
                                  link;
        return (temp->data);
        free(temp);
    }
}
```