Experiment 1

0) WAP to implement array operation.

1) Find avg of 10 no using array

```
#include<stdio.h>
int main() {
int num[10];
int i, sum=0;
float avg;

printf("Enter 10 numbers :");

for(i=0; i<10; i++)
{scanf("%d", &num[i]);
sum += num[i];
}
avg = sum/10;

printf(" Avg of 10 numbers = %.2f", avg);

return 0;
}
```

2) Display following pattern

```
*
# #
* * *
# # # #
* * * * *
```

```
#include<stdio.h>
int main () {
int i,j;
for(i=1 ; i<=4;i++) {
if(i%2==0) {
printf(" # ");
}
else {
printf(" * ");
}
}
return 0;
}
```

3) Find first repeating no in an array.

```
#include<stdio.h>

int main (){
int arr [100],n,j,i;
int found = 0;

printf ("Enter number of elements in the array:");
scanf ("%d",&n);

printf("Enter %d elements:\n", n);
for(i=0;i<n;i++){
scanf("%d", &arr[i]);
}
for (i=0;i<n;i++){
for (j=i+1;j<n;j++){
if(acc[i]==acc[j]){
printf("First repeating number is: %d \n", arr[i]);
found=1;
break;
}
} if (found){
break;
} }
if (!found){
printf ("No repeating number found. \n");
}

return 0;
}
```

o/p:-

Enter number of elements in the array: 4
Enter 4 elements
4
1
4
3
First repeating number is 4.

4. Find greater / lowest element in array.

```
#include<stdio.h>
int main() {
  int arr[100], n, i;
    int max, min;

printf("Enter no. of elements in the array:");
scanf("%d", &n);

printf("Enter %d elements: \n", n);
  for(i=0; i<n; i++){
  scanf("%d", &arr[i]);
    }
      max = min = arr[0];

  for(i=1; i<n; i++){
    if(arr[i] > max){
      max = arr[i];
  }
  if(arr[i] < min){
      min = arr[i];
    }
}
```

```
printf("Greatest element = %d \n", max);
printf("Lowest element: %d \n", min);

return 0;
}
```

O/P :-

Enter no. of elements in the array : 3
Enter 3 elements
1
2
3
Greatest element : 3
Lowest element : 1

O/P :-

Enter no. of elements in array : 3
Enter 3 elements
1
2
3
Greatest element : 3
Lowest element : 1

5)

```
#include<stdio.h>

int main(){
  int arr[100],n,i;

printf("Enter no.of elements in array:");
scanf("%d",&n);

printf("Enter %d elements \n",n);
  for(i=0;i<n;i++){
    scanf("%d",&arr[i]);
  }
  for(i=0;i<n;i++){
   if((i+1)%2!=0){
   arr[i]=arr[i]* arr[i];
   }
   }
  printf("Array after squaring odd position elements\n");
   for(i=0;i<n;i++){
     printf("%d",arr[i]);
     }
     return 0;
     }
```

O/P:-

Enter no.of elements in array: 3
Enter 3 elements 13
34
22
Array after squaring odd position elements: 169 34 484

classmate
Date
Page

Extra

1.
```
#include<stdio.h>
int main () {
 for (int i=0 ; i<5 ; i++){
 for ( int j=1 ; j<=i ;j++) {
  printf ("%d",j );
 }
  printf ("\n");
}
  return 0;
  }
```

O/P:-

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

2.
```
#include<stdio.h>
 int main () {
 for (int i=1 ; i<=4 ; i++) {
 for (int j = 1 ; j <= i ; j++) {
    printf ("#");
  }
   printf("\n");
 }
  return 0;
  }
```

O/P:-

```
#
# #
# # #
# # # #
# # # # #
```

3.
```
#include <stdio.h>
  int main() {
 for (int i= 1; i<=4 ; i++){
 for (int j = 1 ; j<=i ; j++){
  printf ("\n");
  }
   return 0;
   }
```

op:-

```
*
* *
* * *
* * * *
```

4.
```
#include<stdio.h>
 int main() {
 int rows =4
 for (int i=1 ; i<=rows ; i++) {
  for (int j = 1; j<= i; j++){
    switch (i) {
   case 1: printf(" *") break;
   case 2: printf (" #") break;
  case 3: printf(" $") break;
   case 4: printf(" !") break;
    }
 } printf ("\n");
  }
   return 0;
   }
```

Experiment 2

Search data using linear search consider following list to follow.

56	36	89	57	1	0	67	59

1. Search item, 1 from the above list & write if the item is found or not & for 55.

```
#include<stdio.h>
int main() {
int arr[] = {56, 36, 89, 57, 1, 0, 67, 59};
    int n = 8;
int item = 1;
    int found = 0;

for (int i=0; i<n; i++) {
    if (arr[i] == item) {
        found = 1;
printf("Item %d found at position %d\n", item);
    break;
  }
};
if(!found) {
printf("Item %d not found in list \n", item);
 }
 return 0;
   }
```

O/P:-

Item 1 found at position 4.

for key = 55

```
#include<stdio.h>
int main() {
 int arr [] = { 56, 36, 89, 57, 1, 0, 67, 59};
 int n=8;
   int ;item, found = 0;

 printf("Enter  item to search:");
    scanf(" %d", & item);

 for (int i=0; i<n ; i++) {
    ;if(arr[i] == item) {
        found = 1;
  printf("Item %d found at position %d\n",it
  break;
   }
 }

 if (!found) {
 printf(" Item %d not found in list \n",item);
 }
 return 0;
}
```

O/P:-

Item 55 not found in list.

Using Binary Search

~~#include<stdio.h>~~
~~int main(){0,1,36,56,57,59,67,85}~~
~~int~~

```
#include<stdio.h>
int main(){
int arr[]={0,1,36,56,57,59,67,87};

int n=8;
int item, low=0, high=n-1, mid;

printf("Enter the item to search:");
scanf("%d",&item);

while(low<=high){
  mid=(low+high)/2;

if(arr[mid]==item){
  found=1;
printf("Item %d found at position %d\n"
break;
}else if(item<arr[mid]){
      high=mid-1;
    }else{
      low=mid+1;
    }
  }
if(!found){
printf("Item %d not found in list\n");
  }
return 0;
  }
```

O/P:-

Enter the item search: 1

Item 1 found at position 1.

Practice que for Expt 3

1) WAP to copy elements of one array into another array in a reverse order.

```
#include <stdio.h>

int main(){
int original [5]={ 10,20,30,40,50};
   int reversed [5];
      int i;

for(i=0 ;i<5; i++){
 reversed[i] = original[i];
  }
 printf ("Reversed array : ");
   for (i=0 ; i < 5; i++){
 printf("%d", reversed[i]);
 }
  return 0;
}
```

O/P:-

Reversed array : 50 40 30 20 10

2. WAP in C to count total no. of duplicate elements in an array.

```
#include<stdio.h>

int main ()
{
int arr[1500], n, i, j, count=0;
printf("Enter size of array :");
scanf("%d", &n);
}
printf("Enter %d elements :\n", n);
for(i=0; i<n; i++){
scanf("%d", arr[i]);
}
for(i=0; i<n; i++){
scanf("%d", &arr[i]);
}
for(i=0; i<n; i++){
for (j=i+1; j<n; j++){
if (arr[i]==arr[j]){
count++;
}
}
}
printf("Total no. of duplicate elements=%d\n", arr
return 0;
}
```

O/P:-

```
Enter size of array : 7
Enter 7 elements:
1 2 3 2 4 5 6

Total no of duplicate element = 2.
```

3. WAP in C, to print all unique elements in an array.

```
#include <stdio.h>

int main () {
int arr[50], n, i, j, u;

printf(" Enter %d elements :\n", n);
scanf("%d", &arr[i]);
}
printf(" Unique elements are: ");
for (i=0; i<n ; i++) {
        u = 1;

for (j=0; j <n; j++) {
  if(i != j && arr[i] == arr[j] {
        u = 0;
        break;
  }
}
if (u) {
  printf("%d", arr[i]);
  }
}
return 0;
}
```

O/P:- Enter array size : 7
Enter 7 elements:
1 2 3 2 4 1 5
unique element : 3 4 5

4) WAP in C to separate odd & even integers in 2 separated array.

```
#include<stdio.h>

int main(){
int arr[] = {10,21,4,145,66,93};
    int even[10], odd[10];
    int i, evencount=0, oddcount=0;
int n=sizeof(arr)/sizeof(arr[0]);

for(i=0;i<n;i++){
    if(arr[i] %2 ==0){
    even[even[count++]= arr[i];
    }
}
printf("Even no's :\n");
for(i=0; i< evencount; i++){
printf("%d", even[i]);

printf("\n odd numbers : \n");
for(i=0; i<oddcount; i++){
    printf("%d", odd[i]);
}
return 0;
}
```

5) WAP in C to find second smallest element in an array.

```
#include <stdio.h>

int main() {
    int arr[100], n, i, j, temp;

printf("Enter no. of elements:\n");
    scanf("%d", &n);

printf("Enter %d elements:\n", n);
    for(i=0; i<n; i++){
        scanf("%d", &arr[i]);
    }
 for(i=1; i<n; i++){
    if(arr[i] != arr[0]){
    printf("Second smallest element is: %d\n", arr[i]);
    return 0;
}
}
printf("All elements are the same\n");
    return 0;
    }
```

O/P :-

Enter no. of elements : 5

Enter 5 elements : 3 1 4 1 2

Second smallest element is : 2

13/11

Experiment 3

```
#include<stdio.h>
int main()
{
int a[] = {5,1,4,2,8};
int n = sizeof(a)/sizeof(a[0]);
  int i,j,t;
printf("Original array:");
 for(i=0;i<n;i++)
 printf("%d",a[i]);

for(i=0;i<n-1;i++)
{
for(j=0;j<n-i-1;j++)
{
 if(a[j]>a[j+1])
 {
     t= a[j]
     a[j] = a[j+1]
     a[j+1] = t;
   }
}
}
printf("Sorted array:");
 for(i=0;j<n;i++)
  printf("%d",a[i]);

  return 0;
}
```

O/P :- Original array: 5 1 4 2 8
Sorted array: 1 2 4 5 8

2

```
#include<stdio.h>
int main()
{
  int a[] = {5,1,4,2,8};
  int n = sizeof(a) / sizeof(a[0]);
      int i, j, m, t;

  printf("Original array:");
    for(i=0; i<n; i++)
    printf("%d", a[i]);

  for(i=0; i<n-1; i++)
    {
      m=i;
  for(j=i+1; j<n; j++)
      {
    if(a[j] > a[m])
      a[m] = t;
    }
    }
  printf("Sorted array in descending");
      for(i=0; i<n; i++)
      printf("%d", a[i]);
    return 0;
  }
```

3. Given numbers - 100 200 300 400 500

100 200 300 400 500

100 200 300 400 500

100 200 300 400 500

100 200 300 400 500

1. Start
2. Compare each pair of adjacent elements & swap if the first element is greater than 2nd element continue comparing & sorting
3. After each full pass the largest unsorted element moves to the end of array.
4. Repeat
5. Stop

4) => 500 -20 30 14 50

-20 500 30 14 50

-20 30 500 14 50

-20 30 14 500 50

-20 30 14 50 500

Pass 1 ↑

Pass 2

-20 30 14 50 500

-20 14 30 50 500

-20 14 30 50 500

-20 14 30 50 500.

- array is sorted.

- Algorithm

1. Start
2. Set min to 0
3. Swap value at the location of min
4. Increases min to point the next element
5. Repeat till sorted
6. Stop

Experiment 4

1.

```
#include <stdio.h>
int main ()
{
int a [ ]= { 7,3,5,1,9,8,4,6};
int n= size of (a)/size of (a[0]);
   int i,j,k;

for (i=1;i<n;i++)
 {
 k = a[i];
  j = i-1;
while (j >=0 && a [j ]>k)
{
a [j+1]= a [j];
   j =j-1;
}
 a [j+1]=k;
 if(i==2)
 {
 printf("Array:");
 for (int i=0; i<n;i++)
  printf("%d", a[i];
  printf("\n");
 }
}
 printf("Sorted array:");
 for(i=0;i<n;i++)
 printf("%d, a[i]);
```

2.

```
#include<stdio.h>
int setMin (int a [], int n){
        int m=a[0];
  for(int i=1;i<n;i++)
   {
   if (a[i]>m)
      m=a[i];
      return m;
   }
   }
void sort (int a[ ], int n, int exp)
 {
  int a[50], c[10] = {10}

  for(int i=0;i<n; i++)
    ([(a[i]/exp)%10]++;
  for(int i=1; i<10;i++)
   ([i] += ([i-1];

for(int i=n-1;i>=0;i--)
 {
 o [c[[(a[i]/exp)%d]-1] = a[i];
 c [(a[i]/exp)%10]--;
 }
 for[int 1=0; i<n; i++)
      a[i]=0[i];
    }
int main()
 {
 int a[] = {7,5,5,1,9,8,4,6};
```

```
int n=sizeof(a)/size of (a[0]);
    radixSort(a, b);
    printf("Sorted array:");
    for(int i=0; i<n; i++)
      printf("%d", a[i]);
    return 0;
    }
```

O/P:-

Sorted array:
1 3 4 5 6 7 8 9

3.

7 3 5 1 9 8 4 6

3 7 5 1 9 8 4 6

3 5 7 1 9 8 4 6

1 3 5 7 9 8 4 6

1 3 5 7 8 9 4 6

1 3 4 5 7 8 9 6

1 3 4 5 6 7 8 9

13/4

Experiment 4

```
#include <stdio.h>
int main () {
int i, j, n, r, temp = 0;

printf ("Enter the no of array elements:");
scanf ("%d" &n);
  int arr [n];
printf ("\n Enter the array elements:\n");
for (i=0; i<n; i++) {
scanf ("%d", & arr[i]); }

for (i=0; i<n; i++) {
scanf ("%d", &arr[i]); {
for (i = 1; i<n; i++) {
for (j = 0; j < i; j++) {

  if [arr[i] < arr (j)) {
    temp 2 arr(i);
for (k=i-1; k>=j; k--) {
  arr [k+1] = arr [k]; }{
  arr [j] = temp; } }
printf ("\n Sorted array : \n");
  for (i=0; i<n ; i++) {
printf ("%d", arr [i]; }
  return 0;
  }
```

3) What is o/p of insertion sort after the second iteration of following
7, 3, 1, 9, 4, 8, 6

Insertion sort : 7, 3, 1, 9, 4, 8, 6

1st iteration :- 7, 3, 1, 9, 4, 8, 6

2nd iteration 3, 7, 1, 9, 4, 8, 6

o/p :- 1, 3, 7, 9 4, 8, 6

) Sort list using radix sort.

1) 100, 255, 390, 4130, 956, 99, 5431

	0	1	2	3	4	5	6	7	8	9
100	100									
255						255				
390	390									
4130	4130									
956							956			
99										99
5431		5431								

	0	1	2	3	4	5	6	7	8	9
100		100								
390										390
4130				4130						
5431				5431						
225			225							
956							956			
99										99

~~100~~

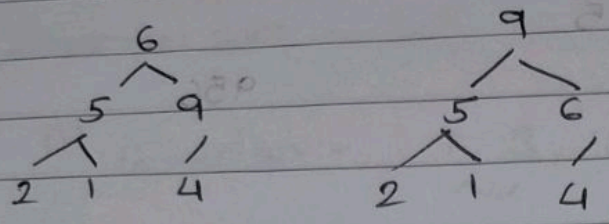
	0	1	2	3	4	5	6	7	8	9
100		100								
225			225							
4130		4130								
5431						5431				
956										956
390				390						
99	099									

	0	1	2	3	4	5	6	7	8	9
0099	0099									
0100	0100									
4130					4130					
~~22~~										
0255	0225									
0390	0390									
5431						5431				
956	0956									

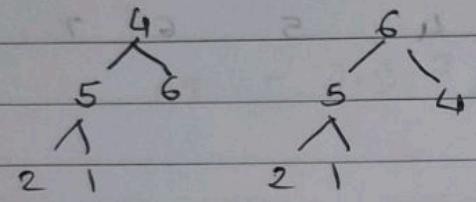
Sorted array: 99, 100, 225, 390, 956, 4130, 5431

5) Sort the following using heap sort

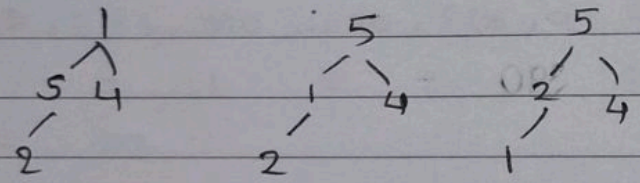
i) 6, 5, 9, 2, 1, 4



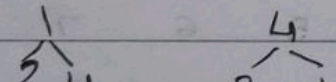
arr = {9, 5, 6, 2, 1, 4}
= {4, 5, 6, 2, 1, [9]}



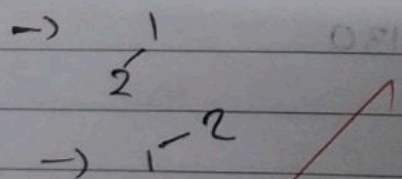
arr = {6, 5, 4, 2, 1, [9]}
= {1, 5, 4, 2, [6], [9]}



arr = {5, 2, 4, 1, [6], [9]}
= {1, 2, 4, [5], [6], [9]}

→ 

arr = {4, 2, 1 [5], [6], [9]}
= {1, 2, [4], [5], [6], [9]}

→ 

arr => {1, [2], [4], [5], [6], [9]}

Sorted array = 1, 2, 4, 5, 6, 9.

(6) Sort the following using shell sort

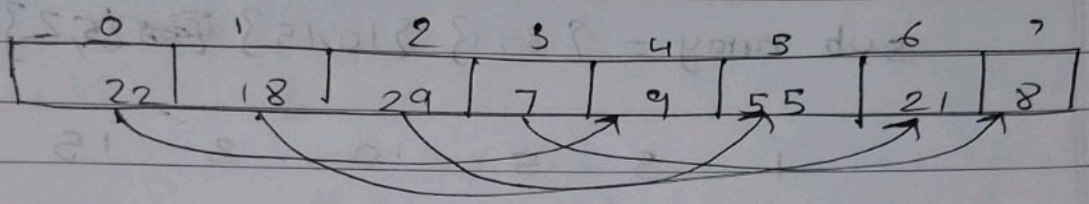
i) 22, 18, 29, 7, 9, 55, 21, 8.

i) 22, 18, 29, 7, 9, 55, 21, 8 →

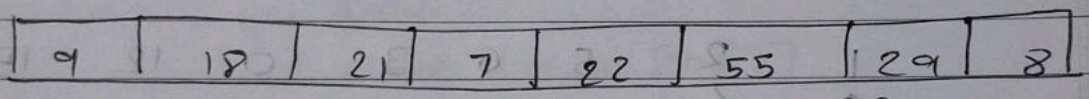
No. of elements = 8

Sequence = N/2, N/4, N/8 ... —

N/2 = 8/2 = 4



N/4 = 8/4 = 2



{ 9, 18, 21, 7, 22, 55, 29, 8 }

9, 21,

{ 9, 21, 22, 29 } { 18, 7, 55, 8 }

{ 9, 21, 22, 29, 7, 8, 15, 55 }

9 21 22 29 7 8 18 55

9 21 22 29 7 8 18 55

9 21 22 29 7 8 18 55

9 21 22 29 7 8 18 55

7 9 21 22 29 8 18 55

7 8 9 21 22 29 18 55

7 8 9 18 21 22 29 55

Sorted array : 7, 8 9 18 21 22 29 55

ii) 7, 10, 15, 12, 1, 5, 2, 6

No. of elements = 8

seq : n/2 , n/4 , n/8 ...

0	1	2	3	4	5	6	7
3	10	15	12	1	5	2	6

sub array = {3, 1} {10, 15} {5, 2} {12, 6}

1 3 5 10 2 15 6 12

sub arrays :- {1, 5, 2, 6} {3, 10, 15, 12}

1 2 5 6 3 10 12 15

1 2 5 6 3 10 12 15

1 2 5 6 3 10 12 15

12 5 6 3 10 12 15

Sorted : 1, 2, 3, 5, 6, 10, 12, 15

classmate
Date
Page

Experiment 6

1) Postfix expression [infix -> postfix]

```
#include<stdio.h>
#include<ctype.h>
  char stack [100],
       top = -1;
      void push (char x)
    { stack [++top] = x;
     }
      char pop (){
      if (top = 2-1)
         return -1;
       }else{
       return stack [top--]
       }
       int priority (char x)
     { if(x==(')
      return 0;
     if (x=='+'||n=='-')
      return 1;
      if (x=2'*'||x=='y')
         return 2;
         return 0;
       }
       char exp[100];
        char e,2a;
      printf("Enter the expression:");
      scanf("%.s",&exp);
       printf("\n");
```

```
e=exp;
while(*e!="0")
{if (isnum(*e))
printf("%c",*e);
}else if{
(*e=="(");
push(*e);
else if(*e==')')
{ while((n=pop())!='(')
printf("%c",x);
}
else{
while (priority(stack[top])>=priority
printf("%c",pop());
push(*e);
}
eAt;
}
while(top!=-1)
{
printf("%c",pop());
}
return 0;
```

2] Convert infix to prefix

Infix :- A+(B*C-(P/E^F)*G)*H

Prefix = H*)G*)F^E/D(-C*B+A

symbol	Stack	output
H	-	H
*	-*	
)	-*)	
G	-*)	HG
*	-*)*	
)	-*)*)	
F	-*)*)^	HGF
^	-*)*)^	
E	-*)*)^	HGFE
/	-*)*)^/	HGFE
D	-*)*)/	HGFE^D
(	-*)-	HGFE^D
-	-*)-	HGFE^D/
G	-*)-*	HGFE^D/*
*	-*)-*	HGFE^D/*C
B	-*)-*	
(	-*)-*	HGFE^D/*CB
+	-+	HGFE^D/*CB*-*A
A	-+	HGFE^D/*CD*-*A+

Prefix O/P :- +A*-*BC*/D^EFGH

WAP to evaluate infix or postfix expression

```
#include<stdio.h>
#include<ctype.h>
#include<stdio.h>
#define size no.

int pop()
void push(int);
char postfix[size];
int stack[size], top=-1;
int main()
int i,a,b,result,pEval;
  char ch;
for(i=0;i<size:i++)
  {
  stack[i]=-1;
}
print("\n Enter a postfix expression")
  scanf("%.s,postfix);
 for(i=0; postfix[i]!='\0';i++)
  ch=postfix[i];
  if(;is digit(ch))
    {push(n-'0');
  }
else if(ch=='+'||ch=='-'||ch=='*'||ch=='/')
    b=pop();
    A=pop();

  switch(ch)
  case '+';
```

```
result = a+b;
    break;
case '-';
 result=a-b;
  break;
 case '/';
  result=a/b;
  break;
  case '*';
   result=a*b;
    break;
  case '%':
   result=a%b;
    break;
    }
    Push (result);
     }}
    pfval=pop();
    printf"\nThe postFix evaluation is:%d\n",pfval;
    return 0;
     }

    void push (int n)
    { if top <(size-1)
    {
    stack [++top ]=n;
    } else {
    printf("stack Full!\n");
        exit (-1);
     }}
```

```
    exit(-1);
}}
  int pop(){
    int n;
    if(pop>-1)

  n = stack[top]
      stack[Top--]
    return n;
  }
  else {
  printf("Stack is empty:\n");
        exit(-1)
    }
  }
```

Evaluate prefix expression:-

+ - * + 1 2 / 4 2 1 $ 4 2

prefix = 2 4 $ 1 2 4 / 2 1 + * - *

Symbol	OP-1	OP-2	Result	O/P
2				2
4				2, 4
$	4 $	2	16	16
1				16, 1
2				16, 1, 2
4				16, 1, 2, 4
/	4 /	2	2	16, 1, 2
2				16, 1, 2
1				16, 1, 2, 1
+	1 +	2	3	16, 1, 2, 3
*	3 *	2	6	16, 1, 6
-	6 -	1	5	16, 5
+	5 +	16	21	21

- Algorithm

Infix to prefix postfix

1) Step 1 : Scan infix expression from left to right
Step 2 : Operand → add to postfix directly
Step 3 : "(" → push to stack
Step 4 : ")" pass from stack until "(" is found.
Step 5 : Operator → for all operators from stack with greater or equal precedence, then push current operator.
Step 6 : After scanning → pop remaining to postfix

2) Infix to prefix

1 : Reverse the infix expression.
2 : Swap '(' with ')' in the reversed expression
3 : Convert the new expression to postfix using same algorithm as above.
4 : Reverse result → Thats the prefix expression

3) Evaluation of postfix

1 : Scan left → right
2 : Operand → push into stack
3 : Operator → push top 2 values apply operator push result back.
4 : [redacted]

At end → only one value remains in stack result.

Evaluate Prefix.

1: Scan expression right -> left

2: Operand -> push into stack

3: Operator -> pop top 2 values apply operation push result back.

4: At end -> only value remains in stack result.

Experiment 5

1)
```
#include <stdio.h>
#include <stdlib.h>

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

struct node* head = NULL;

void create_list();
void display_list();
void add_at_beginning();
void add_at_end();
void add_at_position();
void search_node();
void delete_node();
void reverse_display(struct node * node);

int main() {
    int choice;

    while (1) {
    printf("\n Menu :\n");
    printf(" 1. Create list \n");
    printf("2. Display lists \n");
    printf(" 3. Add at beginning \n");
    printf("4. Add at end \n");
    printf("5 Add at position \n");
```

```
printf("6, Search node \n");
printf("7, Delete node \n");
printf"8, Exit \n");
printf("9, Reverse display \n");
printf("Enter choice:");
scanf("%d, &choice:");

switch(choice){
case 1: create_list();break;
case 2: display_list();break;
case 3: add_at_beginning();
  break;
case 4: add_at_end();
  break;
case 5: add_at_position();
  break;
case 6: search_node();break;
case 7: delete_node();
  break;
case 8: exit(0)
case 9:

if (head==NULL){
  printf("List is empty:\n");
    } else {
  printf("Element in reverse order:");
  reverse_display(head);
    printf("\n");
    }
    break;
default printf("Invalid choice, my again.\n");
```

```
}
return 0;
}
void create_list(){
  int n, i;
  struct node *temp, *tail;

  printf("How among nodes ");
  scanf("%d", &n);

  head = NULL;

  for(i=0; i<n; i++)
  temp = (struct node *) malloc(sizeof(struct node));
  printf("Enter data for node %d: ", i+1);
  scanf("%d", &temp->data);
  temp->next = NULL;

  if(head == NULL){
    head = temp;
    tail = temp;
    }else {
    tail->next = temp;
      tail = temp;
    }
  }
printf("List created with %d nodes\n", n);
}
void display_list(){
    struct node* temp = head;
      if{temp == NULL){
```

```
        printf("List is empty:\n");
        return;
    }
    printf("List elements:");
    while(temp!=NULL){
    printf("%d",temp->data);
    temp=temp->next;
    }
    printf("\n");
}
void add_at_beginning(){
    struct node* temp=(struct node*)malloc(sizeof(
    struct node));
    printf("Enter data to add at beginning:\n");
    scanf("%d",&temp->data);
    temp->next=head;
    head=temp;
    printf("Node added at beginning:\n");
}
void add_at_beginning(){
    struct node* temp=(struct node*)malloc
    (sizeof(struct node));
    struct node* current=head;
printf("Enter data to add at end:");
scanf("%d",&temp->data);
    temp->next=NULL;
if(head==NULL){
    head=temp
    }else{
```

```
while (current -> Next != NULL)
        current = current -> Next;
        current -> next = temp;
    }
printf ("Node added at end \n");

void add_at_position () {
    int pos ;
    struct node *temp, *current = head;

printf("Enter position to add node (starting from 1): ");
scanf("%d", &pos);
    if (pos <= 0) {
    printf("Invalid position !\n");
        return ;
    }
temp = (struct node*) malloc (sizeof (struct node));
    printf("Enter data to add:");
    scanf("%d", &temp -> data);

    if (pos == 1) {
    temp -> next = head;
        head = temp;
    printf("Node added at position \n");
        return ;
    }
for (i = 1; i < pos - 1 && current != NULL; i++)
    current = current -> Next;

    if (current == NULL) {
    printf(" Position out of range \n");
    free(temp);
```

```
        return;
    }
    temp->next = current->next;
    current->next = temp;
    printf("Node added at position : %d\n", pos);

void search_nodes() {
    int value, pos = 1;
    struct node * current = head;
    if(head == NULL) {
        printf("List is empty\n");
        return;
    }
    printf("Enter value to search:");
    scanf("%d", &value);

    while(current != NULL) {
    if(current->date == value) {
    printf("Value %d found at position %d", pos);
        return;
    }
        current = current->next;
        pos++;
    }
    printf("Value of %d not found in list\n", value);
    void delete node() {
        int pos;
        struct node *current = head * temp;

        if (head == NULL) {
```

```
        printf("List is empty:\n");
            return;

    printf("Enter pos to delete node ");
        scanf("%d",&pos);
        if(pos<=0){
            printf("Invalid\n");
            return;

for(i=1;i<pos-1 && current!=NULL;i++)
    current=current->Next;

    if(current==NULL||current->Next==NULL){
    printf("Position out of range\n");
        return;
    }
    temp=current->Next;
    current->next=temp->next;
    free temp;
    printf("Node at position %d deleted\n",pos);
}
void reverse_display(struct node *node){
    if(node==NULL)
        return;
    reverse_display(node->next);
    printf("%d",node->data
    }
    return 0;
}
```

O/P:

Menu:

1. Create list
2. Display list
3. Add at beginning
4. Add at end
5. Add at position
6. Search node
7. Delete node
8. Exit
9. Reverse display

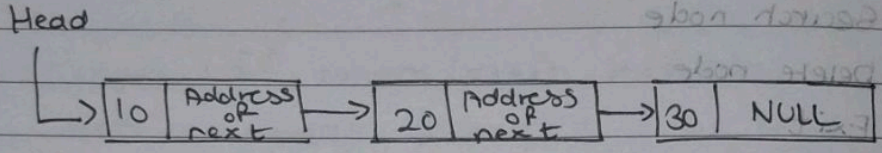
Enter choice:

2. List & explain basic terminology of linked list with example

Term	Meaning
Node	Basic unit of list containing data & pointer
Head	Pointer to first node in linked list
Data	Actual value stored in data
Next	Pointer to next node
NULL	indicates the end of the list
Pointer	A variable that holdes the memory of next pointer

Term	meaning
Traversal	Visits every node
Insertion	Adds a new node
Deletion	Removing a node from list.

Ex:-



- Diff blw linked list & array

	Linked list	Array
1)	Memory allocation is dynamic.	a) Memory allocation is fixed
2)	Element access is sequential access via pointers	b) Element access is direct via index.
3)	Insertion/Deletion just pointer has to be changed.	c) For insertion/deletion pointer is shifted.
4)	Efficient use of memory	d) May waste memory
5)	Extra space is needed for storing pointers.	e) No extra space is needed

Experiment 7

```
#include <stdio.h>
#include <stdlib.h>

void push ();
void pop ();
void display ();
int s[max];
int top = -1;
int main() {
    int choice;
    printf ("MENU:");
    printf ("1. Push:");
    printf ("2. Pop");
    printf ("3. Display:");
    printf ("4, Exit:");
    do {
    printf ("Enter your choice :");
    scanf ("%d", &choice);
    switch (choice) {
    case 1: push ();
        break;
    case 2: pop ();
        break;
    case 3: display ();
        break;
    case 4: exit (0);
    default:
```

```
printf("Invalid choice!");
}
}
while (choice!=4);
return 0;
}
void push (){
int element;
if(top==max-1){
printf("overflow");
}else{
printf("Enter value");
scanf("%d",&element);
top=top+1;
s[top]=element;
}
}
void pop (){
int item;
if (top==-1){
printf("underflow");
}else{
item=s[top];
top--;
printf("The deleted element is %d",item);
}
}
void display(){
if(top==-1){
printf("Stack empty");
}else{
```

Experiment 9

Queue:

```
#include <stdio.h>
#include <stdlib.h>
#define size 5
int Queue [size]
int front = -1, rear = -1;
void insert (){
    int value;
int (rear = size - 1) {
printf ("Queue is full overflow \n");
}else;
printf("Enter value to insert:");
scanf("%d", &value);
if("%d", &value);
if (front==1) {
front=0
rear++;
Queue[rear]=value;
printf(" inserted %d \n", vave);
}
}
void delete(){
if(front==-1 || front>rear) {
printf("Queue is empty (underflow) \n");
}else {
printf("Deleted %d\n", Queue[);
```

```
void display() {
if (front == -1 || front > rear) {
printf("Queue is empty \n");
  }else {
  printf("Queue elements");
for(int i=front; i<=rear; i++){
  printf("%d", Queue[i]);
 }
 printf("\n");
}
}
switch (choice) {
case 1: insert();
        break;
case 2 : delete 1;
        break;
case 3 :
        display()
        break;
case 4 : exit(0)
        default;

printf("; Invalid choice \n");
   }
   return 0;
```

Date
Page

O/P:-

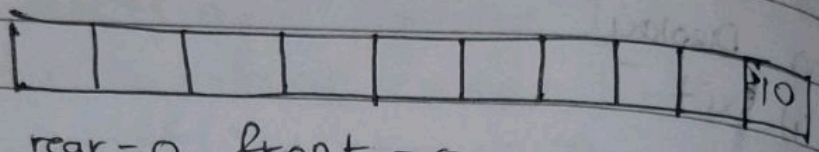
1) Insert
2) Delete
3) Display
4) Exit

Enter your choice : 1
Enter value to insert : 10
inserted 10

Mean:....
Enter your choice = 1
Enter value to insert = 26
inserted 26

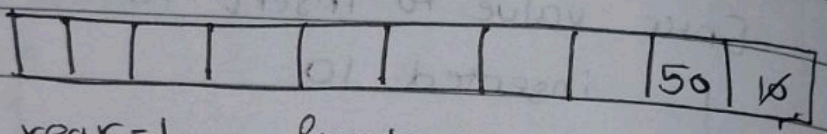
2) Perform following operation on linear queue in array, size 10 with this operation (10), insert(50), delete, insert(100)

-> Insert(10)



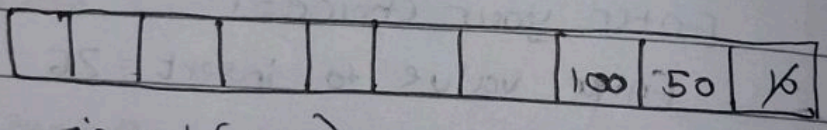
rear = 0, front = 0

-> insert 50

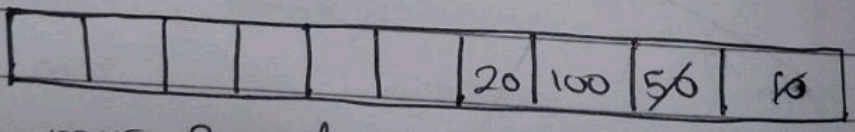


rear = 1 front = 0

insert(100)

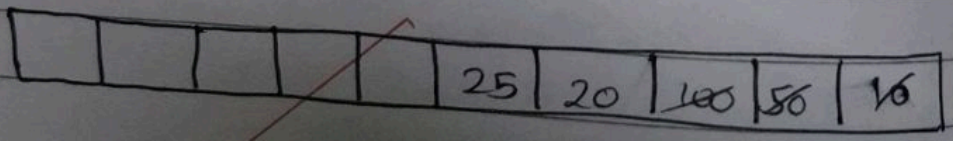


insert(20)

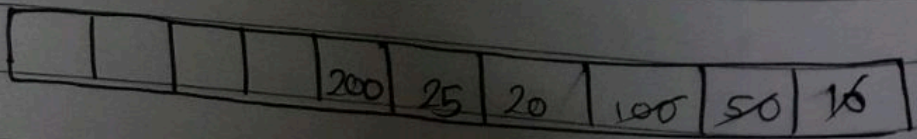


rear = 2 front = 0

Insert (25)



Insert (200)



1) Explain concept of priority queue.

-> A priority queue is a special type of Queue where each element is assigned a priority & element are derived based on their priority not just their average order.

2) Algorithm for insertion & deletion of linear queue.

-> Insertion

1) Check if queue is in overflow condition.
2) If no overflow condition check if front is equal to -1 & change it to 0.
3) Increase rear by 1, equate the rear to value entered.
4) Print required value.

13/11

Expt 10

- Circular queue

```
#include<stdio.h>
#include<stdlib.h>
#define Max 6
    int front=-1;
    int rear=-1;
    int queue[Max];

void enqueue(int val);
    void dequeue();
    void display();
    int main(){
    int choice value;
    while (1){
printf("....Menu....");
printf("1 enqueue\n");
printf("2 dequeue\n");
printf("3.display\n");
printf("4 exit\n");
printf("Choose option:");
scanf("%d",&choice);
switch
case1:
printf("Enter value to enqueue:");
scanf("%d",&value);
break;
```

```
case 2:
dequeue();
break;

case 3:
display();
break;
case 4:
exit(0);
break;
default:
printf("Invalid input");
}
}
return 0;
}
printf("%d inserted successfully", val);
}
}
void dequeue(){
if(front==-1){
printf("Queue empty");
}else{
int item=Queue[front];
if(front==rear){
front=rear=-1;
}else{
front=(front+1)%Max;
}
```

```
printf("%d, dequed", item);
}
}

void display (){
if( front==-1){
printf("Queue empty");
}else {
int i=front; of("Queue elements:");
while(1) {
printf("%d", Queue[i]);
if(i==rear){
break;
}
i=(i+1)%max;
}
}
return 0;
}
```

O/P:-

```
..... Meno .....
1) Queue
2) Dequeue
3) Display
a) exit
Choose option : 1
Enter value to enqueue : 12
12 enqueue successfully
```

Expt 11

Operations on binary tree

```
#include <stdio.h>
#include <stdlib.h>
struct node {
  int data{
  struct node *left;
  struct node *right;
}
struct node *create(int val)
struct node *insertleft(struct node* root,int val);
struct node *insertright(struct node*root,int val);
void deletesubtree(struct node* root);
void deleteleft(struct node* root);
void deleteright(struct node* root);
void display(struct node *root);

int main(){
structnode *root= create(1);
insertleft(root,2);
insertright(root,3);
insertleft(root->left,4);
insertright(root->right,5);
insertleft(root->left,6);
insertright(root->right,7);
insertleft(root->left->left,8);
```

```
printf("Original tree:");
display(root);
printf("\n");

deleteleft(root)
printf("Tree after deleting left subtree of
        root:");
display(root);
return 0;
}

struct node * create(int val){
  struct node *new=(struct node *)malloc
        (sizeof(struct node));
  if(new==NULL){
  printf("Memory alloc failed");
  exit(1);
  }

new->data=val;
new->left=NULL;
new->right=NULL;
return new;
}
struct node * insertright(struct node*root,int val)
if(root==NULL){
printf("Cant insert in NULL root");
return NULL;
}
```

```
root -> right = create(val);
        return root -> right;
    }
    deletesubtree(root -> right);
        root -> right = NULL;
    }
void display(struct node * root){
        if(root == NULL){
            return;
        }
    printf("%d ", root -> data);
    display(root -> left);
        display(root -> right);
    }
```

O/P:-

Original tree : 1 2 4 8 5 3 6 7

Left subtree deleted.

Tree after deletion: 1 3 6 7

Q2] Pre order, Postorder, Inorder Traversal.

```
#include<stdio.h>
#include<stdlib.h>

struct node {
    int data;
    struct node *left;
    struct node right;
```

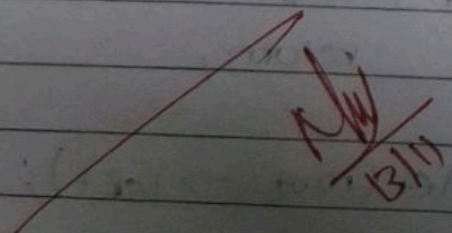
```
void preorder(struct node * root);
void postorder (struct node * root);
void inorder (struct node * root);

    int main () {

  int choice;
    while (1) {

  printf (" ---Menu ---\n");
  printf(" 1. Pre-order traversal \n");
  printf(" 2. Post-order traversal \n");
 printf(" 3. In-order traversal \n");
  printf ("4. Exit ");
  printf ("\n Choose option: ");
   printf(" %d", &choice);
    switch (choice) {
      case 1:
        preorder (root);
          break;

      case 2:
        postorder (root);
          break;

      case 3:
        inorder (root);
          break;
```

```
        exit(0);
        break;

    default:
        printf("Invalid input");
        }
    }
    return 0;
    }

void preorder(struct node *root){
    if(root==NULL){
        return;
    }
    printf("%d",root->data);
    preorder(root->left);
    preorder(root->right);
}
void postorder(struct node *root){
        if(root==NULL){
            return;
    }
    postorder(root->left);
    postorder(root->right);
    printf("%d",root->data);
    }
```

```
void inorder(struct node *root){
    if(root==NULL){
        return;
    }
    inorder(root->left);
    printf("%d",root->data);
    inorder(root->right);
```

O/P :-

```
---Menu---
1. Preorder traversal
2. Postorder traversal
3. Inorder traversal
Choose option: 1
1 2 4 8 5 6 7

---Menu---
Choose option: 2
8 4 5 2 6 7 3 1
```



Expt 12

->
```
#include<stdio.h>
#define v5

void int(int arr[v][v]){
    int i,j;
for(i=0;i< v;i++)
 for(j=0;j<v;j++)
   arr[i][j]=0;
}
void insert edge (int arr[v][v],int :;
     int j>h;
     arr[i][j]=1;
     arr[j][i]=1;
     }
void print adj Matrix (int arr[v][v]){
     int i,j;
 printf("Adjacencg Matrix :\n);
   for(i=0;i<v;i++){
    for(j=0;j<v;j++){
   printf("%d",arr[i][j]);
     }
     printf("\n");
      int main (){
   int adj Matrix[v][v];

      int (adj Matrix);
```

```
insert Edge (adj Matrix 0,1);
insert Edge (adj Matrix 0,4);
insert Edge (adj matrix 1,2);
insert Edge (adj matrix 1,3);
insert Edge (adj matrix 1,4);
insert Edge (adj matrix 2,3);
insert Edge (adj matrix 3,4);

print Adj Matrix (adj Matrix);

return 0;
}
```

O/P:-

Adjacency matrix

0	1	0	0	1
1	0	1	1	1
0	1	0	1	0
0	1	1	0	1
1	1	0	1	0

3/11